ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE SISTEMA DISTRIBUIDO PARA EL CONTROL DE ASISTENCIA DEL PERSONAL ADMINISTRATIVO DE LA ESCUELA POLITÉCNICA NACIONAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

TAI LIN WANG

lin.wang@epn.edu.ec

DIRECTOR: ING. RAÚL DAVID MEJÍA NAVARRETE, M.Sc.

david.mejia@epn.edu.ec

Quito, abril 2019

AVAL

Certifico que el presente trabajo fue desarrollado por Tai Lin Wang, bajo mi supervisión.

Ing. Rául David Mejía Navarrete, M.Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Tai Lin Wang, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Tai Lin Wang

DEDICATORIA

Dedico el presente trabajo a mis padres, por enseñarme la mezcla perfecta entre libertad y responsabilidad. ¡Son los mejores!

A mi amada novia Kathy Simba por llenarme el corazón de alegría y felicidad y por todo el apoyo moral brindado a lo largo de todos estos años. Te amo.

A mis mejores amigos Oso y Java, por estar ahí cuando más los necesitaba.

A mi Director de Trabajo de Titulación M.Sc. David Mejía, quien me apoyó, defendió y guió con paciencia durante todo el trayecto del trabajo.

A todos mis profesores quienes me brindaron los fundamentos teóricos y prácticos para que pueda crecer profesionalmente.

A las futuras generaciones curiosas de conocimiento.

Tai Lin Wang

AGRADECIMIENTO

Agradezco a Dios por protegerme a lo largo de toda mi existencia.

A mis padres Wang Chin Chun y Chang Yueh Hsia por todos sus sacrificios, por confiar en mi, por darme todo su amor y apoyo incondicional, por todos los valores y principios de vida que me han inculcado. "*Wo ái nimen*".

A mi novia Kathy Simba, por ser mi dúo en los caminos de la vida y por "supportearme" amorosamente en todo este trayecto ¡Muchísimas gracias!.

A mis amigos, amigas, compañeros y compañeras, gracias por tantas buenas experiencias vividas durante todos estos años.

Deseo expresar un especial agradecimiento a mi Director y profesor M.Sc. David Mejía por la ayuda, la dedicación y las enseñanzas en estos últimos semestres. Sin usted, este trabajo no se hubiera podido llevar a cabo.

Agradezco a todos mis profesores de la escuela, del colegio y especialmente de la universidad por compartir todos sus conocimientos a lo largo de mi formación académica y profesional.

Por último, le doy gracias a la República de Ecuador y a todos sus ciudadanos por ofrecerme su amistad y compañerismo que me hace sentir como en casa.

Tai Lin Wang

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

El presente trabajo consiste en el desarrollo de un prototipo de sistema para controlar la asistencia del personal administrativo de la Escuela Politécnica Nacional. El prototipo se encuentra conformado principalmente por tres partes: un Cliente Android, un Cliente Gestor y un servicio WCF (*Windows Communication Foundation*) basado en REST (*Representational State Transfer*).

El presente documento se encuentra constituido por cuatro capítulos:

En el primer capítulo se presentan conceptos sobre el servicio WCF basada en REST, sistema operativo Android, API (*Application Programming Interface*) de huella dactilar Android, lectores de huellas, mecanismos de posicionamiento y monitoreo de ubicación Android, bases de datos. Posteriormente se presenta la metodología ágil SCRUM.

El segundo capítulo presenta la metodología empleada para el desarrollo del presente trabajo, el cual consiste en dos partes: diseño e implementación. La parte de diseño presenta el establecimiento de requerimientos de usuario, *backlogs* y diseño de los componentes del sistema. En la parte de implementación se presenta la codificación de la lógica de los componentes.

En el tercer capítulo se presentan los resultados de las pruebas realizadas a los componentes del prototipo. Además, se presentan los resultados de encuestas de validación realizadas por el personal administrativo.

El cuarto capítulo contiene conclusiones y recomendaciones obtenidas durante la realización del presente trabajo.

Por último, se tienen los Anexos. Entre los Anexos se tienen: encuestas de requerimientos, historias de usuario, *sketches*, *wireframes*, *scripts* de la base de datos, solución del servicio WCF, solución del cliente Android y su instalador, solución del Cliente Gestor y su correspondiente instalador, encuesta de validación y manual de usuario. Debido a la extensión, todos los anexos se han incluido en el disco compacto adjunto a este documento.

PALABRAS CLAVE: control de asistencia, servicio WCF basado en REST, Android, monitoreo de ubicación, huella dactilar.

ABSTRACT

The present project consist of the development of a prototype of system that perform attendance control of the administrative staff at the National Polytechnic School. This prototype is conformed by three main parts: an Android Client, a Manager Client and a WCF (Windows Communication Foundation) service based on REST (Representational State Transfer).

The current document is constituted by four chapters:

The first chapter presents concepts about WCF service based on REST, Android operating system, Android fingerprint API (Application Programming Interface), fingerprint reader, Android location monitoring and data bases. Subsequently, it presents the agile methodology SCRUM.

The second chapter presents the methodology used for the development of the current project, which consist of two parts: design and implementation. The first part presents the settlement of users requirements, backlogs and the design of the system's components. The second part presents the logic coding of the system's components.

The third chapter presents the test results done on the prototype's components. Also, it presents the result of validation surveys completed by the administrative staff.

The fourth chapter contains conclusions and recommendations obtained during the development of the current project.

Finally, there are annexes such as: users requirement survey, user's history, sketches, wireframes, database's script, WCF service solution, Android Client solution and it's installer, Manager Client solution and it's corresponding installer, validation survey and user's manual. Because of the size, all annexes were included in the CD attached at the end of this document.

KEYWORDS: Control attendance, WCF service based on REST, Android, location monitoring, fingerprint.

1. INTRODUCCIÓN

A inicios del año 2019, en la Escuela Politécnica Nacional, se usan herramientas biométricas[1] para el control de asistencia de los trabajadores, que permiten tener registros de horas de ingreso y salida. Además el uso de estas herramientas representan un requisito normativo en todas las instituciones públicas del Ecuador [1]. Sin embargo, estas herramientas pueden resultar no tan ágiles al momento de tener grandes cantidades de personas requiriendo hacer uso de la herramienta de control. Adicionalmente, las herramientas actuales obligan a los trabajadores a trasladarse al lugar donde se encuentren instaladas dichas herramientas, para marcar su asistencia.

El uso de aplicaciones móviles junto con sistemas distribuidos[2] pueden integrarse como alternativa del sistema de control de asistencia actual de la Institución, con la finalidad de resolver los inconvenientes por la falta de agilidad de las herramientas biométricas en horas pico y las complicaciones de tener herramientas de control estáticos.

Los beneficios de estas tecnologías incluyen: movilidad para el trabajador, flexibilidad de registrar la asistencia desde cualquier punto con acceso a la red Institucional, reducción de viajes innecesarios, reducción de colas de espera en horas pico.

En este capítulo se presentarán los objetivos generales y específicos, el alcance y el marco teórico, el cual tratará los tópicos referentes a servicios WCF[3] basado en REST[4], sistema operativo Android, API[5] de huella dactilar para Android, lectores de huella en Android, mecanismos de posicionamiento en Android y las bases de datos.

El enfoque de este Proyecto Integrador es desarrollar un sistema que permita controlar la asistencia del personal administrativo de la Escuela Politécnica Nacional para ofrecer una alternativa de control de asistencia ágil y flexible.

[1] Herramientas biométricas: son dispositivos que permiten reconocer la identidad de una persona mediante huellas dactilares, iris, voz, entre otros.
[2] Sistemas Distribuidos: es un conjunto de aplicaciones interconectados entre sí a través de una red para realizar alguna tarea en particular.
[3] WCF (*Windows Communication Foundation*): es un *framework* de .NET para crear aplicaciones distribuidas.
[4] REST (*Representational State Transfer*): es una arquitectura para implementar servicios web
[5] API (*Application Programming Interface*): es un conjunto de métodos ya implementados para el uso del desarrollador.

1.1 Objetivos

El objetivo general de este Proyecto Integrador es desarrollar un sistema que permita controlar la asistencia del personal administrativo de la Escuela Politécnica Nacional.

Los objetivos específicos de este Proyecto Integrador son:

- Analizar el funcionamiento de la API de huella dactilar Android, el sistema operativo Android, el lector de huellas de Android, el mecanismo de posicionamiento en ambientes *outdoor*, el servicio WCF JSON[6] basado en REST y las bases de datos.
- Diseñar los componentes que conforman el sistema de control de asistencia del personal administrativo: Servicio WCF JSON basado en REST, bases de datos, código de control de asistencia, cliente Gestor y cliente Android.
- Implementar los componentes del sistema de control de asistencia del personal administrativo.
- Analizar los resultados de las pruebas realizadas en el sistema de control de asistencia del personal administrativo.

1.2 Alcance

El prototipo de sistema distribuido para el control de asistencial del personal administrativo va a estar formado por tres componentes principales: Cliente Android, Servidor y un Cliente Gestor. Se muestra una descripción general del sistema en la Figura 1.1.

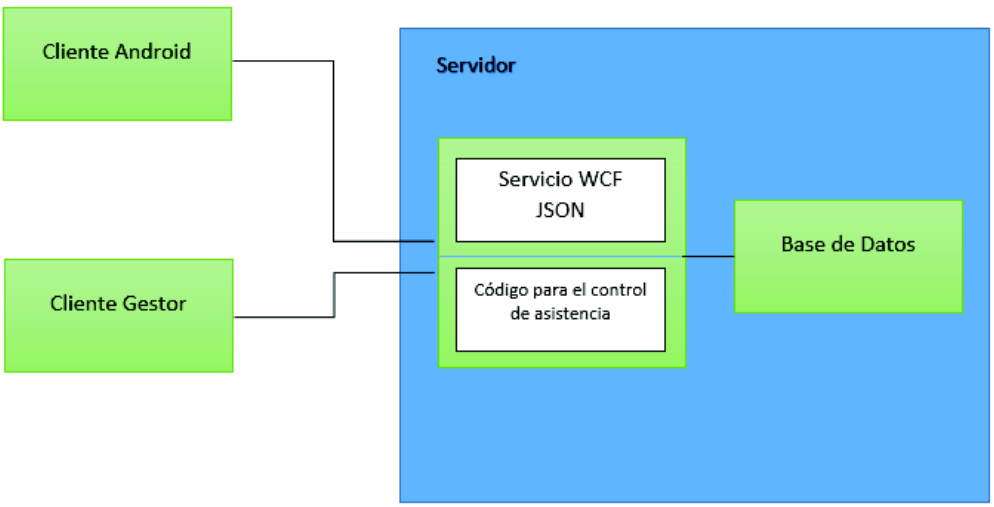


Figura 1.1.- Descripción general del sistema

[6] JSON (*JavaScript Object Notation*): es un formato de representación de datos.

El Cliente Android va a permitir consultar el horario de trabajo del empleado administrativo, así como permitirá marcar la hora de entrada y salida mediante la huella dactilar. Además, se permitirá el monitoreo de ubicación del Cliente Android cuando este se encuentre en las instalaciones del campus. También se emitirán alertas cuando el Cliente Android salga fuera del campus en horario de trabajo.

En el Cliente Gestor se va a poder observar reportes del personal administrativo con respecto a las horas que han ingresado y salido del trabajo, el total de horas trabajadas, también se podrán configurar las jornadas de trabajo que los miembros del personal administrativo van a tener, así como se podrá visualizar las alertas cuando el Cliente Android salga fuera del campus politécnico.

El Servidor alojará a un servicio web JSON, el cual tendrá una interfaz para que los clientes hagan pedidos y obtengan respuestas, además, va a tener el código que controlará la asistencia a los trabajadores y guardará en una base de datos toda la información del sistema.

Se aplicará la metodología SCRUM[7] para la realización del prototipo.

Se harán las pruebas del prototipo solo en la Facultad de Ingeniería Eléctrica y Electrónica de la Escuela Politécnica Nacional.

Se utilizará para las pruebas a un conjunto de al menos 10 personas del personal administrativo de la Facultad de Ingeniería Eléctrica y Electrónica, a quienes se les pedirá que instalen el aplicativo y se aplicará una encuesta para conocer su opinión sobre el prototipo.

El prototipo será una alternativa a los sistemas de control de asistencia que se implementan y no se buscará la integración con el sistema existente de la Escuela Politécnica Nacional.

El alcance del prototipo se limitará al alcance establecido por la red inalámbrica de la Escuela Politécnica Nacional.

1.3 Marco Teórico

Servicio WCF

WCF (Windows *Communication Foundation*) es un *framework*[8] que forma parte de .NET[9] que permite la construcción, configuración y despliegue de aplicaciones distribuidas, de

[7] SCRUM: es una metodología ágil para gestionar el desarrollo de software.
[8] *Framework*: es una estructura conceptual y tecnológica compuesto de herramientas o módulos que sirven como base para el desarrollo de software.
[9] .NET: es un *framework* de Windows para el desarrollo de Software.

una manera fácil y unificada. Además, posee un conjunto de clases para construir aplicativos orientados al servicio los cuales permiten que las aplicaciones de múltiples plataformas puedan interconectarse entre sí.

Los servicios WCF se basan en el patrón de comunicación de tipo solicitud/respuesta con soporte de comunicación *simplex*[10] o *full duplex*[11]. Además soporta diferentes protocolos de comunicación como HTTP[12], TCP[13], MSMQ[14]. También se tiene gran variedad de opciones de alojamiento que pueden ser a través de: IIS[15]; cualquier aplicativo .NET, entre otros.

Los principales beneficios que comprenden los servicios WCF son: interoperabilidad entre diferentes infraestructuras mediante el uso de formatos o lenguajes estandarizadas como XML[16], JSON, para el intercambio de información. Estos servicios están compuestos por una arquitectura de capas, por lo que permite escalabilidad y flexibilidad ante cambios que puedan suceder dentro del servicio [2].

Arquitectura WCF

Los servicios WCF están conformados por una arquitectura en capas como se muestra en la Figura 1.2.

En la capa aplicación se tienen todas las funcionalidades que el servicio ofrece, además, en esta capa se implementan todos los métodos u operaciones definidas dentro de la capa contrato, en un lenguaje de programación como C#.

En la capa de contrato, se especifican las operaciones y métodos que va a tener el servicio, así como los tipos de datos que se van a retornar. En esta capa se tiene cuatro tipos de datos: Contrato de datos, que especifica qué datos van a ser intercambiados por el servicio y también involucra los tipos de parámetros de entrada y de retorno de las operaciones; contrato de mensajes, que sirve para controlar los parámetros de mensajes SOAP[17];

[10] Comunicación *Simplex*: es una forma de comunicación unidireccional en el cual el receptor no puede responder al transmisor.
[11] Comunicación *Full Duplex*: es una forma de comunicación bidireccional que se puede transmitir y recibir información de manera simultánea.
[12] HTTP (*HyperText Transfer Protocol*): es un protocolo de solicitud/respuesta para el intercambio de recursos a través de la red.
[13] TCP (*Transport Control Protocol*): es un protocolo de transmisión confiable y orientada a conexión.
[14] MSMQ (*Microsoft Message Queuing*): es un mecanismo de intercambio de mensajes en el cual se guardan los mensajes en una cola de espera para su posterior recuperación.
[15] IIS (*Internet Information Services*): es una plataforma de Windows para alojar aplicaciones web.
[16] XML (*Extensible Markup Language*): es un lenguaje estandarizado por medio de etiquetas, desarrollado por W3C que sirve para describir información de manera estructurada.
[17] SOAP (*Simple Object Access Protocol*): protocolo en formato XML que permite comunicación entre aplicaciones a través de mensajes por medio de una red.

contrato de servicio, que sirve para proporcionar información sobre protocolos de comunicación del servicio, tipos de datos de mensaje, ubicación y funcionalidades que ofrecen los extremos; vinculación y políticas, que provee información sobre protocolos de seguridad.

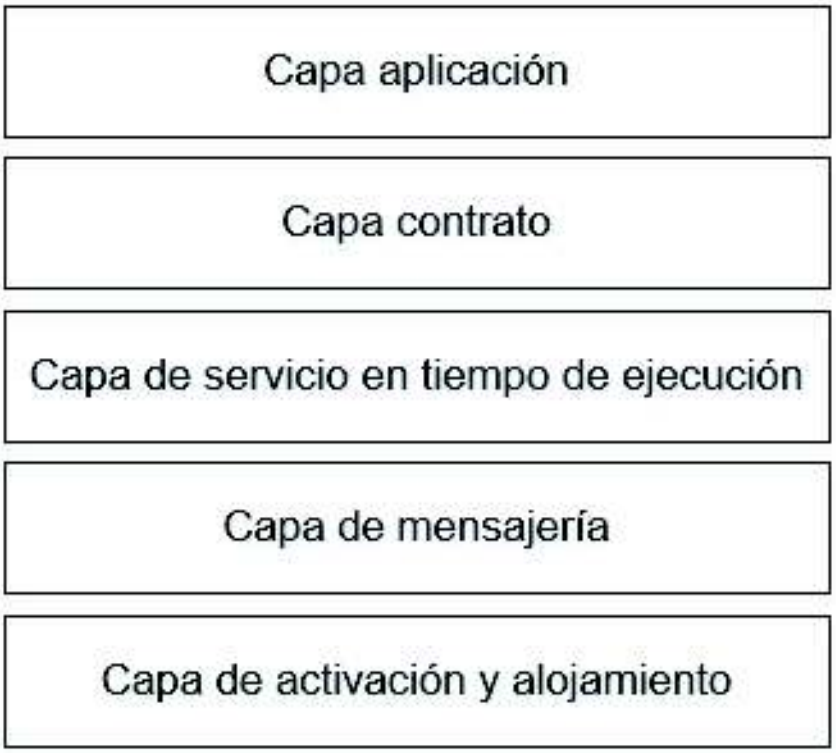


Figura 1.2.- Arquitectura WCF [2]

La capa de servicio en tiempo de ejecución, como su nombre lo indica, controla todos los comportamientos que el servicio va a tener en tiempo de ejecución. Estos comportamientos permiten controlar: la concurrencia, los comportamientos cuando ocurran errores, el número de instancias del servicio, entre otros.

La capa mensajería define varios canales[18] de comunicación que se pueden usar para acceder al servicio. Estos pueden ser: canales seguros con algún mecanismo de cifrado, canales TCP, canales HTTP, canales MSMQ, etc.

La capa de activación y alojamiento proporciona una variedad de opciones de arranque y alojamiento del servicio. Los arranques pueden ser de manera automática, cuando se aloje el servicio dentro de un servidor WAS[19]. Entre las opciones de alojamiento se tienen: Servicios de Windows, cualquier aplicación en .NET, entre otros [2].

[18] Canales: son los medios en los cuales un mensaje es trasmitido.
[19] WAS (*Windows Process Activation Service*): es un programa que inicia aplicaciones que alojen servicios WCF.

Componentes del servicio WCF

Un Servicio WCF está compuesto principalmente por dos elementos: descripción del servicio y de extremos, como se muestra en la Figura 1.3.

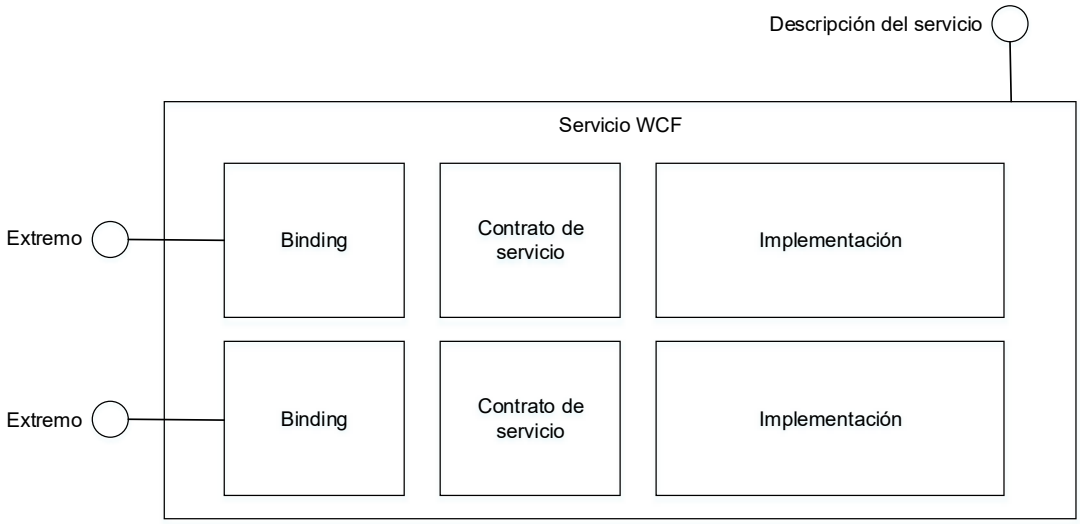


Figura 1.3.- Componentes del servicio WCF [2]

La descripción del servicio define cómo se accede y qué operaciones o funcionalidades se presentan en el servicio. Se encuentran descritas en formato XML y se localizan dentro de archivos `WSDL`[20], `XSD`[21], `WS-Policy`[22], `WS-MEX`[23]. El *framework* de .NET permite crear las descripciones de manera automática, solo se debe adornar el servicio con los atributos correspondientes y modificar los archivos de configuración de acuerdo a lo que se requiera.

Un extremo es una parte del servicio, el cual permite a los clientes conectarse a esta, y se encuentra constituido por: dirección, *binding* y contrato.

- La dirección es una ubicación única en toda la red que se encuentra en formato URI[24]. Y está compuesto por las siguientes partes: esquema, autoridad, ruta del recurso, consultas y el fragmento.

  El esquema establece el protocolo que se empleará para acceder al recurso.

[20] WSDL (*Web Service Description Language*): es un formato estandarizado que sirve para describir las características de un servicio web.
[21] XSD (XML *Schema Definition*): es un formato para definir la estructura de un documento XML.
[22] WS-Policy (*Web Service Policy*): es un formato para describir requisitos, capacidades de los servicios web.
[23] WS-MEX (*Web Service MetadaExchange*): describe diferentes mecanismos para la manipulación de metadatos.
[24] URI (*Uniform Resource Identifier*): es una cadena que identifica inequívocamente un recurso accesible por la red, definido en la RFC 3986.

La autoridad está conformado por el nombre del *host* o dirección IP [25] (que aloja el recurso) además del número de puerto. La autoridad empieza con los caracteres `://`, por ejemplo `//:192.168.100.10:8092`.

La ruta del recurso contiene información acerca de la ubicación del recurso de manera jerárquica, por ejemplo: `/Service.svc/Autenticacion`.

Los términos de consultas son sentencias con estructura clave=valor y separadas con caracteres `&` o `;` para identificar algún recurso en particular que satisfagan dichas sentencias. Las consultas comienzan con el signo de interrogación `?`, por ejemplo: `?edad=24&facultad=electronica`.

Y el fragmento permite mostrar una parte o vista del recurso. Los fragmentos inician con el carácter `#`, por ejemplo: `#top` [3].

- El *binding* define la manera en que se comunican los extremos con el resto del mundo, además de las características de seguridad, confiabilidad, protocolos de comunicación y patrones de mensajes[26].
- El contrato de servicio define los datos a ser intercambiados y las operaciones soportadas por el servicio en cuestión.

Servicios WCF basados en REST

REST (*Representational State Transfer*) es una arquitectura de red que permite implementar servicios web mediante estándares web, con patrones de intercambio de mensajes de tipo solicitud/respuesta. Esta arquitectura permite el manejo de diferentes formatos de datos como: HTML[27], JSON, XML, archivos planos y archivos de media [4]. Además, REST no involucra ningún tipo de mensaje especial y es menos verboso, lo cual resulta ser una arquitectura ligera y simple, sin embargo no posee tantas funcionalidades como los servicios basados en SOAP [5].

Cabe destacar que estos servicios tienen su propia configuración en las diferentes capas de la arquitectura WCF. En la capa de mensajería, REST puede emplear el esquema HTTP o HTTPS[28] para el intercambio de datos. En la capa de contratos, REST especifica un

[25] Dirección IP (Internet *Protocol*): conjunto de cuatro octetos separados por puntos los cuales permiten identificar un dispositivo conectado a una red que emplea el protocolo IP.
[26] Patrones de mensaje: es la forma de intercambiar mensajes, estas pueden ser: petición/respuesta, *simplex*, *full duplex*.
[27] HTML (*HiperText Markup Language*): es un lenguaje que sirve para definir la estructura de una página web.
[28] HTTPs (*HiperText Transport Protocol Secure*): es un protocolo basado en HTTP para transportar datos de manera segura.

contrato uniforme conformada por métodos propios de HTTP tales como: `GET`, `POST`, `PUT`, `DELETE`, para realizar operaciones referentes a leer, actualizar, crear y eliminar información respectivamente. Para implementar la capa de contrato en el *framework* de .NET, se emplea la librería `System.ServiceModel.Web` con sus correspondientes atributos como:

`[ServiceContract]`: Para definir un contrato de servicio, se lo coloca sobre una interfaz o una clase.

`[OperationContract]`: Para definir los métodos u operaciones de un contrato de servicio, se lo coloca sobre métodos, con la finalidad de que estos puedan ser consumidos por los clientes.

`[WebGet]`: Para indicar que un método u operación de servicio es lógicamente una operación de recuperación de información y se lo puede llamar por medio de WCF basado en REST.

`[WebInvoke]`: Para indicar que un método u operación de servicio es lógicamente una operación de creación, actualización y eliminación de información y se lo puede llamar por medio de WCF basado en REST.

En el Código 1.1 se muestra un ejemplo de definición de contrato de servicio (línea 1); se especifica el nombre del contrato de servicio como `IInformacionEmpleado` (línea 2); se especifica solamente una operación (línea 4); en las líneas 5-6, se define una operación de recuperación de información en formato JSON, acerca de un empleado dado un identificador en particular; y para que se pueda acceder la operación en cuestión, se debe llamar a la ruta del recurso `ObtenerEmpleado/?id={idEmpleado}`.

```
1. [ServiceContract]
2. public interface IInformacionEmpleado
3. {
4.     [OperationContract]
5.     [WebGet (ResponseFormat= WebMessageFormat.Json, UriTemplate="ObtenerEmpleado/?id={idEmpleado}")]
6.     Empleado ObtenerEmpleado(int idEmpleado);
7. }
```

Código 1.1.- Ejemplo definición de contrato de servicio

Para la serialización de objetos se emplea la librería `System.Runtime.Serialization` con sus correspondientes atributos:

`[DataContract]`, que permite definir los tipos de datos que van a ser intercambiados y serializados.

`[DataMember]`, para especificar los atributos de los tipos de datos que van a ser intercambiados y serializados.

En el Código 1.2 se muestra un ejemplo de definición de contrato de datos (línea 1), el cual especifica el tipo de dato llamado `Empleado` (línea 2), además se especifican los atributos a ser serializados como `idEmpleado`, `nombre`, `apellido`, `direccion` y `tipoSangre` (líneas 4-13).

```
1.  [DataContract]
2.  public class Empleado
3.  {
4.      [DataMember]
5.      public int idEmpleado { get; set; }
6.      [DataMember]
7.      public string nombre { get; set; }
8.      [DataMember]
9.      public string apellido { get; set; }
10.     [DataMember]
11.     public string direccion { get; set; }
12.     [DataMember]
13.     public string tipoSangre { get; set; }
14. }
```

Código 1.2.- Ejemplo de contrato de datos

En el Código 1.3 se muestra un ejemplo de implementación de funcionalidad del servicio, la línea 1 especifica la clase que implementa el contrato de servicio `IInformacionEmpleado`; en las líneas 3 a 7 se especifica la funcionalidad que va a tener el servicio, en este caso, se va a buscar y retornar un `Empleado` cuyo atributo `idEmpleado` coincida con el identificador entregado por el parámetro de entrada. En las líneas 8 a 13 se define el método `ObtenerListaEmpleado` que retorna una lista de `Empleado` guardados por defecto.

```
1.  public class EmpleadoInfo:IInformacionEmpleado
2.  {
3.  public Empleado ObtenerEmpleado(int idEmpleado)
4.  {
5.  Empleado informacionEmpleado= ObtenerListaEmpleados.Where(empleado=> empleado.idEmpleado == idEmpleado).FirstOrDefault();
6.  return informacionEmpleado;
7.  }
8.  private List<Empleado> ObtenerListaEmpleados(){
9.  return new List<Empleado>{
10. new Empleado{ idEmpleado=1, nombre="Juan", apellido="Pérez", direccion="Av. Amazonas", tipoSangre="AB"},
11. new Empleado{ idEmpleado=2, nombre="Alberto", apellido="Sánchez", direccion="Av. Río Coca", tipoSangre="B"},
12. new Empleado{ idEmpleado=3, nombre="Elizabeth", apellido="Herrera", direccion="Av. los Shyris", tipoSangre="AB"}};
13. }
14. }
```

Código 1.3.- Ejemplo de implementación de la operación del servicio

En cuanto a lo que se refiere con la capa de servicio en tiempo de ejecución, capa de mensajería y capa de activación dentro de la arquitectura WCF, se emplea un archivo de configuración denominado `Web.config`, el cual permite manipular las características que va a tener el servicio mediante elementos XML[29]. En la Tabla 1.1 se muestra algunas etiquetas que se emplean para la configuración de un servicio WCF REST.

Tabla 1.1 Etiquetas empleadas en el archivo `Web.config` [6]

Etiqueta	Descripción
`<configuration>`	Es el elemento raíz del archivo de configuración.
`<system.serviceModel>`	Es una sección que contiene todos los elementos de configuración de WCF.
`<behaviors>`	Permite definir los comportamientos de los extremos del servicio a través de sus dos elementos hijos `<serviceBehaviors>` y `<endpointBehaviors>`.
`<serviceBehaviors>`	Es una sección que representa todos los comportamientos definidos para un servicio específico, además posee un elemento hijo denominado `<behavior>`.
`<endpointBehaviors>`	Es una sección que representa todos los comportamientos definidos para un extremo específico, además contiene un elemento hijo llamado `<behavior>`.

[29] Elemento XML: permite identificar secciones, atributos en un documento XML y se definen mediante etiquetas de marcado.

Tabla 1.1 Etiquetas empleadas en el archivo `Web.config` (Continuación)

Etiqueta	Descripción
`<behavior>`	Contiene un conjunto de configuraciones del comportamiento que va a tener un servicio. Algunos de los comportamientos empleados son `<serviceMetada>` , `<serviceDebug>` y `<webHttp>`.
`<serviceMetada>`	Especifica la publicación de metadatos[30] del servicio y la información asociada a esta. Contiene un atributo denominado `httpGetEnabled`, que especifica si se va a permitir o no la recuperación de metadatos a través de solicitudes HTTP `GET`.
`<serviceDebug>`	Especifica las características de depuración e información de ayuda para un servicio WCF. Posee un atributo llamado `includeExceptionDetailInFaults`, el cual permite exponer detalles sobre excepciones ocurridas.
`<webHttp>`	Especifica que el servicio tenga el comportamiento basado en REST (sin estado).
`<protocolMapping>`	Es una sección que permite añadir los *bindings* y sus correspondientes protocolos de comunicación (HTTP, net.tcp[31], net.pipe[32], etc) empleados por el servicio. Posee un elemento hijo denominado `<add>`.
`<add>`	Permite añadir *binding* y esquemas, contiene dos atributos denominados `binding` y `schema`, los cuales debe ser definido como `webHttpBinding` y `http` respectivamente para definir un servicio WCF basado en REST.
`<serviceHostingEnvironment>`	Permite definir el tipo de entorno de alojamiento del servicio. Contiene un atributo `aspNetCompabilityEnabled`, para especificar si se emplea la compatibilidad entre WCF y ASP.NET. También contiene otro atributo llamado `multipleSiteBindingsEnabled`, que especifica si se encuentra habilitado múltiples esquemas de comunicación para una aplicación web de IIS.

[30] Metadatos: son datos que describen otros datos.
[31] Net.tcp: es un tipo de *binding* que pertenece a los extremos del servicio WCF, que permite una comunicación confiable y orientada a conexión.
[32] Net.pipe: es un tipo de *binding* que pertenece a los extremos del servicio WCF, que permite comunicación entre procesos dentro de la misma máquina.

Tabla 1.1 Etiquetas empleadas en el archivo `Web.config` (Continuación)

Etiqueta	Descripción
`<system.webServer>`	Es un elemento raíz que contiene elementos para definir las configuraciones del servidor IIS.
`<handlers>`	Se emplea para habilitar el manejo del archivos `Service.svc`[33] cuando se aloja el servicio en IIS.
`<directoryBrowse>`	Sirve para permitir o denegar la navegación de directorios.

En el Código 1.4 se muestra un ejemplo de configuración del archivo `Web.config` para el servicio REST. A continuación se describen las partes más importantes.

```
1.  <?xml version="1.0"?>
2.  <configuration>
3.    <system.serviceModel>
4.      <behaviors>
5.        <serviceBehaviors>
6.          <behavior>
7.            <serviceMetadata httpGetEnabled="true"/>
8.            <serviceDebug includeExceptionDetailInFaults="false"/>
9.          </behavior>
10.       </serviceBehaviors>
11.       <endpointBehaviors>
12.         <behavior>
13.           <webHttp/>
14.         </behavior>
15.       </endpointBehaviors>
16.     </behaviors>
17.     <protocolMapping>
18.       <add binding="webHttpBinding" scheme="http"/>
19.     </protocolMapping>
20.     <serviceHostingEnvironment multipleSiteBindingsEnabled="true"/>
21.   </system.serviceModel>
22. </configuration>
```

Código 1.4.- Ejemplo Web.config para servicio REST

En la línea 1, se especifica la versión del lenguaje de marcado XML; entre las líneas 3 a la 21, se especifica la sección que contiene todos los elementos de configuración WCF; las líneas 5 a 10 especifican la sección de comportamientos del servicio, que en este caso define que el servicio va a permitir la recuperación de metadatos a través de HTTP `GET` (línea 7), además de no permitir la exposición de detalles de excepciones (línea 8); en las

[33] `Service.svc`: es un archivo que proporciona información sobre el servicio web. WCF emplea esta información para crear el documento `WSDL`.

líneas 11 a 15, se especifica la sección de comportamiento del extremo, el cual se empleará un extremo de tipo REST, ya que se emplea el elemento `<webHttp>` (línea 13); las líneas 17 a 19 especifican la sección para agregar *bindings* y protocolos de comunicación, el cual en este caso se especifica que se emplean solicitudes HTTP para llevar a cabo la comunicación con el servicio; la línea 20 especifica que se tiene habilitado múltiples esquemas de comunicación para la aplicación web alojada en el servidor IIS.

Sesiones en servicios WCF REST

Una característica fundamental de REST es que no mantiene estado entre solicitudes/respuestas. Es decir, cada solicitud/respuesta es independiente de otras. Sin embargo, existen alternativas que se pueden optar para guardar estado, como en el caso de habilitar la compatibilidad de WCF con ASP.NET[34], el cual ofrece métodos para el almacenamiento y la recuperación de información de usuario cuando se navegue por la aplicación. Además, los estados de sesiones permiten controlar el acceso de los usuarios a determinadas operaciones del servicio y además, permite mostrar configuraciones personalizadas para determinados usuarios.

Para ello, se emplea la librería `System.ServiceModel.Activation` para añadir el atributo `AspNetCompatibilityRequirements`, que permite habilitar la compatibilidad mencionada.

En el Código 1.5 se muestra un ejemplo de activación de la compatibilidad WCF con ASP.NET. En la línea 1, se especifica que se permita la compatibilidad con ASP.NET; en la línea 2, se muestra que se aplica el atributo `AspNetCompatibilityRequirements` a la clase `ServicioAutenticacion`, la cual implementa el contrato de servicio `IServicioAutenticacion`.

```
1. [AspNetCompatibilityRequirements(RequirementsMode=AspNetCompatibilityRequirementsMode.Allowed)]
2.     public class ServicioAutenticacion : IServicioAutenticacion
3.     {
4.         public void DoWork()
5.         {
6.         }
7.     }
```

Código 1.5.- Ejemplo activación compatibilidad WCF con ASP.NET

[34] ASP.NET: es un modelo de desarrollo web de .NET que sirve para crear aplicaciones web.

Para almacenar y recuperar los datos de usuario en una sesión en particular, se emplean variables de sesión mediante el atributo `Session` de `HttpContext`, el cual permite guardar y recuperar variables de cualquier tipo soportado por .NET *Framework* (`ArrayList`, `Object`, `int`, `string`, etc).

Las variables de sesión se encuentran identificados por una cadena de caracteres, como se lo puede observar en el Código 1.6. En las líneas 1 y 2, se crean dos variables de sesión llamadas `Nombre` y `Apellido`, las cuales van a almacenar los valores de los componentes de texto `txtNombre` y `txtApellido` respectivamente; en las líneas 4 a 5, se declaran dos variables que recuperan los valores de sesión `Nombre` y `Apellido` respectivamente. Cabe mencionar que para recuperar los valores de las variables de sesión, se debe realizar la conversión específica del tipo de dato que se desee, por ello en el ejemplo se emplea el método `toString`, para convertirlo a cadena de caracteres .

```
1. Session["Nombre"] = txtNombre.Text;
2. Session["Apellido"] = txtApellido.Text;
3.
4. string nombreRecuperado=Session["Nombre"].toString();
5. string apellidoRecuperado=Session["Apellido"].toString();
```

Código 1.6.- Ejemplo de uso de variables de sesión

Las variables de estado de sesión se encuentran almacenados dentro del servidor web de manera predeterminada y se mantienen durante el tiempo de vida que dure la sesión en cuestión. Además, estas variables solo pueden ser accedidas por un usuario en particular.

Sesiones en cliente .NET

Para mantener estado de sesión en el lado del cliente, se emplean *cookies*, que son fragmentos de texto que se envían junto con las solicitudes/respuestas que se realicen. Las *cookies* permiten almacenar información específica del usuario, manejar estados y facilitar el reconocimiento de los usuarios. Por ejemplo, se podría emplear *cookies* para registrar el inicio de sesión con la finalidad de que el usuario ya no tenga que seguir proporcionando sus credenciales [7].

En el cliente .NET, para manejar *cookies* se emplea la clase `Dictionary`, que representa una colección de `claves/valores` y se lo utiliza para guardar y recuperar un conjunto de contenedores de objetos *cookie* con sus correspondientes nombres de dominio o autoridad de una dirección URI, por ejemplo `//:192.168.100.10:8092`, con la finalidad de que las *cookies* en cuestión solo puedan ser accedidas dentro del dominio especificado.

Además, se crea una instancia de la clase `HttpWebRequest` con la finalidad de crear una solicitud, para que luego se agregue a esta el contenedor de *cookies*. Con esto, ya se puede enviar *cookies* junto con las solicitudes.

En el Código 1.7 se muestra un ejemplo de un método que crea solicitudes HTTP con soporte para *cookies*. En las líneas 3 a 4, se instancia un diccionario que almacena un conjunto de dominios con sus correspondientes contenedores `CookieContainer`; en la línea 5, se especifica un método que crea una solicitud web a una dirección URI determinada; en la línea 6 se instancia una solicitud HTTP; en la línea 7 se extrae una parte de la dirección URI a través del método `GetLeftPart` y como parámetro de entrada `UriPartial.Authority` para especificar que se desea extraer la autoridad o dominio; en la línea 8 se declara un contenedor de objetos *cookies*; entre las líneas 9 a 13, se verifica si en el diccionario ya existe el dominio especificado, en caso de no existir, se crea un nuevo contenedor de *cookies* y se lo guarda en el diccionario de *cookies*; en caso de que ya exista el dominio dentro del diccionario, se procede a añadir el contenedor de *cookies* dentro de la solicitud (línea 15).

```
1.  public class CookiedRequestFactory
2.      {
3.          private static Dictionary<string, CookieContainer> diccionarioContenedoresCookies=
4.  new Dictionary<string,CookieContainer>();
5.          public static HttpWebRequest CrearHttpWebRequest(string url) {
6.              HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
7.              string dominio = (new Uri(url)).GetLeftPart(UriPartial.Authority);
8.              CookieContainer contenedorCookie;
9.
10.             if (!diccionarioContenedoresCookies.TryGetValue(dominio, out contenedorCookie)) {
11.                 contenedorCookie = new CookieContainer();
12.                 diccionarioContenedoresCookies[dominio] = contenedorCookie;
13.             }
14.
15.             request.CookieContainer = contenedorCookie;
16.             return request;
17.         }
18.     }
```

Código 1.7.- Ejemplo para crear solicitudes con *cookies* en cliente .NET

Formato JSON

REST emplea representaciones o formatos de datos en cada solicitud/respuesta, siendo el formato más utilizado JSON, que es un formato de representación de datos basado en texto para representar objetos en JavaScript. Posee una estructura conformada por pares nombre/valor leíble tanto para seres humanos como para computadoras. JSON es una alternativa a XML, pero a diferencia de esta última, representa la misma información empleando menos caracteres, por lo que resulta ser un formato más eficiente.

Para representar objetos en formato JSON, se emplean los símbolos `{ }` seguidos de los atributos que son pares campo/valor. Por ejemplo:

```
{propiedad1:valor1, propiedad2:valor2}
```

Para representar arreglos de objetos en JSON, se emplean los símbolos `[ ]` seguidos por los objetos separados mediante comas. Por ejemplo:

```
[{nombre:'Sarah',apellido:'Parker'}, {nombre:'Bryan',
apellido:'Cranston'}]
```

Alojamiento del servicio WCF

El alojamiento del servicio WCF es el ambiente en el cual el servicio opera y existe. Es importante alojar los servicios en alguna parte, para que los clientes puedan acceder a estos. Se tienen varias opciones de alojamiento:

- Auto-alojamiento: requiere incluir código adicional para la creación y cierre del servicio. Se puede auto-alojar el servicio dentro de una aplicación de consola de .NET mediante la clase `ServiceHost`, a través el método `Open`, con la finalidad de que el servicio empiece a recibir mensajes. El método `Close` termina el servicio e impide que los clientes puedan acceder a este. Además se debe añadir información sobre los extremos del servicio, que puede ser mediante comando o por medio del archivo de configuración.

- Alojamiento en IIS (*Internet Information Services*): el servidor web IIS es una plataforma que facilita el alojamiento, creación, configuración y administración de servicios web [8]. Una de las ventajas que ofrece este método es que los servicios pueden iniciar automáticamente.

  Además, permite la administración de la salud de los servicios, los cuales reconocen qué procesos no responden o cuáles se demoran mucho en responder. En estos casos, IIS facilita el reciclaje de procesos de forma automática.

IIS ayuda con la publicación de servicios web en la intranet o en Internet, con mecanismos de seguridad como: autenticación robusta, listas de acceso y comunicaciones cifradas [9]. Sin embargo, solo se puede emplear el protocolo HTTP para la comunicación.

- Alojamiento en servicios de Windows: este método es conocido como Windows *NT Services*. Esta forma de alojamiento no es muy diferente al método de auto-alojamiento, sin embargo, este método ofrece beneficios en los siguientes casos:
    - Cuando un servicio es de larga duración y no requiera interacción directa con el usuario.
    - Cuando se requiera que el servicio empiece o termine de manera automática.
    - Cuando se requiera protocolos de comunicación ajenos a HTTP como net.tcp o net.pipe.
- Alojamiento en WAS: el servicio WAS maneja la activación y el tiempo de vida de procesos que contengan aplicaciones con servicios WCF alojados. Este método permite el manejo de protocolos de comunicación HTTP y no HTTP. Además, proporciona un manejo más eficiente en cuento a la configuración y administración de aplicaciones. Así mismo, proporciona un reciclaje más eficiente y permite empezar y parar aplicaciones de manera dinámica [2].

Consumo del servicio WCF basado en REST en .NET

Para que una aplicación cliente en entorno de .NET pueda realizar peticiones y obtener respuestas del servicio, se hace uso de las clases `HttpWebRequest` y `HttpWebResponse` respectivamente. Además, dependiendo del método HTTP que se utilice, se debe incluir los datos necesarios dentro de la solicitud. Es decir, cuando se trate de enviar una petición `POST`, `PUT`, `DELETE`, se debe de realizar un proceso de serialización para convertir los datos a un flujo de bytes para que este sea escrito en el cuerpo de la solicitud HTTP. En cambio, si se trata de una petición `GET`, no se debe incluir datos dentro del cuerpo de la solicitud.

Con la finalidad de serializar y deserializar datos `JSON`, se emplean los métodos `Serialize` y `Deserialize` de la clase `JavaScriptSerializer` respectivamente [10].

Y para convertir los datos en formato JSON a un flujo de bytes y viceversa, se emplea la clase `Encoding`.

En la Tabla 1.2 se muestran algunas clases y métodos empleados en el proceso del consumir el servicio.

Tabla 1.2 Clases y métodos para consumir servicio en .NET

Clase	Descripción
`HttpWebRequest`	Permite realizar peticiones HTTP. Posee los siguientes métodos y atributos: `Create`: Para instanciar un objeto `HttpWebRequest`. `GetRequestStream`: retorna un objeto `Stream` que se emplea para enviar datos en la solicitud. `GetResponse`: retorna la respuesta del servicio `Method`: Especifica el método HTTP. `ContentType`: Especifica el tipo contenido dentro de la cabecera HTTP. `Accept`: especifica el valor del encabezado HTTP *Accept*.
`Stream`	Permite manipular secuencias de bytes. Posee los métodos: `Write`: permite escribir una secuencia de bytes a un buffer o un flujo. `Close`: permite liberar todos los recursos del flujo de bytes.
`HttpWebResponse`	Permite crear una instancia que contenga la respuesta del servicio. Posee los siguientes métodos: `GetResponseStream`: Obtiene el flujo de bytes de la respuesta del servicio. `Close`: Cierra el flujo de respuesta
`StreamReader`	Permite leer flujos de bytes. Posee los métodos: `ReadToEnd`: Permite leer los caracteres desde una posición hasta el final. `Close`: Cierra el flujo de `StreamReader`.
`JavaScriptSerializer`	Permite serializar y deserializar objetos JSON. Posee los métodos: `Serialize`: convierte un objeto a formato JSON. `Deserialize`:convierte una cadena en formato JSON a un objeto específico.
`Enconding`	Realiza la codificación de caracteres. Posee el método: `GetBytes`: convierte una cadena de caracteres a una secuencia de bytes.

En el Código 1.8 se muestra un ejemplo de consumo de servicio empleando una petición `POST`. En las líneas 2 a 4, se convierte un objeto que contiene las credenciales de autenticación de usuario a una cadena de caracteres en formato JSON, además, se

convierte la cadena de caracteres a un flujo de bytes; en las líneas 6 a 9 se crea una solicitud HTTP a una dirección URI determinada, se especifica que se emplea el método `POST` y además, se especifica que el cuerpo HTTP contiene valores en formato JSON; en las líneas 12 a 14, se obtiene una referencia al flujo de bytes de la solicitud para que se pueda escribir los datos JSON dentro de esta; las líneas 17 a 19, se envía la solicitud y se obtiene la correspondiente respuesta para luego, convertir el flujo de bytes recuperado a cadena de caracteres en formato JSON; finalmente la línea 24 convierte la cadena JSON a un objeto específico.

```
1.  //Serializar los empleados a json
2.  var serializador= new JavaScriptSerializer();
3.  var jsonRequestString= serializador.Serialize(credenciales);
4.   var bytes= Encoding.UTF8.GetBytes(jsonRequestString);
5.  //Crear solicitud
6.  var request =(HttpWebRequest)WebRequest.Create(url);;
7.  request.Method="POST";
8.  request.ContentType="application/json";
9.  request.Accept="application/json";
10.
11. //enviar datos json al servicio
12. var postStream= request.GetRequestStream();
13. postStream.Write(bytes, 0, bytes.Length);
14. postStream.Close();
15.
16. //obtener el estado del login del servicio
17. var respuesta = (HttpWebResponse)request.GetResponse();
18. var reader = new StreamReader(respuesta.GetResponseStream());
19. var jsonResponseString = reader.ReadToEnd();
20. reader.Close();
21. respuesta.Close();
22. Console.WriteLine(jsonResponseString);
23. //Deserializar el JSON y retornar el resultado
24. EstadoServicio estad= serializador.Deserialize<EstadoServicio>(jsonResponseString);
```

Código 1.8.- Ejemplo de consumo de servicio en cliente .NET

Consumo del servicio WCF REST en Android

Para que una aplicación cliente en entorno Android pueda realizar peticiones y obtener respuestas del servicio WCF, se puede emplear una librería HTTP, llamada Volley.

Volley fue desarrollado por Google para facilitar la interconectividad de las aplicaciones Android, ya que permite el manejo de casi cualquier trabajo que se necesite sobre la red.

Esta librería maneja solicitudes de red sin el uso de `AsyncTask`[35], que típicamente son empleados para realizar llamadas asíncronas. A diferencia de esta última, Volley permite el manejo de concurrencia de manera automática ya que cuenta con tres hilos de ejecución: hilo principal, hilo cache e hilo red. Los cuales permiten realizar tareas principales de la aplicación, realizar peticiones de búsqueda de memoria cache y enviar/obtener solicitudes/respuestas HTTP del servicio remoto respectivamente.

Además, Volley almacena en memoria cache todas las peticiones y respuestas que se realicen, con el fin de reutilizar recursos previamente guardados por causa de cambios en las actividades. Por ejemplo, cuando se rota la pantalla en una aplicación Android, se realiza un proceso de reconstrucción de la actividad, sin embargo, ya no es necesario realizar otra vez la misma solicitud para re-crear la actividad en cuestión, ahorrando así la utilización de la red [11].

Para crear solicitudes, se puede emplear tres diferentes clases según el tipo de dato que responda el servicio. Estas son: `StringRequest`, para solicitar respuestas de tipo `String`; `JsonObjectRequest`, para obtener respuestas de objetos `JSON`; `JsonArrayRequest`, para adquirir respuestas de arreglos de objetos `JSON`. Además, se debe implementar los manejadores de eventos en casos de obtener una respuesta satisfactoria o de fallo, mediante las clases `Response.Listener` y `Response.ErrorListener` respectivamente.

Para enviar la solicitud, se emplea una instancia de la clase `RequestQueue` [12], que permite el manejo de una cola que almacene las solicitudes HTTP. Además, esta clase permite enviar las solicitudes guardadas anteriormente.

En Android, se puede emplear la librería GSON para convertir objetos Java a su correspondiente representación JSON y viceversa, así como para transformar una cadena JSON a su correspondiente objeto Java. Para ello, se emplean los métodos `toJson` y

[35] `AsyncTask`: es una clase que permite correr procesos en segundo plano.

`fromJson` de la clase `Gson`, con el fin de realizar las conversiones antes mencionadas [13].

En el Código 1.9 se muestra un ejemplo de consumo de servicio en el cliente Android. Entre las líneas 1 a 19, se tiene una sección para crear una solicitud para obtener información acerca de un `Empleado`; en la línea 1, se crea una instancia de solicitud `JsonObjectRequest`, la cual especifica el método HTTP `GET` (`Method.GET`), la dirección URI (`urlPedido`), se especifica que no hay datos en el cuerpo HTTP (`null`) y se especifica los manejadores de eventos de respuesta satisfactoria (`Response.Listener`) y de respuesta de fallo (`Response.ErrorListener`); entre las líneas 3 a 12, se implementa la secuencia de código que se va a realizar tras obtener una respuesta satisfactoria; se convierte la respuesta de formato JSON a un objeto de la clase `Empleado` (línea 5); se muestran en los componentes visuales `TextView`[36] los atributos del `Empleado` como `direccion` , `tipoSangre`, `idEmpleado`, `nombre`, `apellido` (líneas 6 a 11); en las líneas 15 a 18, se implementa la secuencia de código que se va a realizar tras no obtener una respuesta satisfactoria; se muestra un mensaje de error en el `Log`[37] de Android (línea 16); se muestra un mensaje de error en un `TextView` de Android; en la línea 21, se agrega la solicitud antes creada en una cola y luego envía la solicitud al servicio.

```
1.  JsonObjectRequest jsonObjReq= new JsonObjectRequest(Method.GET, urlPedido, null, new Response.Listener<JSONObject>(){
2.          @Override
3.          public void onResponse(JSONObject response) {
4.              //para convertir JSON a objeto
5.              Empleado emp= gson.fromJson(response, Empleado.class);
6.              txtDireccion.setText("Address: "+emp.getAddress());
7.              txtTipoSangre.setText("BloodGroup: "+emp.getBloodGroup());
8.              txtCodigoEmpleado.setText("EmployeeId: "+emp.getEmployeeId());
9.              txtNombre.setText("FirstName: "+emp.getFirstName());
10.             txtApellido.setText("LastName: "+emp.getLastName());
11.             txtError.setText("");
12.         }//Fin onResponse
13.     }, new Response.ErrorListener(){
14.         @Override
15.         public void onErrorResponse(VolleyError error) {
16.             VolleyLog.d(TAG,"Error: "+error.getMessage());
17.             txtError.setText("Error no se encuentra ");
18.         }
19.     });
20.     //agregar el request a la cola de request
21.     Volley.newRequestQueue(getApplicationContext()).add(jsonObjReq);
```

Código 1.9.- Ejemplo consumo de servicio con Volley

[36] `TextView`: es un elemento de interfaz de usuario que permite mostrar texto.
[37] `Log`: es una clase que permite mostrar mensajes de depuración en el monitor de Android.

Sistema Operativo Android

Android es un sistema operativo que fue desarrollado por Google y *Open Handset Alliance*, creada para una gran variedad de dispositivos. Además, es un sistema de código abierto basado en el *kernel*[38] de Linux, que permite la interacción entre hardware y software.

El sistema operativo Android tiene una arquitectura como se muestra en la Figura 1.4.

Figura 1.4.- Arquitectura Android [14]

En el nivel de aplicaciones contiene todas las aplicaciones que el usuario tiene incluyendo las que vienen por defecto. Como por ejemplo la lista de contactos, WhatsApp, el

[38] *Kernel*: hace referencia a la parte central o el núcleo de un sistema operativo.

explorador, etc. Estas aplicaciones emplean los servicios, librerías y abstracciones de los niveles inferiores.

Java API *Framework* es el nivel en donde se tienen todas las herramientas necesarias para el desarrollo de una aplicación. Entre estas se encuentran:

- Sistema de vista: compila la interfaz gráfica de una aplicación e incluyen los diferentes componentes visuales como cuadrículas, cuadros de texto, botones, etc.
- Administrador de actividad: gestiona el ciclo de vida de las aplicaciones Android.
- Administrador de ubicación: obtiene información de localización y posicionamiento.
- Administrador de notificaciones: muestra información acerca de los eventos específicos que se produzcan en cada aplicación.
- Administrador de recursos: brinda acceso a recursos como gráficos, archivos de diseño, `strings`, entre otros.

En el siguiente nivel se encuentran las librerías que están codificadas con lenguajes de programación C/C++ y ayudan a las aplicaciones a tener mayores características y funcionalidades. Entre estas librerías se tienen:

- Bionic C: es una librería desarrollada por Google con licencia BSD[39] para compilar código en lenguaje C además interviene en servicios específicos Android como el manejo de *logs* y propiedades del sistema.
- Webkit: proporciona herramientas para navegación web.
- OpenGL/ES[40] y SGL[41]: permite que las aplicaciones Android manejen gráficos 3D o 2D.
- Librería SQLlite: proporciona manejo de bases de datos relacionales en Android.

Al mismo nivel de la capa librerías se encuentra el apartado de tiempo de ejecución Android, el cual contiene librerías de gran variedad de clases en Java, y a partir de Android 5.0, se emplea ART (Android *RunTime*), que permite ejecutar varias máquinas virtuales en dispositivos de bajos recursos ejecutando archivos DEX[42]. Además, ART está basado en

[39] BSD (*Berkeley Software Distribution*): es una licencia de software muy permisiva.
[40] OpenGL/ES (*Open Graphic Library for Embedded Systems*): es una API para gráficos 3D diseñada para dispositivos móviles, consolas entre otros.
[41] SGL (*Scalable Graphics Library*): es una librería de bajo nivel que maneja renderización
[42] DEX (Dalvik *Executable Files*): son códigos en bytes que son optimizados para pesar lo mínimo.

el *kernel* de Linux, lo que permite manejar subprocesos y administración de memoria en bajo nivel.

La capa de abstracción de hardware permite exponer interfaces para que la capa Java API *Framework* haga uso de recursos hardware (cámara, lector de huellas, GPS[43], entre otros).

En el último nivel se encuentra el *kernel* de Linux que permite que Android se beneficie de las funcionalidades de seguridad y además permite a los fabricantes de dispositivos desarrollar hardware para un *kernel* conocido [14].

Android Studio

Es un IDE[44] para la creación de aplicaciones en Android y está basado en IntelliJ IDEA[45]. Posee un editor de texto además de herramientas potentes para desarrolladores de aplicaciones.

También ofrece varias características como: compilación basado en Gradle[46], un emulador con diversas funcionalidades, integración con Github[47], *Instant Run* para evitar re-compilar el código, variedad de herramientas y *frameworks* de prueba, entre otros [15] .

Los proyectos de Android Studio se encuentran conformados por los siguientes archivos:

- Gradle: conjunto de archivos que permiten automatizar y administrar el proceso de compilación y además, facilita la definición de configuraciones personalizadas para compilar y probar aplicaciones Android [16]. En estos archivos se agregan las dependencias o librerías para que estas puedan ser usadas posteriormente.
- Archivo `AndroidManifest`: se encuentra definido en formato XML y define información esencial sobre la aplicación, para que el sistema Android pueda ejecutar el código del proyecto. Entre sus funciones principales se encuentran:

[43] GPS (*Global Positioning System*): es un sistema conformado por veinte y cuatro satélites que sirven para proporcionar estimaciones de posicionamiento geográfico de objetos en la superficie terrestre.
[44] IDE (*Integrated Development Enviroment*): es una aplicación que permite desarrollar otras aplicaciones.
[45] IntelliJ IDEA: es un IDE desarrollado por JetBrains para crear software.
[46] Gradle: es un sistema de compilación basado en JVM (Java *Virtual Machine*) que traduce el código fuente de la aplicación y recursos a código binario y los empaqueta en APK (Android *Package*). Además facilita la actualización y exportación de la aplicación.
[47] Github: es una plataforma que sirve para alojar proyectos de programación en repositorios de manera gratuita.

    - Nombrar un identificador único y se encuentra conformado por el dominio y nombre de la aplicación.
    - Describir todos los componentes de la aplicación, como las actividades, los servicios, los intentos[48], los receptores de mensajes y los proveedores de contenido.
    - Declarar los permisos que van a requerir las aplicaciones para acceder a lugares protegidas de Android, así como las autorizaciones necesarias para interactuar con otros proyectos.
    - Declarar el nivel mínimo del API de Android [17].
- Archivos Java: entre estos archivos se pueden encontrar las Actividades y las Clases en el contexto del paradigma de programación orientada a objetos.
    - Una `Activity` es parte de la aplicación y está relacionada con una GUI[49] con la que los usuarios puedan interactuar y realizar alguna determinada acción [18]. En las actividades se implementan todas las funcionalidades de la aplicación en lenguaje de programación Java.
    - Las clases de Java son moldes o plantillas que contienen características o atributos que representen a un objeto de la vida real.
- Archivos res: contiene todos los recursos de la aplicación, entre estos encuentran:
    - `Drawable`: contiene todas las imágenes de la aplicación.
    - `Layout`: contiene las interfaces gráficas de las actividades y se encuentran definidos en `XML`.
    - `Mipmap`: al igual que los recursos del archivo `drawable`, contiene recursos de imágenes, pero a diferencia de este último, solo guarda íconos.
    - `Values`: contiene valores de cadenas de caracteres, arreglos, estilos y dimensiones.

API de huella dactilar Android

La API de huella dactilar se introduce por primera vez en la API de Android 6.0 *Marshmallow*, que ofrece nuevas funcionalidades a las aplicaciones y a los desarrolladores. Una de estas es que permite autenticar usuarios mediante huellas digitales, sin necesidad de emplear pares de nombre de usuario y contraseña típicos.

[48] `Intent`: es una instancia que permite comunicar diferentes componentes de una aplicación Android y también sirve para iniciar una actividad o un servicio.
[49] GUI (*Graphical User Interface*): es un conjunto de componentes gráficos que permiten a los usuarios navagar y realizar diferentes procesos.

También proporciona un nivel adicional de seguridad para que solamente el propietario del dispositivo pueda tener acceso y hacer uso de las funcionalidades específicas de las aplicaciones.

La autenticación por huellas ofrece ventajas como:

- No requiere que el usuario recuerde credenciales para la autenticación.
- Se tiene un nivel alto de confiabilidad ya que se asegura de que el acceso sea válido solo para el propietario.
- Permite realizar de una manera más cómoda y rápida las operaciones que requieran verificar al usuario. Por ejemplo en el caso de realizar transacciones bancarias [19].
- Las características de los valles y crestas de la epidermis de cada individuo es única y representa unívocamente su identidad y permanece invariante en el tiempo [20].

Para llevar a cabo la autenticación por medio de huellas digitales, la aplicación hace uso de las librerías y clases que proporcionan la API de Android como: `FingerPrintManager`, para realizar el proceso de autenticación. `Keystore` y `KeyGenerator`, para generar y administrar claves involucradas. Además de `Cipher` y `CryptoObject` para el manejo de cifrado y parámetros de seguridad. Así mismo, se emplean sensores que se encuentran disponibles en dispositivos Android para capturar imágenes de las huellas dactilares [21]. En la Tabla 1.3, la Tabla 1.4 y la Tabla 1.5 se muestran librerías, clases y excepciones respectivamente involucradas en el proceso de autenticación por medio de huella digital.

Tabla 1.3.- Librerías para la autenticación por huella digital

Librería	Descripción
`android.app.KeyguardManager`	Para manejar servicios de desbloqueo de pantalla (PIN, contraseña, patrón, huella digital).
`android.hardware.fingerprint.FingerprintManager`	Permite manejar el servicio de autenticación por huella.
`android.security.keystore.KeyGenParameterSpec`	Para especificar parámetros de generación de claves.
`javax.crypto.KeyGenerator`	Permite generar claves.
`javax.crypto.Cipher`	Permite realizar el proceso de cifrado.
`javax.crypto.SecretKey`	Permite agrupar todas las interfaces de claves secretas.
`java.security.KeyStore`	Permite manipular las claves que se encuentran en el contenedor seguro de Android.

Tabla 1.3.- Librerías para la autenticación por huella digital (Continuación)

Librería	Descripción
`android.security.keystore.KeyProperties`	Para especificar diferentes propiedades de las claves que se encuentre en el contenedor seguro de Android. Define las siguientes constantes: `KeyProperties.PURPOSE_ENCRYPT`: especifica que el propósito de la clave es de cifrado. `KeyProperties.PURPOSE_DECRYPT`: especifica que el propósito de la clave es de decifrado. `KeyProperties.BLOCK_MODE_CBC`: representa el modo de bloques CBC[50] `KeyProperties.KEY_ALGORITHM_AES`: representa el algoritmo de cifrado AES[51] `KeyProperties.ENCRYPTION_PADDING_PKCS7`: indica que el tipo de relleno es PKCS#7[52].

Tabla 1.4.- Clases para la autenticación por huellas digitales

Clase	Descripción
`FingerPrintManager`	Permite tener acceso al servicio de autenticación por huella digital. Contiene los siguientes métodos: `getSystemService`: permite recuperar un objeto `FingerPrintManager` para manejo de huellas. `hasEnrolledFingerprints`: verifica si se tiene registrado huellas digitales dentro del dispositivo móvil. `Authenticate`: realiza el proceso de autenticación.
`KeyguardManager`	Permite tener acceso al servicio de desbloqueo de pantalla. Contiene lo siguiente: `getSystemService`: permite recuperar un objeto `NotificationManager` para controlar `keyguard`. `isKeyGuardSecure`: valor booleano para verificar si hay un mecanismo de bloqueo de pantalla en el dispositivo.
`Keystore`	Permite obtener acceso al contenedor de claves criptográficas de Android. Contiene los siguiente: `getInstance`: permite obtener una referencia al contenedor de claves de Android. `load`: permite cargar el contenedor de claves.

[50] CBC (*Cipher Block Chaining*): es un proceso de cifrado en cadena en el cual a cada bloque se le aplica la operación XOR al bloque anterior.
[51] AES (*Advanced Encryption Standard*): es un estándar de cifrado simétrico por bloques.
[52] PKCS#7(*Public-Key Cryptography Standards*): es un estándar sobre la sintaxis del mensaje criptográfico.

Tabla 1.4.- Clases para la autenticación por huellas dactilares (Continuación)

Clase	Descripción
`KeyGenerator`	Permite generar una clave de cifrado. Contiene los siguiente: `getInstance`: permite instanciar una generador de claves que genere claves secretas para un algoritmo de cifrado específico. `Init`: permite inicializar el generador de claves. `generateKey`: permite generar una clave con los parámetros especificados.
`KeyGenParameterSpec`	`Builder`: Permite especificar el tipo de clave que se va a generar, si es simétrica o asimétrica, modo de cifrado de bloques, tipo de relleno. `setBlockModes`: permite especificar el modo de cifrado de bloques (Ejm: CBC). `setUserAuthenticationRequired`: para que el usuario tenga que autorizar la operación cada vez que se emplee la clave para la autenticación de huellas. `setEncryptionPaddings`: permite especificar el tipo de relleno de bloques (Ejm: PKCS#7) `build`: permite crear una instancia `KeyGenParameterSpec`.
`Cipher`	Permite realizar funciones de cifrado y de-cifrado. Contiene los siguiente: `getInstance`: permite obtener un objeto `Cipher` que implementa la transformación especificada. `init`: inicializa el cifrador con una clave y un grupo de parámetros de seguridad.
`CryptoObject`	Esta clase se encuentra asociada con la llamada al proceso de autenticación, que permite hacer más seguro el proceso en cuestión.
`SecretKey`	Permite agrupar todas las interfaces de claves secretas.
`AuthenticationCallback`	Permite manejar los eventos en todo el proceso de autenticación como: `onAuthenticationError`: se encarga de manejar los intentos fallidos de la autenticación. `onAuthenticationHelp`: es llamado cuando un error recuperable ocurre. `onAuthenticationFailed`: es llamado cuando la huella digital es válida pero no reconocida. `onAuthenticationSucceeded`: es llamado cuando la huella es reconocida.
`CancellationSignal`	Permite cancelar una operación en progreso.

Tabla 1.5.- Excepciones de la autenticación por huella digital

Excepción	Descripción
`KeyPermanentlyInvalidatedException`	Indica que la clave ya no puede ser usado porque se ha invalidado permanentemente.
`InvalidAlgorithmParameterException`	Para indicar parámetros de algoritmos inválidos o no apropiados.
`InvalidKeyException`	Para indicar claves inválidos (codificación inválida, longitud equivocada, no se ha inicializado, etc).
`KeyStoreException`	Es una excepción genérica para la clase `KeyStore`.
`NoSuchAlgorithmException`	Indica que el algoritmo solicitado no se encuentra disponible.
`NoSuchProviderException`	Indica que el proveedor de seguridad solicitado no se encuentra disponible.
`CertificateException`	Indica una variedad de problemas con los certificados.
`IOException`	Indica fallos o interrupciones de operaciones de entrada y salida.
`UnrecoverableKeyException`	Se lanza esta excepción cuando una clave no pudo ser recuperada del contenedor de claves.
`NoSuchPaddingException`	Indica que el mecanismo de relleno solicitado no se encuentra disponible.

En el Código 1.10 se muestra un fragmento del método `onCreate`, que se ejecuta cuando se crea una actividad y crea las instancias y llamadas a métodos requeridos durante todo el proceso de autenticación. En la línea 5, se instancia el objeto `KeyguardManager`, que permite tener acceso a claves de desbloqueo de pantalla de dispositivos Android; en la línea 6, se instancia la clase `FigerprintManager`, el cual permite manejar el proceso de autenticación por huella digital; en la línea 14, se realiza una llamada al método `generarClave` , el cual permite crear una clave criptográfica, este método se muestra en el Código 1.11; en la línea 16, se llama al método `cipherInit`, que crea una instancia de `Cipher` y retorna `true` si es que se inicializó correctamente el cifrador, este método se encuentra descrito en el Código 1.12; en la línea 17, se instancia un `CryptoObject` que permite que el proceso de autenticación por huellas sea más seguro; en la líneas 18 y 19, se instancia la clase `ManejoHuellas` que maneja el proceso de autenticación y se llama al método `empezarAutenticacionHuella` de esta última clase para ejecutar el proceso de autenticación, este método se muestra en el Código 1.13.

En el Código 1.11 se muestra un ejemplo de un método para generar claves criptográficas. En las líneas 2 a 6, se obtiene una referencia a un objeto `Keystore`, para tener acceso al contenedor de claves segura de Android, además, se tratan las excepciones que se puedan ocurrir; en las líneas 9 a 13, se crea una instancia de la clase `KeyGenerator` para generar claves empleando el algoritmo de cifrado AES, y se manejan las excepciones correspondientes; en la línea 16, se carga el contenedor de claves de Android; en las líneas 17 a 20, se genera una clave con el nombre especificado en `KEY_NAME`, que se lo emplea

con propósitos de cifrado y descifrado, con el modo de cifrado CBC, que requiere permisos del usuario y con sintaxis PKCS#7.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_autenticacion_huella);
        //Se instancian objetos FingerprintManager y KeyguardManager
        keyguardManager = (KeyguardManager) getSystemService(KEYGUARD_SERVICE);
        fingerprintManager = (FingerprintManager) getSystemService(FINGERPRINT_SERVICE);
        //Se verifican si la pantalla de seguridad esté habilitada.
        ...
        //Se verfica si es que se tiene permiso del uso de huella dactilar.
        ...
        //Se verifica si se tienen huellas dactilares enroladas en teléfono.
        ...
        //Generar clave
        generarClave();
        //Iniciar el cifrador.
        if(cipherInit()){
            cryptoObject= new FingerprintManager.CryptoObject(cipher);
            ManejoDeHuellas manejoDeHuellas = new ManejoDeHuellas(this);
            manejoDeHuellas.empezarAutenticacionHuella(fingerprintManager,cryptoObject);
        }
    }
```

Código 1.10.- Ejemplo método principal autenticación huellas

```
protected void generarClave() {
    try {
        keyStore = KeyStore.getInstance("AndroidKeyStore");
    } catch (Exception e) {
        e.printStackTrace();
    }

    //Se instancia KeyGenerator con los diferentes parámetros de seguridad.
    try {
        keyGenerator=KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES, "AndroidKeyStore");
    }catch (NoSuchAlgorithmException | NoSuchProviderException e){
        throw new RuntimeException("Fallo en obtener instancia de KeyGenerator",e);
    }
    //Generar la clave que va a ser usado por el cifrador en el proceso de encripción.
    try{
        keyStore.load(null);
        keyGenerator.init(new KeyGenParameterSpec.Builder(KEY_NAME, KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT)
        .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
        .setUserAuthenticationRequired(true)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7).build());
    }catch (NoSuchAlgorithmException| InvalidAlgorithmParameterException|CertificateException| IOException e){
        throw new RuntimeException(e);
    }
}
```

Código 1.11.- Ejemplo método para generar claves criptográficas

En el Código 1.12 se muestra un ejemplo de un método que verifica la creación correcta del objeto `Cipher`. En las líneas 2 a 6, se instancia un objeto `Cipher` que emplea el algoritmo de cifrado AES, usando el modo de cifrado CBC y la sintaxis PKCS#7, además se controlan las excepciones correspondientes; en la línea 8, se cargan las claves que se encuentran dentro del contenedor de claves; en la línea 10, se obtiene la clave con el nombre `KEY_NAME` que se encuentra dentro del contenedor de claves, y se lo va a emplear en el proceso de cifrado; en la línea 12, se inicializa el cifrador con la clave secreta obtenida; se retorna `true` cuando se inicializa correctamente el cifrador (línea 14); en las líneas 15 a 19 se manejan las excepciones que puedan ocurrir.

```
1.  public boolean cipherInit(){
2.      try {
3.          cipher=
Cipher.getInstance(KeyProperties.KEY_ALGORITHM_AES+"/"+KeyProperties.BLOCK_MODE_CBC+"/"+KeyProperties.ENCRYPTION_PADDING_PKCS7);
4.      }catch (NoSuchAlgorithmException|NoSuchPaddingException e){
5.          throw new RuntimeException("Fallo en obtener el Cifrador",e);
6.      }
7.      try{
8.          keyStore.load(null);//Se carga el Keystore container
9.          //Se genera una clave secreta
10.         SecretKey key= (SecretKey)keyStore.getKey(KEY_NAME,null);
11.         //Se inicia el cifrador
12.         cipher.init(Cipher.ENCRYPT_MODE, key);
13.         //Si aquí el cifrador inicia correctamente, retorna true.
14.         return true;
15.     }catch (KeyPermanentlyInvalidatedException e){
16.         return false;
17.     }catch (KeyStoreException|CertificateException|UnrecoverableKeyException|IOException|NoSuchAlgorithmException|InvalidKeyException e){
18.         throw new RuntimeException("Fallo en iniciar el Cifrador",e);
19.     }
20. }
```

Código 1.12.- Ejemplo código para inicializar un objeto `Cipher`

El Código 1.13 se muestra un ejemplo para realizar la autenticación y manipular los eventos de la autenticación. En la línea 1, la clase `ManejoHuellas` se extiende de la interfaz `FingerprintManager.AuthenticationCallback`, para implementar los métodos que manejen eventos de autenticación satisfactoria o de fracaso; en la línea 2, se define un método que permite empezar el proceso de autenticación por huella; en la línea 6, se emplea el método `authenticate` que solicita la autenticación de un `CryptoObject`, esto hace que el hardware de lector de huellas esté listo y que empiece a escanear por huellas dactilares, se especifica un `CancellationSignal` para cancelar el proceso en cuestión, se especifica que no se tienen banderas (`0`), se especifica el contexto que recibe los

eventos de la autenticación (`this`) y por último, no se especifica ningún manipulador de eventos *callback* (`null`); en las líneas 8 a 11, se imprime en pantalla un mensaje de error cuando se produzca un evento de intento fallido de autenticación; en las líneas 12 a 15, se muestra un mensaje de ayuda cuando el usuario no coloca bien el dedo en el sensor; en las líneas 16 a 19 se muestra un mensaje cuando no se reconoce la huella digital; y entre las líneas 20 a 23, se muestra un mensaje de autenticación satisfactoria cuando se reconoce la huella del usuario.

```
1.  public class ManejoDeHuellas extends FingerprintManager.AuthenticationCallback {
2.      public void empezarAutenticacionHuella(FingerprintManager manager, FingerprintManager.CryptoObject cryptoObject){
3.          cancellationSignal = new CancellationSignal(); //para poder cancelar una operación en progreso.
4.          //Verificar otra vez si es que se tienen permisos de usaar huella dactilar
5.          ...
6.          manager.authenticate(cryptoObject,cancellationSignal,0,this,null);
7.      }
8.      @Override
9.      public void onAuthenticationError(int errMsgId, CharSequence errString){
10.         Toast.makeText(appContext,"Error en la Autenticacion\n"+errString,Toast.LENGTH_LONG).show();
11.     }
12.     @Override
13.     public void onAuthenticationHelp(int helpMsgId,CharSequence helpString){
14.         Toast.makeText(appContext,"Ayuda en la autenticacion\n"+helpString,Toast.LENGTH_LONG).show();
15.     }
16.     @Override
17.     public void onAuthenticationFailed(){
18.         Toast.makeText(appContext,"Fallo en la autenticación.", Toast.LENGTH_LONG).show();
19.     }
20.     @Override
21.     public void onAuthenticationSucceeded(FingerprintManager.AuthenticationResult result){
22.     Toast.makeText(appContext, "Autenticación satisfactoria :D",Toast.LENGTH_LONG).show();
23.     }
```

Código 1.13.- Ejemplo de clase para manejar eventos de autenticación por huella

Para que el proceso de autenticación de huellas funcione correctamente, se debe añadir el elemento `<uses-permission android:name="android.permission.USE_FINGERPRINT"/>` dentro del archivo `AndroidManifest`. Además, se debe añadir código para verificar estos permisos adecuadamente en tiempo de ejecución.

Lectores de huellas en Android

Una de las tendencias más actuales en el ámbito del desarrollo de teléfonos inteligentes es la integración de sensores de huella digital a una amplia gama de dispositivos móviles, esto resulta ser una característica atractiva para el usuario ya que le ayuda a realizar procesos que requieran autenticación de manera rápida y sencilla.

Para llevar a cabo esto, los lectores de huellas constan principalmente de tres partes: sensor de lectura, que permite capturar una imagen de la huella con los patrones de crestas y valles de la epidermis del individuo; conversor analógico-digital, que sirve para convertir la imagen analógica capturada por el sensor en una señal digital; interfaz de comunicaciones, con la finalidad de transmitir los datos digitales a otro lugar, que puede ser otro dispositivo, un espacio de almacenamiento, entre otros.

Los sensores pueden emplear diferentes características físicas durante el proceso de obtención de imágenes, estas pueden ser por medio de: luz, electricidad y sonido.

Para capturar imágenes a través de luz se emplean sensores ópticos, cuando el dedo entra en contacto con la superficie del sensor, un diodo LED proyecta un haz de luz sobre los valles y crestas de la epidermis reflejándose en múltiples direcciones. Esta luz direccional se focaliza mediante un sistema de lentes hacia un dispositivo CCD[53] o CMOS[54], obteniéndose la imagen de la huella.

Para obtener las huellas mediante electricidad, se emplean sensores capacitivos, que forman un conjunto de decenas de miles de micro-capacitores en una superficie plana, el cual se extiende un dieléctrico. La medida de voltaje de estos capacitores permite capturar la imagen de la huella. Obteniéndose niveles de voltaje más altos en las crestas de la yema del dedo que en los valles, debido a que en lo primero, se encuentran más próximos a la superficie. La superficie en contacto con el dedo cuenta con una fina capa protectora con su debida puesta a tierra, el cual protege de la electricidad estática de los dedos. Además, estos sensores cuentan con mecanismos eléctricos para mejorar la calidad de la imagen capturada cuando el dedo se encuentre húmedo o muy seco.

Para capturar huellas dactilares a través del sonido se utilizan sensores ultrasónicos, el cual proyecta pulsos ultrasónicos sobre la yema del dedo. El eco reflejado por la superficie del dedo permite determinar si se trata de los valles o crestas de acuerdo a la distancia

[53] CCD (*Charge Coupled Device*): es un sensor de luz, presente en cámaras digitales que sirve para capturar imágenes a un rango amplio de tonos en alta calidad y poco ruido.
[54] CMOS (*Complementary Metal Oxide Semiconductor*): es un sensor de luz, presente en cámaras profesionales que consume menos energía y es menos sensible a la luz que CCD.

total reflejada. Este tipo de sensores resultan ser más fiables puesto que son menos susceptibles a la humedad, la grasa y a la suciedad de la piel [20].

Mecanismo de posicionamiento *en* Android

A inicios del año 2019, la mayoría de personas llevan consigo un teléfono inteligente a donde sea que vayan. Es por ello que si se le añade conocimiento de ubicación a las aplicaciones móviles, estos pueden ofrecer a los usuarios una experiencia personalizada, de acuerdo a su ubicación [22].

Las aplicaciones con acceso a estos servicios pueden brindar al usuario contenido referente a: lugares de interés cercanos, direcciones, mapas, entre otros.

En Android se tienen dos mecanismos para acceder a los servicios de ubicación, el primero es mediante Android *Framework Location* API, que se encuentra disponible por defecto dentro del nivel de *Application Framework* en la arquitectura Android, como se mostró anteriormente en la Figura 1.4. La segunda forma, es acceder mediante Google *Location Services* API, la cual forma parte de los servicios de Google y se lo utiliza con la finalidad de ofrecer a las aplicaciones Android funcionalidades como: el conocimiento de ubicación, el monitoreo de ubicación y *Geofencing*, los cuales se explicarán más adelante.

De acuerdo con [23], se recomienda utilizar Google *Location Services* porque provee un *framework* más favorable, el cual facilita el trabajo de escoger el proveedor de ubicación (GPS, WiFi[55], GSM[56]), así como para mejorar el uso de la batería.

Google Location Services API

Los servicios de Google permiten aprovechar características propias de Google como Mapas, Google+, Google Drive, entre otros. Además permite a los usuarios recibir actualizaciones de manera rápida a través de Google Play Store. También, facilita la integración de nuevas funcionalidades que va a desarrollar Google en un futuro.

Para llevar a cabo esto, los servicios de Google cuentan con una librería cliente, la cual contiene las interfaces para acceder a los servicios en cuestión, como se muestra en la Figura 1.5. Cabe destacar que se debe tener instalado las herramientas de servicio Google Play en el SDK *Tools* de Android Studio.

[55] WiFi (*Wireless Fidelity*): es el nombre comercial del estándar 802.11 para comunicaciones inalámbricas en área local.
[56] GSM (*Global System for Mobile Communications*): es un sistema digital estandarizada que ofrece servicios de telefonía móvil de segunda generación a los usuarios.

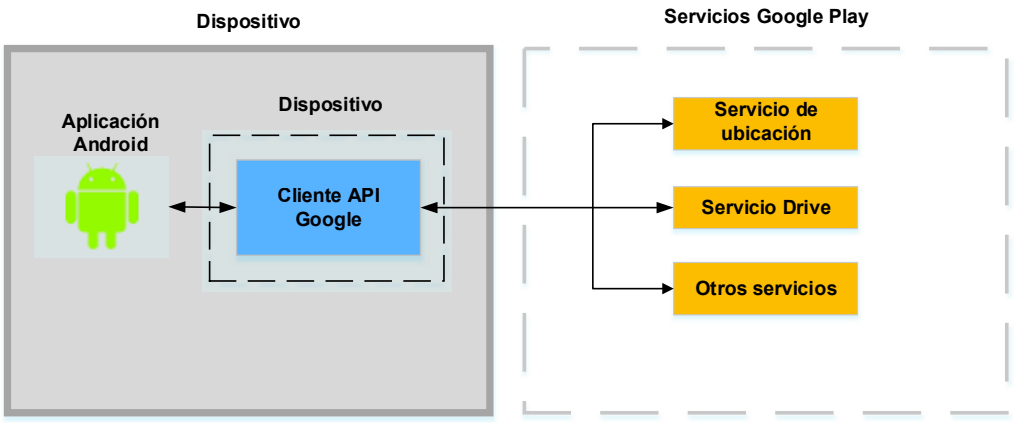


Figura 1.5.- Componentes servicios Google Play [24]

En el Código 1.14 se muestra un ejemplo para configurar el cliente `GoogleApiClient`. En la línea 1, se llama al constructor de `GoogleApiClient` para realizar la configuración del cliente con diversos parámetros que siguen a continuación; en la línea 2, se especifica que se van a emplear los servicios de ubicación de Google; la línea 3 permite registrar un manejador de eventos de conexión del cliente; en las líneas 4 a 7, se registra un mensaje de éxito en *logs* cuando el cliente se conecte satisfactoriamente; en las líneas 9 a 12, se registra un mensaje de suspensión en *logs* cuando el cliente se suspende; en las líneas 14 a 20, se registra un manejador de eventos con errores de conexión, el cual, se escribe mensajes de error en *logs* cuando no se pueda conectar con el cliente; en la línea 21 se crea una instancia de `GoogleApiClient` para que se pueda comunicarse con la API de ubicación de Google.

```
1.  googleApiClient = new GoogleApiClient.Builder(this)
2.                  .addApi(LocationServices.API)
3.                  .addConnectionCallbacks(new GoogleApiClient.ConnectionCallbacks() {
4.                      @Override
5.                      public void onConnected(@Nullable Bundle bundle) {
6.                          Log.d(TAG, "Conectado a GoogleApiClient");
7.                      }
8.                      //Si esque se suspende la conexión
9.                      @Override
10.                     public void onConnectionSuspended(int i) {
11.                         Log.d(TAG, "Conexión suspendida a GoogleApiClient");
12.                     }
13.                 })
14.                 .addOnConnectionFailedListener(new GoogleApiClient.OnConnectionFailedListener() {
15.                     //cuando hay un error en conectar con el cliente.
16.                     @Override
17.                     public void onConnectionFailed(ConnectionResult connectionResult) {
18.                         Log.d(TAG, "fallo en la conexión con GoogleApiClient-" + connectionResult.getErrorMessage());
19.                     }
20.                 })
21.                 .build();
```

Código 1.14.- Ejemplo de configuración `GoogleApiClient`

Geofencing

Geofencing o monitoreo de geovallas es una de las funcionalidades más llamativas de los servicios de ubicación de Google porque permite crear áreas virtuales en los cuales se disparen eventos cuando un dispositivo entre, salga, o permanezca por cierto tiempo dentro del área en cuestión, con la finalidad de monitorear la posición de los dispositivos en lugares específicos.

Antes de empezar a utilizar *geofencing*, se debe primero crear una instancia de la clase `GoogleApiClient`, que sirve para acceder a los servicios de la API de ubicación, como se lo puede ver en el Código 1.14.

Después, se debe crear una solicitud de ubicación mediante el objeto `LocationRequest`, para especificar la frecuencia y la precisión de las actualizaciones de ubicación. En el Código 1.15, se muestra un ejemplo de instancia de una solicitud de ubicación que va a pedir actualizaciones en intervalos de 5 y 10 segundos, con requerimientos de ubicación de alta precisión, que puede emplear GPS como proveedor principal.

```
1.  LocationRequest locationRequest = LocationRequest.create()
2.      .setInterval(10000) //cada 10 segs
3.      .setFastestInterval(5000) // 5 segs
4.      .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
```

Código 1.15.- Ejemplo instancia de la clase `LocationRequest`

Posteriormente, se envía la solicitud a través del método `requestLocationUptdates` de la clase `LocationServices.FusedLocationApi`, que permite a la aplicación obtener actualizaciones constantes de coordenadas geográficas. El Código 1.16 se muestra un ejemplo para enviar solicitudes de ubicación y recibir actualizaciones de ubicación. En la línea 4, se envía la solicitud de ubicación al cliente `GoogleApiClient`, además se registra un manejador de eventos; en las líneas 6 y 7, se escribe en *logs* la latitud y longitud, cada vez que se reciban actualizaciones de ubicación.

```
1. //Verificar permisos
2. ...
3. //Se envían las solicitudes
4. LocationServices.FusedLocationApi.requestLocationUpdates(googleApiClient, locationRequest, new LocationListener() {
5. @Override
6. public void onLocationChanged(Location location) {
7. Log.d(TAG,"Actualizacion ubicacion latitud/longitud"+ location.getLatitude()+" "+location.getLongitude());
8. }
9. });
```

Código 1.16.- Ejemplo empezar actualizaciones de ubicación

Luego, se definen los parámetros de la geovalla como: identificador, latitud, longitud, radio, tiempo de expiración, tiempo de respuesta y tipo de transición. Para ello se emplea la clase `Geofence` con sus correspondientes métodos `set`. Se tienen tres tipos de transiciones dentro de las geovallas como se muestra en la Figura 1.6: `GEOFENCE_TRANSITION_ENTER`, que monitorea cuando un dispositivo entre a la geovalla; `GEOFENCE_TRANSITION_EXIT`, que monitorea cuando un dispositivo sale de la geovalla; `GEOFENCE_TRANSITION_DWELL`, que monitorea cuando un dispositivo permanece dentro de la geovalla por un intervalo de tiempo. En el Código 1.17 se instancia una geovalla con id `GEOFENCE_ID` (línea 2), de región circular con centro en las coordenadas (`-0.271467, -78.481058`) de radio 100 metros (línea 3), sin tiempo de expiración (línea 4), con un tiempo de respuesta de 1 segundo luego de que se dispare una transición (línea 5), con especificación de tipo de transición de entrada y salida (línea 6).

Figura 1.6.- Tipos de transiciones geovalla [25]

```
1. Geofence geofence = new Geofence.Builder()
2.                     .setRequestId(GEOFENCE_ID)
3.                     .setCircularRegion(-0.271467, -78.491058, 100)
4.                     .setExpirationDuration(Geofence.NEVER_EXPIRE)
5.                     .setNotificationResponsiveness(1000)
6.                     .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER | Geofence.GEOFENCE_TRANSITION_EXIT)
7.                     .build();
```

Código 1.17.- Ejemplo instancia de la clase `Geofence`

Después, se crea una instancia de la clase `GeofenceRequest`, que sirve para añadir las geovallas previamente creadas. Esta instancia contiene un conjunto de geovallas con un número máximo de cien. En el Código 1.18 se instancia un `GeofencingRequest` que establece como evento disparador inicial la transición de entrada de la geovalla (línea 2); en la línea 3, se añade la geovalla previamente creada y se instancia el objeto `GeofencingRequest`.

```
1. GeofencingRequest geofencingRequest = new GeofencingRequest.Builder()
2.       .setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER)
3.       .addGeofence(geofence).build();
```

Código 1.18.- Ejemplo instancia de la clase `GeofencingRequest`

Así mismo, se debe implementar una clase que extienda de `IntentService`, para manipular intentos pendientes, los cuales permiten realizar tareas de larga duración en segundo plano. En esta clase se manejan las transiciones ocurridas en las geovallas durante el tiempo de ejecución. En el Código 1.19, se muestra un ejemplo del manejo de las transiciones. En la línea 3, se captura el evento disparado por el intento pendiente; se obtiene el tipo de transición ocurrida (línea 6); se obtiene una lista de geovallas que tuvieron transiciones (línea 7); en las líneas 8 a 9, se obtiene la primera geovalla y su correspondiente identificador de la lista `geofences`; en las líneas 10 a 11, se maneja la transición de entrada; en las líneas 12 a 13 se maneja la transición de salida.

```
1.    @Override
2.    protected void onHandleIntent(Intent intent) {
3.        GeofencingEvent event= GeofencingEvent.fromIntent(intent);
4.        if(event.hasError()){
5.        }else {
6.            int transicion= event.getGeofenceTransition();
7.            List<Geofence> geofences = event.getTriggeringGeofences();
8.            Geofence geofence= geofences.get(0);
9.            String requestId= geofence.getRequestId();
10.           if(transicion==Geofence.GEOFENCE_TRANSITION_ENTER){
11.               Log.d(TAG, "Entrando geofence "+requestId);
12.           }else if(transicion== Geofence.GEOFENCE_TRANSITION_EXIT){
13.               Log.d(TAG, "Saliendo geofence "+ requestId);
14.           }
15.       }
16.   }
```

Código 1.19.- Ejemplo de manejo de transiciones de geovalla

Con la finalidad de iniciar el monitoreo de geovallas, se utiliza el método `addGeofences` de la clase `LocationServices.GeofencingApi`. En el Código 1.20 se añade una nueva geovalla al cliente de la API de servicios de Google con su respectivo intento pendiente (línea 1); en las líneas 2 a 9, se manejan los eventos de éxito o fracaso del proceso en cuestión.

```
1.  LocationServices.GeofencingApi.addGeofences(googleApiClient, geofencingRequest,pendingIntent)
2.                          .setResultCallback(new ResultCallback<Status>() {
3.                              @Override
4.                              public void onResult(Status status) {
5.                                  if(status.isSuccess()){
6.                                      Log.d(TAG, "Se añadió exitosamente geofence");
7.                                  }else {
8.                                      Log.d(TAG, "Fallo en añadir geofence +"+ status.getStatus());
9.                                  }
10.
11.                             }
12.                         });
```

Código 1.20.- Ejemplo iniciar monitoreo *geofencing*

Por último, para detener el monitoreo de las geovallas se lo realiza a través el método `removeGeofences` de la clase `LocationServices.GeofencingApi`[24]. En el Código 1.21 se muestra un ejemplo para detener el monitoreo de geovallas. En las líneas 1 y 2, se crea una lista que almacena los identificadores de las geovallas; en la línea 3, se eliminan las geovallas especificadas del cliente `googleApiClient`.

```
1.  ArrayList<String> geofenceIds= new ArrayList<String >();
2.          geofenceIds.add(GEOFENCE_ID);
3.          LocationServices.GeofencingApi.removeGeofences(googleApiClient, geofenceIds);
```

Código 1.21.- Ejemplo detener *geofencing*

Para que funcione la aplicación con monitoreo de geovallas, se debe añadir en el archivo `Gradle.build` las dependencias de los servicios de Google: `compile 'com.google.android.gms:play-services:11.0.4'`. Además se debe añadir los permisos de ubicación `android.permission.ACCESS_COARSE_LOCATION` y `android.permission.ACCESS_FINE_LOCATION` en el archivo `AndroidManifest`. Además, se debe añadir código para el correcto manejo de la verificación de estos permisos en tiempo de ejecución.

Bases de datos

Un sistema gestor de bases de datos (SGBD) es un conjunto de datos relacionados entre sí y un grupo de programas que permiten el acceso y la manipulación los datos en cuestión. Además, estos sistemas tienen la finalidad de ofrecer almacenamiento y recuperación de información de una base de datos de manera práctica y eficiente.

Las bases de datos son colecciones de datos que contienen información importante sobre salarios, empleados, estudiantes, cuentas bancarias, etc. Por lo que forman parte esencial en casi todas las empresas e instituciones actuales.

Para describir los datos, las relaciones y restricciones de consistencia de una base de datos, se consideran varios conceptos:

- Entidades: Son objetos de la vida real distinguible de otros objetos y se describen mediante un conjunto de atributos.
- Atributos: Representan características de las entidades y proporcionan detalles sobre ellos [26].
- Clave primaria: Se trata de un atributo de una entidad que sirve como elemento principal para identificar alguna entidad en particular.
- Clave foránea: Es un atributo de una entidad que se relaciona con la clave primaria de otra entidad.
- Relación: Es una asociación ente varias entidades.
- Cardinalidad: Expresa el número de entidades con las que otra entidad se puede asociar a través de un conjunto de relaciones.
- Tablas: Conjunto de filas y columnas que sirven para representar entidades.
- Columnas: Cada una de las columnas poseen un nombre único, el cual sirve para representar los valores de los atributos de las entidades.
- Registros: Hace referencia a las filas de una tabla, los cuales representan los objetos con sus correspondientes atributos.
- Consulta: Conjunto de sentencias que permiten solicitar la recuperación de información [27].

Uno de los gestores de bases de datos más empleados es Microsoft SQL Server Management Studio el cual es un IDE que proporciona herramientas necesarias para configurar, monitorear, administrar las bases de datos, además permite crear consultas y *scripts* [28]. Así mismo, este programa emplea el lenguaje SQL (*Structured Query Language*) con la finalidad de definir la estructura de los datos, además de permitir expresar consultas y manipular (recuperar, insertar, borrar, modificar) la información de las

bases de datos. Cabe mencionar que este lenguaje solo requiere que el usuario especifique los datos que se necesiten, sin especificar cómo obtener dichos datos, por lo que, este lenguaje resulta ser fácil de aprender y usar [27].

Metodología de desarrollo de software SCRUM

SCRUM es un marco de trabajo para el desarrollo y mantenimiento de productos complejos. Los usuarios de esta metodología deben inspeccionar los artefactos de SCRUM[57] de manera frecuente, así mismo del progreso hacia un objetivo específico.

En SCRUM se tienen algunos conceptos que son:

- *Sprint*: Es un bloque de tiempo de duración en el cual se desarrolla un producto utilizable y consisten en:
- *Sprint planning meeting* o reunión de planificación de *sprint*: Es una reunión en la cual se planifica el trabajo que se va a realizar durante el *sprint*.
- *Sprint goal*: Es una meta establecida para el *sprint*
- *Sprint review*: Es una reunión informal que se lleva a cabo al final del *sprint*, en el cual se presentan los avances del producto con el objetivo de facilitar la retroalimentación.

Algunos de los artefactos empleados en SCRUM son:

- *Product backlog* o lista de producto: Es un lista ordenada de requerimientos de usuario y contiene todo lo que podría ser necesario en el producto. Esta lista va evolucionando a medida que el producto y el entorno lo hacen, es decir, no es estático. Además, la lista de producto enumera las características, funcionalidades, requisitos, mejoras y correcciones, que se pueden realizar sobre el producto para entregas futuras.
- *Sprint backlog* o lista de pendientes: Es un conjunto de elementos seleccionados de la lista de producto para el *sprint* junto con un plan para completar los objetivos o requisitos de la iteración [29].

Herramientas que se van a emplear

El presente trabajo de titulación se van a usar servicios WCF REST basados en JSON, se desarrollarán aplicaciones en .NET, se desarrollarán aplicaciones en Android, se empleará la API de huella dactilar Android, se usará *Geofencing*, se desarrollará una base de datos,

[57] Artefactos SCRUM: representa elementos físicos que se producen como resultado de la aplicación de SCRUM.

se empleará la librería Volley para consumir servicios en Android y se usará la librería GSON para serializar y de-serializar objetos JSON en Android.

1.4 Relación con trabajos afines

Antes del desarrollo de este trabajo se determinaron que existen trabajos afines como:

- Sistema que permite controlar el acceso y asistencia del personal para la empresa Human Trend. [30].
- Sistema de control de asistencia del personal mediante el uso de tecnología biométrica de huella dactilar [31].

En los trabajos mencionados y en el presente trabajo de titulación se perseguirán los mismos objetivos de controlar la asistencia del personal de una determinada Institución o empresa a través de herramientas biométricas. Sin embargo, se diferenciarán en estos puntos en particular:

- Los trabajos afines emplean una hardware biométrico externo para el control de asistencia, mientras que en el presente trabajo se utilizarán los sensores biométricos integrados en los teléfonos inteligentes.
- En los trabajos mencionados se desarrollaron terminales estáticos que no permite movilidad en el control de asistencia, mientras que en el presente trabajo, a través del aplicativo móvil y la red inalámbrica, se podrá controlar la asistencia desde cualquier punto dentro de la cobertura de red inalámbrica.
- Los trabajos afines no manejan ninguna información de ubicación, en cambio, en el presente trabajo, permite monitorear cuando un empleado entre o salga de un espacio en particular.
- Por último, los trabajos afines no permiten al personal consultar sobre sus propios registros de asistencia, mientras que, el presente trabajo permite al personal consultar sus registros de asistencia, faltas injustificadas, atrasos, días festivos, entre otros.

2. METODOLOGÍA

El presente proyecto integrador se desarrollará empleando una investigación de tipo aplicada, en el cual se emplearán los conocimientos obtenidos con respecto al desarrollo de servicios WCF basados en JSON, aplicaciones Android, autenticación de huella dactilar Android, *Geofencing* y bases de datos para el desarrollo de los componentes del prototipo, el cual realizará control de asistencia a los usuarios Android y presentarán los reportes de asistencias a los usuarios gestores.

Se recolectará información referente a los requerimientos de usuario por medio de encuestas. Mediante el análisis de estas encuestas, se determinarán los requerimientos del usuario y se levantarán historias de usuarios para detallarlos.

La metodología de desarrollo de software SCRUM será empleada para el desarrollo del prototipo y se emplearán los artefactos de esta metodología como: *product backlog* y *sprint backlog* con la finalidad de realizar el seguimiento de los requerimientos y dividirlos en iteraciones.

El diseño del prototipo se lo realizará mediante diagramas de casos de uso, diagramas entidad-relación, diagramas de clases, *sketches*, *wireframes*, y diagramas de actividades.

La implementación se lo realizará empleando tecnologías de Android Studio, Visual Studio, SQL Server y servidor IIS, siguiendo los *backlogs* de SCRUM.

Al finalizar de cada iteración se presentarán los avances del prototipo para su debida retroalimentación.

Una vez implementado el prototipo, se probará el funcionamiento de todos los componentes del sistema y se realizarán las correcciones respectivas de ser necesario. Se realizarán y se analizarán encuestas de validación dirigidos al personal administrativo para verificar el cumplimiento de los requerimientos establecidos.

2.1. Diseño del prototipo

Arquitectura del prototipo

El prototipo estará compuesto por: un Cliente Android, un Servicio WCF, el Código para el control de asistencia, un Cliente Gestor y una Base de Datos, como se lo observa en la Figura 2.1. Se empleará el protocolo de comunicación HTTP y se empleará una arquitectura Cliente-Servidor.

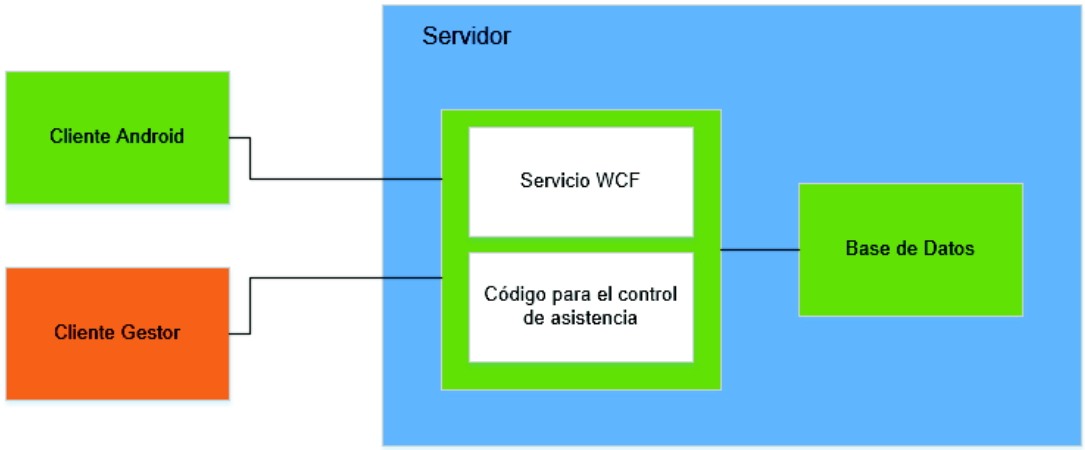


Figura 2.1.- Arquitectura del prototipo

Requerimientos de usuario

Para determinar el Sistema Operativo para el cual se desarrollará el aplicativo móvil, qué información se debe mostrar al usuario y cómo se debe realizar el proceso de registro de asistencia tanto para la entrada como la salida, se realizó una encuesta a los usuarios. La encuesta se encuentra en el Anexo I. Esta encuesta permitirá justificar los requerimientos de usuario establecidos en el plan de trabajo de titulación y permitirá además establecer los requerimientos de usuario iniciales. La mayoría de preguntas de la encuesta son de tipo cerradas y dicotómicas, las cuales los encuestados deben elegir una respuesta entre dos alternativas (si - no). Estos tipos de preguntas tienen como ventaja su fácil respuesta y procesamiento. Sin embargo, la información que ofrece es limitada [32]. Por este motivo, se puede hacer un análisis de resultado más conveniente si se tienen preguntas cerradas que preguntas abiertas, porque si bien este último permite a los encuestados exponer todos sus criterios, da la posibilidad a que estos establezcan funcionalidades fuera del alcance del presente trabajo de titulación. Quedó a discreción del autor en realizar la encuesta de esta manera por los motivos mencionados. A continuación, se muestran los resultados obtenidos de esta encuesta.

La primera pregunta permite conocer si los usuarios poseen un dispositivo móvil. En la Figura 2.2 se observa que todos los encuestados poseen un dispositivo móvil.

La segunda pregunta permite conocer el Sistema Operativo más usado en los teléfonos inteligentes de los usuarios. En la Figura 2.3 se muestra que el 90% de los encuestados emplean el Sistema Operativo Android en sus dispositivos móviles.

La tercera pregunta permite conocer el sistema de control de asistencia más empleado por los usuarios a inicios del año 2019. En la Figura 2.4 se muestra que todos los encuestados emplean dispositivos biométricos para registrar la asistencia diaria.

La cuarta pregunta permite conocer si los usuarios desean contar con una aplicación móvil para registrar la asistencia. En la Figura 2.5 se muestra que a todos los encuestados les parece una buena idea contar con una aplicación móvil que les permita realizar su registro de asistencia.

Figura 2.2.- Respuestas a la primera pregunta de la encuesta

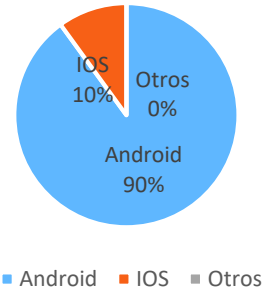


Figura 2.3.- Respuestas a la segunda pregunta de la encuesta

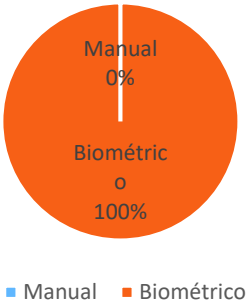


Figura 2.4.- Respuestas a la tercera pregunta de la encuesta

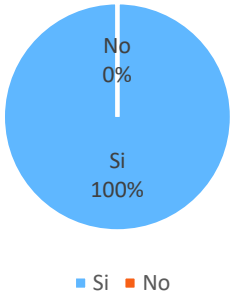


Figura 2.5.- Respuestas a la cuarta pregunta de la encuesta

La quinta pregunta permite determinar si los usuarios desean observar su propio horario de trabajo en el aplicativo móvil. En la Figura 2.6 se muestra que a todos los encuestados les parece una buena idea que la aplicación mostrara su horario de trabajo.

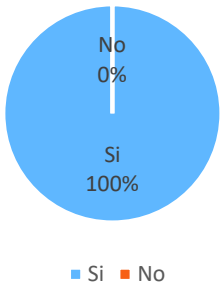


Figura 2.6.- Respuestas a la quinta pregunta de la encuesta

La sexta pregunta permite determinar si los usuarios desean emplear su huella dactilar para registrar su asistencia. En la Figura 2.7 se muestra que a todos los encuestados desean que el aplicativo móvil emplee su huella dactilar para realizar el registro de asistencia.

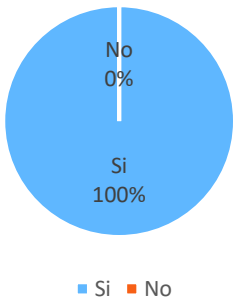


Figura 2.7.- Respuestas a la sexta pregunta de la encuesta

La séptima pregunta permite conocer si los usuarios desean contar con un PIN (4 dígitos numéricos) como mecanismo de respaldo para el registro de asistencia. En la Figura 2.8

se muestra que a todos los encuestados les interesa que el aplicativo móvil cuente con un PIN como mecanismo de respaldo para el registro de asistencia.

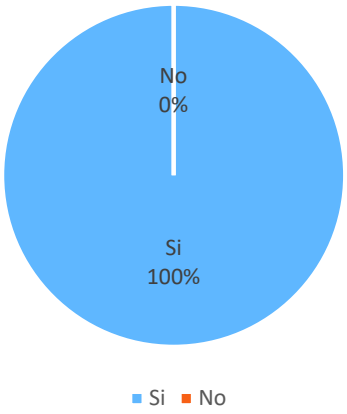


Figura 2.8.- Respuestas a la séptima pregunta de la encuesta

La octava pregunta permite conocer si los usuarios desean emplear su huella dactilar para realizar su registro de salida. En la Figura 2.9 se muestra que a todos los encuestados les parecería una buena idea que cuando terminen su jornada de trabajo, tengan que colocar su huella dactilar en la aplicación móvil para que se registre su salida.

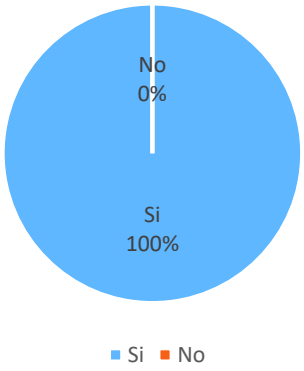


Figura 2.9.- Respuestas a la octava pregunta de la encuesta

El manejo de eventos de transición de salida del *Geofencing* da la posibilidad de proporcionar una alternativa para el registro de salida, para cuando el usuario salga de la geovalla, se registre automáticamente su salida. Por lo que en la novena pregunta permite conocer si los usuarios desean registrar las salidas de manera automática, sin que ellos actúen. En la Figura 2.10 se muestra que el 80% de los encuestados si les parecería una buena idea que cuando terminen sus jornadas de trabajo, se registre la salida sin que ellos intervengan.

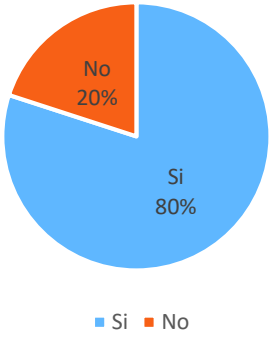


Figura 2.10.- Respuestas a la novena pregunta de la encuesta

La décima pregunta permite determinar si los usuarios desean que el aplicativo móvil les muestre información sobre sus asistencias. En la Figura 2.11 se muestra que a todos los encuestados les gustaría que la aplicación les muestre sus horas de ingreso, salida y atrasos.

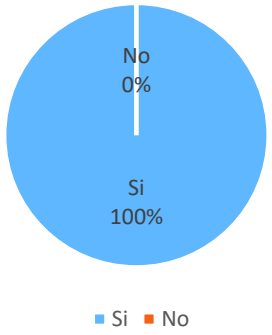


Figura 2.11.- Respuestas a la décima pregunta de la encuesta

De los resultados de la encuesta se han obtenido los siguientes requerimientos funcionales iniciales que van a formar parte del prototipo:

- El aplicativo móvil se lo desarrollará para el Sistema Operativo Android.
- Se le mostrará al usuario su horario de trabajo.
- Se realizará el registro de entrada de asistencia a través de huella dactilar.
- Se empleará un PIN como mecanismo de respaldo tanto para el registro de entrada como la salida.
- Se presentarán dos opciones de registro de salida: por medio de huella dactilar o de manera automática sin que el usuario intervenga
- Se mostrará al usuario registros referentes a horas de ingreso, salida y atrasos.

Historias de usuario

El sistema de control de asistencia va a estar conformado por dos actores: usuario del Cliente Android y usuario del Cliente Gestor. El primero hace referencia a empleados del personal administrativo, y en el segundo se trata del personal de talento humano que administran al personal de trabajo. La encuesta realizada en la sección anterior permitió establecer los requerimientos iniciales de usuario que sirvió como punto de partida, y por medio de reuniones con los usuarios se pudieron establecer requerimientos funcionales adicionales y específicos en forma de historias de usuario. El formato que tendrán las historias de usuarios son:

- ID: es el identificador de la historia de usuario, empieza con las siglas HU (Historia Usuario) seguido de un número con formato de tres dígitos, por ejemplo: HU002.
- Prioridad: denota la prioridad que representa el historia de usuario
- Nombre: es un nombre descriptivo que permite identificar el requerimiento del usuario.
- Descripción: detalla el quién, qué y para qué del requerimiento de usuario.

En la Tabla 2.1 se describe un requerimiento de nivel de prioridad alta para que el usuario del Cliente Android pueda registrar su entrada por medio de huella dactilar.

Tabla 2.1.- Historia de usuario registrar entrada con huella dactilar

ID: HU002	**PRIORIDAD: 4**
NOMBRE: Los usuarios del Cliente Android pueden registrar su hora de entrada con huella dactilar	
DESCRIPCIÓN: Como usuario del Cliente Android, quiero poder ingresar mi huella dactilar para registrar mi hora de entrada.	

En la Tabla 2.2 se muestran cuatro niveles de prioridad de historias de usuario, ordenadas de forma ascendente, siendo el nivel cuatro el más prioritario. En el Anexo II se detallan todas las historias de usuario tanto para el usuario del Cliente Android como para el usuario del Cliente Gestor.

Tabla 2.2.- Tabla de prioridades de historias de usuario

Nivel bajo de prioridad	Nivel medio bajo prioridad	Nivel medio alto prioridad	Nivel alto de prioridad
1	2	3	4

Pila del producto

La pila del producto permite recopilar y ordenar los requerimientos funcionales del sistema de acuerdo a la prioridad que representen, también permite realizar el seguimiento de estos requerimientos. Además, la pila del producto evoluciona con el paso del tiempo y describe las funcionalidades, las mejoras y las correcciones del sistema en forma de historias de usuario. Por lo que, después de cada reunión con los usuarios, la pila del producto aumenta en caso de que el sistema requiera nuevas funcionalidades.

La Tabla 2.3, la Tabla 2.4 y la Tabla 2.5 describen el conjunto de requerimientos que tendrá el sistema de manera priorizada, así como el *sprint* o iteración en que se lo llevará a cabo.

Tabla 2.3.- Pila del producto (Parte 1)

ID HU	Nombre de la Historia	Sprint
HU001	Los usuarios del cliente Android pueden ingresar al sistema con credenciales	1
HU002	Los usuarios del cliente Android pueden registrar su hora de entrada con huella dactilar	1
HU003	Los usuarios del cliente Android pueden registrar su hora de entrada con un PIN	1
HU004	Los usuarios del cliente Android pueden registrar su hora de salida con huella dactilar	2
HU005	Los usuarios del cliente Android pueden registrar su hora de salida con un PIN	2
HU006	Los usuarios del cliente Android pueden registrar su hora de salida automáticamente	2
HU007	Los usuarios del cliente Android pueden recibir notificaciones del estado de registro de asistencia automática	2
HU008	Los usuarios del cliente Android pueden registrar entrada en horario no convencional	2
HU009	Los usuarios del cliente Android pueden registrar salida en horario no convencional	2
HU010	Los usuarios del cliente Android pueden consultar su horario de trabajo	3
HU011	Los usuarios del cliente Android pueden consultar sus registros de entrada	3
HU012	Los usuarios del cliente Android pueden consultar sus registros de salida	3
HU013	Los usuarios del cliente Android pueden consultar sus atrasos	3
HU014	Los usuarios del cliente Android pueden consultar sus horas totales trabajadas	3
HU015	Los usuarios del cliente Android pueden consultar sus asistencias	3
HU016	Los usuarios del cliente Android pueden consultar los días de feriado	3

Tabla 2.4.- Pila del producto (Parte 2)

ID HU	Nombre de la Historia	Sprint
HU017	Los usuarios del cliente Android pueden consultar las solicitudes de permiso emitidas para faltar al trabajo	4
HU018	Los usuarios del cliente Android pueden consultar las solicitudes de trabajo no convencionales	4
HU019	Los usuarios del cliente Android pueden consultar los días que han faltado	4
HU020	Los usuarios del cliente Android pueden solicitar permiso para faltar al trabajo	4
HU021	Los usuarios del cliente gestor pueden ingresar al sistema con credenciales	5
HU022	Los usuarios del cliente gestor pueden agregar empleados	5
HU023	Los usuarios del cliente gestor pueden modificar empleados	5
HU030	Los usuarios del cliente gestor pueden asignar horarios de trabajo a empleados	5
HU031	Los usuarios del cliente gestor pueden modificar el horario de trabajo de empleados	5
HU032	Los usuarios del cliente gestor pueden eliminar el horario de trabajo	5
HU033	Los usuarios del cliente gestor pueden cambiar las contraseñas de credenciales de empleados	5
HU034	Los usuarios del cliente gestor pueden buscar lista de credenciales de empleados	5
HU035	Los usuarios del cliente gestor pueden agregar registros de asistencia de los empleados	5
HU036	Los usuarios del cliente gestor pueden modificar registros de asistencia de los empleados	6
HU037	Los usuarios del cliente gestor pueden eliminar registros de asistencia de los empleados	6
HU038	Los usuarios del cliente gestor pueden ver los registros de asistencia de los empleados	6
HU039	Los usuarios del cliente gestor pueden mantener un historial de cambios de registros de asistencias	6
HU040	Los usuarios del cliente gestor pueden ver los atrasos de los empleados	6
HU041	Los usuarios del cliente gestor pueden ver las faltas de los empleados	6
HU042	Los usuarios del cliente gestor pueden justificar faltas de los empleados	6
HU043	Los usuarios del cliente gestor pueden mantener un historial de cambios de registros de faltas	6
HU044	Los usuarios del cliente gestor pueden agregar el área de monitoreo	6
HU045	Los usuarios del cliente gestor pueden modificar las áreas de monitoreo existentes	6
HU046	Los usuarios del cliente gestor pueden ver notificaciones cuando un empleado entre o salga de la universidad	6
HU047	Los usuarios del cliente gestor pueden ver el calendario de días de descanso obligatorio de los empleados	6
HU048	Los usuarios del cliente gestor pueden crear el calendario de días de descanso obligatorio de los empleados	6
HU049	Los usuarios del cliente gestor pueden modificar el calendario de días de descanso obligatorio de empleados	6

Tabla 2.5.- Pila del producto (Parte 3)

ID HU	Nombre de la Historia	Sprint
HU050	Los usuarios del cliente gestor pueden eliminar el calendario de días de descanso obligatorio de los empleados	6
HU051	Los usuarios del cliente gestor pueden aceptar solicitudes de permisos que fueron solicitados por los empleados	6
HU052	Los usuarios del cliente gestor pueden rechazar solicitudes de permisos que fueron solicitados por los empleados	6
HU053	Los usuarios del cliente gestor pueden buscar solicitudes de permisos que fueron solicitados por los empleados	6
HU054	Los usuarios del cliente gestor pueden mantener un historial de cambios de solicitudes de permiso	6
HU055	Los usuarios del cliente gestor pueden solicitar trabajo no convencional	6
HU056	Los usuarios del cliente gestor pueden buscar solicitudes de trabajo no convencional	6
HU057	Los usuarios del cliente gestor pueden modificar solicitudes de trabajo no convencional	6
HU058	Los usuarios del cliente gestor pueden eliminar solicitudes de trabajo no convencional	6
HU059	Los usuarios del cliente gestor pueden mantener un historial de cambios de solicitudes de trabajo no convencional	6
HU060	Los usuarios del cliente gestor pueden ver los cambios realizados a registros de asistencia	7
HU061	Los usuarios del cliente gestor pueden ver los cambios realizados a registros de atraso	7
HU062	Los usuarios del cliente gestor pueden ver los cambios realizados a registros de falta	7
HU063	Los usuarios del cliente gestor pueden ver los cambios realizados a solicitudes de trabajo no convencional	7
HU064	Los usuarios del cliente gestor pueden ver los cambios realizados a solicitudes de permiso	7
HU065	Los usuarios del cliente gestor pueden configurar la hora mínima para el registro de entrada de empleados	7
HU066	Los usuarios del cliente gestor pueden configurar la hora máxima para el registro de salida de empleados	7
HU067	Los usuarios del cliente gestor pueden configurar la hora en que se marcará atraso	7
HU068	Los usuarios del cliente gestor pueden ver el total de horas trabajadas de los empleados	7

Pila de sprints

Con la finalidad de recopilar una lista de tareas a realizar durante una iteración o *sprint* se emplea la pila de *sprints* o *sprint backlog*. En esta pila se detallan las tareas de diseño, implementación y pruebas para cada requerimiento de usuario, sin embargo, en el presente trabajo de titulación, no se especificará tal detalle. Por otra parte, se enfocará más en el cumplimiento de los requerimientos en su totalidad.

En la Tabla 2.6 se describen los objetivos a alcanzar durante el primer *sprint.*

Tabla 2.6.- Pila de *sprint* 1

ID HU	Nombre de la Historia
Sprint 1	
HU001	Los usuarios del cliente Android pueden ingresar al sistema con credenciales
HU002	Los usuarios del cliente Android pueden registrar su hora de entrada con huella dactilar
HU003	Los usuarios del cliente Android pueden registrar su hora de entrada con un PIN

Diagramas de casos de uso

Después del análisis de requerimientos, se definen dos actores que van a intervenir en el sistema de control de asistencia, y estos son: usuario del Cliente Android y usuario del Cliente Gestor. El primero empleará el prototipo de aplicación móvil con el objetivo principal de registrar su asistencia y para realizar consultas relacionadas a estas. El segundo actor empleará el prototipo gestor con la finalidad de realizar seguimiento referente a asistencias y mantener las configuraciones del sistema.

Los casos de uso del Cliente Android se muestra en la Figura 2.12.

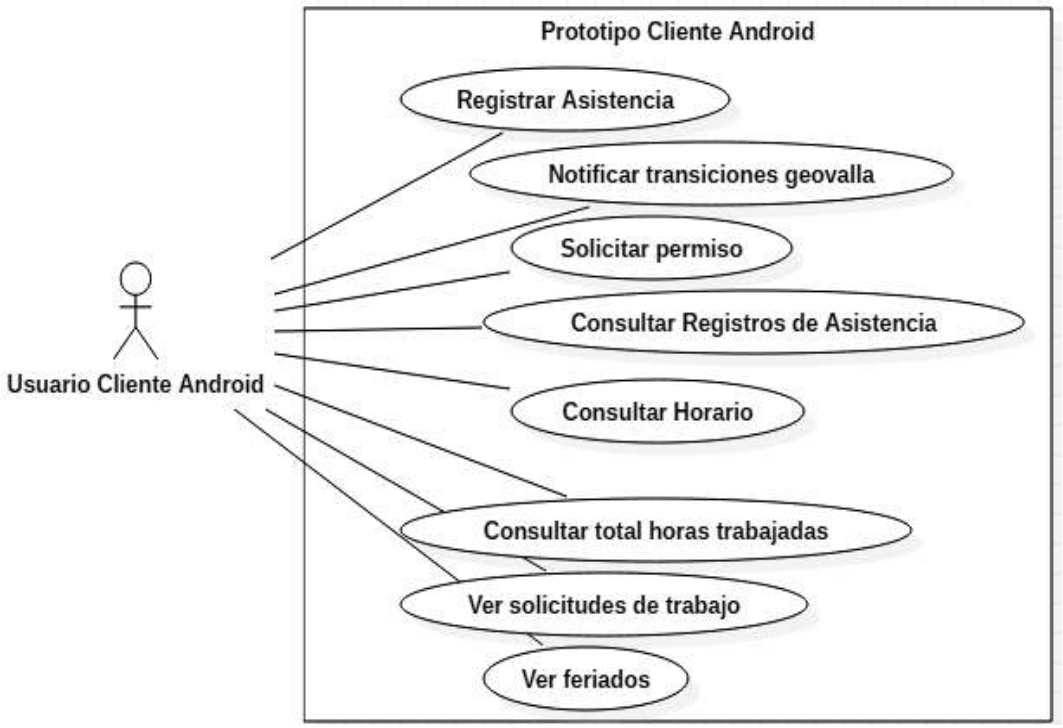


Figura 2.12.- Diagrama de Casos de Uso del Cliente Android

Registrar asistencia, permitirá al usuario registrar tanto la entrada como la salida para marcar su asistencia. Registrar entrada incluye registro de entrada por huella dactilar y por PIN. Registrar salida incluye registro de salida por huella, por PIN o de manera automática, sin que el usuario intervenga.

Notificar transiciones geovalla, permitirá al usuario enviar notificaciones de forma automática cuando el usuario entre o salga de un área de monitoreo.

Solicitar permiso, permitirá al usuario pedir permiso para ausentarse del trabajo por un motivo en particular.

Consultar registros de asistencia, permitirá al usuario realizar consultas referentes a asistencias, atrasos, faltas. Se permitirá al usuario realizar consultas relacionados a horarios, total horas trabajadas, días de feriado.

Por último, ver solicitudes de trabajo le permitirá al usuario consultar cuando las autoridades le convocan a trabajar en un día fuera de su jornada de trabajo normal.

En la Figura 2.13 se muestran los casos de uso del Cliente Gestor. Los casos de uso principales son: Gestionar empleados, Gestionar registros de asistencias, Configurar geovalla, Gestionar solicitudes de permiso, Gestionar solicitudes de trabajo y Configurar horas del sistema.

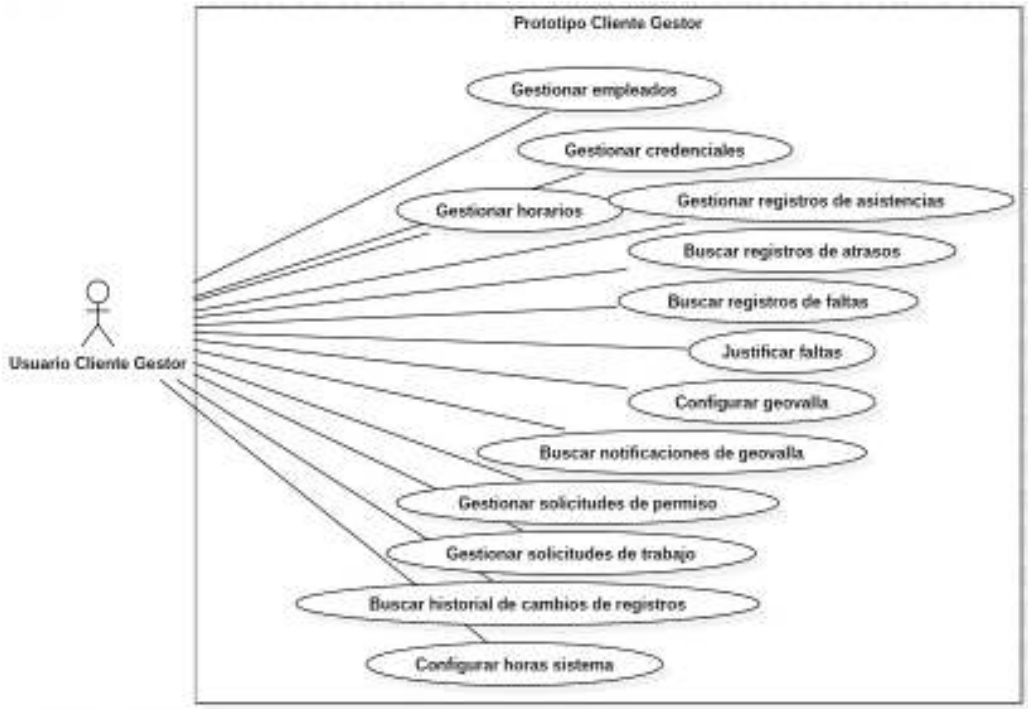


Figura 2.13.- Diagrama de Casos de Uso del Cliente Gestor

Gestionar empleado, permitirá al usuario administrar empleados del sistema, esto incluye agregar, modificar, deshabilitar, habilitar y buscar empleados.

Gestionar registros de asistencia, permitirá agregar o realizar cambios referentes a registros de asistencias.

Configurar geovalla, permitirá al usuario establecer el área virtual de monitoreo de los empleados.

Gestionar solicitudes de permiso, permitirá al usuario aprobar o rechazar solicitudes de permisos emitidas por los empleados.

Gestionar solicitudes de trabajo, permitirá al usuario solicitar a un empleado a trabajar en un día fuera de su jornada convencional.

Por último, Configurar horas del sistema, permitirá al usuario configurar la hora en que se considere atraso, así como la hora límite en que los empleados puedan marcar su correspondiente salida.

Diagrama Entidad Relación

La base de datos es el componente del sistema que permitirá guardar toda la información. Conforme a los requerimientos de usuario establecidos, se realizaron diagramas entidad – relación como se muestran en la Figura 2.14 y en la Figura 2.15.

La tabla `HorarioTrabajo` almacenará la información con respecto al horario del empleado, como la hora de entrada, la hora de salida, la hora de receso y los días de jornada laborales del empleado. Esta tabla tendrá un identificador único denominado `idHorarioTrabajo`.

La tabla `Credencial` contendrá información de los usuarios, como el nombre de usuario y contraseña, además del tipo de usuario y estado del usuario. El campo `TipoCuenta` permitirá establecer si se trata de un usuario o un administrador. El campo `estado` permitirá establecer si un usuario se encuentra habilitado o deshabilitado.

La tabla `Empleado` guardará toda la información referente a un empleado, como el nombre, apellido, cédula, teléfono, el horario de trabajo, la credencial asociada, la foto del empleado en cuestión y el código de cuatro dígitos de seguridad. El identificador único se denominará `idEmpleado`.

La tabla `SolicitudPermiso` es una tabla que permitirá a los empleados solicitar permiso para faltar. Contiene información con respecto al motivo, las fechas y horas en las que se llevará acabo el permiso, además del tiempo total del permiso en cuestión. El campo

`estado` permitirá establecer si el permiso se encuentra pendiente, aprobado o rechazado. Y en cuanto al campo `tipoSolicitudPermiso` permitirá distinguir el tipo de solicitud.

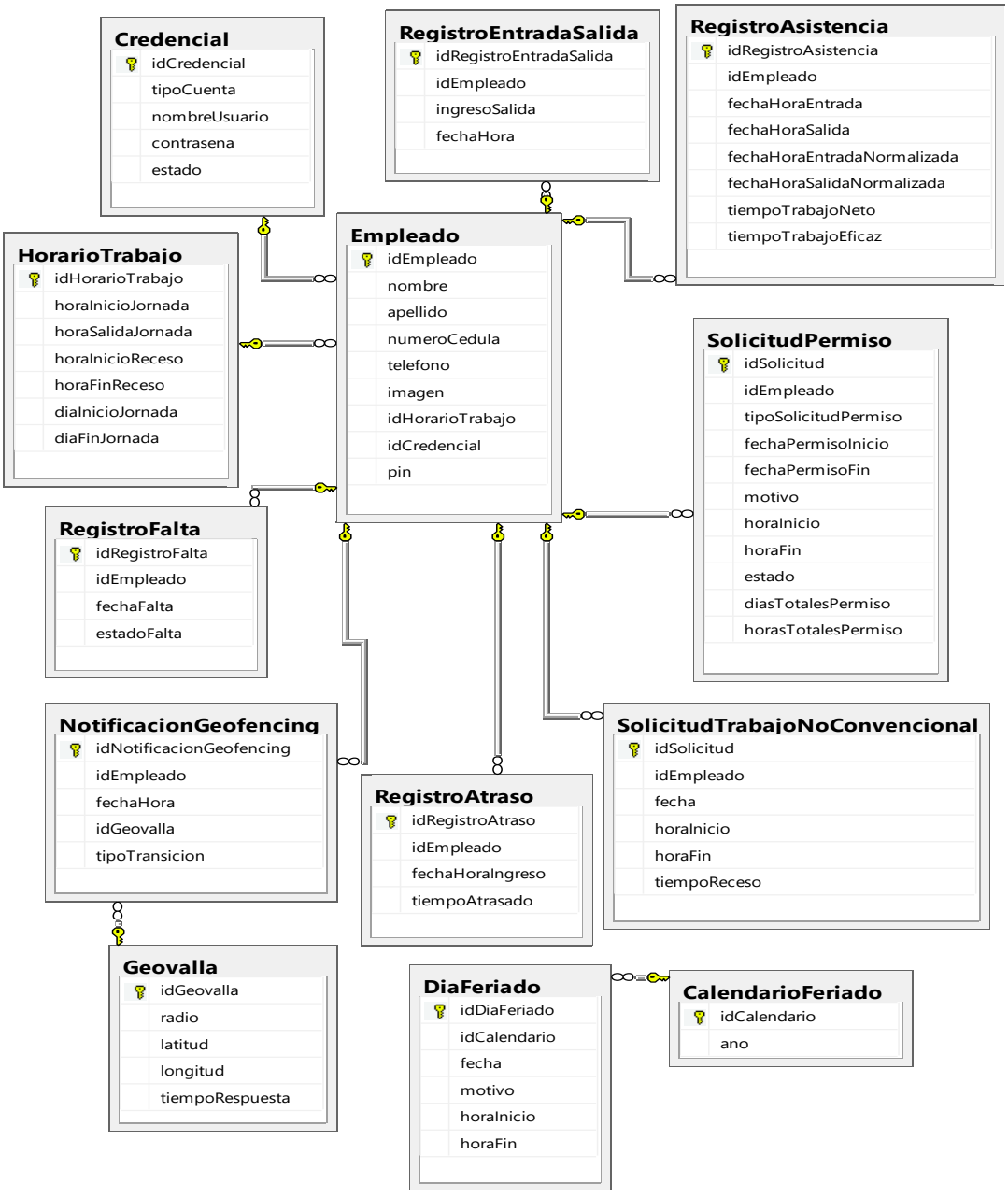


Figura 2.14.- Diagrama Entidad - Relación (Parte 1)

La tabla `RegistroEntradaSalida` servirá como una tabla auxiliar para crear registros de asistencias, y contendrá información sobre las fechas y horas de ingreso o salida del empleado. El campo `ingresoSalida` permitirá distinguir si un registro se trata de una entrada o de una salida.

La tabla `RegistroFalta` guardará información sobre los días en los que no asistan los empleados.

La tabla `RegistroAsistencia` guardará los días asistidos por el empleado. Los campos `tiempoTrabajoNeto` y `tiempoTrabajoEficaz` hacen referencia al tiempo total transcurrido. El segundo campo a diferencia del primero, solamente cuenta el tiempo transcurrido entre `horaInicioJornada` hasta `horaSalidaJornada` del horario del empleado, mientras que el primero cuenta el tiempo transcurrido desde que el empleado marque su entrada hasta que marque su salida. Los campos `fechaHoraEntradaNormalizada` y `fechaHoraSalidaNormalizada` sirven para confinar las entradas o salidas respectivamente, en un rango de tiempo entre `horaInicioJornada` y `horaSalidaJornada`.

La tabla `SolicitudTrabajoNoConvencional` almacenará información referente a la convocatoria de un empleado en un día fuera de su jornada de trabajo normal. Contiene información sobre la fecha y la hora en que se solicite al empleado asistir, además del receso que tendrá el empleado.

La tabla `CalendarioFeriado` permitirá categorizar los días de feriado por año, y permitirá a los empleados tener días de descansos. La tabla `DiaFeriado` permitirá describir las fechas, horas y motivos de los feriados.

La tabla `Geovalla` permitirá configurar el área de monitoreo de los empleados, y tiene información con respecto a las coordenadas, radio y tiempo de respuesta del área en cuestión. Esta tabla almacenará solamente un área de monitoreo.

La tabla `NotificacionGeofencing` permitirá almacenar información acerca de la fecha y hora en que un empleado traspase el área de monitoreo. El campo `tipoTransicion` distingue si el traspaso del área en cuestión es de entrada o salida.

En la Figura 2.15 se muestran las tablas `SolicitudPermisoHistorialCambios`, `RegistroAtrasoHistorialCambios`, `RegistroFaltaHistorialCambios`, `RegistroAsistenciaHistorialCambios`, `SolicitudTrabajoNoConvencionalHistorialCambio`, las cuales permitirán guardar respaldos e identificar los responsables de las modificaciones o eliminaciones realizadas a las tablas `SolicitudPermiso`, `RegistroAtraso`, `RegistroFalta`, `RegistroAsistencia` ,`SolicitudTrabajoNoConvencional` respectivamente.

Por último, la tabla `ConfiguracionHorasSistema` permitirá configurar la hora en que los empleados puedan marcar sus entradas, así como el tiempo en que se marque atraso. También el tiempo máximo para que el empleado marque su salida.

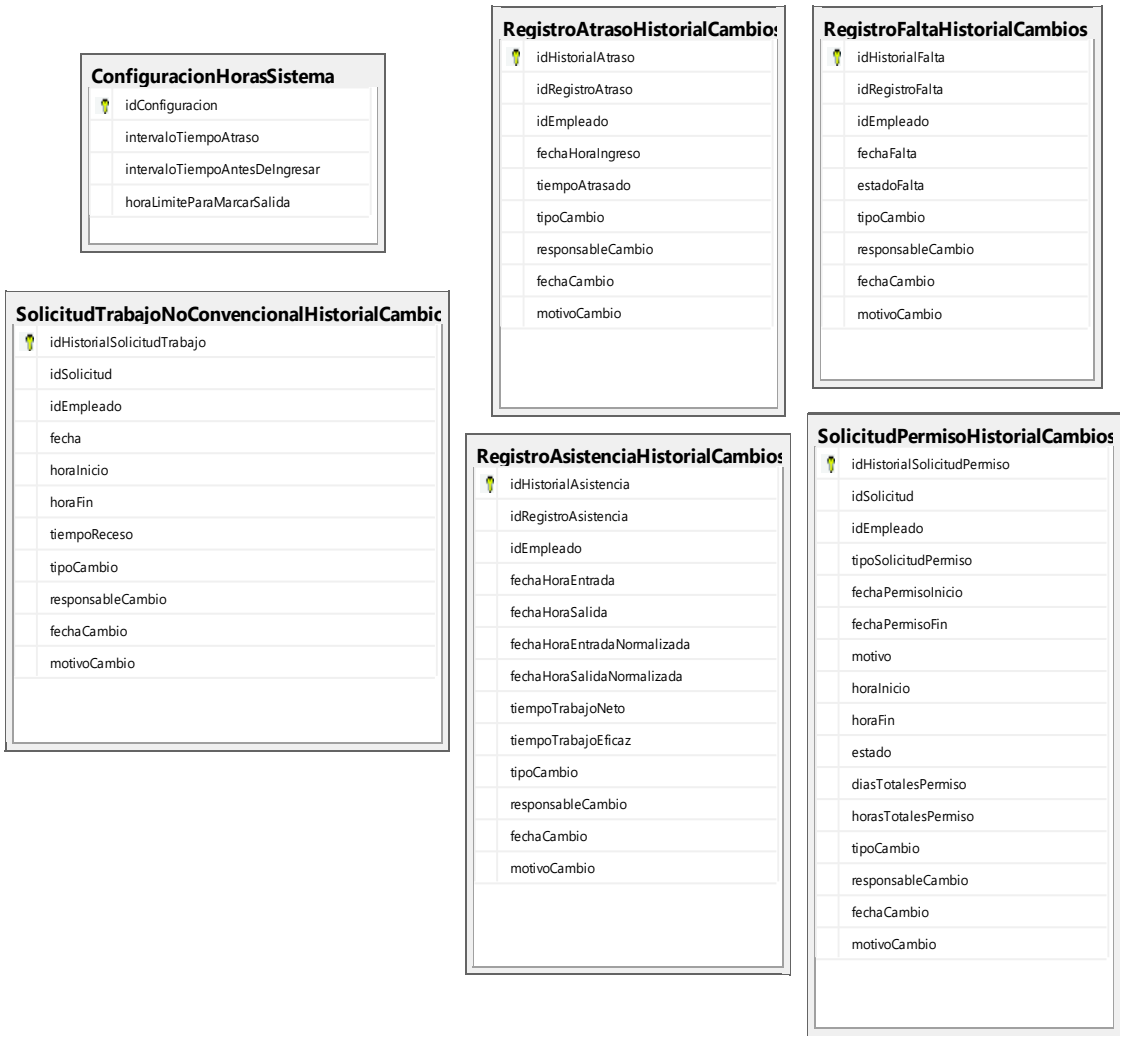


Figura 2.15.- Diagrama Entidad - Relación (Parte 2)

Diagrama de clases del servicio WCF

El servicio WCF permitirá exponer las operaciones descritas en los diagramas de casos de uso, para que el Cliente Android y Cliente Gestor puedan consumir de estas.

En la Figura 2.16 se muestran las clases `HorarioTrabajo`, `Credencial`, `Empleado`, `SolicitudPermiso`, `RegistroEntradaSalida`, `RegistroAtraso`, `RegistroFalta`, `RegistroAsistencia`, `SolicitudTrabajoNoConvencional`, `CalendarioFeriado`, `DiaFeriado`, `ConfiguracionHorasSistema`, `Geovalla`, `NotificacionGeofencing`, `SolicitudPermisoHistorialCambios`, `RegistroAtrasoHistorialCambios`, `RegistroFaltaHistorialCambios`,

`RegistroAsistenciaHistorialCambios`, `SolicitudTrabajoNoConvencionalHistorialCambio`, las cuales servirán para representar los datos almacenados en la base de datos.

La clase `MensajeServicio` estará compuesto por un código y un mensaje, los cuales permitirán ofrecer mayor información sobre el estado de una operación en particular del servicio WCF.

La instancia de la clase `ManejoBBDD` ofrecerá métodos que permitirán ejecutar sentencias SQL con la finalidad de agregar, recuperar, modificar y eliminar información de tablas en la base de datos.

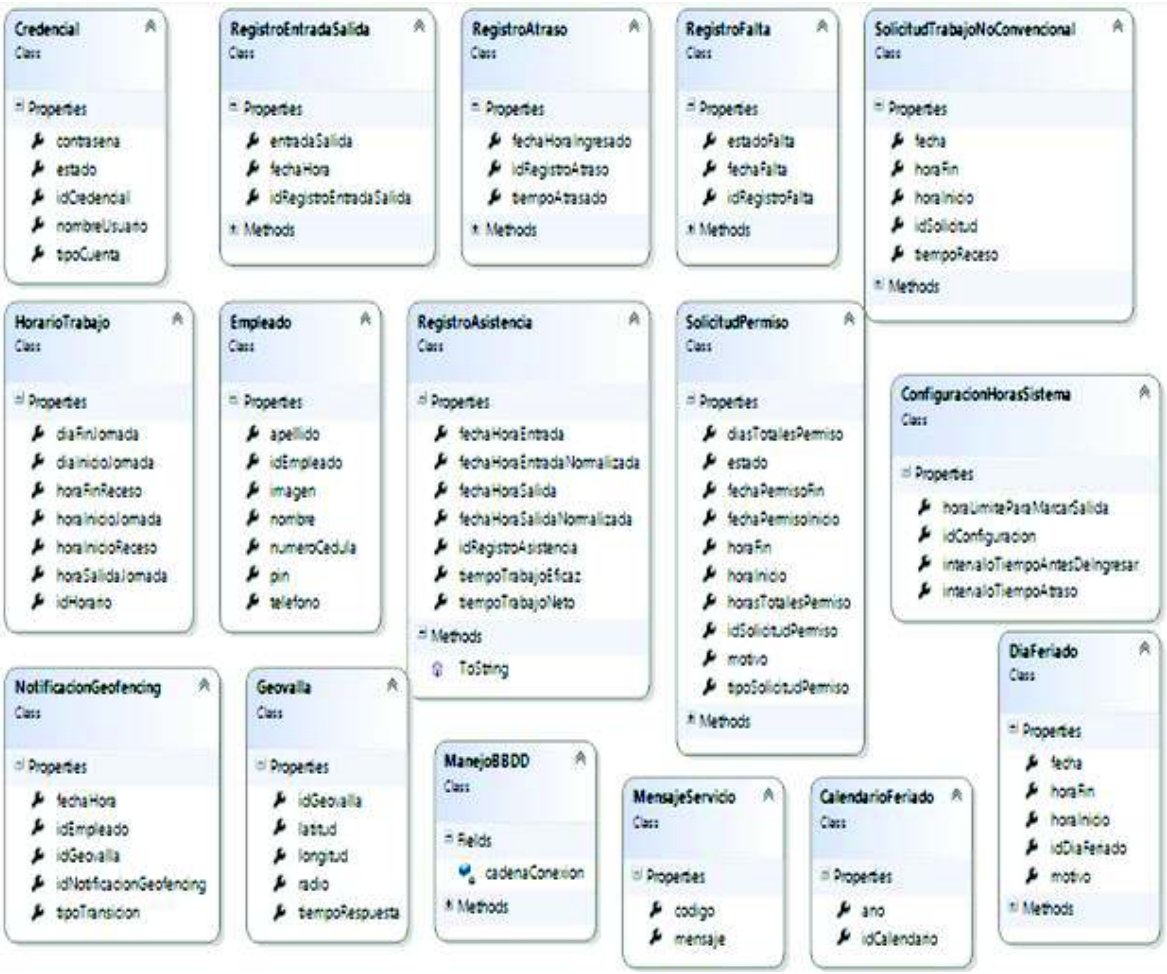


Figura 2.16.- Diagrama de clases del servicio WCF (Parte 1)

El servicio se encuentra compuesto por el contrato de servicio `IServicioControlAsistencia` y la clase que implementa dicho contrato denominado `ServicioControlAsistencia`, como se lo muestra en la Figura 2.17.

El contrato de servicio `IServicioControlAsistencia` define las operaciones soportadas por el servicio con sus respectivas direcciones URI. La clase `ServicioControlAsistencia` permitirá implementar las funcionalidades de las operaciones establecidas por el contrato. Esta clase empleará instancias de las clases `ManejoBBDD` y `ManejoRegistrosControlAsistencia` denominados `miDb` y `controlAsistencia` respectivamente, con la finalidad de llevar a cabo el control de asistencia, así como la gestión de toda información del sistema.

A continuación, se describen algunos métodos de `ServicioControlAsistencia`.

- `Login`: retorna una instancia de `Credencial` con un identificador mayor a cero cuando se ingresen parámetros de credenciales correctos.
- `RegistrarEntrada`: permite registrar la entrada del empleado proporcionado como argumento, además retorna un código y un mensaje que indican si se realizó correctamente o no el registro en cuestión.
- `VerificarRegistroEntradaHoy`: retorna un código y un mensaje que indican si el empleado ingresado como parámetro registró su entrada en el día actual.
- `VerificarRegistroSalidaHoy`: retorna un código y un mensaje que indican si el empleado ingresado como parámetro registró su salida en el día actual.
- `ValidacionesRegistrarSalidaHoy`: retorna un código y un mensaje que indican si el empleado ingresado como parámetro puede o no registrar su salida.
- `RegistrarSalida`: permite registrar la salida del empleado proporcionado como argumento, además retorna un código y un mensaje que indican si se realizó correctamente o no el registro en cuestión.
- `AutenticarPin`: retorna un código y un mensaje que indica si la información de seguridad del empleado ingresado como parámetro coincide o no con uno existente en la base de datos.
- `RegistrarEntradaPin`: permite registrar la entrada del empleado ingresado como parámetro, siempre y cuando la información de seguridad de este sea correcto. Retorna un mensaje y un código que indican si se realizó correctamente el registro en cuestión.

- `RegistrarSalidaPin`: permite registrar la salida del empleado ingresado como parámetro cuando la información de seguridad sea la correcta. Además retorna un código y un mensaje que indican si se realizó correctamente el registro en cuestión.
- `BuscarGeovallaUnica`: retorna información referente al área de monitoreo como latitud, longitud, área, entre otros.
- `RegistrarNotificacionGeofencingEntrada`: permite registrar una transición de entrada del área de monitoreo proveniente del empleado ingresado como parámetro. Retorna un código y mensaje indicando el éxito del registro en cuestión.
- `RegistrarNotificacionGeofencingSalida`: permite registrar una transición de salida del área de monitoreo proveniente del empleado ingresado como parámetro. Retorna un código y mensaje indicando el éxito del registro en cuestión.
- `RegistrarSalidaAutomatica`: permite registrar una notificación con transición de salida del área de monitoreo del empleado ingresado como parámetro, además permite registrar su salida. Retorna un código y un mensaje indicando el éxito del registro en cuestión.
- Los métodos `BuscarRegistrosAsistencia`, `BuscarRegistrosAtraso`, `BuscarRegistrosFaltas` reciben como argumentos la información del empleado y un intervalo de fechas, y retornan una lista de registros de asistencias, registros de atrasos y registros de faltas respectivamente, los cuales coincidan con los parámetros ingresados.
- `BuscarHorasTotalTrabajadas`: recibe como argumentos un empleado y un intervalo de fechas y retorna el número total de horas trabajadas por el empleado durante el periodo ingresado.
- `BuscarDiasFestivos`: recibe como parámetro un número que denota el año y retorna una lista de días de descanso para el año especificado.
- `RegistrarSolicitudPermiso`: recibe como parámetros la información de un empleado y una solicitud de permiso y los guarda en la base de datos. Además, retorna un mensaje y un código que indica el éxito del proceso en cuestión.

En la Figura 2.18 se muestra la clase `ManejoRegistrosControlAsistencia`, la cual posee una instancia de `ManejoBBDD` denominado `miDb` que permitirá manejar

información de tablas en la base de datos. Además, esta clase tendrá métodos que facilitarán la lógica del control de asistencia, los cuales se describirán a continuación.

- `VerificarHorasJornadasFeriadosPermisoTodoEntrada`: retorna un número entero que indica si el empleado especificado como parámetro ingresó puntual, atrasado, en días de feriado, en días de permiso o si el empleado ingresó en un día fuera de su jornada de trabajo.
- `VerificarTrabajarHoy`: retorna un valor booleano que indica si el empleado especificado como parámetro debe trabajar para la fecha actual.
- `VerificarTrabajarDiaEspecificado`: retorna un valor booleano que indica si el empleado especificado como parámetro debe trabajar para una fecha específica.
- `TrabajarHoy`: retorna un valor booleano que indica si el día de hoy se encuentra entre un intervalo de dos días ingresados como parámetros. Se tiene un método sobrecargado de este, que realiza la misma funcionalidad, sin embargo, para una fecha específica.
- `ConvertirDia`: permite convertir un día de la semana en formato de cadena de caracteres a tipo `DayOfWeek`.
- `VerificarRangoHoraIngreso`: retorna un número entero que permite verificar si la hora actual se encuentra cerca o dentro de la hora de inicio de jornada de un empleado en particular. Se tiene un método sobrecargado que realiza lo mismo, pero para una hora específica.
- `VerificarSiEstoyDentroPermisosAprobadoHoy`: retorna un valor booleano que indica si el empleado ingresado como parámetro se encuentra dentro del periodo de permisos aprobados.
- `VerificarPermisoAprobadoHoyTodoDia`: retorna un valor booleano que indica si el empleado ingresado como parámetro posee un permiso aprobado para la fecha actual y que dure todo el día.
- `VerificarEmpleadoDentroFeriado`: retorna un valor booleano que indica si en la fecha y hora actual, el empleado ingresado como parámetro se encuentra dentro de horas de feriado. Se tiene otro método del mismo tipo, pero sobrecargado con una fecha determinada.

Figura 2.17.- Diagrama de clases servicio WCF (Parte 2)

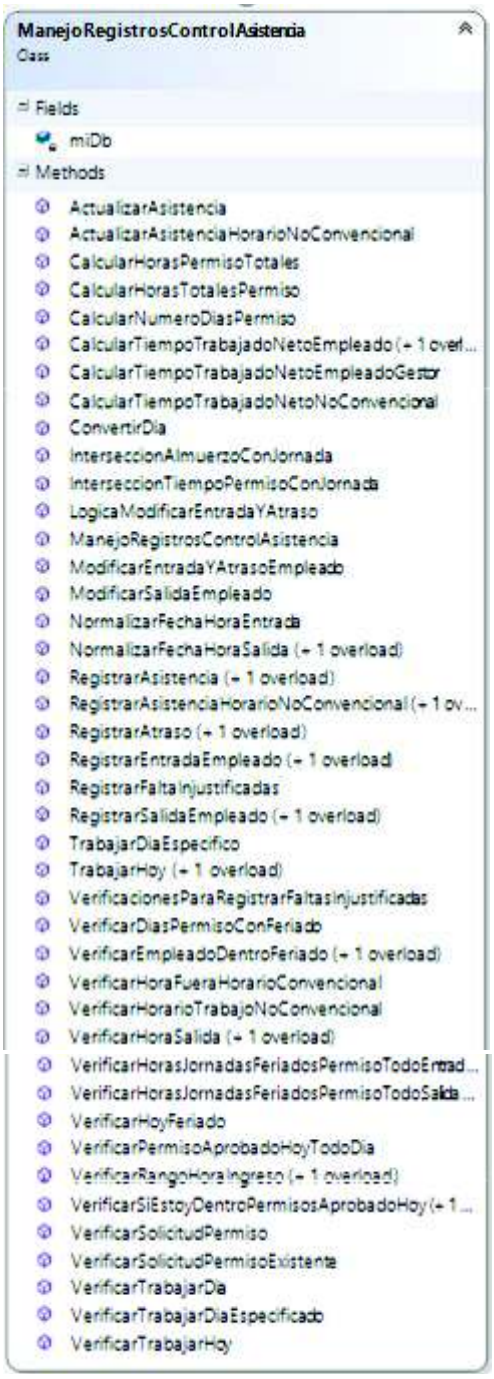


Figura 2.18.- Diagrama de clases del servicio WCF (Parte 3)

- `RegistrarEntradaEmpleado`: retorna un entero que indica si se pudo o no registrar una entrada del empleado ingresado como parámetro.
- `ModificarEntradaYAtrasoEmpleado`: retorna un entero que indica si se pudo o no actualizar un registro de entrada o un registro de atraso, del empleado ingresado como parámetro.

- `LogicaModificarEntradaYAtraso`: permite manejar la lógica asociada a la modificación, inserción, eliminación de registros de entradas y atrasos.
- `RegistrarAtraso`: retorna un entero que indica si se pudo registrar un atraso del empleado ingresado como parámetro.
- `VerificarHorasJornadasFeriadosPermisoTodoSalida`: retorna un entero que indica si el empleado ingresado como parámetro puede o no registrar su salida.
- `VerificarHoraSalida`: retorna un entero que indica si el empleado ingresado como parámetro registró su salida dentro del horario permitido.
- `RegistrarSalidaEmpleado`: retorna un entero que indica si se pudo registrar una salida del empleado ingresado como parámetro.
- `RegistrarAsistencia`: retorna un entero que indica si se pudo registrar una asistencia del empleado ingresado como parámetro.
- `ActualizarAsistencia`: permite actualizar un registro de asistencia del empleado ingresado como parámetro.
- `NormalizarFechaHoraEntrada`: recibe como parámetros una fecha y hora de entrada y una hora de inicio de jornada, además, retorna una fecha y hora ajustada el cual evita que existan horas antes de la entrada.
- `NormalizarFechaHoraSalida`: recibe como parámetros una fecha y hora de salida y una hora de fin de jornada, adicionalmente, retorna una fecha y hora ajustada el cual evita que existan horas después de la salida.
- `RegistrarAsistenciaHorarioNoConvencional`: retorna un entero que indica si el empleado ingresado como parámetro pudo registrar su asistencia en un día fuera de su jornada convencional.
- `CalcularTiempoTrabajadoNetoNoConvencional`: recibe como parámetros un empleado, fechas horas de entrada y salida, información de solicitud de trabajo y retorna un intervalo de tiempo que indica el tiempo total trabajado por el empleado para el día de trabajo no convencional.
- `CalcularHorasTotalesPermiso`: recibe como parámetros un empleado y su correspondiente horario y permite realizar el cálculo correspondiente a la duración total de un permiso.

- `InterseccionAlmuerzoConJornada`: retorna un entero que indica si un empleado salió antes o después de su hora de receso. Permite evitar la resta de horas indebidas.
- `InterseccionTiempoPermisoConJornada`: retorna un entero que indica si un empleado salió antes del inicio de un determinado permiso. Permite evitar la resta de horas indebidas.
- `CalcularTiempoTrabajadoNetoEmpleado`: recibe como parámetros un empleado, fecha hora de entrada, fecha hora de salida y el horario del empleado en cuestión. Retorna el tiempo total de trabajo del empleado.
- `RegistrarFaltaInjustificadas`: retorna un entero que indica si se pudo registrar una falta injustificada del empleado ingresado como parámetro.
- `VerificacionesParaRegistrarFaltasInjustificadas`: retorna un entero que indica si se debe o no registrar una falta injustificada al empleado ingresado como parámetro.
- `VerificarHoyFeriado`: retorna un valor booleano que indica si en la fecha y hora actual se tiene feriado.
- `VerificarSolicitudPermiso`: retorna un entero que indica si las fechas de la solicitud ingresada como parámetro son válidas.
- `CalcularNumeroDiasPermiso`: recibe como parámetros una solicitud de permiso y un empleado, además retorna el número de días totales que dure la solicitud.
- `VerificarHorarioTrabajoNoConvencional`: retorna un entero que indica si una fecha y hora ingresada como parámetro es válida para una solicitud de trabajo no convencional.

Diagrama de clases cliente Android

A continuación se presentan descripciones del Cliente Android representadas por diagramas de clases. En la Figura 2.19 se muestran las clases empleadas para representar las respuestas de peticiones HTTP. La clase `Constantes` permitirá establecer un conjunto de variables tipo `string` que almacenarán las direcciones URI de operaciones del servicio WCF.

Todas las actividades se derivan de `AppCompatActivity`[58] por lo que implementan el método `onCreate` con la finalidad de inicializar los componentes y variables necesarios para el funcionamiento de la actividad. Además, se tiene el método `onSupportNavigateUp` que habilita el botón de regreso y permitirá al usuario retornar a la actividad previa. Cada actividad empleará métodos de la instancia `ManejadorPeticiones`, para consumir las operaciones establecidas por el servicio web. También manejarán objetos `Context`[59] con la finalidad de tener acceso a recursos y clases específicas de la aplicación y emplear llamadas a métodos de nivel aplicación como por ejemplo, llamar a otra actividad.

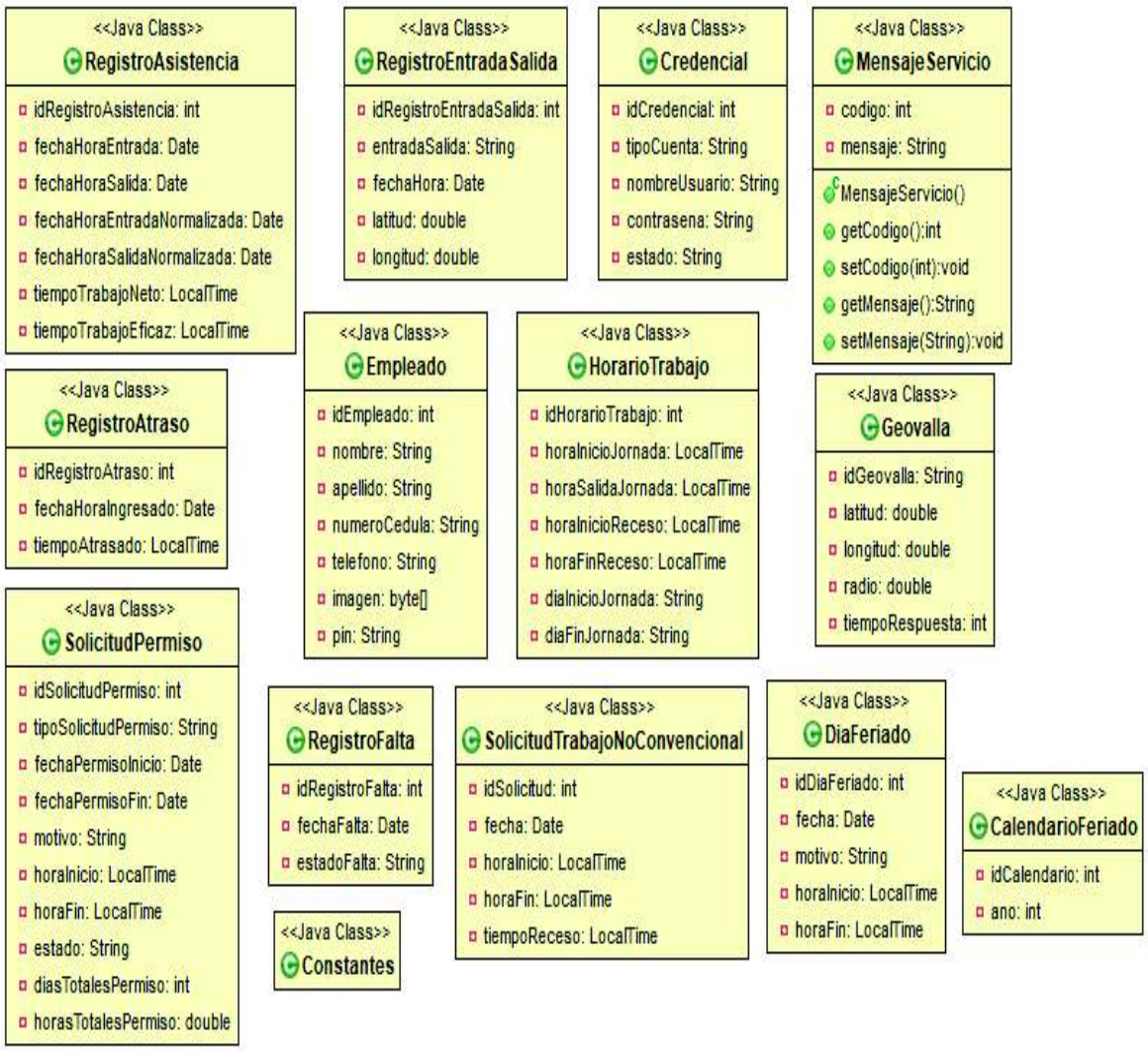


Figura 2.19.- Diagrama de clases Cliente Android (Parte 1)

[58] `AppCompatActivity`: es una clase base que heredan las actividades con la finalidad de poder emplear características de barras de herramientas

[59] `Context`: es una clase abstracta que permite tener acceso a recursos y clases de la aplicación.

En la Figura 2.20 Figura 2.20, la Figura 2.21, la Figura 2.22 y la Figura 2.23 se presentan las actividades que forman parte del Cliente Android, las cuales emplearán métodos y atributos necesarios para llevar a cabo los procesos referentes al registro de asistencia, consultas varias y emisión de solicitudes de permiso. A continuación se describen estas actividades.

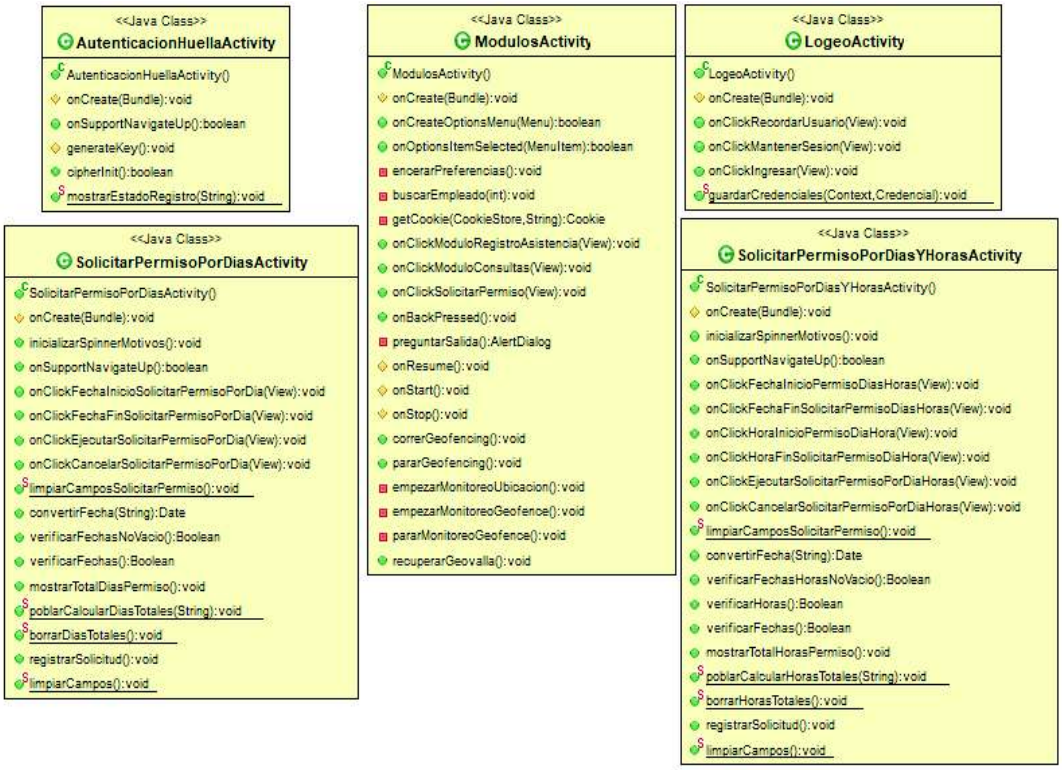


Figura 2.20.- Diagrama de clases Cliente Android (Parte 2)

- `LogeoActivity`: es la actividad raíz del Cliente Android. Aquí el usuario podrá ingresar al sistema con sus credenciales. Poseerá los métodos necesarios para recordar usuario y mantener sesión activa.
- `ModulosActivity`: permitirá al usuario acceder a los módulos de registro de asistencia, de consultas, de solicitar permiso y al menú de configuraciones de usuario. Aquí es donde se emplearán los métodos `correrGeofencing`, `pararGeofencing`, `empezarMonitoreoUbicacion`, `pararMonitoreoGeofence`, `recuperarGeovalla` para empezar, parar y buscar el área de monitoreo. Además posee los métodos `onResume`, `onStart`, `onStop`, los cuales permitirán controlar la reconexión con el cliente de Google, este último hará posible el monitoreo en cuestión.

- `ModuloRegistroAsistenciaActivity`: actividad que permitirá al usuario registrar entrada/salida. Además, se tendrán los métodos `verificarDisponibilidadBotonEntrada`, `verificarDisponibilidadBotonSalida`, `deshabilitarRegistroEntrada`, `habilitarRegistroEntrada`, `deshabilitarRegistroSalida`, `habilitarRegistroSalida` los que controlarán duplicidad de registros y se mostrarán mensajes de ayuda para los usuarios mediante los métodos `estado0BotonSalida`, `estado1BotonSalida`, `estado2BotonSalida`, `estado3BotonSalida`. A través del método `mostrarFechaActualServicio` permitirá mostrar la fecha actual del servicio.

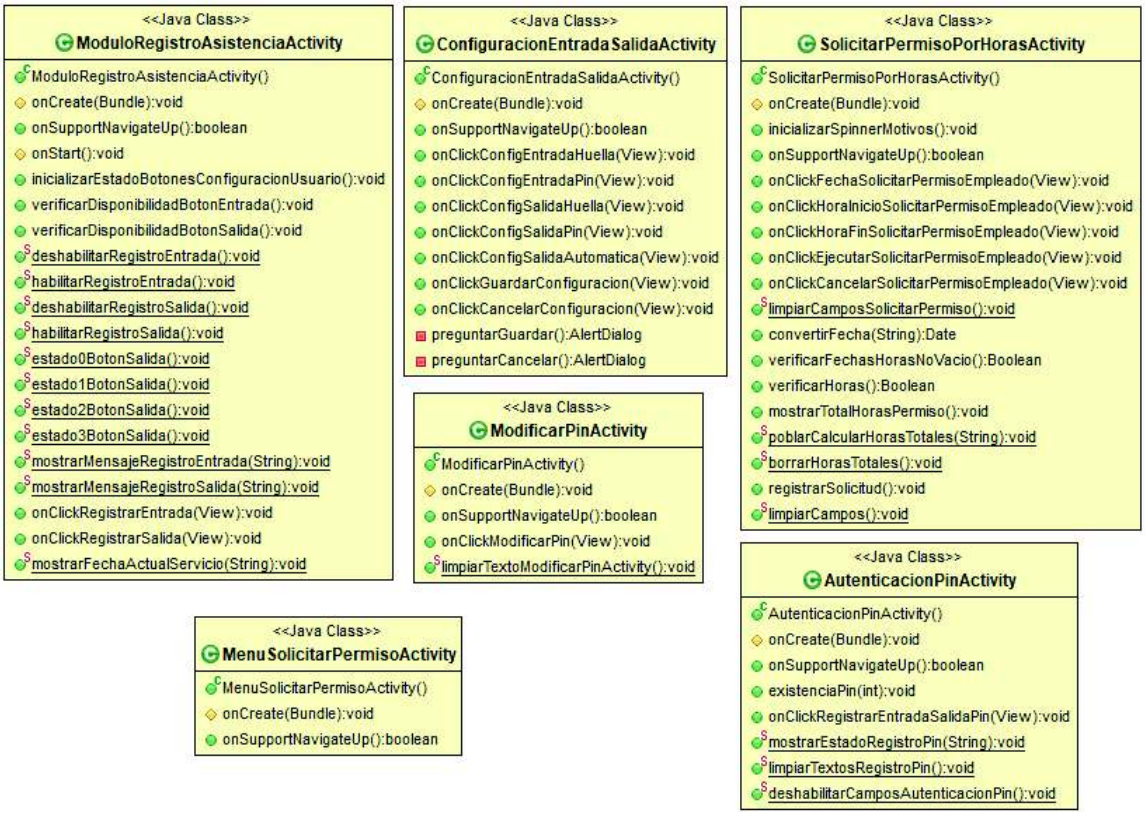


Figura 2.21.- Diagrama de clases Cliente Android (Parte 3)

- `AutenticacionHuellaActivity`: actividad que realizará la autenticación de huellas dactilares para registrar la entrada/salida del usuario. Los métodos `generateKey` y `cipherInit` permitirán configurar los mecanismos de seguridad para la autenticación de huellas. `mostrarEstadoRegistro` mostrará un mensaje de éxito o de fallo en el proceso de autenticación por huellas.

- `AutenticacionPinActivity`: actividad en donde se realizará la autenticación por medio del PIN para registrar la entrada/salida del usuario.
- `ModificarPinActivity`: actividad que permitirá al usuario cambiar su código de seguridad. El método `existenciaPin` permitirá verificar si el usuario tiene un PIN.
- `MenuSolicitarPermisoActivity`: actividad que permitirá al usuario elegir el tipo de solicitud que desea emitir.
- `SolicitarPermisoPorHorasActivity`: actividad que permitirá al usuario solicitar permiso por un periodo menor a un día. Se tendrán escuchadores de eventos de pulsación del ratón en los botones de la interfaz gráfica para desplegar `DatePickerDialog` y `TimePickerDialog` con la finalidad de que el usuario pueda especificar fechas y horas de la solicitud. `convertirFecha` permitirá transformar una fecha en formato `string` a un objeto `Date`. Además, proporcionará los métodos necesarios para verificar campos vacíos y para mostrar/borrar información del permiso en cuestión.
- `SolicitarPermisoPorDiasActivity`: actividad que permitirá al usuario solicitar permiso por un intervalo de fechas. Se tendrán las mismas descripciones de métodos antes mencionadas en la actividad para solicitar horas.
- `SolicitarPermisoPorDiasYHorasActivity`: actividad que permite al usuario solicitar permiso por un intervalo de fechas y horas. Se tendrán las mismas descripciones de métodos antes mencionadas en la actividad para solicitar horas.
- `ConfiguracionEntradaSalidaActivity`: actividad que permite al usuario escoger la manera en que desea registrar su entrada/salida, ya sea por medio de huella dactilar, PIN o de forma automáticamente, este último medio solamente se encuentra disponible para el registro de salida. Se desplegarán diálogos para que el usuario confirme los cambios realizados.
- `ConsultasIndiceActivity`: actividad que permitirá mostrarle al usuario una lista de opciones de consultas y permitirá redirigirle a la actividad correspondiente según su elección.
- Las actividades `ConsultaHorarioActivity`, `ConsultaEntradaSalidaActivity`, `ConsultaAtrasoActivity`, `ConsultaAsistenciasActivity`, `ConsultaTotalHorasActivity`,

`ConsultaPermisosActivity`, `ConsultaFeriadosActivity`, `ConsultaFaltasActivity` y `ConsultaSolicitudTrabajoNoConvencionalActivity` permitirán al usuario consultar, como los métodos indican, el horario, los registros de entrada/salida, los atrasos, las asistencias, el total de horas trabajadas, los permisos, los días de feriado, las faltas y las convocatorias de trabajo no convencional respectivamente. Estas actividades posibilitarán al usuario filtrar sus búsquedas de acuerdo a un intervalo de fechas. Además, tendrán los métodos requeridos para mostrar los resultados en una lista visual denominada `ListView`[60] empleando sus correspondientes `ArrayAdapter`[61].

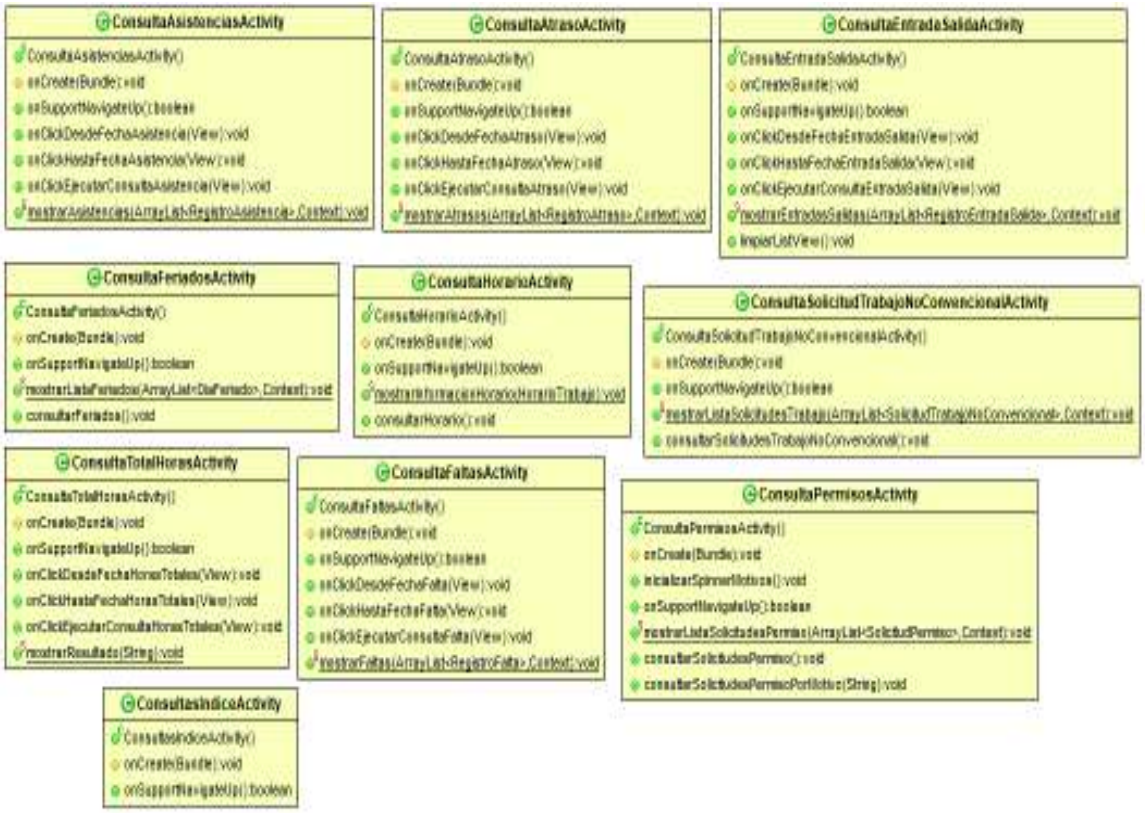


Figura 2.22.- Diagrama de clases Cliente Android (Parte 4)

- Las actividades `InfoRegistroAsistenciaActivity`, `InfoDiaFeriadoActivity`, `InfoRegistroAtrasoActivity`, `InfoRegistroFaltaActivity`, `InfoSolicitudPermisoActivity` y `InfoSolicitudTrabajoNoConvencionalActivity` permitirá mostrar

[60] `ListView`: es un componente de la interfaz gráfica que despliega resultados en forma de una lista
[61] `ArrayAdapter`: permite enlazar arreglos de objetos a una lista visual

información detallada sobre un registro de asistencia, un día de feriado, un registro de atraso, un registro de falta, una solicitud de permiso o de una solicitud de trabajo respectivamente. Tendrán métodos respectivos para mostrar sus detalles en componentes visuales `TextView`[62].

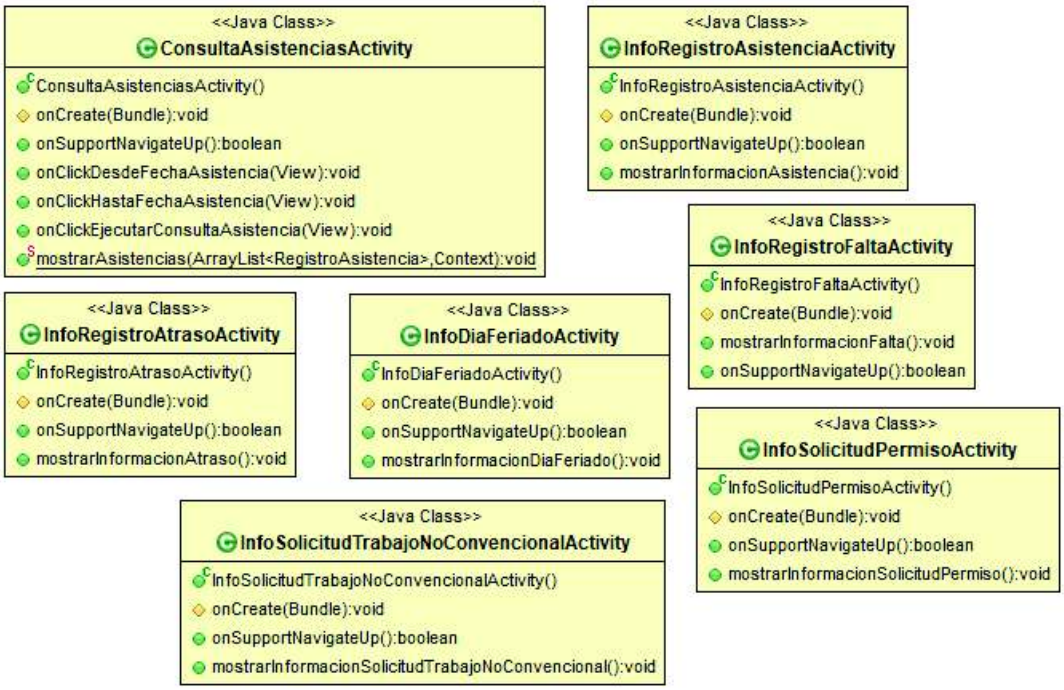


Figura 2.23.- Diagrama de clases Cliente Android (Parte 5)

En la Figura 2.24 se presentan las siguientes clases:

- `ManejoDeHuellas`: clase que implementa la interfaz `AuthenticationCallback`, por lo tanto, especifica los métodos `onAuthenticationError`, `onAuthenticationHelp`, `onAuthenticationFailed` y `onAuthenticationSucceded`, los cuales permitirán definir la lógica del manejo de error, de ayuda, de fallo o de éxito del proceso de autenticación de huellas dactilares respectivamente. El método `empezarAutenticacionHuella` realizará la autenticación en cuestión.
- `ServicioGeofence`: clase que extiende de `IntentService`, por lo tanto, implementa el método `onHandleIntent`, que permitirá ejecutar el servicio de monitoreo de geovallas en segundo plano. Aquí es donde se establecerá la lógica del manejo de transiciones del área de monitoreo.

[62] `TextView`: es un componente de la interfaz gráfica de una actividad que permite mostrar texto

- `ManejoConfiguracionUsuario`: clase que permitirá guardar información del usuario a través de un par clave/valor en `SharedPreferences`[63]. Los métodos `setString`, `obtenerString`, `setInt`, `obtenerInt`, `setBool`, `obtenerBool`, permitirán guardar/recuperar información de tipo cadena de caracteres, entero o booleano respectivamente.
- `ManejoNotificaciones`: clase que permitirá configurar y mostrar notificaciones que aparecerán en el dispositivo móvil.

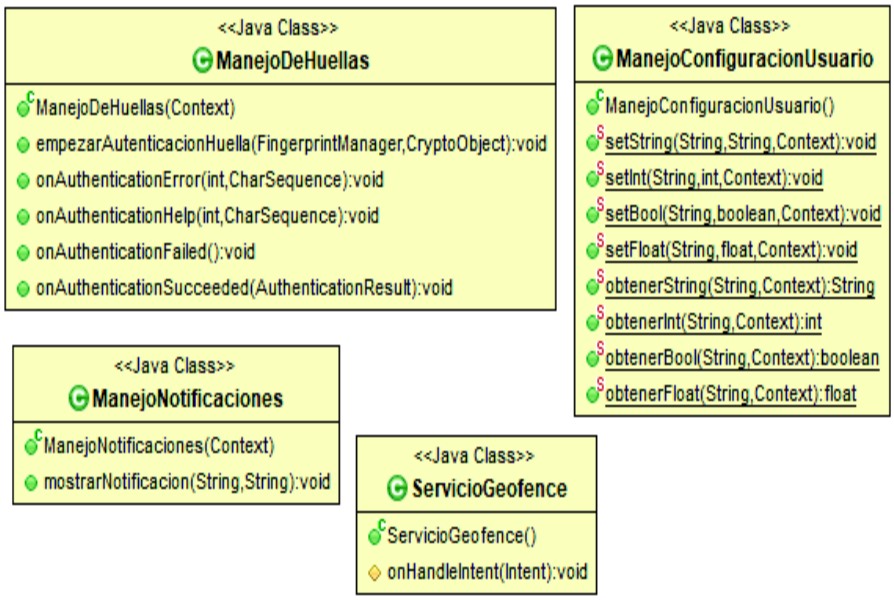


Figura 2.24.- Diagrama de clases Cliente Android (Parte 6)

En la Figura 2.25 se muestra la clase `ManejadorPeticiones`, la cual permite al Cliente Android consumir operaciones establecidas por el servicio WCF. Emplea una instancia de la clase `RequestQueue` para enviar peticiones HTTP, además, emplea una instancia `AbstractHttpClient` para inicializar la cola de peticiones.

Cada uno de los métodos de `ManejadorPeticiones`, implementarán los métodos `onResponse` y `onErrorResponse` para especificar la lógica de lo que ocurrirá tras obtener una respuesta satisfactoria y de error respectivamente. Además se empleará una instancia `ProgressDialog` para desplegar un círculo giratorio para indicarle al usuario que espere. Adicionalmente, se realizarán los procesos respectivos de serialización/deserialización de objetos JSON.

[63] `SharedPreferences`: es un archivo en formato XML que permite guardar información en campos clave/valor

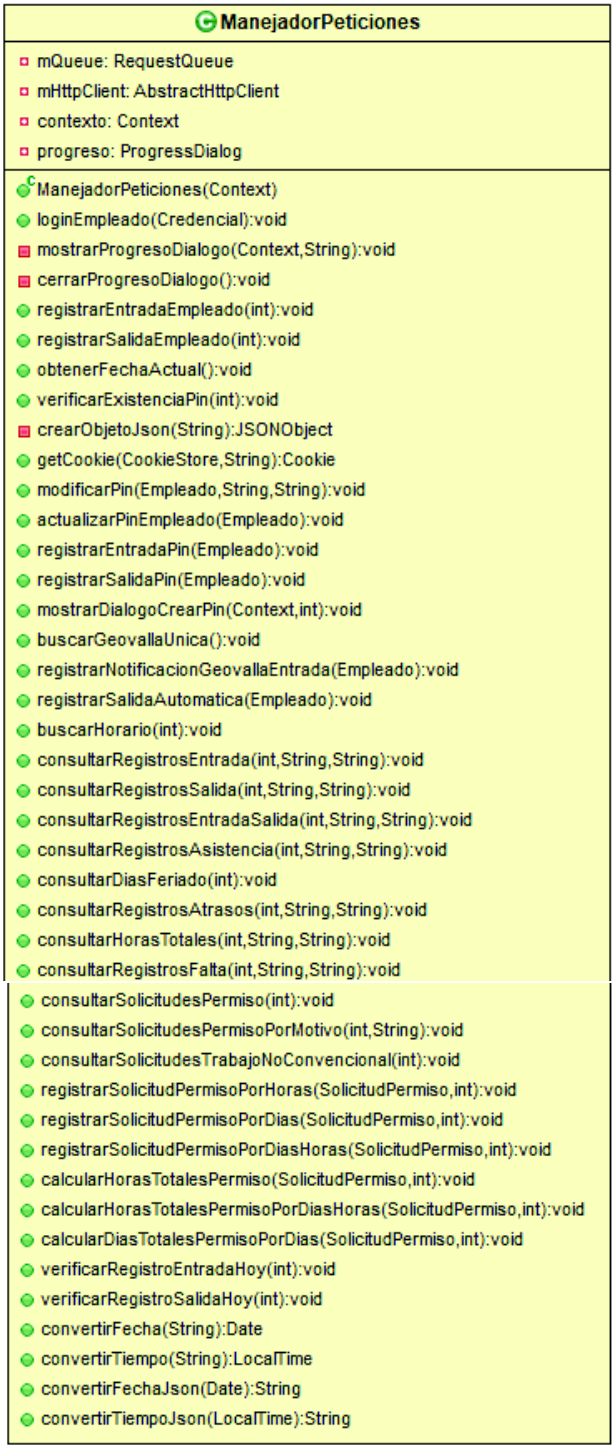


Figura 2.25.- Diagrama de clases Cliente Android (Parte 7)

El método `getCookie` permite buscar y recuperar cookies. Los métodos `convertirFechaJson`, `convertirFecha`, `convertirTiempoJson`, `convertirTiempo` permitirán serializar/deserializar los tipos de datos fechas e intervalos de tiempo a formato JSON y viceversa, ya que la librería GSON empleada no soporta conversión de estos tipos de datos.

Diagrama de clases cliente Gestor

En la Figura 2.26 se muestran clases del Cliente Gestor los cuales permitirán representar las respuestas obtenidas por peticiones HTTP. Cada uno de estas tendrá su correspondiente constructor para inicializar sus atributos respectivos. Además, se especificará el método `toString` para establecer la información que se desee mostrar.

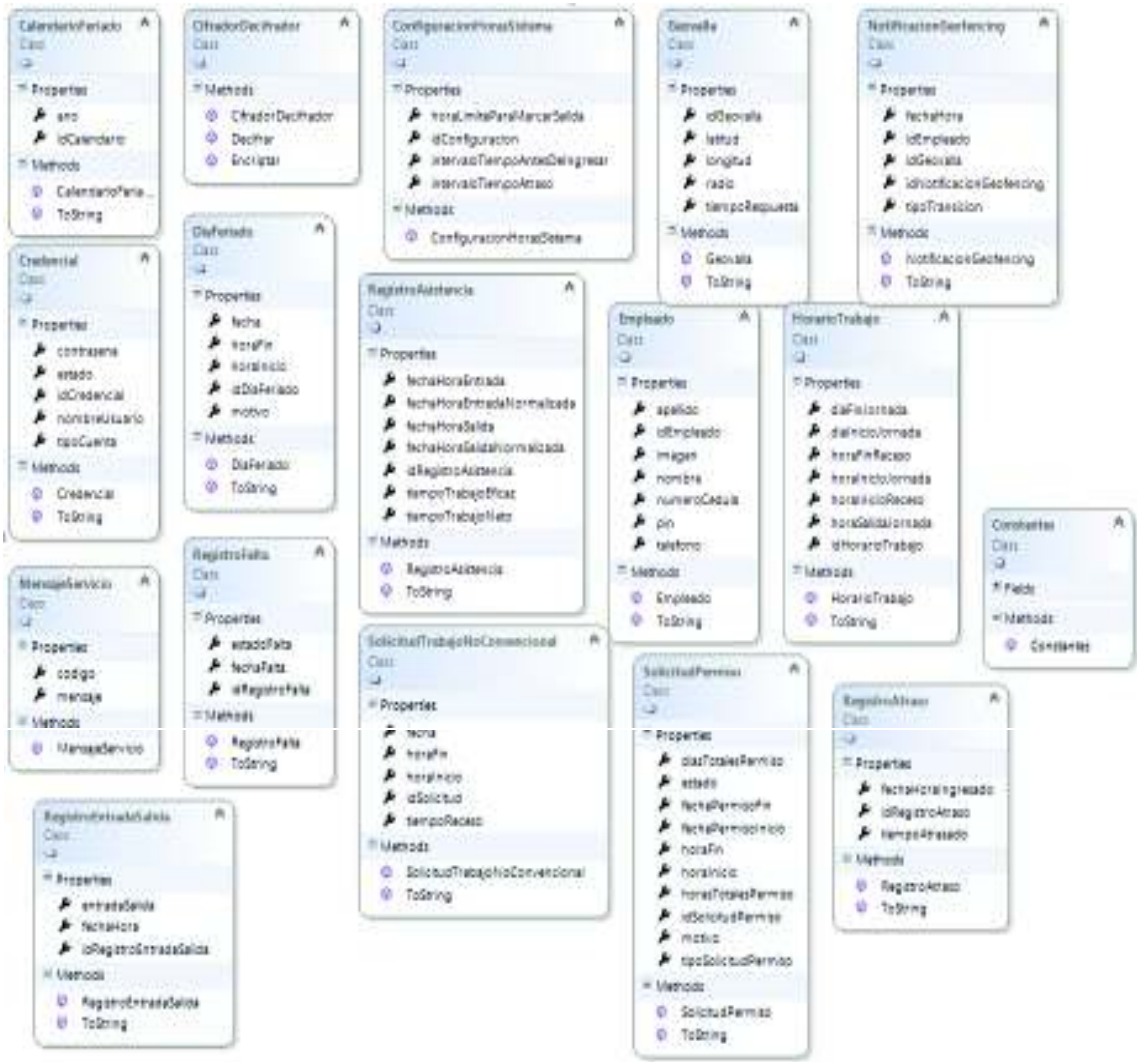


Figura 2.26.- Diagrama de clases del Cliente Gestor (Parte 1)

La clase `Constantes` almacenará las direcciones URI de operaciones del servicio web.

La clase `CookiedRequestFactory`[64] permitirá crear peticiones con soporte a *cookies* para mantener estado de sesión. Para consumir el servicio WCF y acceder al conjunto de operaciones de gestión del sistema, se emplearán los métodos de la clase `ServicioControlAsistenciaProxy`, los cuales se encargarán de enviar/recibir

[64] `CookiedRequestFactory`: clase que permite a una aplicación de escritorio manejar *cookies*, se basa en el código que se encuentra en [10]

peticiones/respuestas, serializar/deserializar objetos JSON. Esta clase hace uso de los métodos de la clase `CookiedRequestFactory` para el manejo de *cookies*.

Todos los formularios que requieran obtener respuestas del servicio, utilizará la clase `ServicioControlAsistenciaProxy` (ver Figura 2.27).

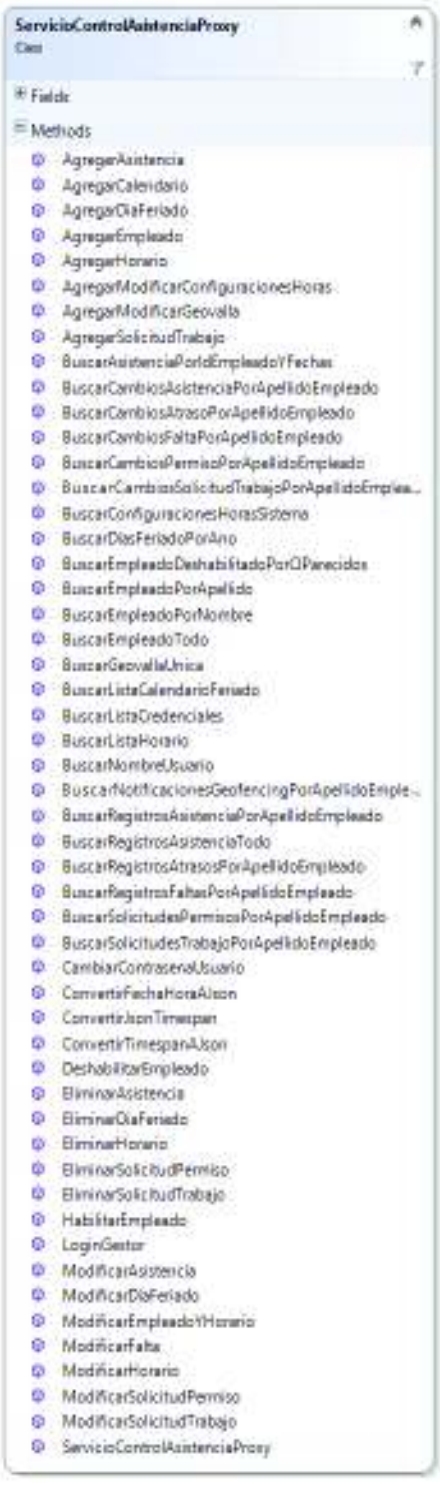


Figura 2.27.- Diagrama de clases del Cliente Gestor (Parte 2)

Sketches

Los bocetos o *sketches* permiten tener una visión general de los componentes visuales que tendrá una aplicación.

A continuación, en la Figura 2.28, la Figura 2.29 y la Figura 2.30 se presentan bocetos de las interfaces gráficas de las actividades `LogeoActivity`, `ModulosActivity` y `ConfiguracionEntradaSalidaActivity` del Cliente Android.

Figura 2.28.- *Sketch* de la actividad `LogeoActivity`

Figura 2.29.- *Sketch* de la actividad `ModulosActivity`

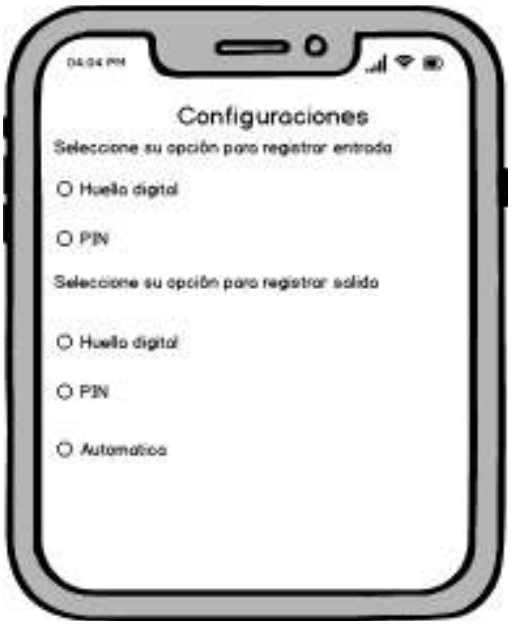


Figura 2.30.- *Sketch* de la actividad `ConfiguracionEntradaSalidaActivity`

Wireframes

Los *Wireframes* permiten establecer la funcionalidad y la navegabilidad de los diferentes componentes gráficos. En la Figura 2.31, se muestra el *wireframe* de la actividad `LogeoActivity`, el cual permitirá al usuario ingresar al sistema, y acceder al módulo de registro de asistencia, al módulo de consultas, o al módulo para solicitar permiso.

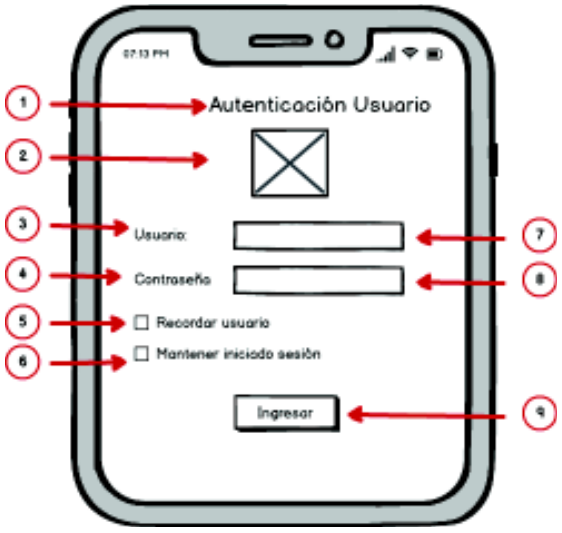


Figura 2.31.- *Wireframe* de la actividad `LogeoActivity`

En la Tabla 2.7 se muestra la descripción de los elementos del *wireframe* de la actividad `LogeoActivity`. En el Anexo III se recopila todos los *sketches* y *wireframes* con sus correspondientes descripciones del Cliente Android y del Cliente Gestor.

Tabla 2.7.- Descripción de la actividad `LogeoActivity`

Identificador	Descripción	Control
1	Permite mostrar el título de la actividad	`TextView`
2	Permite mostrar un ícono de usuario	`ImageView`
3	Permite desplegar texto	`TextView`
4	Permite desplegar texto	`TextView`
5	Permite conocer si el usuario desea recordar el nombre de usuario	`Checkbox`
6	Permite conocer si el usuario desea mantener iniciado sesión	`Checkbox`
7	Permite al usuario ingresar su nombre de usuario	`EditText`
8	Permite al usuario ingresar su contraseña	`EditText`
9	Permite al usuario autenticarse y acceder a la actividad `ModulosActivity`	`Button`

En la Figura 2.32 se presentan los pasos para llegar a la actividad `ModulosActivity`. Al ingresar a la aplicación, el usuario ingresa sus credenciales y este presiona sobre el botón “Ingresar”. Si los credenciales son correctos, se llama a la actividad `ModulosActivity`, para que el usuario pueda acceder a los diferentes módulos de la aplicación.

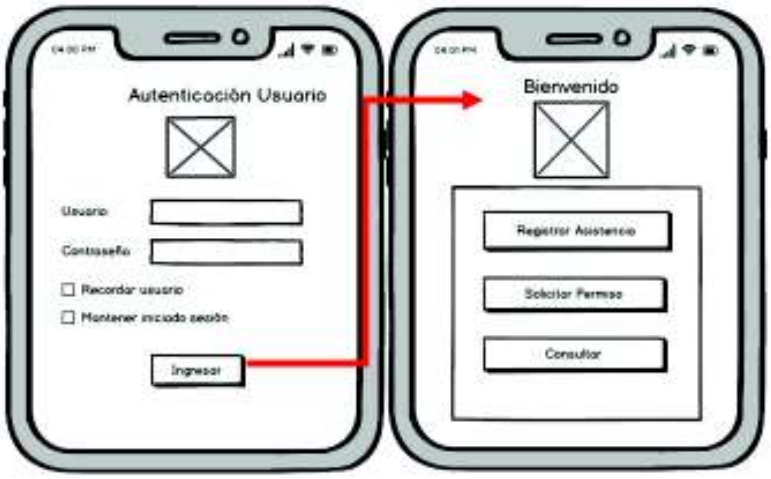


Figura 2.32.- Navegación hacia la actividad `ModulosActivity`

En la Figura 2.33 se muestran los pasos que el usuario debe seguir para registrar su entrada a través de huella digital (considerando que el usuario seleccionó dicha opción). Dentro de la actividad `ModulosActivity`, se presiona sobre el botón “Registrar Asistencia” y se llega hasta la actividad `ModuloRegistroAsistenciaActivity`, aquí se presiona sobre el botón “Registrar Ingreso” y aparece la actividad

`AutenticacionHuellaActivity`, que autenticará las huellas digitales del usuario con la finalidad de realizar el proceso en cuestión.

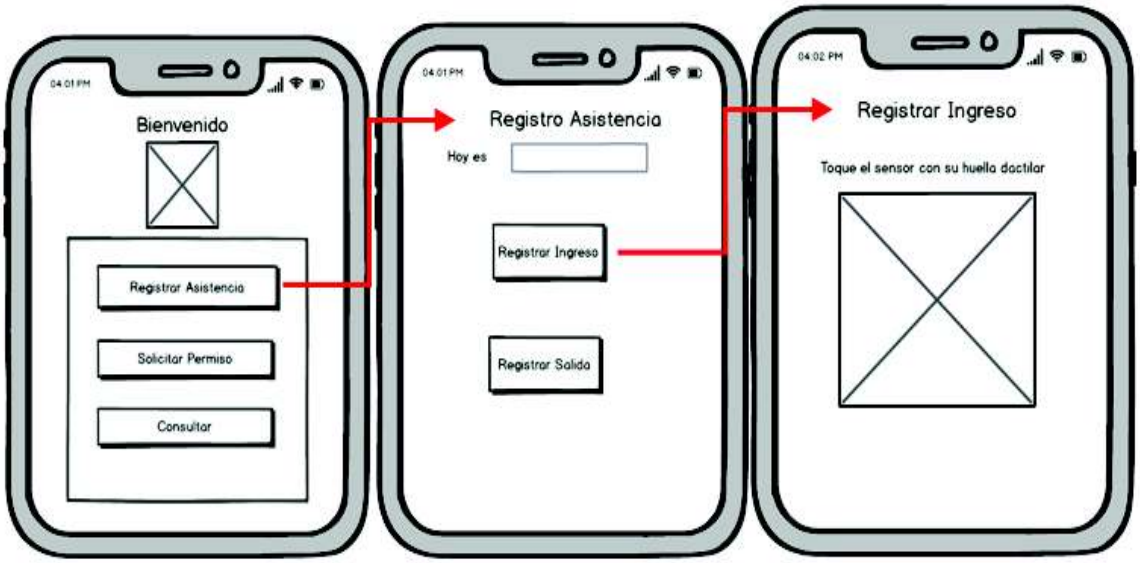


Figura 2.33.- Navegación hacia la actividad `AutenticacionHuellaActivity`

Diagrama de actividades

Los diagramas de actividades permiten describir procesos que intervienen en los casos de uso.

A continuación se presentan diagramas de actividades para la autenticación de usuario, registro de entrada y consulta de asistencias para el usuario del Cliente Android.

Autenticación de usuario

El prototipo de Cliente Android permitirá al usuario ingresar sus credenciales para poder acceder al mismo.

La Figura 2.34 describe el proceso de autenticación del usuario y es el siguiente: cuando el usuario desee mantener su sesión activa y haya ingresado al sistema alguna vez, este podrá acceder al sistema sin que el usuario intervenga, enviando automáticamente una petición al servicio web para autenticar al usuario con credenciales previamente guardados.

Por otro lado, cuando el usuario no desee mantener activa su sesión, o si es la primera vez que intenta ingresar al sistema, este deberá ingresar su nombre de usuario y contraseña. Después se enviará una petición al servicio de autenticación con los credenciales del

usuario y se recibirá la respuesta. Si la respuesta es correcta, el usuario podrá acceder al prototipo, caso contrario, este deberá ingresar nuevamente sus credenciales.

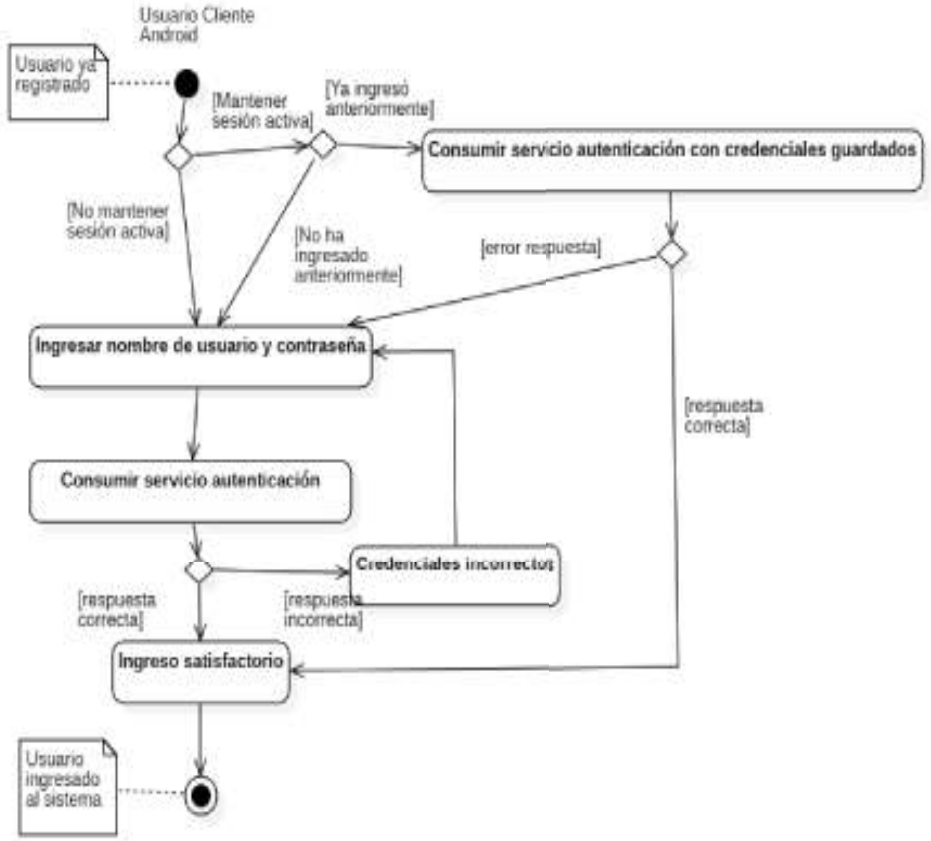


Figura 2.34.- Diagrama de actividades para autenticar usuario

Registrar entrada

Una vez que un usuario del Cliente Android ha ingresado al prototipo, podrá acceder al mismo para registrar su entrada. El prototipo contará con un apartado de configuraciones, que permitirá al usuario elegir el método de registro de entrada, ya sea por huella dactilar o por medio del PIN, como se muestra en la Figura 2.35, cuando el usuario presione el respectivo botón para registrar entrada, se verificará la opción que eligió el usuario.

En caso de que la opción escogida fue registro por medio de huella, se mostrará la actividad respectiva, por lo que, el usuario tocará el sensor con su huella y se verificará la validez del mismo. Si la autenticación es correcta, se enviará una petición al servicio para que registre la entrada del usuario.

En caso de que la opción escogida por el usuario fue el registro por medio de PIN, se mostrará la actividad respectiva. Luego el usuario ingresará su código de seguridad el cual

se lo enviará en una petición al servicio para que este verifique y registre la entrada cuando el código sea válido.

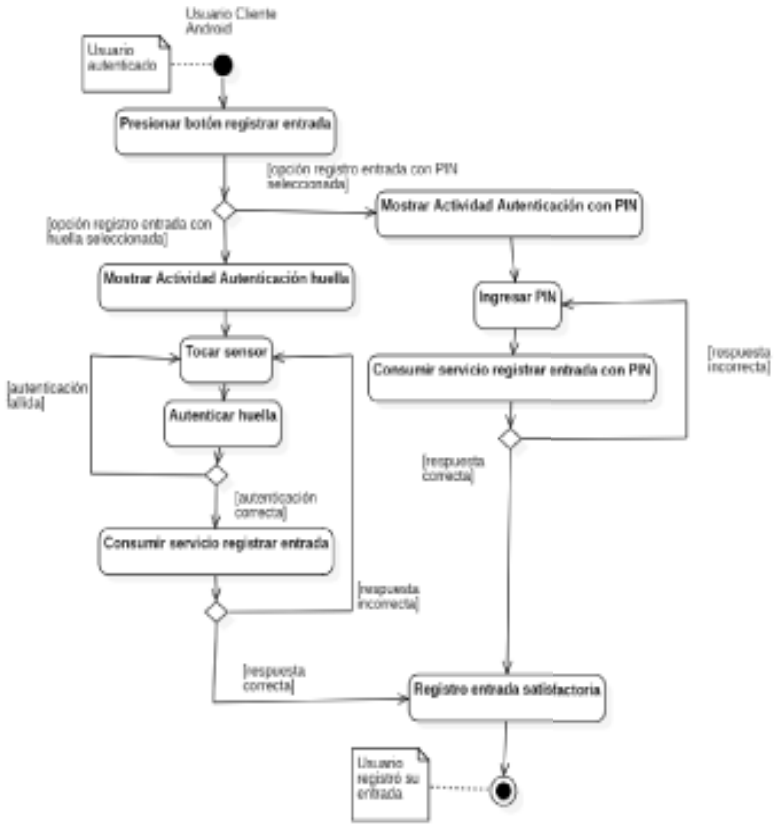


Figura 2.35.- Diagrama de actividades para registrar entrada

Consultar asistencias

Al ingresar el usuario al sistema, este podrá realizar consultas con respecto a sus registros de asistencias. En la Figura 2.36 se muestra el proceso para consultar estos registros.

Primeramente el usuario ingresará un intervalo de fechas para la búsqueda y luego, al presionar el botón respectivo, se enviará una petición al servicio para obtener una lista de registros de asistencias del usuario que cumplan con el criterio de búsqueda.

En caso de obtener resultados, se los mostrarán a través de una lista visual, el cual permitirá al usuario escoger cualquier registro con la finalidad de desplegar mayor detalle acerca del mismo.

En caso de no obtener resultados, se mostrará el mensaje respectivo y el usuario podrá ingresar otras fechas y realizar nuevamente la consulta.

Figura 2.36.- Diagrama de actividades para consultar asistencias

2.2. Implementación del prototipo

Se codificará la base de datos, se codificará la lógica del servicio WCF, se codificará la lógica de control de asistencia, se alojará el servicio web en IIS, se generarán las interfaces gráficas del Cliente Android, se codificará la lógica del Cliente Android, se generarán las interfaces gráficas del Cliente Gestor, se codificará la lógica del Cliente Gestor, se establecerá una red inalámbrica de pruebas y se instalará tanto el Cliente Android como el Cliente Gestor. Adicionalmente, por motivo a que es necesario realizar el control de asistencia de manera periódica, se codificará la lógica para controlar faltas y se lo alojará en el programador de tareas *Task Scheduler*[65] para llevar a cabo lo antes mencionado. A continuación se presenta un resumen de la implementación de cada componente.

Bases de datos

Para la codificación de la base de datos se empleó el sistema de gestión de base de datos Microsoft SQL Server Management Studio 2012. A continuación se muestra una parte del

[65] Planificador de Tareas (*Task Scheduler*): es una herramienta que permite programar tareas de manera periódica.

script para la creación de las tablas. El resto de tablas fueron implementadas de forma similar. Todos los *scripts* de la base de datos se encuentran en el Anexo IV.

En el Código 2.1, en la línea 5, se crea la base de datos denominada `ControlAsistencia`, la cual alojará toda la información del sistema. En las líneas 6 y 7 se especifica al motor que use la base de datos previamente creada.

```
5  create database ControlAsistencia
6  use ControlAsistencia
7  go
```

Código 2.1.- Sentencia SQL para crear la base de datos

Se implementó el horario de trabajo tomando en consideración los requerimientos del usuario, por lo que estos van a tener un día en que inicien su jornada, un día en que terminen su jornada, una hora en que inician y finalizan su jornada y una hora determinada para el almuerzo. En el Código 2.2 se muestran las sentencias para crear una tabla que guardará información acerca del horario de trabajo de un empleado. En la línea 9 se especifica el nombre de la tabla el cual tendrá las columnas establecidas dentro de las líneas 11 a la 17. La línea 11 establece que el identificador único de la tabla es `idHorarioTrabajo`, de tipo entero, y mediante la palabra clave `identity`, especifica que se va a incrementar el identificador automáticamente, cada vez que se ingresen nuevos registros. En las líneas 12 al 15 se definen las columnas denominadas `horaInicioJornada`, `horaSalidaJornada`, `horaInicioReceso` y `horaFinReceso`, de tipo de dato `time(0)`, el cual representa horas con precisión de dos dígitos de segundos decimales. Las líneas 16 y 17 definen los días de inicio y fin de jornada respectivamente, de tipo de dato `varchar(20)`, que denota cadena de caracteres con un número máximo de 20 de estos.

```
9   create table HorarioTrabajo
10  (
11  idHorarioTrabajo int not null identity primary key,
12  horaInicioJornada time(0),
13  horaSalidaJornada time(0),
14  horaInicioReceso time(0),
15  horaFinReceso time(0),
16  diaInicioJornada varchar(20),
17  diaFinJornada varchar (20)
18  )
```

Código 2.2.- Sentencia SQL para crear tabla `HorarioTrabajo`

Lógica del control de asistencia

Se codificó la lógica de control de asistencia en el lenguaje C#, con el entorno de desarrollo Visual Studio 2013. Toda la lógica se la implementó en la clase `ManejoRegistrosControlAsistencia`.

Un fragmento de la clase `ManejoRegistrosControlAsistencia` se muestra en el Código 2.3. Los métodos de esta clase permiten o rechazan el registro de entradas, de salidas, de asistencias, de atrasos, de faltas, etc. En la línea 9 se instancia un objeto de la clase `ManejoBBDD` dentro del constructor con la finalidad de manipular registros de la base de datos.

```
1.  namespace ControlAsistenciaEPN.ServicioControlAsistencia
2.  {
3.      public class ManejoRegistrosControlAsistencia
4.      {
5.          private ManejoBBDD miDb;
6.
7.          public ManejoRegistrosControlAsistencia()
8.          {
9.          miDb=new ManejoBBDD();
10.         }
11. ...
12.     }
13. }
```

Código 2.3.- Clase `ManejoRegistrosControlAsistencia`

A continuación se describen algunos métodos de la clase `ManejoRegistrosControlAsistencia`. El código de toda la clase se encuentra en el Anexo V.

Se muestra el método `VerificarEmpleadoDentroFeriado` en el Código 2.4. Este método retorna un valor booleano indicando si el empleado ingresado como parámetro actualmente se encuentra en feriado o no. En la línea 6, se busca en la base de datos un día de feriado para la fecha actual. Las líneas 7 a la 19 establecen la lógica cuando si exista feriado. En las líneas 10 al 13 se establece el valor `true` a la variable `verificacion` cuando la hora actual se encuentre dentro del periodo que dure el feriado en cuestión, caso contrario, se establece el valor `false` (línea 16). Las líneas 20 al 23 definen la lógica cuando no exista el feriado para la fecha actual. Por lo que, en la línea 21, se establece el valor `false` para `verificación`.

Código 2.4.- Método `VerificarEmpleadoDentroFeriado`

El método `VerificarSiEstoyDentroPermisosAprobadoHoy` retorna un valor booleano indicando si el empleado ingresado como parámetro se encuentra o no dentro de permisos aprobados. Este método se lo muestra en el Código 2.5. En la línea 6 se busca una solicitud de permiso aprobada para la fecha actual del empleado en cuestión. Las líneas 7 al 18 establecen la lógica cuando se encuentre una solicitud aprobada. En las líneas 9 al 12 se retorna el valor `true` cuando el empleado se encuentre dentro del intervalo del permiso aprobado. Caso contrario, retorna el valor `false`. Las líneas 19 al 23 retorna el valor `false` cuando no se pudo encontrar ninguna solicitud de permiso para la fecha especificada.

Código 2.5.- Método `VerificarSiEstoyDentroPermisosAprobadoHoy`

Para verificar si un empleado en particular se encuentra dentro de sus días de jornada de trabajo, se emplea el método `VerificarTrabajarHoy`, el cual se muestra en el Código 2.6. En la línea 8 se busca en la base de datos el horario del empleado. En las líneas 10 y 11 se convierten los días de inicio y fin de jornada de tipo `string` a tipo `DayOfWeek` por medio del método `ConvertirDia`. En la línea 12 se emplea el método `TrabajarHoy` el cual devuelve `true` cuando el empleado se encuentre dentro de los días de su jornada de trabajo. Por último, se retorna la respuesta anterior.

```
1.  public Boolean VerificarTrabajarHoy(int idEmpleado)
2.  {
3.      Boolean valorRetorno = false;
4.      //Ira acceder al horario de trabajo del empleado.
5.      int idHorario = miOb.BuscarIdHorarioEmpleado(idEmpleado);
6.      if (idHorario > 0)
7.      {       //Obtener el horario del empleado.
8.          HorarioTrabajo horarioTrabajador = miOb.BuscarHorario(idHorario);
9.          //verificar si aquie estoy dentro de la jornada de trabajo
10.         DayOfWeek diaInicioJornada = ConvertirDia(horarioTrabajador.diaInicioJornada);
11.         DayOfWeek diaFinJornada = ConvertirDia(horarioTrabajador.diaFinJornada);
12.         valorRetorno = TrabajarHoy(diaInicioJornada, diaFinJornada);
13.         return valorRetorno;
14.     }
15.     return valorRetorno;
16. }
```

Código 2.6.- Método `VerificarTrabajarHoy`

Para distinguir si un empleado se encuentra demasiado temprano, puntual, atrasado o demasiado tarde para registrar su entrada se emplea el método `VerificarRangoHoraIngreso`, el cual se lo muestra en el Código 2.7. En las líneas 4 al 8 permiten retornar el valor `-1` cuando el empleado ingrese antes de la hora en que se pueda registrar su entrada. Las líneas de la 8 a la 12 permite retornar el valor `0` cuando el empleado se encuentre puntual. Las líneas de la 12 a la 16 permite retornar el valor `1` cuando el empleado se encuentre atrasado. Por último, en las líneas de la 16 a la 20 se retorna el valor `2` cuando el empleado no pueda registrar su entrada debido a que se excedió de la hora límite para ello.

Se emplea el método `VerificarHorasJornadasFeriadosPermisoTodoEntrada` con la finalidad de integrar todas las verificaciones antes mostradas en un solo método. Debido a la extensión del código de este método, se lo divide en tres partes.

La primera parte del método para validar la entrada se lo muestra en el Código 2.8. En las líneas de la 4 a la 9 se retorna el valor `-1` cuando el empleado ingresado como parámetro se encuentre en feriado.

Código 2.7.- Método `VerificarRangoHoraIngreso`

```
1.  public int VerificarHorasJornadasFeriadosPermisoTodoEntrada(int idEmpleado)
2.  {
3.      int verificacion = -2;
4.      if (VerificarEmpleadoDentroFeriado() == true)
5.      {
6.          //Usted se encuentra dentro del periodo de feriado, no puede marcar asistencia
7.          verificacion = -1;
8.          return verificacion;
9.      }
10. ...
```

Código 2.8.- Método para validar entrada (Parte 1)

La segunda parte del método para validar la entrada se lo muestra en el Código 2.9. Las líneas 4 al 8, de igual manera retornan el valor `-1` cuando el empleado ingresado como parámetro se encuentre dentro de un intervalo de permisos aprobados. Las líneas 11 al 22 especifican la lógica de verificaciones cuando el empleado en cuestión se encuentre dentro de su jornada de trabajo. En la línea 12 se llama al método `VerificarRangoHoraIngreso`. Se retorna el valor `-1` cuando el empleado se encuentre demasiado temprano (líneas 13 al 17). Se retorna el valor `0` cuando el empleado se encuentre puntual (líneas 18 a 22).

En el Código 2.10 se encuentra la tercera y última parte del método para validar la entrada, la cual permite retornar el valor `1` cuando el empleado se encuentre atrasado (línea 26 y

27), y finalmente, se retorna el valor `-1` cuando el empleado se encuentre demasiado tarde (línea 31 y 32).

```
1.  public int VerificarHorasJornadasFeriadosPermisoTodoEntrada(int idEmpleado)
2.  {
3.  //Verificar Feriados
4.  if (VerificarSiEstoyDentroPermisosAprobadoHoy(idEmpleado))
5.          {            //Usted se encuentra dentro del periodo de permisos, NO puede marcar asistencia
6.              verificacion = -1;
7.              return verificacion;
8.          }
9.          else
10.         {
11.              if (VerificarTrabajarHoy(idEmpleado))
12.             {            int ingresoPuntualAtrasado=VerificarRangoHoraIngreso(idEmpleado);
13.                  if (ingresoPuntualAtrasado == -1)
14.                 {                    //No se puede registrar porque está demasiado temprano
15.                     verificacion = -1;
16.                     return verificacion;
17.                 }
18.                 else if (ingresoPuntualAtrasado == 0)
19.                 {                    //SI se puede registrar, usted está puntual
20.                     verificacion = 0;
21.                     return verificacion;
22.                 }
23. ...
```

Código 2.9.- Método para validar entrada (Parte 2)

```
23. ...
24.                 else if (ingresoPuntualAtrasado == 1)
25.                 {                    //SI se puede registrar pero usted está atrasado
26.                     verificacion = 1;
27.                     return verificacion;
28.                 }
29.                 else
30.                 {                    //Se encuentra demasiado tarde
31.                     verificacion = -1;
32.                     return verificacion;
33.                 }
34.             }
35.         }
36. }
```

Código 2.10.- Método para validar entrada (Parte 3)

Lógica para controlar faltas

Se creó un programa llamado `ControladorFaltas` con el lenguaje de programación C# que permite controlar las faltas de los empleados, y se ejecuta de manera periódica empleado la herramienta Task Scheduler de Windows 10. A continuación se explican las partes más importantes de la lógica para realizar el proceso antes mencionado. En el Código 2.11 se muestra el método `Main` de `ControladorFaltas`. En la línea 6 se busca en la base de datos una lista de todos los empleados habilitados. En las líneas 9 al 14 se realiza un proceso de control de faltas de forma iterativa, para cada empleado de la lista en cuestión se empleará el método `RegistrarFaltasInjustificadas`.

```
3.      static void Main(string[] args)
4.          {
5.              ManejoBBDD miDb = new ManejoBBDD();
6.              List<Empleado> listaEmpleado = miDb.BuscarEmpleadoTodo("Habilitado");
7.              if (listaEmpleado != null)
8.              {
9.                  foreach (var item in listaEmpleado)
10.                 {
11.                     if (item.idEmpleado > 0)
12.                     {
13.                         RegistrarFaltaInjustificadas(item.idEmpleado);}}}
14.                     }
```

Código 2.11.- Método `Main` de `ControladorFaltas`

En el Código 2.12 se muestra la primera parte del método `RegistrarFaltaInjustificadas`. En la línea 3 se llama al método `VerificacionesParaRegistrarFaltasInjustificadas` el cual retorna el código 0 cuando el empleado no registró su asistencia para la fecha actual. Por lo que, en las líneas 6 al 10 se inserta un nuevo registro de falta verificando que esta no se encuentre duplicada.

```
1.  public static int RegistrarFaltaInjustificadas(int idEmpleado)
2.        {...
3.            int verificar = VerificacionesParaRegistrarFaltasInjustificadas(idEmpleado);
4.            if (verificar == 0)
5.            {
6.                RegistroFalta registroExistente = miDb.BuscarFaltaInjustificada(idEmpleado, DateTime.Now);
7.                if (registroExistente.idRegistroFalta > 0)
8.                {//No insertar falta
9.                }else{
10.               return miDb.InsertarRegistroFalta(idEmpleado);}}
11. ...
```

Código 2.12.- Método `RegistrarFaltaInjustificadas` de `ControladorFaltas` (Parte 1)

La segunda parte del método `RegistrarFaltaInjustificadas` se lo presenta en el Código 2.13. Aquí cuando el método `VerificacionesParaRegistrarFaltasInjustificadas` retorne el código `-2` (línea 12) indica que el empleado en cuestión no se encuentra dentro de su jornada de trabajo convencional. Por lo que se verifica si el empleado fue convocado para trabajar en un horario no convencional (líneas 14 y 15). Las líneas 16 al 24 muestran que cuando si le convocaron al empleado en cuestión, se verificará si este asistió o no a la convocatoria. En la línea 18 se busca la asistencia. En las líneas 19 al 21 se especifica que no se inserte la

falta debido a que el empleado si asistió a la convocatoria, por el contrario, en las líneas 21 al 23 se inserta la falta cuando no exista tal asistencia.

```
11. ...
12.             else if (verificar == -2)
13.             {
14.                 SolicitudTrabajoNoConvencional solicitudTrabajoHoy =
15. miDb.BuscarSolicitudTrabajoNoConvencionalHoy(idEmpleado, DateTime.Now);
16.                 if (solicitudTrabajoHoy.idSolicitud > 0)
17.                 {
18.                     RegistroAsistencia registroAsistenciaHoy = miDb.BuscarRegistroAsistenciaHoy(idEmpleado, DateTime.Now);
19.                     if (registroAsistenciaHoy.idRegistroAsistencia > 0)
20.                     {//No insertar falta
21.                     } else  {
22.                         return miDb.InsertarRegistroFalta(idEmpleado);
23.                     }
24.                 }
25. ...
```

Código 2.13.- Método `RegistrarFaltaInjustificadas` de `ControladorFaltas` (Parte 2)

Con la finalidad de controlar las faltas de los empleados, cada día, de manera periódica y a una determinada hora, se optó por emplear el *Task Scheduler* de Windows 10. Para ello se ingresa a la herramienta y en la ventana de Acciones, se selecciona la opción Crear tarea básica, como se muestra en la Figura 2.37.

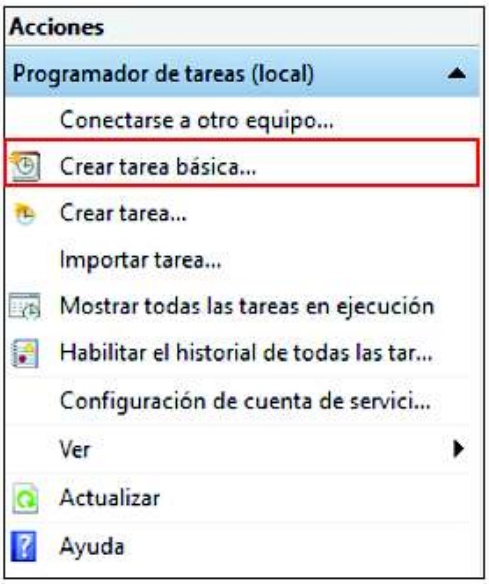


Figura 2.37.- Crear tarea básica *Task Scheduler*

Se ingresa el nombre y la descripción de la tarea, como lo indica en la Figura 2.38.

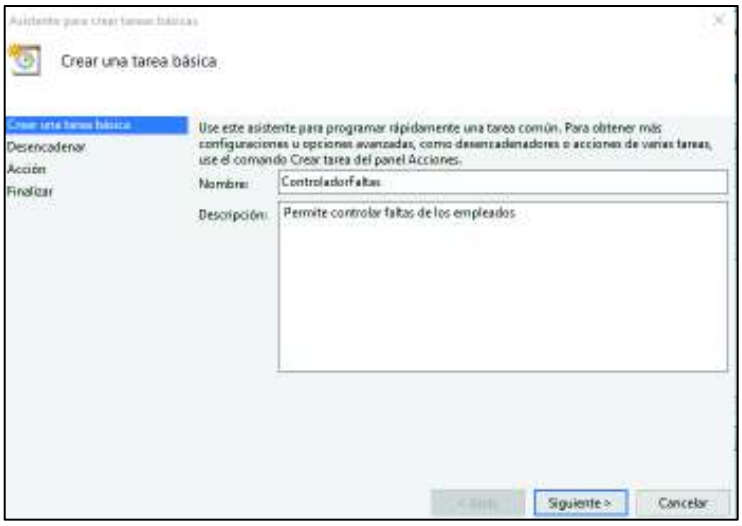


Figura 2.38.- Detalles de la tarea básica

En la Figura 2.39 se escoge la opción Diariamente y se configura la hora en que se inicia la tarea. En este caso, se establece que se realice la tarea cada día, a las 21:30.

Figura 2.39.- Configuración diaria de la tarea

En la Figura 2.40 y Figura 2.41 se establece la tarea a realizar, en este caso se selecciona el archivo ejecutable del Controlador de Faltas, ubicado dentro de la carpeta `bin\Debug` del proyecto.

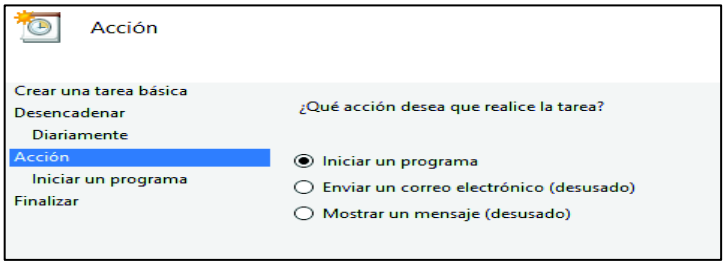


Figura 2.40.- Configurar Acción

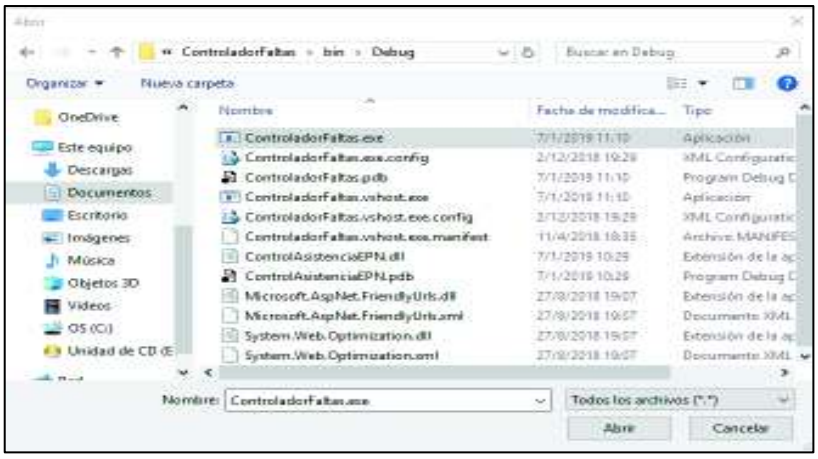


Figura 2.41.- Seleccionar el ejecutable `ControladorFaltas.exe`

Por último, en la **Figura 2.42** se presenta un resumen de toda la configuración de la tarea y se finaliza el proceso.

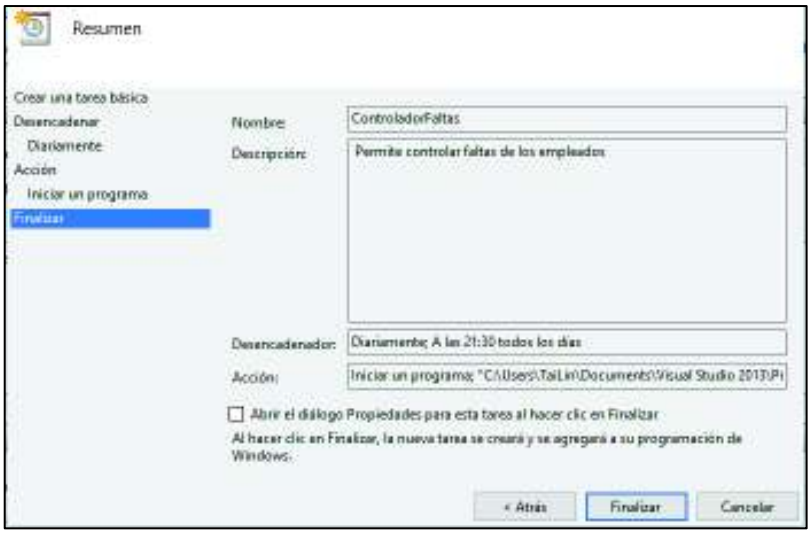


Figura 2.42.- Resumen de la configuración de la tarea

Servicio WCF

La lógica del servicio WCF fue codificado en lenguaje C#, y fue desarrollado en el entorno Visual Studio 2013.

Se creó un nuevo proyecto de tipo ASP.NET *Web Application* llamado `ControlAsistenciaEPN`, como se lo muestra en la Figura 2.43. El servicio WCF se encuentra en el Anexo V.

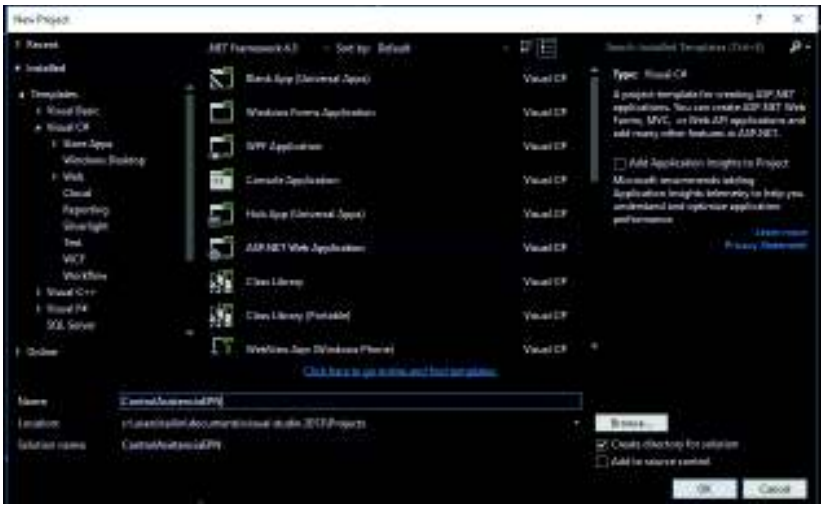


Figura 2.43.- Creación del proyecto `ControlAsistenciaEPN`

Se agregó una carpeta al proyecto con el espacio de nombres `ServicioControlAsistencia`, el cual albergará el servicio WCF. Se agregó un servicio WCF a la carpeta creada anteriormente con el nombre `ServicioControlAsistencia.svc`, como se muestra en la Figura 2.44. Se generaron los archivos `IServicioControlAsistencia.cs`, `ServicioControlAsistencia.svc.cs` y `Web.config`.

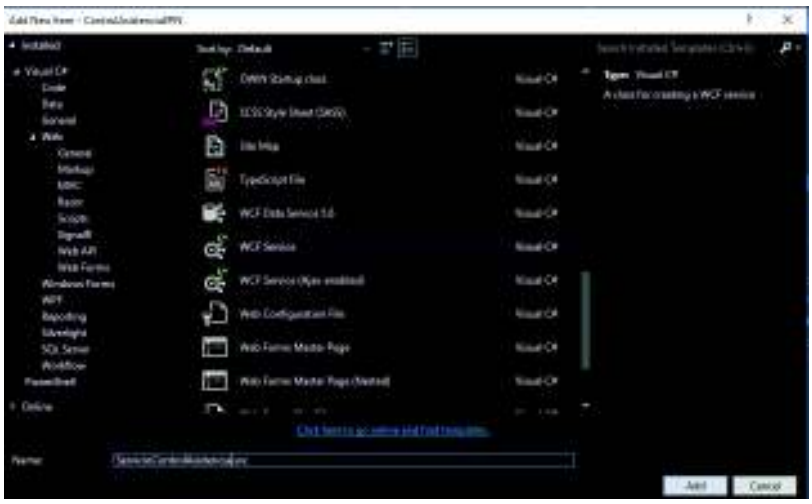


Figura 2.44.- Creación del servicio WCF

Para definir los tipos de datos a ser intercambiados por el servicio, se agregó una carpeta al proyecto con el espacio de nombres `ClasesContrato`. Se crearon dentro de esta carpeta las clases `CalendarioFeriado`, `ConfiguracionHorasSistema`, `Credencial`, `DiaFeriado`, `Empleado`, `Geovalla`, `HorarioTrabajo`, `MensajeServicio`, `NotificacionGeofencing`, `RegistroAsistencia`, `RegistroAsistenciaHistorialCambios`, `RegistroAtraso`,

RegistroAtrasoHistorialCambios, RegistroEntradaSalida, RegistroFalta, RegistroFaltaHistorialCambios, SolicitudPermiso, SolicitudPermisoHistorialCambios, SolicitudTrabajoNoConvencional, SolicitudTrabajoNoConvencionalHistorialCambio. Además, esta carpeta contendrá a la clase ManejoBBDD, el cual tendrá métodos que permitirán ejecutar sentencias SQL y convertir registros de la base de datos en objetos.

En el Código 2.14 se muestra una parte de la definición de la clase horario de trabajo. Las líneas 1 y 2 establecen que la clase HorarioTrabajo es un tipo de dato que va a ser serializado e intercambiado por el servicio. Las líneas 4 al 17 definen los atributos del horario de trabajo, los adornos [DataMember] que se encuentran antes de estos especifican que se serialicen los atributos en cuestión. El resto de las clases del contrato de datos fueron implementadas de forma similar.

```
1.  [DataContract]
2.      public class HorarioTrabajo
3.      {
4.          [DataMember]
5.          public int idHorario { get; set; }
6.          [DataMember]
7.          public TimeSpan horaInicioJornada {get; set;}
8.          [DataMember]
9.           public TimeSpan horaSalidaJornada {get; set;}
10.         [DataMember]
11.          public TimeSpan horaInicioReceso {get; set;}
12.         [DataMember]
13.         public TimeSpan horaFinReceso {get; set;}
14.         [DataMember]
15.         public string diaInicioJornada { get; set; }
16.         [DataMember]
17.         public string diaFinJornada { get; set; }
18.     }
```

Código 2.14.- Clase HorarioTrabajo del servicio WCF

En el Código 2.15 se muestra una parte de la interfaz del contrato de servicio, el cual definirá todas las operaciones que ofrecerá el servicio. En la línea 1 se muestra el espacio de nombres. En las líneas 3 y 4 se establece que la interfaz IServicioControlAsistencia es el contrato de servicio.

En el Código 2.16 se muestra un ejemplo de definición de la operación que permite registrar la entrada de un empleado. En las líneas 1 y 2 se especifica que se debe emplear el método HTTP POST y la dirección relativa del recurso /ControlAsistencia/RegistrarEntrada para que se pueda consumir la operación en cuestión, además se especifica que la petición/respuesta debe estar en formato JSON.

En la línea 3 se especifica un parámetro de entrada de tipo `Empleado` y el tipo de retorno (`MensajeServicio`) que tiene la operación.

```
1. namespace ControlAsistenciaEPN.ServicioControlAsistencia
2. {
3. [ServiceContract]
4.     public interface IServicioControlAsistencia
5.     {
6. ...
7.     }
8. }
```

Código 2.15.- Contrato de servicio `IServicioControlAsistencia`

```
1.  [WebInvoke(Method = "POST", UriTemplate = "/ControlAsistencia/RegistrarEntrada", ResponseFormat = WebMessageFormat.Json,
2. RequestFormat = WebMessageFormat.Json)]
3.         MensajeServicio RegistrarEntrada(Empleado empleado);
```

Código 2.16.- Definición de la operación `RegistrarEntrada`

En el Código 2.17 se muestra un ejemplo de definición de una operación que permite consultar registros de asistencias por un determinado periodo de tiempo. En la línea 1 se establece que para consumir el servicio se debe emplear el método HTTP `GET` y que la petición/respuesta se encuentra en formato JSON. En la línea 2 se establece la dirección del recurso, el cual define los parámetros de entrada `idEmpleado`, `desdeFecha` y `hastaFecha`. En la línea 3 se muestra que la operación posee como tipo de retorno una lista de registros de asistencias y acepta como parámetros el identificador del empleado y un intervalo de fechas.

```
1. [WebGet(ResponseFormat = WebMessageFormat.Json,
2. UriTemplate = "/Consultar/RegistrosAsistencia/?id={idEmpleado}&desdeFecha={desdeFecha}&hastaFecha={hastaFecha}")]
3.         List<RegistroAsistencia> BuscarRegistrosAsistencia(int idEmpleado, string desdeFecha, string hastaFecha);
```

Código 2.17.- Definición de la operación `BuscarRegistrosAsistencia`

Cuando una operación especifica el método HTTP `POST` y esta requiera más de un parámetro de entrada, se debe especificar el atributo `BodyStyle = WebMessageBodyStyle.WrappedRequest` como se lo muestra en la línea 5 del Código 2.18. En este caso, la operación en cuestión recibe como argumentos un registro de asistencia y un identificador de un empleado en particular, además, establece el tipo de dato `MensajeServicio` como retorno. El resto de definiciones de operaciones del servicio fueron implementadas de forma similar.

```
1. [WebInvoke(Method = "POST",
2. UriTemplate = "/Gestion/Asistencia/AgregarAsistencia",
3.  ResponseFormat = WebMessageFormat.Json,
4. RequestFormat = WebMessageFormat.Json,
5. BodyStyle = WebMessageBodyStyle.WrappedRequest)]
6.         MensajeServicio AgregarRegistroAsistencia(RegistroAsistencia registro, int idEmpleado);
```

Código 2.18.- Definición de la operación `AgregarRegistroAsistencia`

Hasta el momento, solo se han prestado los fines de operaciones del servicio, a continuación se muestran la implementación de la lógica de las operaciones `RegistrarEntrada`, `RegistrarSalida` y `BuscarRegistrosAsistencia` establecidas en la clase que implementa la interfaz de servicio `ServicioControlAsistencia`. En el Código 2.19 se muestra una parte de la clase `ServicioControlAsistencia`. En la línea 3 se habilita la compatibilidad de ASP.NET con el servicio WCF para que este último pueda mantener estado de sesión. En las líneas 6 al 8 se declaran variables con el modificador de acceso privado, para que estos puedan ser empleados posteriormente. En las líneas 10 al 14 se establece el constructor de la clase, el cual instancia objetos de las clases `HttpContext` (línea 11), `ManejoBBDD` (línea 12) y `ManejoRegistrosControlAsistencia` (línea 13), los cuales permiten manipular variables de sesión, manipular registros de tablas en la base de datos y validar los registros de asistencias (registros de entrada, salidas, atrasos, faltas, etc) respectivamente.

```
1  namespace ControlAsistenciaEPN.ServicioControlAsistencia
2  {
3      [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]
4      public class ServicioControlAsistencia : IServicioControlAsistencia
5      {
6          private readonly HttpContext contexto;
7          private ManejoBBDD miDb;
8          private ManejoRegistrosControlAsistencia controlAsistencia;
9
10         public ServicioControlAsistencia() {
11             contexto = HttpContext.Current;
12             miDb = new ManejoBBDD();
13             controlAsistencia = new ManejoRegistrosControlAsistencia();
14         }
15 ...
16     }
17 }
```

Código 2.19.- Clase `ServicioControlAsistencia`

A continuación se presenta el método `RegistrarEntrada`, el cual ha sido dividido en cuatro partes debido a su extensión.

La primera parte se lo muestra en Código 2.20. En las líneas 2 y 3 se inicializan el identificador del empleado y un mensaje por defecto. En la línea 4 se llama al método

`VerificarHorasJornadasFeriadosPermisoTodoEntrada` el cual retornará códigos de valores enteros del `-3` al `1` que indican si el empleado ingresado como parámetro puede o no registrar su entrada. Las líneas 5 a la 21 implementan la lógica del registro cuando el empleado ingresó puntualmente en días dentro de jornadas de trabajo. En la línea 8 se llama al método `RegistrarEntradaEmpleado` el cual permite insertar un nuevo registro de entrada siempre y cuando esta sea única para la fecha actual. Las líneas 10 a la 17 permiten establecer un mensaje con mayor información acerca del registro. La línea 12 establece un mensaje de fallo en el registro de entrada debido a que el empleado ya se ha registrado anteriormente. En la línea 16 se establece un mensaje de éxito en el registro de entrada. En las líneas 18 y 19 se almacena el mensaje y se lo retorna.

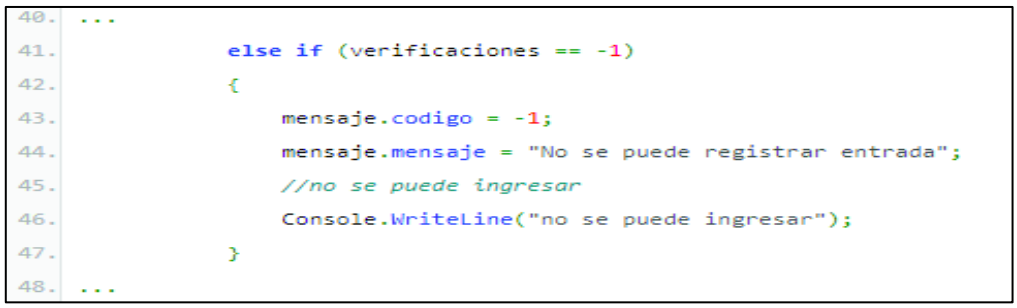
```
1       public MensajeServicio RegistrarEntrada(Empleado empleado) {
2           int idEmpleado = empleado.idEmpleado;
3           MensajeServicio mensaje = new MensajeServicio();
4           int verificaciones = controlAsistencia.VerificarHorasJornadasFeriadosPermisoTodoEntrada(idEmpleado);
5           if (verificaciones == 0)
6           {
7               //se llegó puntual
8               int codigo=controlAsistencia.RegistrarEntradaEmpleado(idEmpleado);
9               mensaje.codigo = codigo;
10              if (codigo == -1)
11              {
12                  mensaje.mensaje = "No se puede registrar otra vez su entrada";
13              }
14              else
15              {
16                  mensaje.mensaje = "Registro de entrada satisfactorio";
17              }
18              mensaje.codigo = codigo;
19              return mensaje;
20
21          }
22  ...
```

Código 2.20.- Implementación del método `RegistrarEntrada` (Parte 1)

La segunda parte del método para registrar la entrada del empleado se lo muestra en el Código 2.21, e indica la lógica de lo que sucede cuando el empleado se encuentre atrasado. En la línea 22 se indica que cuando se tenga el código igual a `1` como respuesta del método `VerificarHorasJornadasFeriadosPermisoTodoEntrada`, el empleado se encuentra atrasado y se lo registra (línea 24). En la línea 25 se llama al método `RegistrarEntradaEmpleado` para que se inserte un registro de entrada del empleado en cuestión. En la línea 29 se establece un mensaje de falla cuando no se pudo registrar la entrada. En las líneas 31 a 37 se establece un mensaje indicando que el empleado se encuentra atrasado y recordándole de su hora de entrada.

Código 2.21.- Implementación del método `RegistrarEntrada` (Parte 2)

La tercera parte del método para registrar entrada se lo muestra en el Código 2.22. Cuando se tenga el código igual a `-1` como respuesta del método `VerificarHorasJornadasFeriadosPermisoTodoEntrada`, el empleado no puede registrar su entrada debido a que este tiene permiso o tiene feriado. Por lo que, en la línea 44 se establece un mensaje de fallo en el registro de entrada.



Código 2.22.- Implementación del método `RegistrarEntrada` (Parte 3)

En la cuarta y última parte del método para registrar la entrada, se establece lo que sucede cuando un empleado ingrese en días fuera de su jornada de trabajo (ver Código 2.23). En las líneas 51 y 52 se busca una solicitud de trabajo no convencional del empleado en cuestión para la fecha actual. En las líneas 53 a 65 se registra la entrada del empleado, cuando exista una solicitud de trabajo y retorna un mensaje sobre el éxito o fracaso del registro en cuestión. En las líneas 66 a 71 se indica que el empleado no tiene ninguna solicitud de trabajo por lo que no puede registrar su entrada.

```
49.             else if (verificaciones == -3)
50.             {
51.                 SolicitudTrabajoNoConvencional miSolicituTrabajoNoConvencionalHoy =
52. miDb.BuscarSolicitudTrabajoNoConvencionalHoy(idEmpleado, DateTime.Now);
53.                 if (miSolicituTrabajoNoConvencionalHoy.idSolicitud > 0)
54.                 {    int codigo=controlAsistencia.RegistrarEntradaEmpleado(idEmpleado);
55.                      mensaje.codigo = codigo;
56.                      if (codigo == -1)
57.                      {
58.                          mensaje.mensaje = "No se puede registrar otra vez su entrada";
59.                      }
60.                      else
61.                      {
62.                          mensaje.mensaje = "Se registró su entrada de manera correcta";
63.                      }
64.                      return mensaje;
65.                 }
66.                 else
67.                 {    mensaje.codigo = -4;
68.                     mensaje.mensaje = "No se puede registrar su entrada porque usted
69. no tiene una solicitud de trabajo para el día de hoy";
70.                      return mensaje;
71.                 }
```

Código 2.23.- Implementación del método `RegistrarEntrada` (Parte 4)

A continuación, en el Código 2.24 se muestra una parte de la implementación de la lógica para registrar la salida de un empleado. Este método es similar al método `RegistrarEntrada`, sin embargo este a diferencia del anterior, emplea el método `VerificarHorasJornadasFeriadosPermisoTodoSalida` (líneas 4 y 5) para validar el registro en cuestión. En las líneas 6 a 23 se registra la salida, en los días en los que el empleado trabaje. En la línea 7 se llama al método `RegistrarSalidaEmpleado` el cual retorna un entero que indica si se pudo o no realizar el registro en cuestión. Se establecen los mensajes de éxito o de fallo para el proceso de registro de salida (líneas 9, 11 y 15). Después de registrar la salida, se procede a registrar la asistencia por medio del método `RegistrarAsistencia` (línea 17). En la línea 20 se establece un mensaje que indica si el registro de asistencia falló. Por último se retorna el mensaje (línea 22).

En el Código 2.25 se presenta la implementación del método que permite consultar registros de asistencias de un empleado en particular dado un intervalo de fechas. En la línea 2 se llama al método `BuscarRegistrosAsistencia` de la instancia `ManejoBBDD`, el cual ejecuta la sentencia SQL correspondiente para retornar una lista de registros que coincidan con los criterios de búsqueda.

```
1.  public MensajeServicio RegistrarSalida(Empleado empleado) {
2.      MensajeServicio mensaje = new MensajeServicio();
3.      int idEmpleado = empleado.idEmpleado;
4.      int verificacionesSalida =
5.  controlAsistencia.VerificarHorasJornadasFeriadosPermisoTodoSalida(idEmpleado);
6.      if (verificacionesSalida == 0)
7.      { int codigo=controlAsistencia.RegistrarSalidaEmpleado(idEmpleado);
8.          if (codigo == -2)
9.          {mensaje.mensaje = "Sucedió un error al registrar su salida, inténtelo más tarde";
10.         }
11.         else if (codigo == 2) {mensaje.mensaje = "Se registró su salida correctamente";
12.         }
13.         else
14.         {
15.             mensaje.mensaje = "Se registró su salida correctamente";
16.         }
17.         codigo=controlAsistencia.RegistrarAsistencia(idEmpleado);
18.         mensaje.codigo = codigo;
19.         if (codigo == -1) {
20.             mensaje.mensaje = "No se pudo registrar asistencia debido a que solo asistió por poco tiempo";
21.         }
22.         return mensaje;
23.     }
24. ...
```

Código 2.24.- Implementación del método `RegistrarSalida`

```
1.  public List<RegistroAsistencia> BuscarRegistrosAsistencia(int idEmpleado, string desdeFecha, string hastaFecha) {
2.              return miDb.BuscarRegistrosAsistencia(idEmpleado, desdeFecha, hastaFecha);
3.          }
```

Código 2.25.- Implementación del método `BuscarRegistrosAsistencia`

En el Código 2.26 se muestra una parte de la clase `ManejoBBDD`. En las líneas 3 y 4 se definen la cadena de conexión de la base de datos `ControlAsistencia`.

```
1.  public class ManejoBBDD
2.      {
3.       private string cadenaConexion =
4.          @"Server=TAI\SQLEXPRESS;Database=ControlAsistencia;User ID=usuarioPrueba;Password=1234";
5.      ...
6.      }
```

Código 2.26.- Clase `ManejoBBDD`

En el Código 2.27 se muestra un método de la clase `ManejoBBDD` que permite ejecutar una determinada sentencia SQL. La línea 1 especifica que se retorna un número entero y recibe como parámetro una cadena de caracteres que representa una sentencia SQL. En la línea 3 se instancia una nueva conexión a la base de datos por medio de la cadena de conexión. En la línea 4 se instancia un comando SQL para la conexión creada anteriormente. En las líneas 8 y 9 se conecta con la base y se ejecuta el comando, además, se guarda el número total de filas afectadas en `resultado`. Las líneas 11 a 15 controlan

excepciones. Las líneas 16 a 19 se encargan de cerrar la conexión con la base de datos. Y por último, en la línea 20, se retorna el número de filas afectadas por la ejecución del comando.

```
1.  private int EjecutarSentencia(string sentenciaSql)
2.        {
3.            SqlConnection conexion = new SqlConnection(cadenaConexion);
4.            SqlCommand comando = new SqlCommand(sentenciaSql, conexion);
5.            int resultado = 0;
6.            try
7.            {
8.                conexion.Open();
9.                resultado = comando.ExecuteNonQuery();
10.           }
11.           catch (SqlException e)
12.           {
13.               Console.WriteLine(e.Message);
14.               resultado = 0;
15.           }
16.           finally
17.           {
18.               conexion.Close();
19.           }
20.           return resultado;
21.       }
```

Código 2.27.- Método `EjecutarSentencia`

En el Código 2.28 se muestra un método que permite insertar en la base de datos un registro de entrada del empleado especificado como parámetro y retorna un valor entero (Línea 1). Las líneas 3 a 7 establecen la sentencia SQL para agregar un nuevo registro de entrada. En la línea 9 se ejecuta la sentencia y se retorna el número total de filas afectadas. El resto de métodos de la clase `ManejoBBDD` fueron implementadas de forma similar.

```
1.      public int InsertarRegistroEntrada(int idEmpleado)
2.       {
3.           string sentenciaSql = "INSERT INTO RegistroEntradaSalida ";
4.           sentenciaSql += "(idEmpleado , ingresoSalida, fechaHora) VALUES (";
5.           sentenciaSql += idEmpleado + ",";
6.           sentenciaSql += "'Ingreso' ,'";
7.           sentenciaSql += DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fffffff") + "')";
8.           Console.WriteLine(sentenciaSql);
9.           return EjecutarSentencia(sentenciaSql);
10.      }
```

Código 2.28.- Método `InsertarRegistroEntrada`

Para establecer que el servicio sea de tipo REST y para establecer los protocolos de comunicación, se debe configurar el archivo `Web.Config`. A continuación se muestran las partes más importantes de la configuración.

En el Código 2.29 se muestra el segmento del archivo de configuración del servicio que permite agregar una cadena de conexión a la base de datos con el nombre `ControlAsistencia` (línea 4).

```
1. <?xml version="1.0"?>
2. <configuration>
3.   <connectionStrings>
4.     <add name="ControlAsistencia" connectionString="Server=[illegible]" />
5.   </connectionStrings>
```

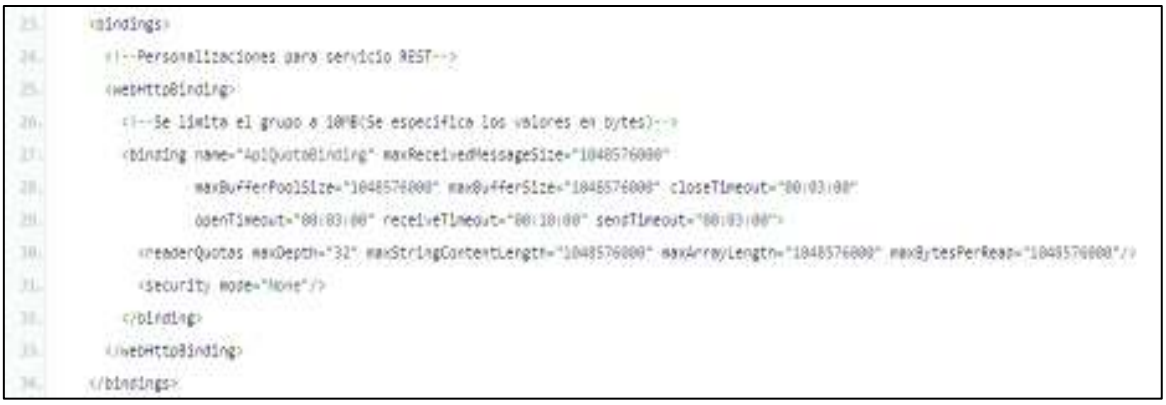
Código 2.29.- Segmento `connectionStrings` de `Web.config`

En el Código 2.30 se muestra una sección para configurar el servicio. La línea 15 establece el nombre del servicio en cuestión. Las líneas 16 a 20 definen el extremo del servicio. La línea 16 indica que se establece una dirección del extremo. En la línea 17 se especifica que se emplea el protocolo HTTP. La línea 18 establece el contrato del servicio. En la línea 19 indica que se emplea una configuración de canales personalizada llamado `ApiQuotaBinding` y se lo define en la sección `bindings`. En la línea 20 se especifica el comportamiento `webHttpBehavior` el cual se lo define en la sección `behaviors`.

```
15.   <system.serviceModel>
16.     <services>
17.       <service name="ControlAsistenciaEPN.ServicioControlAsistencia.ServicioControlAsistencia">
18.         <endpoint address=""
19.                   binding="webHttpBinding"
20.                   contract="ControlAsistenciaEPN.ServicioControlAsistencia.IServicioControlAsistencia"
21.         bindingConfiguration="ApiQuotaBinding"
22.         behaviorConfiguration="webHttpBehavior"/>
23.       </service>
24.     </services>
```

Código 2.30.- Segmento `services` de `Web.config`

En el Código 2.31 se muestra el segmento de `bindings` del archivo de configuración. Aquí en las líneas 27 a 30, se personaliza el rendimiento del servicio, para que se reciba datos como máximo de 10 MB. Además se configuran los tiempos límites para abrir y cerrar conexiones con el servicio, así mismo para configurar los tiempos en que se pueda realizar operaciones de lectura y escritura asociadas a mensajes recibidos o enviados. Estos tiempos permiten mitigar los ataques de denegación de servicio [33].



Código 2.31.- Segmento `bindings` de `Web.config`

Se muestra en el Código 2.32 la descripción del comportamiento de extremos, en el cual se define el comportamiento llamado `webHttpBehavior` (línea 37), y por medio del elemento `webHttp` se especifica que el extremo se comporte como REST (línea 38).

```
35.     <behaviors>
36.       <endpointBehaviors>
37.         <behavior name="webHttpBehavior">
38.           <webHttp/>
39.         </behavior>
40.       </endpointBehaviors>
41.     </behaviors>
42.   </system.serviceModel>
```

Código 2.32.- Segmento `behaviors` de `Web.Config`

Hosting del servicio WCF

Para alojar el servicio WCF se empleó de IIS para Windows 10.

Para publicar el servicio se seleccionó al servicio `ControlAsistenciaEPN` en el explorador de soluciones del IDE Visual Studio y se le dio *Click* derecho, *Publish*, y aparece una ventana como se muestra en la Figura 2.45. Aquí se debe seleccionar un perfil para que se pueda publicar y luego presionar en el botón *Next*.

Se debe configurar el método de publicación como se lo muestra en la Figura 2.46, de tipo *File System* y se elige la ubicación en el cual se exportará el servicio. Después se selecciona el botón *Next*.

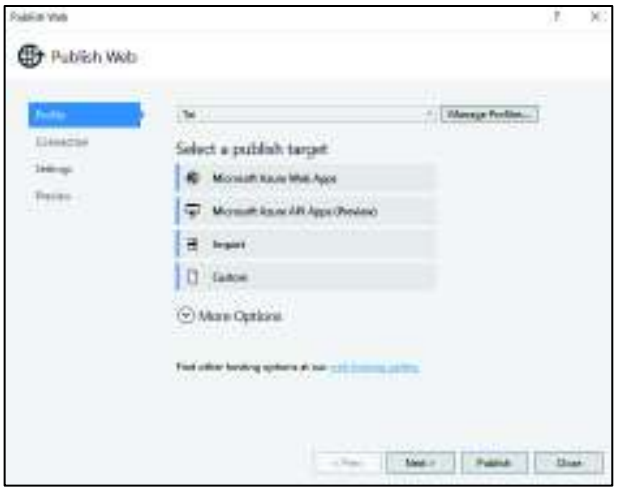


Figura 2.45.- Publicación del servicio (Parte 1)

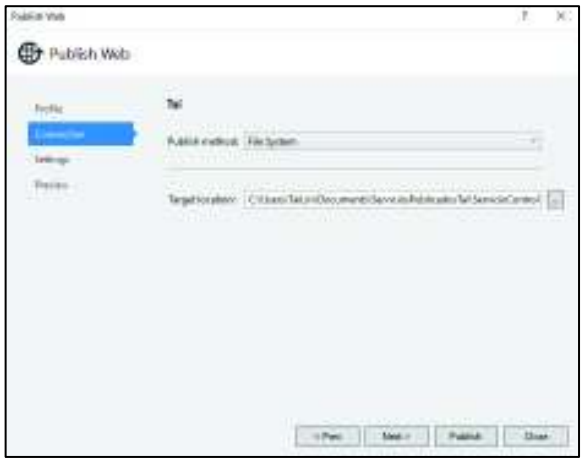


Figura 2.46.- Publicación del servicio (Parte 2)

Por último, se selecciona la configuración de la publicación como *Release* y se da *click* al botón *Publish* como se lo observa en la Figura 2.47.

Figura 2.47.- Publicación del servicio (Parte 3)

Se copió el servicio publicado y se lo pegó en la carpeta `C:\inetpub\wwwroot` como se lo muestra en Figura 2.48.

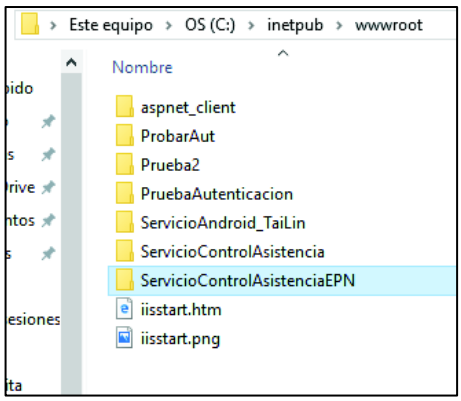


Figura 2.48.- Traslado del servicio publicado

Se creó un nuevo sitio web en el Administrador de IIS como se muestra en la Figura 2.49. Se estableció el nombre del sitio web como `ControlAsistenciaEPN`. Se seleccionó la ruta de acceso física al servicio publicado anteriormente, en la carpeta `C:\inetpub\wwwroot\ServicioControlAsistenciaEPN`. Además se estableció como dirección IP del sitio `a 192.168.137.1` y el puerto `8094`.

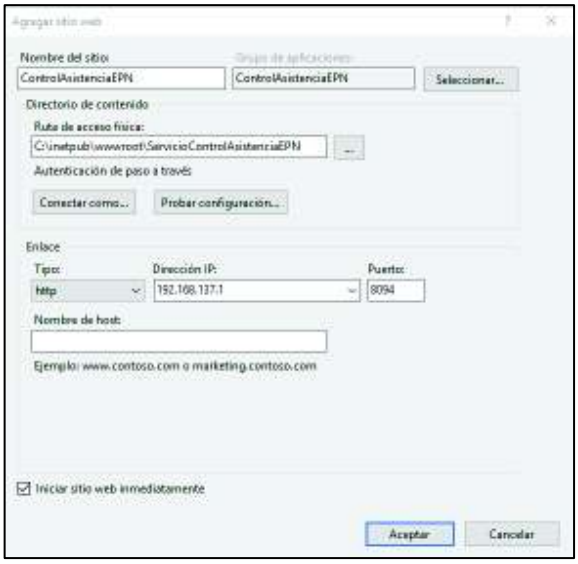


Figura 2.49.- Agregar sitio web

Se configuró los permisos de los archivos `Web.config` y `ServicioControlAsistencia.svc` para que se tenga el usuario `IIS_IUSRS`. Para ello se editan los dos archivos mencionados y se les da *click* derecho, Editar permisos, Seguridad, Editar, Agregar y se ingresa el usuario, por ejemplo `NombreUsuario\IIS_IUSRS` como se muestra en la Figura 2.50. Y por último se da *click* en Aceptar.

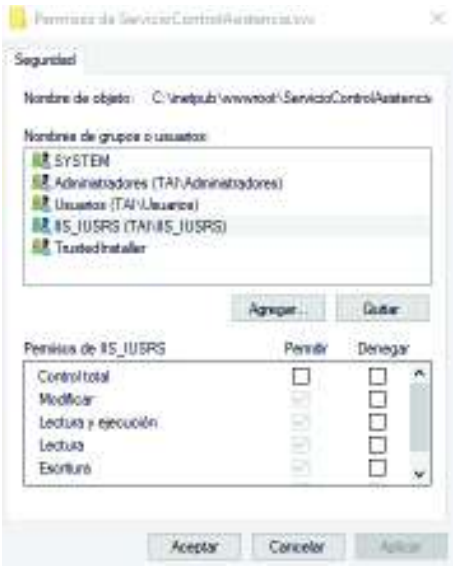


Figura 2.50.- Configuración Permisos

Se agregó una cadena de conexión para el sitio web navegando hacia la opción “Cadenas de Conexión” en la sección de ASP.NET, en la Vista de Características del Administrador de IIS como se lo muestra en Figura 2.51. Y se ingresó toda la información referente a la base de datos y sus correspondientes credenciales como se muestra en la Figura 2.52.

Figura 2.51.- Opción para agregar cadenas de conexión

Nombre	Cadena de conexión
ControlAsistencia	Server=TAI\\SQLEXPRESS;Database=ControlAsistencia;User ID=usuarioPrueba;Password=●●●●
LocalSqlServer	data source=.\SQLEXPRESS;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true

Figura 2.52.- Configuración de la cadena de conexión del sitio web

Por último, se configuró las ACL[66] para establecer las direcciones IP que puedan acceder al sitio web. Para ello se ingresó a la opción de “Restricciones de direcciones IP y dominios” en la vista de características del Administrador de IIS y se agrega una dirección IP como se muestra en la Figura 2.53.

Figura 2.53.- Agregar regla de restricción de permiso

Cliente Android

Se implementó el Cliente Android usando el IDE Android Studio 3.2.1. La lógica del Cliente Android fue realizada en lenguaje Java, y las interfaces gráficas de las actividades fueron implementadas en lenguaje XML con la ayuda del editor de *layouts* de Android Studio[67]. La aplicación del Cliente Android se lo puede encontrar en el Anexo VI.

Interfaz gráfica Cliente Android

A través del editor de *layouts* de Android Studio, se realizó el diseño de las interfaces gráficas de las actividades. En la Figura 2.54 se muestra un ejemplo de este editor. En la parte izquierdo superior de la pantalla se muestra la paleta de *widgets*[68], el cual permite mostrar un conjunto de *widgets* para que el desarrollador seleccione cualquier componente y lo arrastre hacia la ubicación deseada de la interfaz gráfica.

A continuación se muestra la implementación de la actividad `activity_configuracion_entrada_salida`, la cual permite mostrar al usuario diferentes opciones o métodos para registrar entrada y salida, ya sea por medio de huella

[66] ACL (Lista de Control de Acceso): permite configurar la aceptación o rechazo de información para determinadas direcciones IP
[67] Editor de *layouts* de Android Studio (Android Studio *Layout Editor*): es una herramienta de Android Studio que permite diseñar una interfaz gráfica para una determinada actividad.
[68] *Widget*: es un componente o elemento de la interfaz gráfica de usuario de Android.

dactilar, PIN o de forma automática (este último solo se encuentra disponible para el registro de salida). Todos los archivos de *layouts* se encuentran en el Anexo VI.

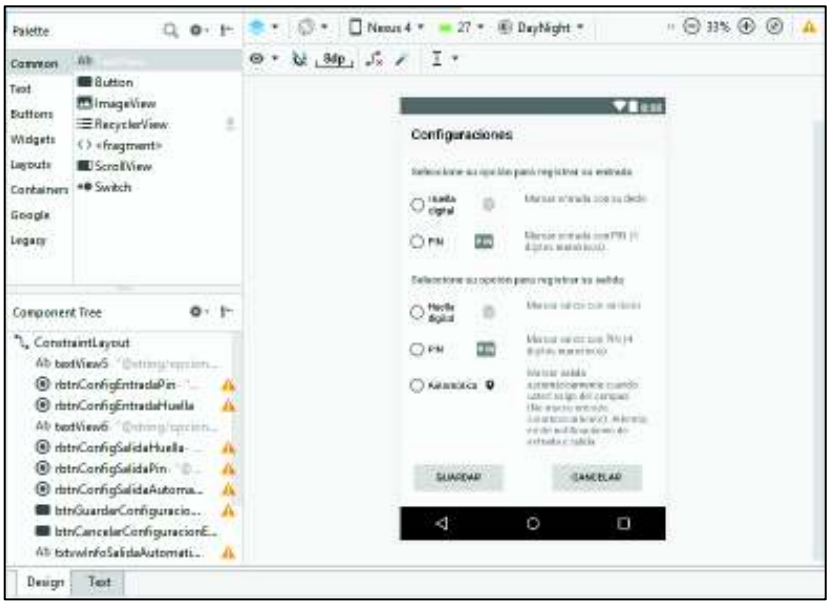


Figura 2.54.- Editor de *layouts* de Android Studio

En el Código 2.33 se presenta un fragmento del código fuente del *layout* `activity_configuracion_entrada_salida`. En las líneas 2 a 7 se define un `ConstraintLayout`, el cual permite simplificar la creación de *layouts* complejos en Android. Desde la línea 8 hasta la 22 se tienen todos los *widgets* del *layout* en cuestión. Cada uno de estos tienen la propiedad `android:id`, el cual establece el identificador único del componente. En la línea 8 se muestra el elemento `TextView`, el cual permite mostrar texto a través de la propiedad `android:text`. Es recomendable que cualquier texto que se vaya a mostrar se encuentre definido en el archivo de recursos `strings`. En la línea 9 se muestra el elemento `RadioButton`, el cual tiene la propiedad `android:onClick` que permite habilitar la escucha de eventos clic para el componente en cuestión. En la línea 10 se especifica la propiedad `android:drawableRight`, la cual permite agregar una pequeña imagen en el lado derecho del `RadioButton`. Todos los demás elementos especifican propiedades similares.

En el Código 2.34 se muestra una parte del archivo de recursos `strings`, el cual almacena todos los textos de los *widgets*. En la línea 3 se definen los campos clave/valor del mensaje para indicarle al usuario que escoja una opción de registro de entrada.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ConfiguracionEntradaSalidaActivity">
     <TextView android:id="@+id/textView5" android:text="@string/opcion_registro_entrada"        .../>
     <RadioButton android:id="@+id/rbtnConfigEntradaPin" android:onClick="onClickConfigEntradaPin"
        android:text="@string/forma_pin" android:drawableRight="@drawable/ic_pin".../>
     <RadioButton        .../>
     <TextView ...." />
     <RadioButton .../>
     <RadioButton .../>
     <RadioButton.../>
     <Button .... />
     <Button ... />
     <TextView ..../>
     <TextView ... />
     <TextView... />
     <TextView... />
     <TextView.../>
</android.support.constraint.ConstraintLayout>
```

Código 2.33.- Código fuente `activity_configuracion_entrada_salida`

```
<resources>
...
 <string name="opcion_registro_entrada">Seleccione su opción para registrar su entrada</string>
...
</resources>
```

Código 2.34.- Archivo `strings.xml`

En la Figura 2.55 se muestra el resultado de la implementación del *layout*. Se muestran las diferentes opciones de registro de entrada y salida, con sus respectivas descripciones e imágenes.

Figura 2.55.- Vista diseño `activity_configuracion_entrada_salida`

Lógica Cliente Android

En el Código 2.35 se muestra un fragmento del contenido del archivo `AndroidManifest.xml`. Las líneas 3 a 6 definen los permisos que requiere la aplicación para que funcione correctamente, se establecen que la aplicación requiere el uso de Internet, del sensor de huella digital y del GPS. En las líneas 8 a 10 se establece tanto el icono como el título y el estilo de la aplicación. Las líneas 11 a 13 especifican que la clase `ServicioGeofence` es un servicio que se ejecuta en segundo plano. Las líneas 14 a 21 definen que `LogeoActivity` va a tener una orientación vertical en su *layout* (línea 16), además es la actividad principal y va a ser lanzado primero cuando se abra la aplicación (líneas 18 y 19). El resto de actividades están definidos dentro del elemento `application`.

```
2.  <manifest...>
3.      <uses-permission android:name="android.permission.INTERNET" />
4.      <uses-permission android:name="android.permission.USE_FINGERPRINT" />
5.      <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
6.      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
7.      <application
8.         android:icon="@mipmap/ic_asistencia_dos"
9.         android:label="@string/app_name"
10.        android:theme="@style/Theme.AppCompat.DayNight">
11.         <service
12.            android:name=".ServicioGeofence"
13.            android:enabled="true" ... />
14.         <activity
15.            android:name=".LogeoActivity"
16.            android:screenOrientation="portrait">
17.             <intent-filter>
18.                 <action android:name="android.intent.action.MAIN" />
19.                 <category android:name="android.intent.category.LAUNCHER" />
20.             </intent-filter>
21.         </activity>
22.         <activity... />
23.         ...
24. </manifest>
```

Código 2.35.- Contenido del archivo `AndroidManifest.xml`

A continuación se explican las clases `Empleado`, `ManejoConfiguracionUsuario`, `Constantes`, `ManejadorPeticiones`, `ManejoDeHuellas`, `AutenticacionHuellaActivity`, `ServicioGeofence`. Todas estas clases se encuentran en el Anexo VI.

En el Código 2.36 se muestra la clase `Empleado`, la cual permite representar los datos recibidos del servicio asociados a empleados. En la línea 1 se indica que esta clase implementa la interfaz `Serializable`, lo que permite que todos los atributos de esta clase puedan ser serializados o de serializados. Las líneas 2 al 8 definen todos los atributos. Por

último, en las líneas 10 a 17 se especifican los métodos `GET` y `SET` de los atributos con la finalidad de poder manipular estos.

```
1.  public class Empleado implements Serializable{
2.      private int idEmpleado;
3.      private String nombre;
4.      private String apellido;
5.      private String numeroCedula ;
6.      private String telefono ;
7.      private byte[] imagen ;
8.      private String pin;
9.      //Getters y Setters
10.     public int getIdEmpleado() { return idEmpleado; }
11.     public void setIdEmpleado(int idEmpleado) {this.idEmpleado = idEmpleado;}
12.     public String getNombre() {return nombre;}
13.     public void setNombre(String nombre) {this.nombre = nombre;}
14.     public String getApellido() {return apellido;}
15.     public void setApellido(String apellido) {this.apellido = apellido;}
16.     public String getNumeroCedula() {return numeroCedula;}
17.     public void setNumeroCedula(String numeroCedula) {this.numeroCedula = numeroCedula;}
18.    ...
19. }
```

Código 2.36.- Clase `Empleado`

En el Código 2.37 se muestra una parte de la clase `ManejoConfiguracionUsuario`, la cual permite guardar/recuperar información de en un archivo de configuración. En las líneas 2 al 6 se define el método que permite guardar información de tipo `string` en el archivo en cuestión. En las líneas 3 y 4 se instancian las clases `SharedPreferences` y `SharedPreferences.Editor` para tener acceso al archivo de configuración. En la línea 5 se emplea el método `putString` para agregar un nuevo par clave/valor y se guardan los cambios (línea 6). Las líneas 7 al 9 definen el método que permite retornar un valor de tipo `string` dado una determinada clave por medio del método `getString`, además, se especifica que si no se encuentra dicha clave, se establece por defecto un valor con el caracter vacío (línea 9). El resto de métodos del código realizan un proceso similar, pero para guardar/recuperar valores tipo entero y booleano por medio de métodos `putInt`, `getInt`, `putBoolean` y `getBoolean`.

En el Código 2.38 se muestra una parte de la clase `Constantes`. Las líneas 3 al 4 permiten establecer el esquema y la autoridad de la dirección URI del servicio. En la línea 5 se concatena lo anterior con la dirección del recurso para completar la dirección URI que apunta hacia la operación del servicio para registrar la entrada de un empleado. Esto permite que cuando se requiera realizar algún cambio en la dirección IP del *host*, se lo pueda hacer fácilmente. El resto de direcciones URI fueron establecidas de manera similar. Por último, en las líneas 7 y 8 se establecen estáticamente textos de fallo y de espera.

```
1.  public class ManejoConfiguracionUsuario {
2.      public static void setString(String key, String value, Context context) {
3.          SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(context);
4.          SharedPreferences.Editor editor = preferences.edit();
5.          editor.putString(key, value);
6.          editor.commit();}
7.   public static String obtenerString(String key, Context context) {
8.          SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(context);
9.          return preferences.getString(key, "");}
10.     public static void setInt(String key, int value, Context context) {
11.         ...
12.         editor.putInt(key, value);
13.         editor.commit();}
14.   public static int obtenerInt(String key, Context context) {
15.         ...
16.         return preferences.getInt(key,0);}
17.     public static void setBool(...) {... }
18.     public static boolean obtenerBool(...) {...}}
```

Código 2.37.- Clase `ManejoConfiguracionUsuario`

```
1.  public  class Constantes {
2.  ...
3.      public  static final String dominioUrl=
4.  "http://192.168.137.1:8094/ServicioControlAsistencia/ServicioControlAsistencia.svc";
5.  public static final String urlRegistrarEntradaEmpleado=dominioUrl+"/ControlAsistencia/RegistrarEntrada";
6.  ...
7.      public static final String mensajeErrorServicio="No se puede conectarse con el servidor";
8.      public static final String mensajeDialogoProgreso="Por favor, espere...";
9.  ...
10. }
```

Código 2.38.- Clase `Constantes`

Para que el Cliente Android pueda consumir las operaciones establecidas por el servicio WCF, se emplea la clase `ManejadorPeticiones` (ver Código 2.39). Esta clase se encarga de serializar y deserializar objetos y de enviar/recibir peticiones/respuestas. Dentro del constructor (líneas 3 al 6) se instancian un cliente HTTP (línea 4), una cola de peticiones (línea 5) y se inicializa el contexto (línea 6). Además, se especifican los métodos `registrarEntradaEmpleado` y `crearObjetoJson`, los cuales se explicarán más adelante.

El Código 2.40 muestra el método `crearObjetoJson`. Este método recibe como parámetro una cadena de caracteres en formato JSON y lo convierte en un objeto `JSONObject` (líneas 3 y 4). Por último, se controlan excepciones en la línea 5.

El método `registrarEntradaEmpleado` permite consumir el servicio para registrar entrada. Debido a la extensión del método, se lo divide en tres partes.

```
1.  public class ManejadorPeticiones {
2.  ...
3.   public ManejadorPeticiones(Context contexto) {
4.          mHttpClient = new DefaultHttpClient();
5.          mQueue = Volley.newRequestQueue(contexto, new HttpClientStack(mHttpClient));
6.          this.contexto = contexto;
7.      }
8.  ...
9.   //Método que consume el servicio de registro de entrada
10.     public void registrarEntradaEmpleado(int idEmpleado) {...}
11.      //Método para crear un objeto Json
12.     private JSONObject crearObjetoJson(String cadenaJson) {...}
13. ...
14. }
```

Código 2.39.- Clase `ManejadorPeticiones`

```
1.   private JSONObject crearObjetoJson(String cadenaJson) {
2.          JSONObject obj = null;
3.          try {obj = new JSONObject(cadenaJson);
4.          return obj;
5.          } catch (Throwable t) {Log.e(TAG, "Error conversion JSONObject ", t); }
6.          return obj;
7.      }
```

Código 2.40.- Método `crearObjetoJson` de la clase `ManejadorPeticiones`

La primera parte se lo muestra en el Código 2.41. Aquí se inicializa un objeto `Empleado` (líneas 3 y 4) y se lo serializa por medio del método `toJson` del objeto `Gson` (línea 5). En la línea 6 se convierte lo anterior en un objeto `JSONObject`.

```
1.      public void registrarEntradaEmpleado(int idEmpleado) {
2.          final Gson gson = new Gson();
3.          Empleado miEmpleado = new Empleado();
4.          miEmpleado.setIdEmpleado(idEmpleado);
5.          String empleadoSerializado = gson.toJson(miEmpleado);
6.          JSONObject jsonObject = crearObjetoJson(empleadoSerializado);
7.  ...
```

Código 2.41.- Método `registrarEntradaEmpleado` de la clase `ManejadorPeticiones` (Parte 1)

La segunda parte del método `registrarEntradaEmpleado` se muestra en el Código 2.42. En las líneas 8 y 9 se crea una petición especificando el método HTTP `POST` , la dirección URI y el objeto JSON serializado de la primera parte. Las líneas 12 al 22 definen lo que sucede cuando se reciba una respuesta satisfactorio del servicio. En las líneas 13 y 14 se guardan las *cookies* enviadas por el servicio. En la línea 16 se deserializa la respuesta del servicio. Se muestran los resultados en la actividad que realizó la llamada al método en cuestión (líneas 17 al 21). Y se muestran mensajes de error cuando el cliente no pudo contactarse con el servicio (líneas 23 al 28).

Código 2.42.- Método `registrarEntradaEmpleado` de la clase `ManejadorPeticiones` (Parte 2)

La tercera y última parte del método `registrarEntradaEmpleado` se lo muestra en el Código 2.43. En la línea 30 se envía la petición creada anteriormente y por último, en la línea 31 se muestra una animación de espera en la actividad que llamó al método en cuestión, la cual se detendrá cuando se reciba una respuesta del servicio.

```
29.  ...
30.           mQueue.add(registrarEntrada);
31.          mostrarProgresoDialogo(contexto, Constantes.mensajeDialogoProgreso);
32.      }
```

Código 2.43.- Método `registrarEntradaEmpleado` de la clase `ManejadorPeticiones` (Parte 3)

En el Código 2.44 se muestra una parte del método `onCreate` de la actividad `AutenticacionHuellaActivity`. En las líneas 6 al 8 se instancian objetos de las clases `KeyguardManager`, `FingerprintManager` y `PackageManager` con la finalidad de poder manejar el servicio de desbloqueo de pantalla, el servicio de autenticación de huella y paquetes de la aplicación. En la línea 9 se debe verificar si se tiene un método de bloqueo de pantalla habilitada, se debe verificar si se tiene permiso para emplear el sensor de huella digital y se debe verificar si existen huellas dactilares enroladas en el dispositivo móvil. En la línea 10 se genera una clave criptográfica. En la línea 12 se instancia un cifrador. Las líneas 12 al 25 solo se podrán ejecutar cuando el cifrador se haya inicializado correctamente. En la línea 13 se genera un `cryptoObject` el cual posibilita la autenticación de huellas. En la línea 14 se instancia un objeto de la clase

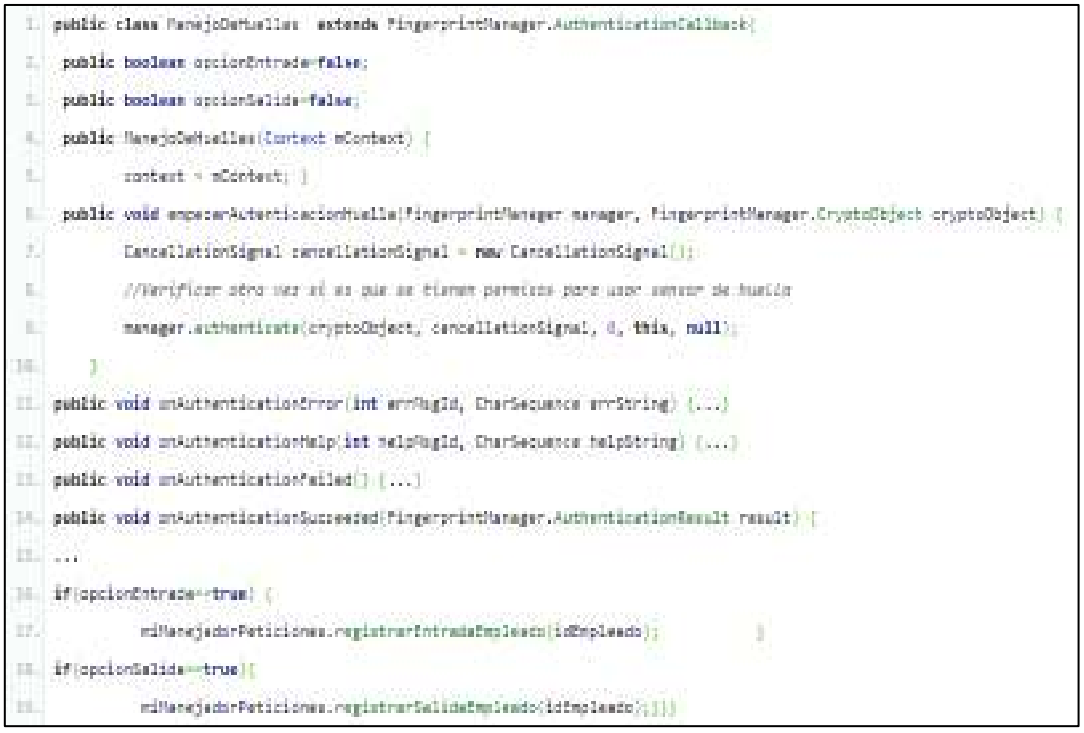
ManejoDeHuellas, el cual se lo explica en el Código 2.45. En las líneas 15 y 16 se obtiene el identificador único del empleado enviado desde la actividad anterior al actual. Las líneas 17 al 22 permiten inicializar variables auxiliares que indican si el usuario escogió la opción de registro de entrada o de salida. Por último, en la línea 24 se llama al método que inicia la autenticación de huellas.

```
3.     protected void onCreate(Bundle savedInstanceState) {
4.         ....
5.  //Se instancian objetos FingerprintManager y KeyguardManager
6.         KeyguardManager keyguardManager = (KeyguardManager) getSystemService(KEYGUARD_SERVICE);
7.         FingerprintManager fingerprintManager = (FingerprintManager) getSystemService(FINGERPRINT_SERVICE);
8.         packageManager=getPackageManager();
9.         //Verificaciones varias...
10.        generateKey();
11.        //Iniciar el cifrador.
12.        if (cipherInit()) {
13.            FingerprintManager.CryptoObject cryptoObject = new FingerprintManager.CryptoObject(cipher);
14.            ManejoDeHuellas manejadorHuellas = new ManejoDeHuellas(this);
15.            Intent intent= getIntent();
16.            int idEmpleado= intent.getExtras().getInt("idEmpleado");
17.        if(opcionEntrada==true){
18.            manejadorHuellas.opcionEntrada=true;
19.             manejadorHuellas.opcionSalida=false;
20.        }else {
21.            manejadorHuellas.opcionEntrada=false;
22.            manejadorHuellas.opcionSalida=true; }
23.            manejadorHuellas.idEmpleado=idEmpleado;
24.            manejadorHuellas.empezarAutenticacionHuella(fingerprintManager, cryptoObject);}
25.      }//Fin onCreate
```

Código 2.44.- Actividad AutenticacionHuellaActivity

La clase que maneja la lógica de lo que sucede luego de una autenticación exitosa o de fallo es ManejoDeHuellas y se lo presenta en el Código 2.45. Las líneas 2 y 3 definen variables auxiliares que indican si el usuario eligió registrar una entrada o una salida. Las líneas 9 al 10 definen el método que permite realizar la autenticación. En la línea 7 se instancia un objeto CancellationSignal el cual permite cancelar el proceso en cuestión. En la línea 9 se empieza el proceso en cuestión. La clase ManejoDeHuellas implementa la interfaz FingerprintManager.AuthenticationCallback, por lo que debe implementar los métodos listados en las líneas 11 a la 14, las cuales permiten mostrar mensajes de error, ayuda, fallo y éxito de la autenticación. En las líneas 16 al 19 se consume el servicio para registrar la entrada o la salida según la opción que haya elegido el usuario, cuando este se haya autenticado correctamente.

La actividad ModulosActivity ofrece al usuario un menú de todos los módulos que tiene la aplicación. Además, aquí se ejecuta el monitoreo de geovallas para registrar la salida automática, porque es la primera actividad que aparece cuando el usuario logre ingresar a la aplicación.

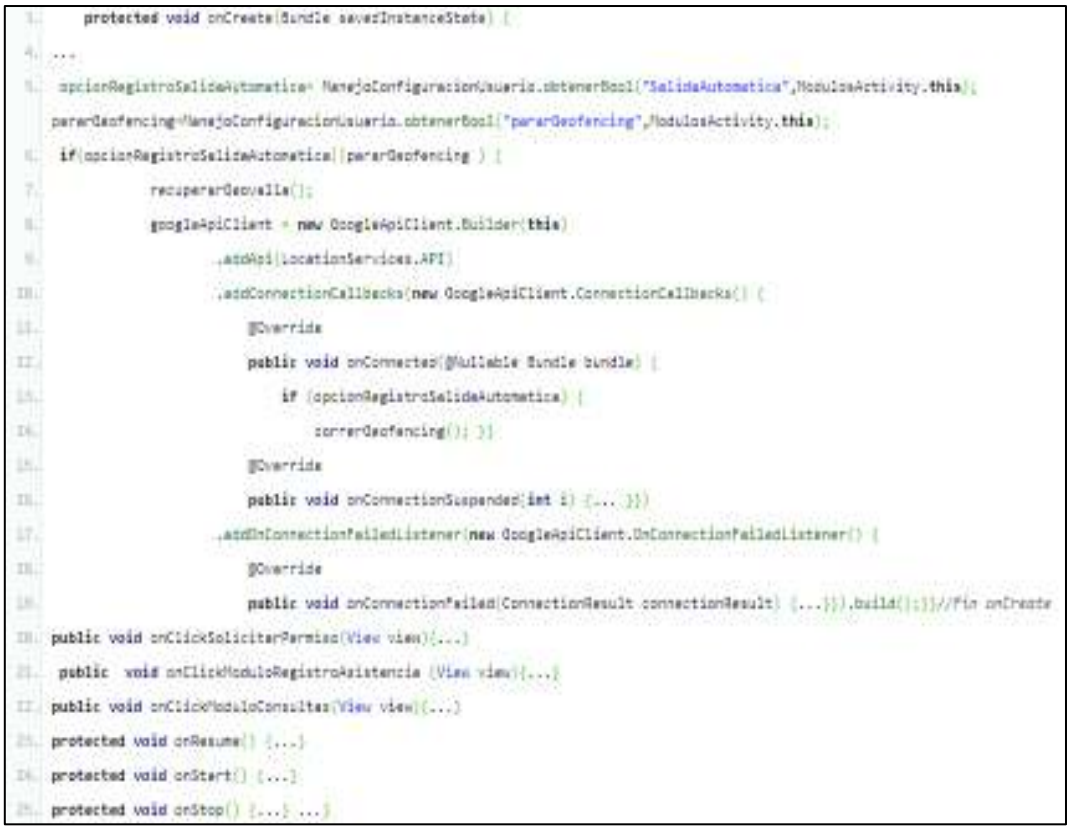


```
1.  public class ManejoDeHuellas  extends FingerprintManager.AuthenticationCallback{
2.   public boolean opcionEntrada=false;
3.   public boolean opcionSalida=false;
4.   public ManejoDeHuellas(Context mContext) {
5.         context = mContext; }
6.   public void empezarAutenticacionHuella(FingerprintManager manager, FingerprintManager.CryptoObject cryptoObject) {
7.         CancellationSignal cancellationSignal = new CancellationSignal();
8.         //Verificar otra vez si es que se tienen permisos para usar sensor de huella
9.         manager.authenticate(cryptoObject, cancellationSignal, 0, this, null);
10.     }
11. public void onAuthenticationError(int errMsgId, CharSequence errString) {...}
12. public void onAuthenticationHelp(int helpMsgId, CharSequence helpString) {...}
13. public void onAuthenticationFailed() {...}
14. public void onAuthenticationSucceeded(FingerprintManager.AuthenticationResult result) {
15. ...
16. if(opcionEntrada==true) {
17.          mManejadorPeticiones.registrarEntradaEmpleado(idEmpleado);          }
18. if(opcionSalida==true){
19.          mManejadorPeticiones.registrarSalidaEmpleado(idEmpleado);}}}
```

Código 2.45.- Clase `ManejoDeHuellas`

A continuación, en el Código 2.46 se muestra un fragmento del método `onCreate` de la actividad `ModulosActivity`. En la línea 5 se recupera un valor booleano del archivo de configuraciones de usuario, el cual indica si el usuario seleccionó la opción de registro de salida automática. También se recupera otro valor booleano que indica si se debe parar o no el monitoreo de geovallas. Las líneas 6 al 19 definen la lógica cuando el usuario quiera registrar su salida automáticamente. En la línea 7 se busca información de la geovalla única del sistema. En las líneas 8 al 19 se instancia el cliente de Google para que se tenga acceso a servicios de ubicación. En las líneas 12 al 14 se establece que cuando se conecte el cliente de Google, se empieza a correr la actualización de ubicación y monitoreo de geovallas. Las líneas 16 al 19 definen la lógica cuando se suspenda y falle la conexión con el cliente. Las líneas 20 al 22 permiten al usuario ingresar a las actividades para solicitar permiso (línea 20), registrar asistencia (línea 21) y consultar registros (línea 22). Las líneas 23 al 25 permiten controlar la reconexión/desconexión con el cliente de Google.

En el Código 2.47 se presenta la primera parte del método `empezarMonitoreoGeofence`. En las líneas 2 al 7 se guarda toda la información recuperada de la geovalla. En la línea 8 se realizan verificaciones de permiso para el acceso a Internet y al GPS.



Código 2.46.- Actividad `ModulosActivity`

```
1.  private void empezarMonitoreoGeofence(){
2.          String idGeovalla= geovallaGuardado.getIdGeovalla();
3.          double latitud= geovallaGuardado.getLatitud();
4.          double longitud= geovallaGuardado.getLongitud();
5.          double radio=geovallaGuardado.getRadio();
6.          int tiempoRespuesta= geovallaGuardado.getTiempoRespuesta();
7.          float radioFloat= (float) radio;
8.        //Se pide permisos
```

Código 2.47.- Método `empezarMonitoreoGeofence` (Parte 1)

La segunda parte del método `empezarMonitoreoGeofence` se presenta en el Código 2.48. Aquí, en las líneas 10 al 16 se inicializa una geovalla con los valores previamente recuperados de la base de datos. En las líneas 17 al 19 se instancia una petición de monitoreo para la geovalla previamente creada, y en las líneas 20 al 22 se llama al servicio `ServicioGeofence`.

La última parte del método `empezarMonitoreoGeofence` se encuentra en el Código 2.49. En las líneas 27 al 32 se agrega la geovalla y se empieza a ejecutar el monitoreo de *geofencing*. Las líneas 31 y 32 muestran mensajes sobre el estado del proceso en cuestión. Y por último, en la línea 33 se controlan excepciones.

```
9.      try {
10.         Geofence geofence = new Geofence.Builder()
11.                 .setRequestId(idGeovalla)
12.                 .setCircularRegion(latitud, longitud, radioFloat)
13.                 .setExpirationDuration(Geofence.NEVER_EXPIRE)
14.                 .setNotificationResponsiveness(tiempoRespuesta)
15.                 .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER | Geofence.GEOFENCE_TRANSITION_EXIT)
16.                 .build();
17.         GeofencingRequest geofencingRequest = new GeofencingRequest.Builder()
18.                 .setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER)
19.                 .addGeofence(geofence).build();
20.         Intent intent = new Intent(this, ServicioGeofence.class);
21.         intent.putExtra("idEmpleado",idEmpleado );
22.         PendingIntent pendingIntent= PendingIntent.getService(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
```

Código 2.48.- Método `empezarMonitoreoGeofence` (Parte 2)

```
23.          //Asegurarse de que se encuentra conectado con el servicio de Google
24.          if(!googleApiClient.isConnected()){
25.              ...
26.          }else {
27.              LocationServices.GeofencingApi.addGeofences(googleApiClient, geofencingRequest,pendingIntent)
28.                      .setResultCallback(new ResultCallback<Status>() {
29.                          @Override
30.                          public void onResult(Status status) {
31.                              if(status.isSuccess()){ ...
32.                              }else {... }}});}
33.      }catch (SecurityException e){...}}
```

Código 2.49.- Método `empezarMonitoreoGeofence` (Parte 3)

Se emplea la clase `ServicioGeofence` para determinar la lógica de lo que suceda después de que ocurra una transición en la geovalla. A continuación se explica esta clase en el Código 2.50. En la línea 6 se recupera el identificador único del empleado. En las líneas 8 y 9 se inicializa un empleado con el identificador obtenido previamente. En la línea 10 se recupera el evento disparado por la geovalla a través del `intent`. En la línea 14 se obtiene el tipo de transición de la geovalla en cuestión. Se obtiene la geovalla disparada (línea 16). En la línea 19 se consume la operación del servicio para registrar una notificación de entrada de geovalla. Finalmente, en las líneas 22 y 23 se consume la operación para registrar la salida de forma automática y se muestra en la pantalla del dispositivo móvil una notificación acerca del registro en cuestión.

```
1.  public class ServicioGeofence extends IntentService {
2.      ...
3.      @Override
4.      protected void onHandleIntent(Intent intent) {
5.          ManejoNotificaciones notificador= new ManejoNotificaciones(this);
6.          int idEmpleado= intent.getExtras().getInt("idEmpleado");
7.          ManejadorPeticiones miManejador= new ManejadorPeticiones(this);
8.          Empleado miEmpleado= new Empleado();
9.          miEmpleado.setIdEmpleado(idEmpleado);
10.         GeofencingEvent event= GeofencingEvent.fromIntent(intent);
11.         if(event.hasError()){
12.           ...
13.           }else {
14.             int transicion= event.getGeofenceTransition();
15.             List<Geofence> geofences = event.getTriggeringGeofences();
16.             Geofence geofence= geofences.get(0);
17.             String requestId= geofence.getRequestId();
18.             if(transicion==Geofence.GEOFENCE_TRANSITION_ENTER){
19.                 miManejador.registrarNotificacionGeovallaEntrada(miEmpleado);
20.             }else if(transicion== Geofence.GEOFENCE_TRANSITION_EXIT){
21.                 Log.d(TAG, "Saliendo geofence "+ requestId);
22.                 mostrarNotificacion("Salida","Saliendo del campus");
23.                 miManejador.registrarSalidaAutomatica(miEmpleado);}}}}
```

Código 2.50.- Clase `ServicioGeofence`

Cliente Gestor

El Cliente Gestor fue desarrollado en el entorno de desarrollo Visual Studio 2013. La lógica de este Cliente fue realizada en el lenguaje de programación C#. La aplicación del Cliente Gestor se encuentra en el Anexo VII.

Interfaz gráfica del Cliente Gestor

Para diseñar la interfaz visual de los formularios del Cliente Gestor se emplearon el Windows *Forms Designer*[69] y *Toolbox*[70], los cuales permiten al desarrollador seleccionar un determinado componente visual del *Toolbox* (parte izquierda de la Figura 2.56) y arrastrarlos a la ubicación deseada dentro del formulario.

Se creó un formulario de tipo MDI[71] llamado `FrmMenuPrincipal`, el cual permite al usuario ingresar a todos los módulos de gestión de la aplicación y permite contener el resto de formularios,el cual se lo presenta en la Figura 2.57.

En la Figura 2.58 se muestra un ejemplo de interfaz gráfica que permite buscar registros de asistencias. Se emplean elementos gráficos como `RadioButton`, `TextBox`, `Label`,

[69] Windows *Forms Designer*: es una herramienta que permite el diseño de los formularios de Visual Studio
[70] *Toolbox*: Ofrece: es una herramienta que ofrece al desarrollador una lista de componentes visuales
[71] MDI (*Multiple Document Interface*): es una propiedad que permite a un formulario contener a otros formularios.

`DateTimePicker` y `Button` para seleccionar opciones, mostrar/ingresar texto, seleccionar fecha/hora y para realizar una acción respectivamente. Para mostrar los resultados de cualquier búsqueda que se realice, se emplea el control `ListBox`. El resto de interfaces gráficas emplean elementos gráficos similares.

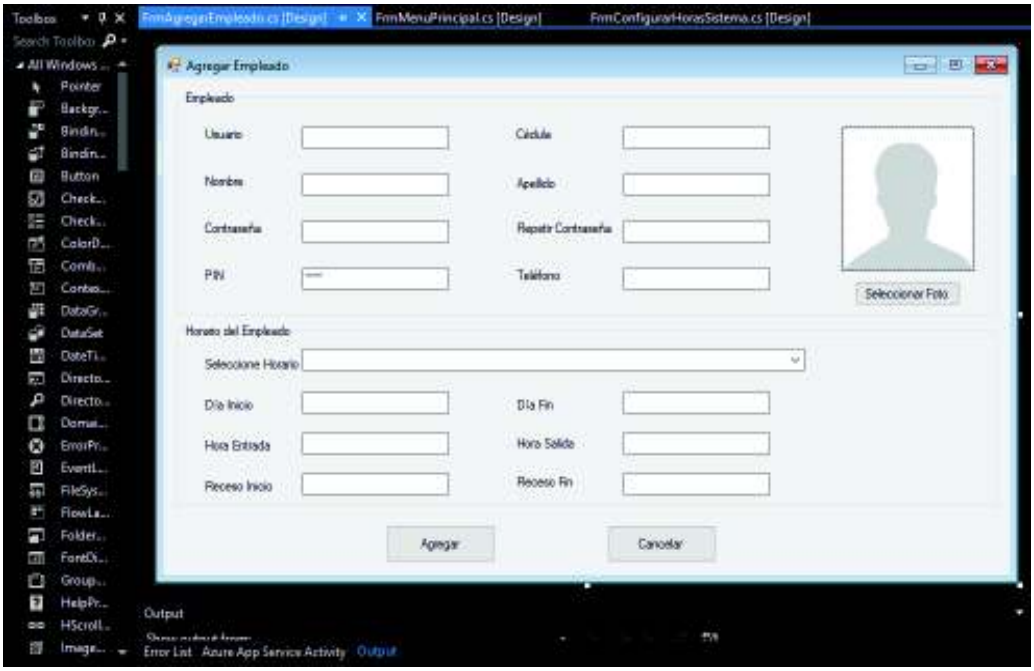


Figura 2.56.- Windows *Forms Designer* y *Toolbox*

Figura 2.57.- Interfaz gráfica de `FrmMenuPrincipal`

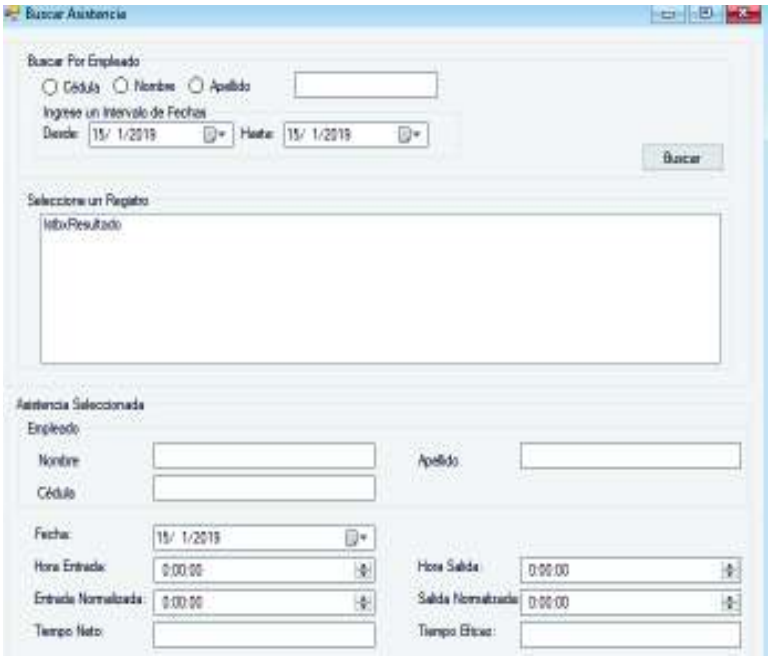


Figura 2.58.- Interfaz gráfica del formulario `FrmBuscarAsistencia`

Lógica del Cliente Gestor

A continuación se explica la lógica para buscar registros de asistencia. En el formulario `FrmBuscarAsistencia` el usuario ingresa información del empleado y un intervalo de fechas para realizar la búsqueda de registros de asistencias asociados al empleado.

En el Código 2.51, se muestra el método `onClick` del botón de búsqueda del formulario en cuestión . En las líneas 3 al 9 se indica lo que sucede cuando el usuario escoge la búsqueda de registros por cédula del empleado. En la línea 5 se obtiene la cédula ingresada por el usuario. En las líneas 6 y 7 se llama al método `BuscarRegistrosAsistenciaPorCedulaEmpleado` con la finalidad de obtener una lista de registros de asistencias perteneciente al empleado y que se encuentre dentro del intervalo de fechas especificada. En la línea 9 se muestran los resultados en un `ListBox`. Finalmente, un similar proceso se hace para el resto de líneas de código, sin embargo, bajo el criterio de búsqueda de registros por nombre y por apellido del empleado como se muestran en las líneas 10 al 15.

En el Código 2.52 se muestra una parte del método `SelectedIndexChanged` de la lista visual que muestra los resultados de búsqueda de registros de asistencias. Este método permite manejar el evento que se produce cuando el usuario seleccione un ítem de la lista. En la línea 3 se obtiene el registro de asistencia seleccionado. En la línea 4 se verifica que lo anterior no esté vacío. En las líneas 5 y 6 se obtiene toda la información del empleado asociado al registro en cuestión. En las líneas 7 al 9 se muestran en los componentes

visuales los datos del empleado. Y por último, en las líneas 11 al 14 se muestran los detalles del registro en cuestión en los elementos visuales.

```
1.  private void btnBuscar_Click(object sender, EventArgs e)
2.          {
3.              if (this.rbtnCI.Checked)
4.              {           ...
5.                      string ci = this.txtBusqueda.Text;
6.                      listaAsistenciasEncontradas =
7.          servicio.BuscarRegistrosAsistenciaPorCedulaEmpleado(ci, desdeFecha, hastaFecha);
8.                      //Verificar resultado no vacío ...
9.                          mostrarListaAsistenciaEncontrada(listaAsistenciasEncontradas);}

10.             if (this.rbtNombre.Checked)
11.             {     ...
12.                     listaAsistenciasEncontradas =
13.             servicio.BuscarRegistrosAsistenciaPorNombreEmpleado(nombre, desdeFecha, hastaFecha);
14.                     mostrarListaAsistenciaEncontrada(listaAsistenciasEncontradas); }
15.             if (this.rbtnApellido.Checked) {...}
16. }
```

Código 2.51.- Método `onClick` de `btnBuscar`

```
1.  private void lstbxResultado_SelectedIndexChanged(object sender, EventArgs e)
2.          {
3.              RegistroAsistencia registro = (RegistroAsistencia)this.lstbxResultado.SelectedItem;
4.              if (registro != null){
5.                  Empleado empleadoRelacionadoAsistencia =
6.              servicio.BuscarEmpleadoPorIdRegistroAsistencia(registro.idRegistroAsistencia);
7.                  this.txtNombre.Text = empleadoRelacionadoAsistencia.nombre;
8.                  this.txtApellido.Text = empleadoRelacionadoAsistencia.apellido;
9.                  this.txtCedula.Text = empleadoRelacionadoAsistencia.numeroCedula;
10.
11.                 this.dtpFecha.Value = registro.fechaHoraEntrada;
12.                 this.dtpHoraEntrada.Value = registro.fechaHoraEntrada;
13.                 this.dtpHoraSalida.Value = registro.fechaHoraSalida;
14.                 this.dtpEntradaNormalizada.Value = registro.fechaHoraEntradaNormalizada;
15.                ...}}
```

Código 2.52.- Método `SelectedIndexChanged` de `lstbxResultado`

Para consumir las operaciones expuestas por el servicio WCF en el Cliente Gestor se emplea la clase `ServicioControlAsistenciaProxy`. A continuación se explica el método `BuscarRegistrosAsistenciaPorCedulaEmpleado` de esta clase. El resto de métodos emplean una lógica similar y se encuentran en el Anexo VII.

En el Código 2.53 se muestra la primera parte de este método. En las líneas 5 al 7 se crea una cadena de caracteres que contiene la dirección URI con sus respectivos parámetros para la operación de buscar registros de asistencia por cédula del empleado. En la línea 8 se crea una petición con soporte de *cookies*. En las líneas 9 al 11 se especifica que la petición emplea el método HTTP `GET` y que contiene información en formato JSON. En la línea 13 se envía la petición y se obtiene la respuesta. En las líneas 14 y 15 se convierte

la respuesta de un flujo de bytes a formato JSON. Y en la línea 17 se cierra la conexión con el servicio.

```
1  public List<RegistroAsistencia> BuscarRegistrosAsistenciaPorCedulaEmpleado(string ci, string desdeFecha, string hastaFecha)
2          {
3              var serializador = new JavaScriptSerializer();
4              //Se crea la solicitud
5              string url = Constantes.urlBuscarRegistroAsistenciaPorCedulaEmpleado + ci;
6              url += Constantes.urlDesdeFecha + desdeFecha;
7              url += Constantes.urlHastaFecha + hastaFecha;
8              var request = CookiedRequestFactory.CreateHttpWebRequest(url);
9              request.Method = "GET";
10             request.ContentType = "application/json";
11             request.Accept = "application/json";
12
13             var respuesta = (HttpWebResponse)request.GetResponse();
14             var reader = new StreamReader(respuesta.GetResponseStream());
15             var jsonResponseString = reader.ReadToEnd();
16             reader.Close();
17             respuesta.Close();
18 ...
```

Código 2.53.- Método `BuscarRegistrosAsistenciaPorCedulaEmpleado` (Parte 1)

En el Código 2.54 se muestra la continuación del método `BuscarRegistrosAsistenciaPorCedulaEmpleado`. En la línea 19 se convierte la respuesta del servicio a un objeto `JArray`. En la línea 20 se instancia una lista que guardará los resultados. En las líneas 22 al 35 se deserializada cada objeto JSON que se encuentre dentro de la lista JSON. Por motivo a que no se puede deserializar automáticamente un tipo de dato JSON que represente un intervalo de tiempo, se emplea el método `ConvertirJsonTimespan` como se lo muestra en las líneas 33 y 34. En la línea 35 se guarda en la lista el registro deserializado. Finalmente, en la línea 36 se retorna la lista de asistencias.

Como se explicó anteriormente, para serializar/deserializar tipos de datos que representen un intervalo de tiempo, la clase empleada `JavaScriptSerializer` no lo realiza de manera automática. Por lo que se optó por implementar métodos que permitan realizar tales acciones. A continuación se detallan algunos de estos.

En el Código 2.55 se muestra el método `ConvertirJsonTimespan` el cual permite convertir una cadena de caracteres que representa una duración de tiempo en formato JSON al tipo de dato `Timespan`. Para ello, en la línea 2, se emplea el método `ToTimeSpan` de la clase `XmlConvert` para llevarlo a cabo.

El modo de representación de datos JSON en sí no define el formato para representar fechas y horas por lo que deja libre al desarrollador escoger el estándar para representar

estos. En este caso, se pudo observar que el servicio WCF implementado enviaba respuestas de intervalos de tiempo en notación ISO 8601[72]

En el Código 2.56 se muestra el método que permite convertir un dato de tipo `TimeSpan` a cadena de caracteres JSON en notación ISO 8610.

```
...
            JArray jArray = JArray.Parse(jsonRespuestaString);
            List<RegistroAsistencia> listaAsistencias = new List<RegistroAsistencia>();

            foreach (JObject item in jArray)
            {
                RegistroAsistencia registroAsistencia = new RegistroAsistencia();
                int idRegistro = Convert.ToInt32(item.GetValue("idRegistroAsistencia").ToString());
  //Se convierte fecha y hora de formato JSON a Datetime
                registroAsistencia.idRegistroAsistencia = idRegistro;
                registroAsistencia.fechaHoraEntrada = Convert.ToDateTime(item.GetValue("fechaHoraEntrada").ToString());
                registroAsistencia.fechaHoraSalida = Convert.ToDateTime(item.GetValue("fechaHoraSalida").ToString());
                registroAsistencia.fechaHoraEntradaNormalizada =
Convert.ToDateTime(item.GetValue("fechaHoraEntradaNormalizada").ToString());
                registroAsistencia.fechaHoraSalidaNormalizada =
Convert.ToDateTime(item.GetValue("fechaHoraSalidaNormalizada").ToString());
                 //Se convierte el formato de duración ISO8601 a tipo TimeSpan
                registroAsistencia.tiempoTrabajoNeto = ConvertirJsonTimespan(item.GetValue("tiempoTrabajoNeto").ToString());
                registroAsistencia.tiempoTrabajoEficaz = ConvertirJsonTimespan(item.GetValue("tiempoTrabajoEficaz").ToString());
                listaAsistencias.Add(registroAsistencia);}
            return listaAsistencias;}
```

Código 2.54.- Método `BuscarRegistrosAsistenciaPorCedulaEmpleado` (Parte 2)

```
1.      public TimeSpan ConvertirJsonTimespan(string duracionISO8601) {
2.                  TimeSpan ts = XmlConvert.ToTimeSpan(duracionISO8601);
3.                  return ts;
4.              }
```

Código 2.55.- Método `ConvertirJsonTimespan`

```
5.              public string ConvertirTimespanAJson(TimeSpan ts) {
6.                  string duracionISO8601 = XmlConvert.ToString(ts);
7.                  return duracionISO8601;
8.              }
```

Código 2.56.- Método `ConvertirTimespanAJson`

[72] ISO 8601: es una norma que especifica la notación estándar de fechas, horas, instantes e intervalos de tiempo, con la finalidad de evitar ambigüedades.

Establecimiento del ambiente de pruebas

Para establecer una red inalámbrica de pruebas se optó por configurar una computadora portátil con Sistema Operativo Windows 10 en una zona con cobertura[73] para que este actúe como enrutador y permita conectividad entre los dispositivos que se asocien a esta red inalámbrica.

El principal motivo por el que se empleó la zona de cobertura es que permite que el servidor IIS exponga el servicio de control de asistencia en una dirección IP estática. A continuación se detallan algunos pasos para configurar la zona con cobertura.

Se accede a la conexión de red inalámbrica que se encuentra en la barra de herramientas de Windows como se muestra en la **Figura 2.59**.

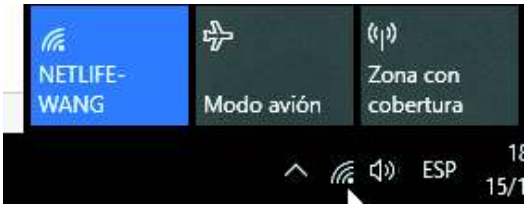


Figura 2.59.- Ventana de conexiones inalámbricas

Luego se da clic derecho sobre “Zona con cobertura” y después seleccionar “Ir a la Configuración” y se muestra una ventana como en la Figura 2.60.

Por último, se ingresa el nombre de la red y su correspondiente contraseña como se observa en la Figura 2.61. Con esto ya se encuentra configurado la zona con cobertura.

Figura 2.60.- Ventana configuraciones de la zona con cobertura

[73] Zona con cobertura: Transforma un computador/teléfono inteligente en un enrutador permitiendo compartir el acceso a Internet con otros dispositivos.

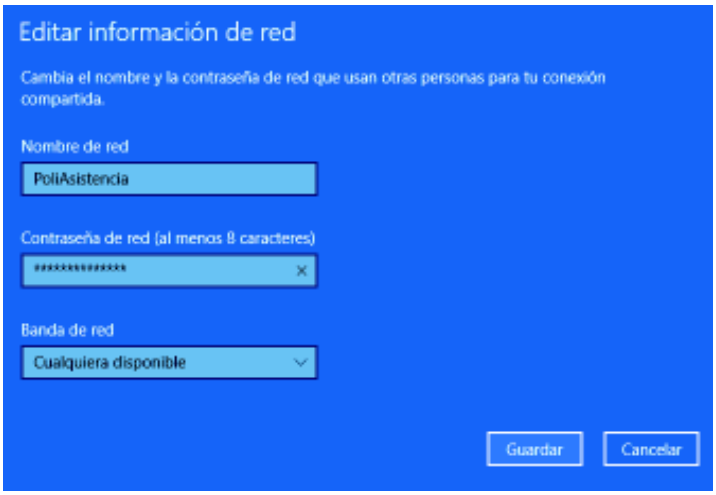


Figura 2.61.- Editar información de red

La dirección de red del ambiente de pruebas es 192.168.137.0/24 y las direcciones IP de los diferentes dispositivos se encuentran dentro de esa red, como se muestra en la Tabla 2.8.

Tabla 2.8.- Direcciones IP del ambiente de pruebas

Dispositivo	Dirección IP/Máscara
Servidor	192.168.137.1/24
Dispositivo Android	Asignado por DHCP[74] en la zona de cobertura
Computadora portátil	Asignado por DHCP en la zona de cobertura

Instalación del Cliente Android

Ya terminado la codificación del Cliente Android, se procedió a generar el archivo de extensión `.apk` y se lo instaló en un dispositivo Samsung Galaxy S6 con Android 7.0. El instalador del Cliente Android se encuentra en el Anexo VIII.

A continuación se muestran ejemplos de las interfaces gráficas.

En la Figura 2.62 se muestra la actividad `LogeoActivity`, la cual permite autenticar al usuario.

En la Figura 2.63 se muestra la actividad `ModulosActivity`, la cual permite ingresar a los módulos de registro, de consultas, de solicitar permiso y de configuraciones.

[74] DHCP(*Dynamic Host Configuration Protocol*): permite asignar direcciones IP a dispositivos de manera dinámica.

Figura 2.62.- Actividad `LogeoActivity`

Figura 2.63.- Actividad `ModulosActivity`

En la Figura 2.64 se presenta la actividad `ConfiguracionEntradaSalidaActivity`, la cual permite configurar opciones de registro de entrada/salida.

Figura 2.64.- Actividad `ConfiguracionEntradaSalidaActivity`

En la Figura 2.65 se presenta la actividad `ModuloRegistroAsistenciaActivity`, la cual permite registrar entrada/salida.

Figura 2.65.- Actividad `ModuloRegistroAsistenciaActivity`

En la Figura 2.66 se muestra la actividad `AutenticacionHuellaActivity`, la cual permite registrar la entrada/salida por huella dactilar.

Figura 2.66.- Actividad `AutenticacionHuellaActivity`

En la Figura 2.67 se muestra la actividad `ConsultasIndiceActivity`, la cual permite mostrar una lista de consultas que se pueden realizar.

Figura 2.67.- Actividad `ConsultasIndiceActivity`

En la Figura 2.68 se muestra la actividad `ConsultaTotalHorasActivity`, la cual permite consultar el total de horas trabajadas de un determinado intervalo de tiempo.

Figura 2.68.- Actividad `ConsultaTotalHorasActivity`

La actividad `SolicitarPermisoPorDiasYHorasActivity` se muestra en la Figura 2.69 y permite al usuario pedir permiso por un intervalo de tiempo.

Figura 2.69.- Actividad `SolicitarPermisoPorDiasYHorasActivity`

Instalación del Cliente Gestor

Después de codificar el Cliente Gestor, se generó el instalador del Cliente Gestor y se procedió a realizar la instalación como se muestra en la Figura 2.70. El instalador de esta aplicación se encuentra en el Anexo IX.

Figura 2.70.- Instalador del Cliente Gestor

A continuación se presentan ejemplos de las interfaces gráficas del Cliente Gestor.

En la Figura 2.71 se presenta el formulario `FrmMenuPrincipal`, el cual permite acceder a diferentes módulos de gestión.

Figura 2.71.- Formulario `FrmMenuPrincipal`

En la Figura 2.72 se muestra el formulario `FrmAgregarEmpleado`, el cual permite agregar un nuevo empleado con su respectivo horario de trabajo y credencial.

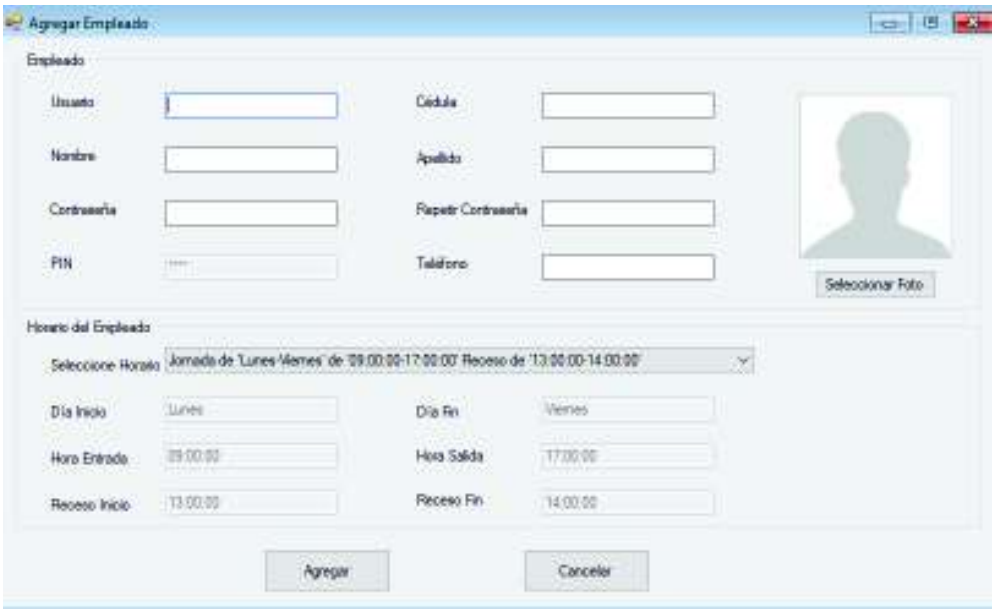


Figura 2.72.- Formulario `FrmAgregarEmpleado`

En la Figura 2.73 se presenta el formulario `FrmBuscarAsistencia`, el cual permite buscar registros de asistencias de un empleado.

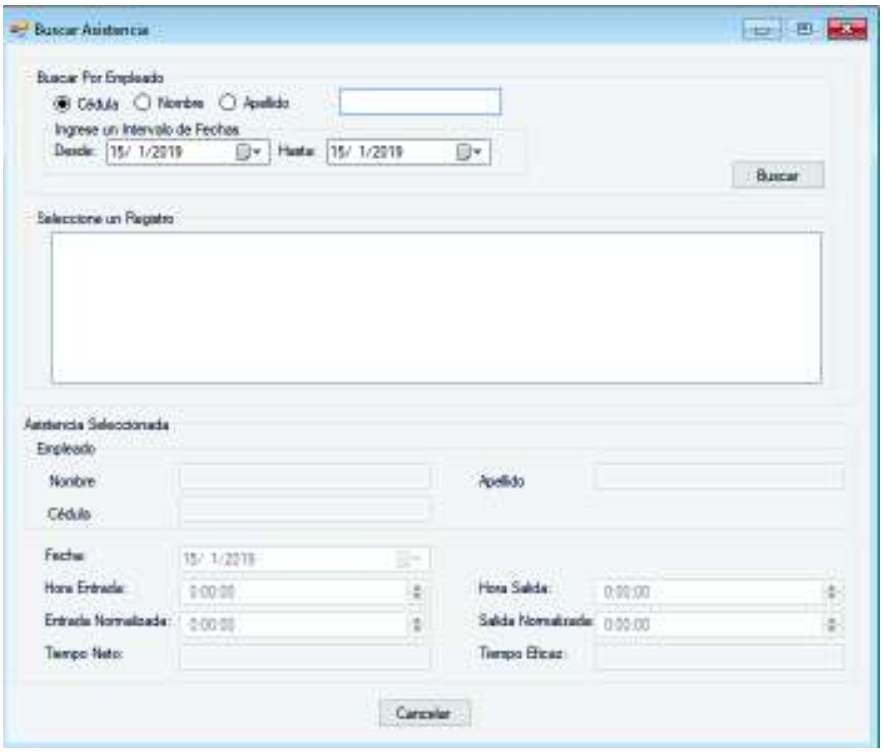


Figura 2.73.- Formulario `FrmBuscarAsistencia`

En la Figura 2.74 se muestra el formulario `FrmBuscarNotificacionGeovalla`, el cual permite buscar transiciones de entrada o salida ocurridas en la geovalla.

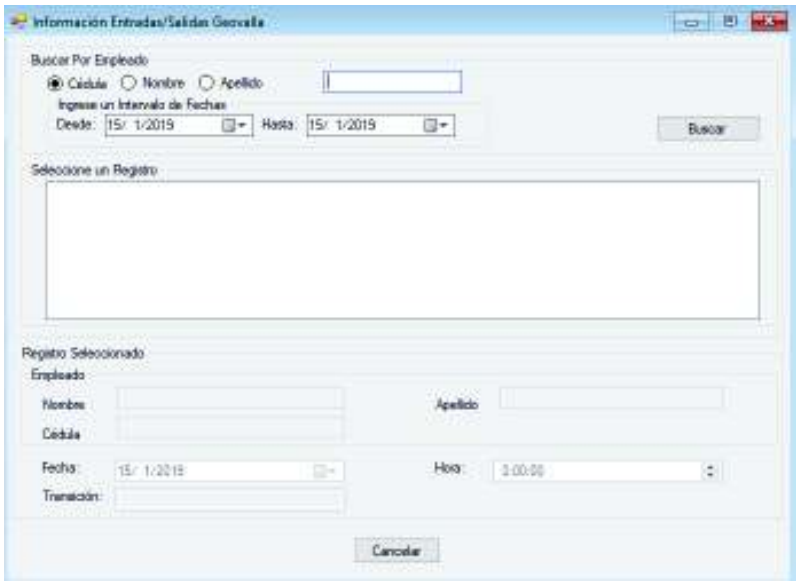


Figura 2.74.- Formulario `FrmBuscarNotificacionGeovalla`

3. RESULTADOS Y DISCUSIÓN

En este capítulo se presentan resultados a las pruebas realizadas al Cliente Android para verificar su funcionamiento, también se realizaron pruebas a la aplicación Gestor en una computadora portátil. Se efectuaron pruebas de conectividad tanto para la aplicación Android como para la aplicación Gestor. Se realizaron pruebas de funcionamiento tanto para el servicio WCF como para la base de datos. Se probó el funcionamiento de la lógica de control de asistencia y se verificó el correcto funcionamiento del monitoreo de geovallas para el registro automático del Cliente Android y sus correspondiente generación de notificaciones. Se realizaron 10 encuestas dirigidos a los trabajadores administrativos de la Facultad de Ingeniería Eléctrica y Electrónica para validar el aplicativo móvil.

Para la realización de pruebas de conectividad se configuró una zona con cobertura con el direccionamiento indicado en la Tabla 2.8.

Pruebas de conectividad en la aplicación Android

Para comprobar la conectividad entre el Cliente Android y el servicio WCF primeramente se habilitó la zona con cobertura denominado “PoliAsistencia”, luego desde el dispositivo Android se asoció a esta red, como se observa en la Figura 3.1. Por tanto, a través del protocolo DHCP, se le asignó al dispositivo Android una dirección IP de 192.168.137.211.

Figura 3.1.- Asociación a la zona con cobertura desde dispositivo Android

Después, se empleó el comando `ping 192.168.137.211` para verificar la conectividad desde la computadora portátil que aloja el servicio hasta el dispositivo Android, como se muestra en la Figura 3.2. Como resultado de esto, se observa que se tiene conectividad entre el dispositivo Android y en la computadora que aloja el servicio.

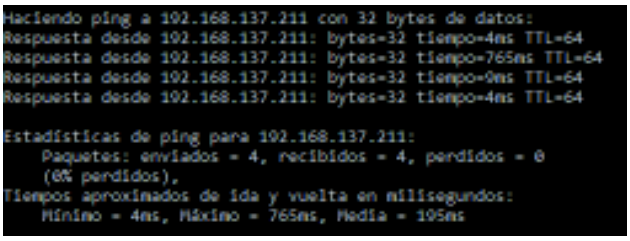


Figura 3.2.- Prueba de conectividad Cliente Android

Pruebas de conectividad en la aplicación Gestor

Debido a que el Cliente Gestor y el servicio se encuentran alojados dentro de la mismo computadora portátil, ambos tendrán conectividad por defecto, y para comprobarlo se empleó el comando `ping 192.168.137.1` como se muestra en la Figura 3.3.

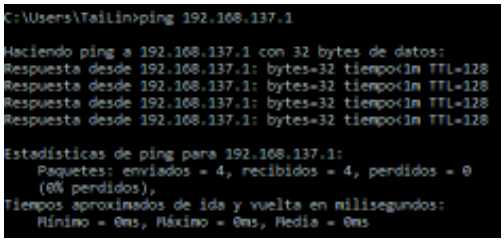


Figura 3.3.- Prueba de conectividad Cliente Gestor

En la Figura 3.4 se muestra el adaptador de red que proporciona la zona con cobertura. La dirección 192.167.137.1 es una dirección estática en el cual pasará todo el tráfico de los dispositivos conectados a la zona en cuestión y se lo empleó para definir la dirección IP del servicio.

Figura 3.4.- Adaptador de red de la zona con cobertura

Pruebas de funcionamiento de la base de datos

Para probar el funcionamiento de la base de datos, se ejecutarán sentencias SQL y se observarán los resultados de estos.

En la Figura 3.5 se muestran los resultados de la consulta a la tabla `HorarioTrabajo` por medio del comando `SELECT * FROM HorarioTrabajo`. Aquí se observa que se tienen

dos horarios de trabajo. El primero tiene una hora de entrada desde las 9:00 y el segundo desde las 8:00. El resto de datos del horario de trabajo son iguales.

	idHorarioTrabajo	horaInicioJornada	horaSalidaJornada	horaInicioReceso	horaFinReceso	diaInicioJornada	diaFinJornada
1	1	09:00:00	17:00:00	13:00:00	14:00:00	Lunes	Viernes
2	2	08:00:00	17:00:00	13:00:00	14:00:00	Lunes	Viernes

Figura 3.5.- Resultados de la consulta a la tabla `HorarioTrabajo`

En la Figura 3.6 se muestran los resultados de la consulta a la tabla `Credencial` por medio del comando `SELECT * FROM Credencial`. Aquí se observa que la primera credencial es la cuenta del usuario del Cliente Gestor mientras que los demás cuentas vienen a ser de los usuarios del Cliente Android.

	idCredencial	tipoCuenta	nombreUsuario	contrasena	estado
1	1	Administrador	admin	[redacted]	Habilitado
2	3	Usuario	user2		Habilitado
3	13	Usuario	user1		Habilitado

Figura 3.6.- Resultados de la consulta a la tabla `Credencial`

En la Figura 3.7 se muestran los resultados de la consulta a la tabla `Empleado` por medio del comando SQL `SELECT * FROM Empleado`. Se observa que se tiene en total dos empleados. El primero con identificador único `2` y de nombre Juana, y el segundo con identificador `11` y de nombre Tai. El primer empleado trabaja de 8:00 a 17:00 y el segundo de 9:00 a 17:00. Estos dos empleados serán los actores principales en todas las pruebas futuras.

	idEmpleado	nombre	apellido	numeroCedula	telefono	imagen	idHorarioTrabajo	idCredencial	pin
1	2	Juana	Arcos	1716705323	2225508	0x8950...	2	3	[redacted]
2	11	Tai	Wang	1716705312	2225507	0xFFD8...	1	13	

Figura 3.7.- Resultados de la consulta a la tabla `Empleado`

Por último, en la Figura 3.8 se muestra el resultado de la consulta `SELECT * FROM ConfiguracionHorasSistema`. En esta tabla se establece que 15 minutos después de la hora de entrada del empleado se considerará como atraso. También se establece que el empleado puede registrar su entrada 30 minutos antes de su hora de entrada, y que la hora límite para registrar la salida son las 21:30. Todos estos valores son configurables a través del Cliente Gestor.

	idConfiguracion	intervaloTiempoAtraso	intervaloTiempoAntesDeIngresar	horaLimiteParaMarcarSalida
1	2	00:15:00.0000000	00:30:00.0000000	21:30:00.0000000

Figura 3.8.- Resultados de la consulta a la tabla `ConfiguracionHorasSistema`

Pruebas de funcionamiento de la lógica de control de asistencia

Para probar el funcionamiento de la lógica de control de asistencias, se creó una aplicación Windows Forms[75] como se muestra en la Figura 3.9, la cual permite realizar las validaciones para el registro de entrada y el registro de salida dado un identificador único de un determinado empleado.

Todas las pruebas de funcionamiento de la lógica de control de asistencia se los realizó para el empleado con el identificador igual a 2. Este empleado posee un horario de trabajo de 8:00 a 17:00, de lunes a viernes, con receso de 13:00 a 14:00. Además, se empleó las configuraciones de horas del sistema como se muestra en la Figura 3.8, los cuales permiten establecer horas de atraso, horas límites de registro de salida y la hora que se habilita el registro de entrada.

En total se realizaron cinco pruebas para verificar el funcionamiento de la lógica de registro de entrada y tres pruebas para verificar el funcionamiento de la lógica de registro de salida, los cuales consisten en ingresar diferentes horas de entrada y salida y ver si la aplicación permite o no realizar sus correspondientes registros, estas pruebas se detallan en la Tabla 3.1 y Tabla 3.2.

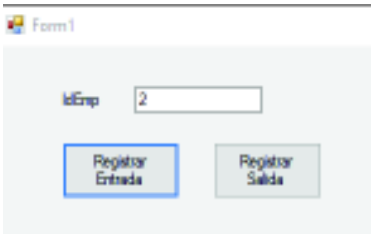


Figura 3.9.- Aplicación de prueba lógica control de asistencias

Tabla 3.1.- Pruebas de la lógica para registrar entrada

N° Prueba Registro Entrada	Hora Entrada	Descripción
1	7:15 (Entre semana)	Registrar entrada en una hora antes de la permitida
2	8:00 (Entre semana)	Registrar entrada en una hora puntual
3	8:20 (Entre semana)	Registrar entrada en una hora tarde
4	8:00 (Fin de semana)	Registrar entrada en fines de semana
5	8:00 (Durante Feriados)	Registrar entrada en días de feriado

[75] Windows Forms: es una aplicación con interfaz gráfica de usuario

Tabla 3.2.- Pruebas de la lógica para registrar salida

N° Prueba Registro Salida	Hora Salida	Descripción
1	17:00 (Entre semana)	Registrar salida sin antes registrar entrada
2	17:00 (Entre semana)	Registrar salida cuando si se registró una entrada
3	21:31(Entre semana)	Registrar salida cuando si se registró una entrada, excediendo el límite de la hora para marcar el registro en cuestión

Para tratar de depurar la lógica de control de asistencia y observar lo que está haciendo la aplicación, se crearon mensajes como se pueden ver en la Figura 3.10, donde se presentan los resultados de la primera prueba. Esta prueba consistió en tratar de ingresar un registro de entrada a una hora antes de la hora permitida. En la línea 1 se observa que primero se valida que el empleado no ingrese en día de feriados. En la línea 2 se muestra el resultado de la validación previa e indica que el empleado en cuestión no se encuentra dentro de feriados. En las líneas 3 y 4 se valida que el empleado no ingrese en días que pidió permiso. En la línea 5 se valida que el empleado en cuestión se encuentre dentro de sus días de jornadas laborales. En las líneas 6 y 7 se verifica que la hora de ingreso se encuentre dentro del periodo permitido. Por último en la línea 8 se obtiene el código `-1`, el cual indica que no se puede registrar la entrada, debido a que se ha configurado que solo se pueda registrar entrada 30 minutos antes de la hora de ingreso (7:30) y el empleado en cuestión trató de ingresar 15 minutos antes de lo permitido (7:15), por lo que este se encuentra muy temprano. Esto demuestra que la lógica funciona de manera adecuada.

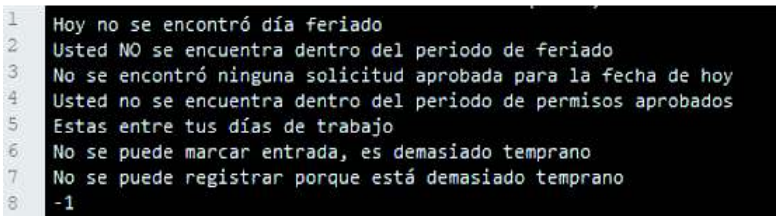


Figura 3.10.- Prueba 1 lógica registro entrada

En las siguientes pruebas solo se explicarán los resultados de estos.

En la Figura 3.11 se muestra el resultado de la segunda prueba del registro de entrada, el cual se obtuvo el código `0` , que indica que el empleado registra su entrada puntualmente.

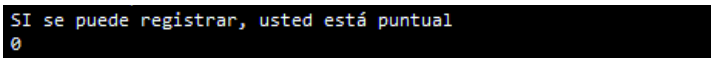


Figura 3.11.- Prueba 2 lógica registro entrada

En la Figura 3.12 se muestra el resultado de la tercera prueba del registro de entrada, se obtuvo el código `1`, el cual indica que el empleado realizó su registro de entrada con atraso.

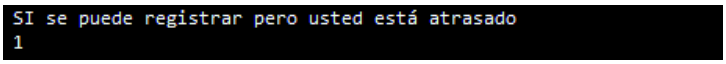


Figura 3.12.- Prueba 3 lógica registro entrada

En la Figura 3.13 se muestran los resultados de la cuarta prueba de la lógica de registro de entrada, se obtuvo el código `-3` el cual indica que el empleado intenta registrar su entrada en un día fuera de su jornada de trabajo convencional (sábado y domingo).

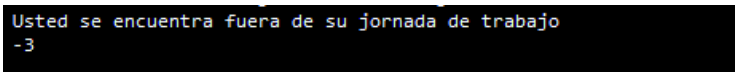


Figura 3.13.- Prueba 4 lógica registro entrada

En la Figura 3.14 se muestra el resultado de la quinta prueba para el registro de entrada del empleado. Se observa que se tiene el código igual a `-1`, el cual indica que el empleado tiene feriado y que no puede realizar su registro.

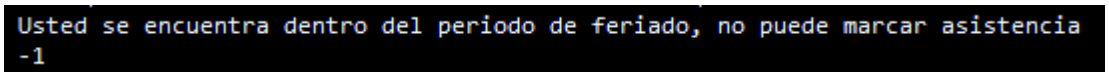


Figura 3.14.- Prueba 5 lógica registro entrada

A continuación se presentan los resultados de las pruebas de la lógica de registro de salida.

En la Figura 3.15 se muestra el resultado de la primera prueba de la lógica para registrar salida. Esta prueba consistió en registrar la salida sin antes registrar una entrada. Por lo que se obtuvo un código igual a `-1`, el cual indica lo antes mencionado.

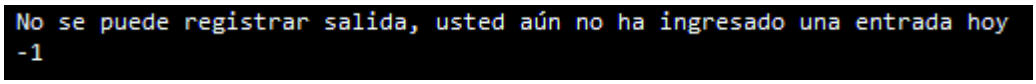


Figura 3.15.- Prueba 1 lógica registro salida

En la Figura 3.16 se muestra el resultado de la segunda prueba del registro de salida. En esta prueba sí se registró primero una entrada, luego se registró la salida en la hora fin de jornada del empleado (a las 17:00). Se observa que se obtiene el código `0`, el cual indica que si se puede registrar salida y su correspondiente asistencia.

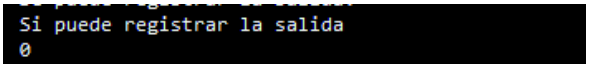


Figura 3.16.- Prueba 2 lógica registro salida

En la Figura 3.17 se muestra el resultado de la tercera prueba del registro de salida. En esta prueba también se registró primero una entrada, luego se registró la salida en un instante después de la hora límite para marcar salida. Se observa que se obtiene el código `-1`, el cual indica que no se puede registrar salida debido a que se configuró que la hora límite sea a las 21:30, por lo que se demuestra que la lógica para registrar salida funciona de manera adecuada.

```
No se puede registrar la salida
-1
```

Figura 3.17.- Prueba 3 lógica registro salida

Pruebas de funcionamiento del servicio WCF

Se empleó una aplicación de Google Chrome[76] denominado ARC[77], la cual permite enviar peticiones y obtener respuestas HTTP para verificar el funcionamiento del servicio WCF. La Figura 3.18 y Figura 3.19 se muestran las interfaces gráficas de la herramienta y en la Tabla 3.3 se muestra una descripción de sus diferentes componentes.

Figura 3.18.- Interfaz gráfica de la herramienta ARC (Parte 1)

Figura 3.19.- Interfaz gráfica de la herramienta ARC (Parte 2)

[76] Google Chrome: es una navegador web el cual permite interpretar lenguajes HTML, CSS, JavaScript y se lo muestra al usuario.
[77] ARC (*Advanced* REST *Client*): es un programa que permite enviar peticiones y recibir respuestas HTTP

Tabla 3.3.- Descripción de la interfaz gráfica de la herramienta ARC

ID	Descripción
1	Especifica el método HTTP
2	Especifica la dirección URI de una operación de servicio determinado
3	Permite enviar la petición
4	Permite Ingresar los parámetros del método POST en formato JSON
5	Muestra el código de la respuesta HTTP
6	Muestra la respuesta de la petición enviada

En la Figura 3.20 se presenta el envío de una petición con método HTTP `POST` a la operación que se encarga de registrar la entrada del empleado especificado como parámetro de entrada en formato JSON, a las 8:00 de la mañana. Y se obtiene como respuesta el código `1` que indica que se registró la entrada satisfactoriamente.

Figura 3.20.- Petición/Respuesta operación `RegistrarEntrada`

En la Figura 3.21, de igual manera, se presenta el envío de una petición con método HTTP `POST` a la operación que se encarga de registrar la salida del empleado especificado como parámetro de entrada en formato JSON a las 17:00 de la tarde. Y se obtiene como respuesta el código `0` que indica que se registró la salida satisfactoriamente.

Figura 3.21.- Petición/Respuesta operación `RegistrarSalida`

En la Figura 3.22 se observa que se han registrado correctamente la entrada y salida del empleado con identificador igual a `2` en las horas especificadas, al emplear el comando `SELECT * FROM RegistroEntradaSalida`.

	idRegistroEntradaSalida	idEmpleado	ingresoSalida	fechaHora
1	216	2	Ingreso	2019-01-21 08:00:37.9156074
2	217	2	Salida	2019-01-21 17:00:08.2736368

Figura 3.22.- Resultados de la consulta a la tabla `RegistroEntradaSalida`

En la Figura 3.23 se observa que se registró correctamente la asistencia del empleado en cuestión dentro de la base de datos, con horas de trabajo totales de aproximadamente 8 horas, a través del comando SQL `SELECT * FROM RegistroAsistencia`.

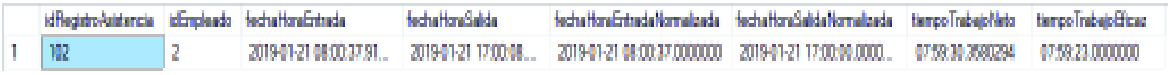


Figura 3.23.- Resultados de la consulta a la tabla `RegistroAsistencia`

Prueba de funcionamiento del Cliente Android

Para probar el funcionamiento del Cliente Android se procedió a crear un escenario de pruebas que consta de tres días de prueba, los cuales se emplearon diferentes métodos de registro de entrada y salida como se muestra en la Tabla 3.4. Las pruebas se las realizó para el empleado con el identificador único igual a `2`, el cual tiene un horario de trabajo de 8:00 a 17:00.

Tabla 3.4.- Días de prueba

Prueba	Día	Método Entrada	Hora Entrada	Método Salida	Hora Salida
1	Martes 22 de enero 2019	Huella Digital	8:00	Huella Digital	17:00
2	Miércoles 23 de enero 2019	PIN	8:00	PIN	17:00
3	Jueves 24 de enero 2019	PIN	8:00	Automático	17:00

Primer día de prueba

En la Figura 3.24 se muestra la configuración del Cliente Android para el primer día de prueba. Aquí se ingresó a la actividad `ConfiguracionEntradaSalidaActivity` a través de la actividad `ModulosActivity` y se escogió como opción de registro de entrada y salida el método de huella digital.

Figura 3.24.- Configuración prueba 1 Cliente Android

En la Figura 3.25 se muestra el proceso para registrar entrada en la hora especificada para el primer día de prueba por medio de huella digital. Primero se accede al módulo de registros de asistencias y se ingresa a la actividad `AutenticacionHuellaActivity`. Y

después se toca el sensor con el dedo y luego aparece un mensaje de registro de entrada satisfactoria cuando se haya autenticado correctamente el empleado.

Figura 3.25.- Resultados registro entrada con huella digital

En la Figura 3.26 se muestra el proceso para registrar la salida a la hora de salida especificada por medio de huella. De manera similar al proceso de registro de entrada, se accede a la actividad `AutenticacionHuellaActivity` y se toca el sensor dando un mensaje de registro satisfactorio.

Figura 3.26.- Resultados registro salida con huella digital

Para verificar si se realizó el registro de asistencia, se ingresa a `ModulosActivity` -> `ConsultasIndiceActivity` -> `ConsultasAsistenciasActivity`. En esta actividad se ingresa la fecha de búsqueda del primer día de pruebas (2019-01-22) y se ejecuta la consulta. Luego se selecciona el primer registro de asistencia y se muestra mayor información en la actividad `InfoRegistroAsistenciaActivity`. Aquí se observa que se tiene un tiempo de trabajo neto aproximadamente de 8 horas, por lo tanto, se registró correctamente la asistencia para el primer día de pruebas (ver Figura 3.27).

Figura 3.27.- Resultado consulta registro asistencia para prueba 1

Segundo día de prueba

En la Figura 3.28 se configuran las opciones de registros de entrada y salida para usar el PIN del empleado.

Figura 3.28.- Configuración prueba 2 Cliente Android

En la Figura 3.29 se ingresa a `ModulosActivity` -> `AutenticacionPinActivity`. Aquí aparece un diálogo el cual indica que el empleado tiene un PIN por defecto y le da la opción de cambiar. Luego el usuario ingresa su PIN y se registra su entrada en la hora especificada del segundo día de pruebas.

Figura 3.29.- Resultados registro entrada con PIN

En la Figura 3.30 se registra la salida en la hora especificada de manera similar al registro de entrada. Se obtuvo un mensaje de registro de salida exitoso.

Figura 3.30.- Resultados registro salida con PIN

En la Figura 3.31 se observa que se registró correctamente la asistencia del empleado para el segundo día de pruebas (2019-01-23), con un tiempo aproximado de 8 horas de trabajo.

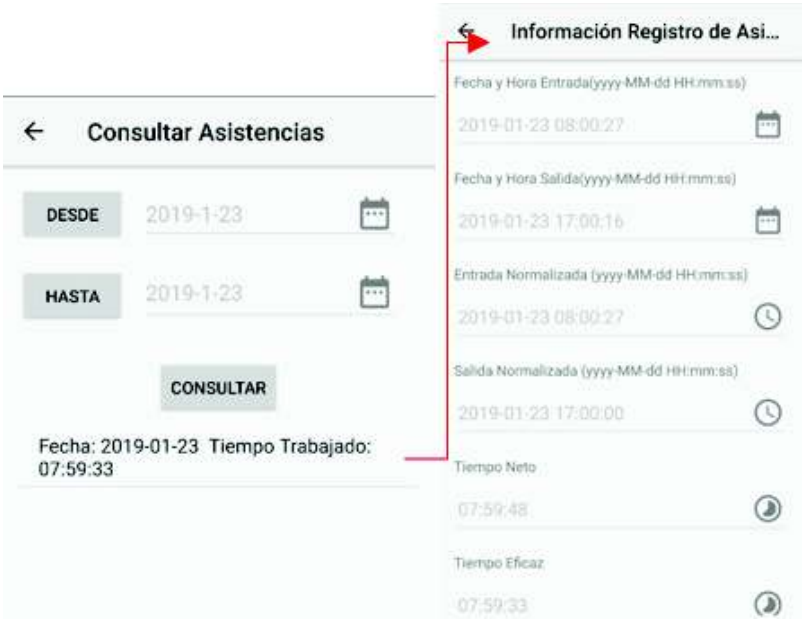


Figura 3.31.- Resultado consulta registro asistencia para prueba 2

Tercer día de prueba

Para el tercer día de prueba, se debió configurar primero la geovalla para el registro de salida automático, para ello se estableció una geovalla de radio 32 metros y con una

latitud/longitud de `-0.209215/-78.489521`, esta geovalla se encuentra ubicada en las coordenadas del centro de la Facultad de Ingeniería Eléctrica y Electrónica, como se presenta en la Figura 3.32.

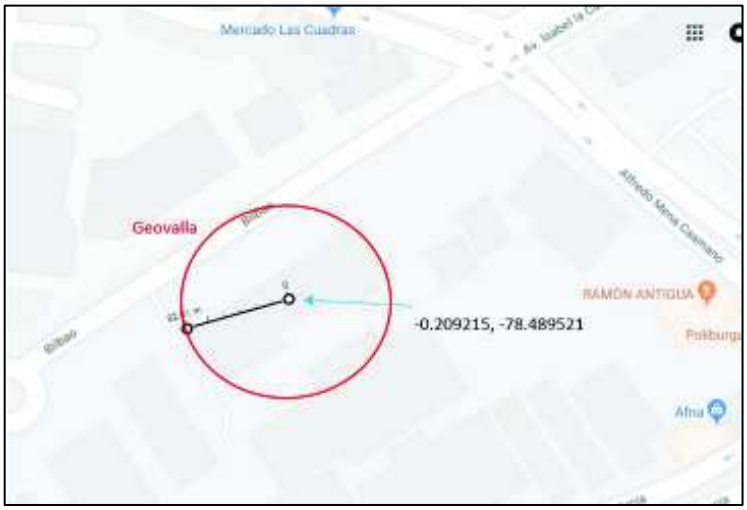


Figura 3.32.- Configuración Geovalla

Se configuró el Cliente Android para que el registro de entrada sea a través de PIN y la salida sea de manera automática como se muestra en la Figura 3.33.

Figura 3.33.- Configuración prueba 3 Cliente Android

En la Figura 3.34 se realiza el registro de entrada por medio de PIN a la hora de entrada establecida para el tercer día de pruebas.

Figura 3.34.- Registro entrada PIN tercer día prueba

Para simular la entrada y salida del área de monitoreo, se empleó la aplicación *Fake* GPS de Android, la cual permite establecer coordenadas determinadas para un dispositivo Android. En la Figura 3.35 se estableció las coordenadas del dispositivo para simular que este se sitúe en el centro de la Facultad de Ingeniería Eléctrica y Electrónica. Luego de hacerlo, se registra una notificación de geovalla con transición entrada.

Figura 3.35.- Resultados notificación transición entrada geovalla

En la Figura 3.36 se simula que el dispositivo se sitúe en un lugar fuera de la Facultad de Ingeniería Eléctrica y Electrónica a través de *Fake* GPS, por lo que, se registra la salida del empleado de forma automática, cuando sucede tal evento.

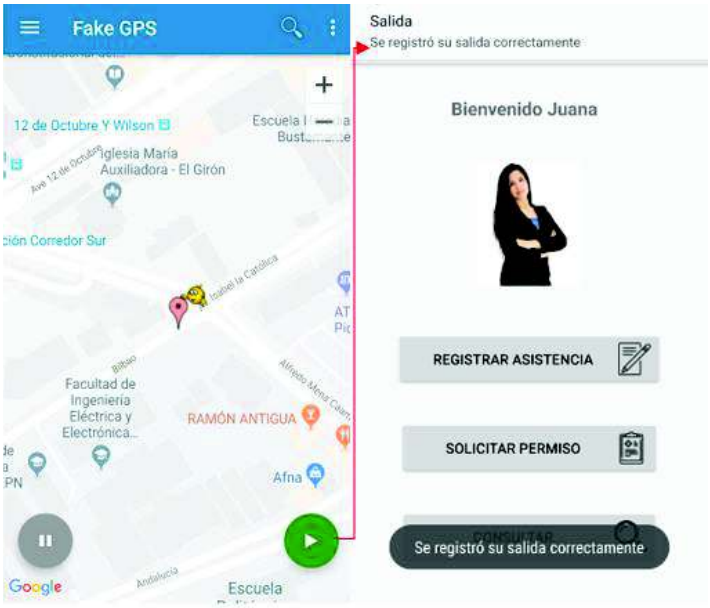


Figura 3.36.- Resultados registro salida automática

En la Figura 3.37 se observa que se registró la asistencia del empleado correctamente, con un total de horas trabajadas de 8.

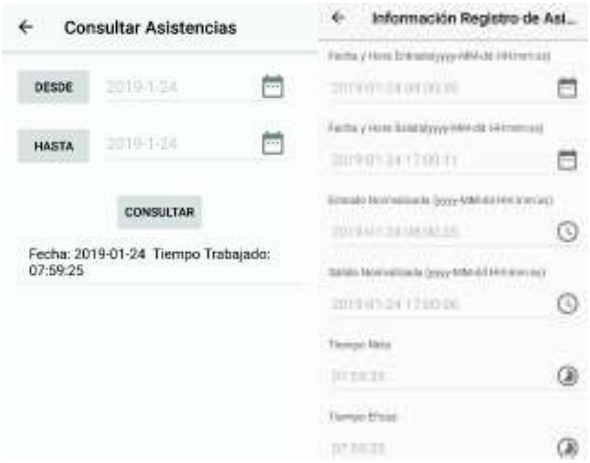


Figura 3.37.- Resultado consulta registro asistencia para prueba 3

En la Figura 3.38 se presentan las notificaciones de geovallas de entrada y salida ocurridas en el tercer día de pruebas, a través del comando SQL `SELECT * FROM NotificacionGeofencing`, por lo que se puede ver que se registraron notificaciones de manera correcta.

	idNotificacionGeofencing	idEmpleado	fechaHora	idGeovalla	tipoTransicion
1	117	2	2019-01-24 08:00:20.0000000	geo001	Entrada
2	119	2	2019-01-24 17:00:11.0000000	geo001	Salida

Figura 3.38.- Resultado consulta a la tabla `NotificacionGeofencing`

En la Figura 3.39 se presentan los resultados de la consulta del horario de trabajo del empleado en cuestión. Se muestra que el empleado trabaja de lunes a viernes, desde las 8:00 hasta las 17:00 con receso de 13:00 a 14:00.

Figura 3.39.- Consulta horario de trabajo

Por último, en la Figura 3.40 se presentan los resultados de la consulta del total de horas trabajadas de estos tres días de pruebas para el empleado en cuestión, desde la fecha 2019-01-22 hasta 2019-01-24, por lo que se tiene un total de aproximadamente 24 horas de trabajo.

Figura 3.40.- Consulta Total horas trabajadas

Prueba de funcionamiento del Controlador de Faltas

Para verificar el funcionamiento del Controlador de Faltas, se empleó dos escenarios de pruebas que se presentan en la Tabla 3.5. Estas pruebas se realizaron durante cinco días, de martes a sábado, el cual el empleado con identificador `2` registra su asistencia de martes a jueves, mientras que el empleado con el identificador `11` no registra su asistencia en ningún día. Se ejecutó el Controlador de Faltas para cada día del escenario de pruebas para verificar si efectivamente se registran las faltas de manera adecuada.

Tabla 3.5.- Escenario de pruebas para el Controlador de Faltas

Id Empleado	Martes 22 de enero 2019	Miércoles 23 de enero 2019	Jueves 24 de enero 2019	Viernes 25 de enero 2019	Sábado 26 de enero 2019
2	Asistió	Asistió	Asistió	Faltó	Descanso
11	Faltó	Faltó	Faltó	Faltó	Descanso

A continuación se presentan los resultados.

Para depurar la lógica del Controlador de Faltas se muestran mensajes como los de la Figura 3.41, el cual muestra los resultados del control de asistencias para el día miércoles 23 de enero de 2019. En la línea 1 se verifica que el empleado `2` se encuentre dentro de sus días de trabajo. En la línea 2, se busca el registro de asistencia del empleado `2` y se observa que se encontró el registro en cuestión, por lo que el empleado `2` sí asistió al trabajo. En la línea 3 se verifica que el empleado `11` se encuentre dentro de sus días de

trabajo. La línea 4 indica que no se encontró registro de asistencia del empleado `11`. En la línea 5 se verifica que el empleado `11` no tiene permiso para faltar. En la línea 6 se verifica que el empleado `11` no tiene feriado. Por lo que, en las líneas 7 al 9 se realiza el proceso para registrar la falta del empleado `11` debido a que este faltó injustificadamente. Los resultados obtenidos del control de asistencia para los días martes y jueves son similares a lo descrito anteriormente.

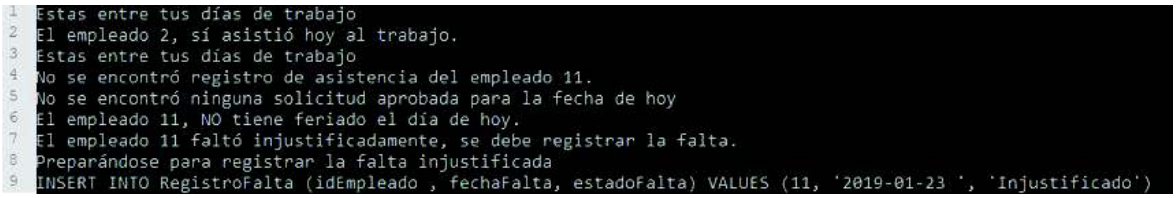


Figura 3.41.- Resultado del Controlador de Faltas para el día miércoles

En las siguientes pruebas, solo se mostrarán los resultados de estas, haciendo énfasis en si se controló correctamente o no la asistencia.

En la Figura 3.42 se muestran los resultados del control de asistencia para el día viernes. Como se puede observar aquí, ninguno de los empleados registraron sus correspondientes asistencias, por lo que, se registraron faltas para ellos.

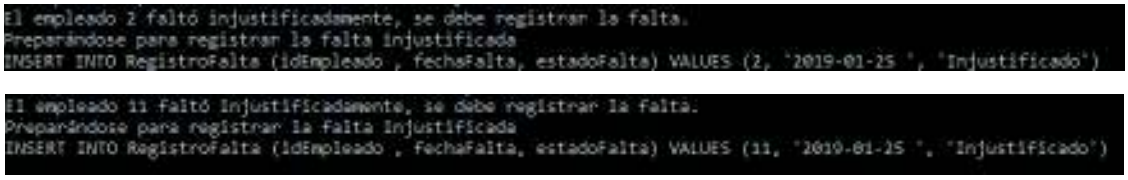


Figura 3.42.- Resultado de Controlador de Faltas para el día viernes

En la Figura 3.43 se muestra que no se registraron faltas para los empleados en cuestión debido a que ambos descansan el día sábado 26 de enero del 2019. Aquí en la línea 1 y 4 se verifica que los empleados `2` y `11` respectivamente no se encuentran dentro de su jornada de trabajo. En las líneas 2 y 5 se verifica que los empleados `2` y `11` respectivamente no tengan solicitudes de trabajo. En las líneas 3 y 6 se verifica que no se solicitó a los empleados `2` y `11` trabajar en el día especificado, por lo que no se registraron faltas para ellos.

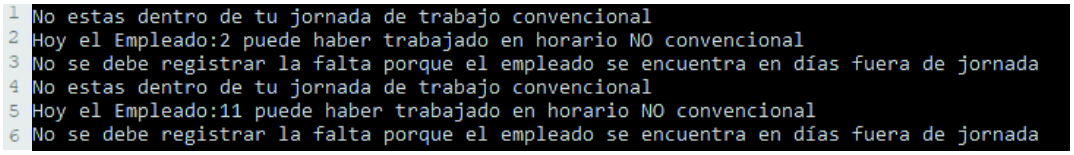


Figura 3.43.- Resultado de Controlador de Faltas para el día sábado

En la Figura 3.44 se muestran los resultados del comando SQL `SELECT * FROM RegistroFalta`. Aquí se presentan todas las faltas del escenario de pruebas. El empleado con identificador `11` faltó injustificadamente cuatro días en total, mientras que el otro empleado faltó solamente un día. Por lo tanto, se pudo verificar el correcto funcionamiento del Controlador de Faltas para el periodo de pruebas.

	idRegistroFalta	idEmpleado	fechaFalta	estadoFalta
1	12	11	2019-01-22	Injustificado
2	13	11	2019-01-23	Injustificado
3	16	11	2019-01-24	Injustificado
4	14	2	2019-01-25	Injustificado
5	15	11	2019-01-25	Injustificado

Figura 3.44.- Resultados de la consulta a la tabla `RegistroFalta`

Prueba de funcionamiento del Cliente Gestor

Para verificar el correcto funcionamiento del Cliente Gestor se generarán reportes de asistencias para el mismo escenario de pruebas de la Tabla 3.4 y Tabla 3.5 (de martes a sábado), en los cuales el empleado con el identificador `2` (Juana Arcos) registra su asistencia de martes a jueves y falta injustificadamente el día viernes, mientras que el empleado con el identificador `11` (Tai Lin Wang) falta todos los días. Por lo que se espera que Juana tenga una sola falta y tres asistencias, mientras que para Tai, se espera que este tenga cuatro faltas injustificadas y ninguna asistencia.

En la Figura 3.45 se presenta el reporte de asistencias de la empleada, el cual contiene información correspondiente a horas de entrada y salida durante el escenario de pruebas. Se observa que esta empleada asistió tres días al trabajo en las fechas 22, 23 y 24 de enero. Por lo tanto, el reporte de registros de asistencias coincide con los resultados esperados para el escenario de pruebas.

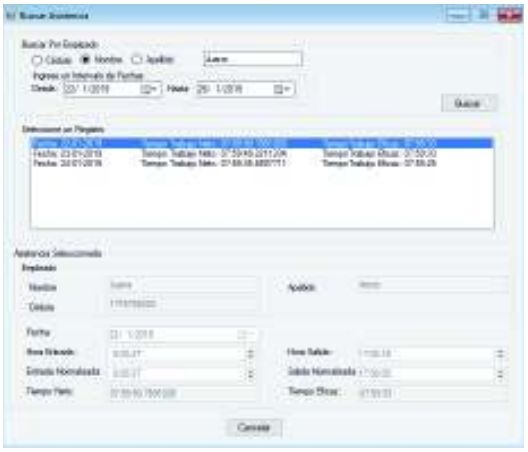


Figura 3.45.- Reportes asistencias de la primera empleada

En la Figura 3.46 se presentan los reportes de asistencias del empleado. Se observa que no se encontró ningún registro de asistencia. Por lo tanto, coincide con los resultados esperados del escenario de pruebas.

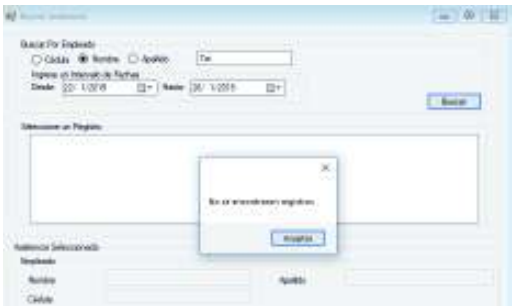


Figura 3.46.- Reportes asistencias del segundo empleado

En la Figura 3.47 se presenta el reporte total de horas trabajadas de la empleada. Se observa que durante el escenario de pruebas, esta empleada trabajó aproximadamente 24 horas, lo cual es correcto debido a que la empleada trabajó durante tres días, cada día por 8 horas.

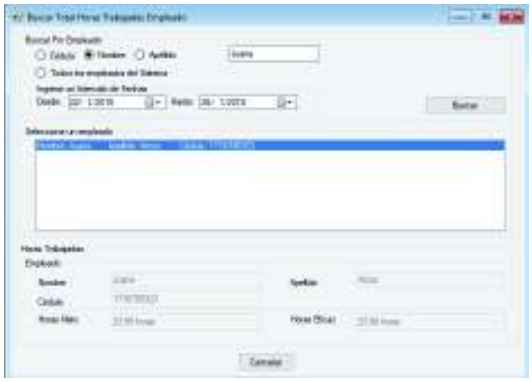


Figura 3.47.- Total horas trabajadas de la empleada

En la Figura 3.48 se presenta el reporte de total de horas trabajadas del empleado y se observa que este trabajó 0 horas. Este resultado coincide con lo esperado.

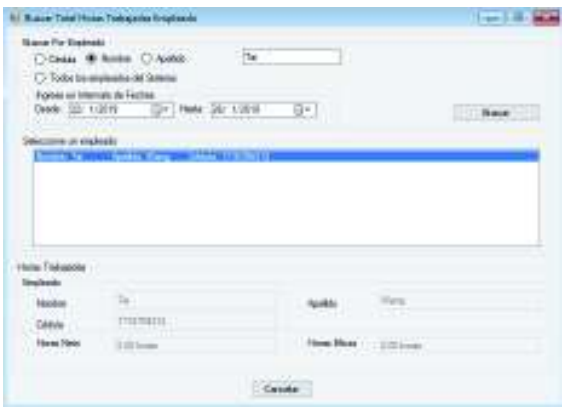


Figura 3.48.- Total horas trabajadas del empleado

En la Figura 3.49 se presentan los datos de la geovalla, los datos de latitud/longitud y radio fueron configurados según lo establecido en la Figura 3.32.

Figura 3.49.- Datos de la geovalla

De acuerdo con la Tabla 3.4 del escenario de pruebas, el día jueves, la empleada (Juana) registró su salida de manera automática, por lo que en la Figura 3.50 se muestran las notificaciones resultantes de ese día. Se observa que se tienen en total dos notificaciones, una de entrada a las 8:00 y otra de salida a las 17:00. Por lo tanto, las notificaciones resultantes coinciden con las notificaciones esperadas para el escenario de pruebas.

Figura 3.50.- Búsqueda de notificaciones del área de monitoreo

En la Figura 3.51 se muestra una falta injustificada el día viernes 25 de enero de la empleada, por lo tanto, coincide con lo establecido en la Tabla 3.5.

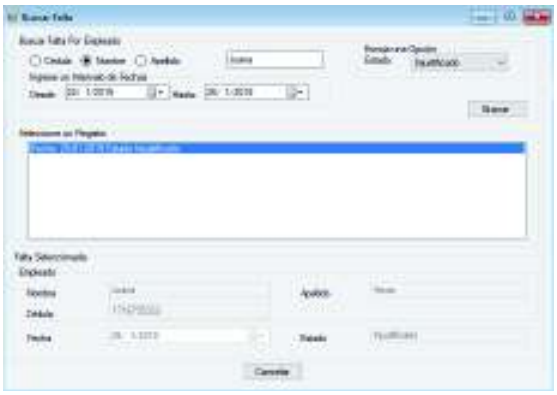


Figura 3.51.- Faltas injustificadas del primer empleado

En la Figura 3.52 se muestra el reporte de faltas injustificadas del empleado, se observa que este faltó injustificadamente los días 22, 23, 24 y 25 de enero, los cuales coinciden con los resultados de faltas esperadas para el escenario de pruebas.

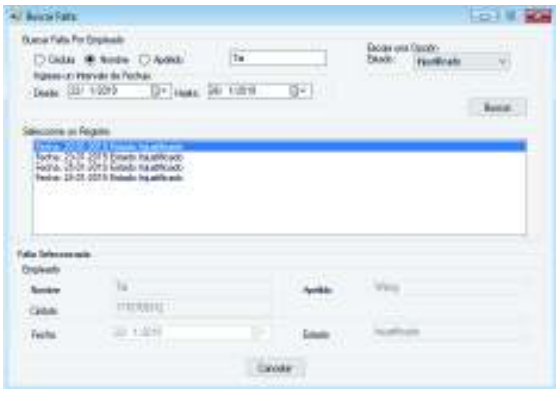


Figura 3.52.- Faltas injustificadas del segundo empleado

En la Figura 3.53 se muestra el formulario que permite crear un nuevo horario de trabajo, y en la Figura 3.54 se presenta la creación de un nuevo empleado y se le asigna un horario determinado.

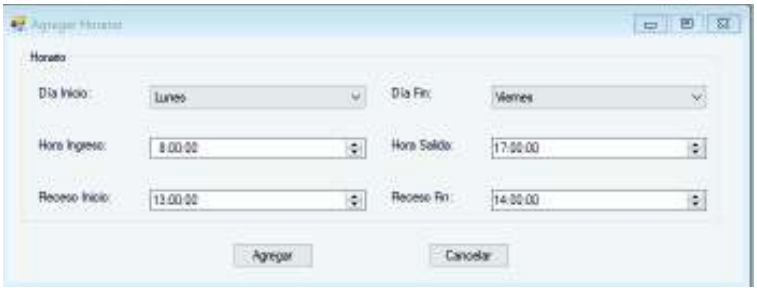


Figura 3.53.- Creación horario de trabajo

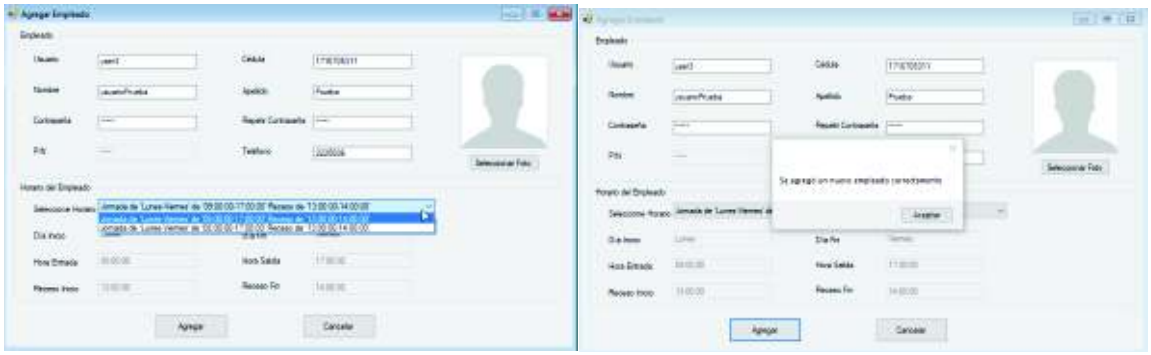


Figura 3.54.- Asignación del horario de trabajo al empleado

Por último, en la Figura 3.55 se muestran los cambios ocurridos tras la creación de un nuevo empleado. Se observa que se ingresó correctamente el empleado en la base de datos por medio de la consulta SQL `SELECT * FROM Empleado`. Por lo tanto, se agregó correctamente un nuevo empleado.

	idEmpleado	nombre	apellido	numeroCedula	telefono	imagen	idHorarioTrabajo	idCredencial	pin
1	2	Juana	Arcos	1716705323	2225508	0x89504E470D0A1A0A0000000D4948445200000C8000000...	2	3	
2	11	Tai	Wang	1716705312	2225507	0xFFD8FFE000104A4649460001010100000000000FFDB004...	1	13	
3	25	usuarioPrueba	Prueba	1716705311	2225506	0xFFD8FFE000104A4649460001010100000000000FFDB004...	1	27	

Figura 3.55.- Cambios ocurridos en la tabla `Empleado`

Encuestas de validación

Se distribuyó el Cliente Android a 10 personas que forman parte del personal administrativo de la Facultad de Ingeniería Eléctrica y Electrónica los cuales se les pidió llenar una encuesta de validación después de que estos probaran el aplicativo en cuestión. A continuación se muestran los resultados obtenidos de la encuesta, la cual se presenta en el Anexo X.

La primera pregunta permite confirmar si el usuario pudo registrar su entrada a través de su huella digital. En la Figura 3.56 indica que todos los encuestados pudieron registrar sus entradas a través de sus huellas digitales.

Figura 3.56.- Respuesta a la primera pregunta de la encuesta de validación

La segunda pregunta permite confirmar si el usuario pudo registrar su entrada por medio de su PIN. En la Figura 3.57 indica que todos los encuestados pudieron registrar sus entradas por medio de sus códigos de seguridad PIN.

Figura 3.57.- Respuesta a la segunda pregunta de la encuesta de validación

La tercera pregunta permite confirmar si el usuario pudo registrar su salida por medio de su huella digital. En la Figura 3.58 indica que todos los encuestados pudieron registrar sus salidas por medio de sus huellas digitales.

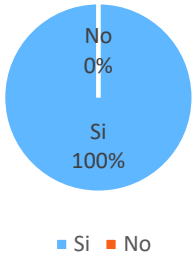


Figura 3.58.- Respuesta a la tercera pregunta de la encuesta de validación

La cuarta pregunta permite confirmar si el usuario pudo registrar su salida por medio de su PIN. En la Figura 3.59 indica que todos los encuestados pudieron registrar sus salidas por medio de sus códigos de seguridad PIN.

Figura 3.59.- Respuesta a la cuarta pregunta de la encuesta de validación

La quinta pregunta permite confirmar si el usuario pudo seleccionar la opción de registro de salida automático. En la Figura 3.58 indica que todos los encuestados pudieron seleccionar la opción de registro de salida automático.

Figura 3.60.- Respuesta a la quinta pregunta de la encuesta de validación

La sexta pregunta permite confirmar si el usuario pudo registrar su salida de forma automática, sin necesidad de ingresar a la aplicación, cuando este eligió la opción de registro automático. En la Figura 3.61 indica que todos los encuestados pudieron registrar sus salidas de forma automática sin ingresar a la aplicación.

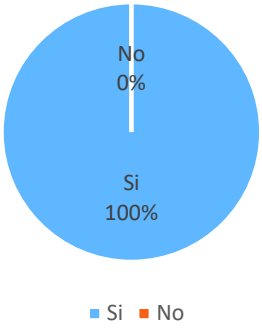


Figura 3.61.- Respuesta a la sexta pregunta de la encuesta de validación

La séptima pregunta permite confirmar si el usuario pudo consultar su horario de trabajo. En la Figura 3.62 indica que todos los encuestados pudieron consultar sus horarios de trabajo.

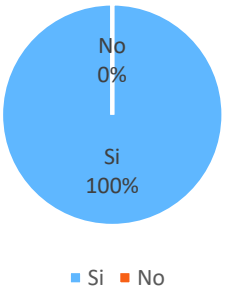


Figura 3.62.- Respuesta a la séptima pregunta de la encuesta de validación

La octava pregunta permite confirmar si el usuario pudo consultar sus horas de ingreso. En la Figura 3.63 indica que todos los encuestados pudieron consultar sus horas de ingreso.

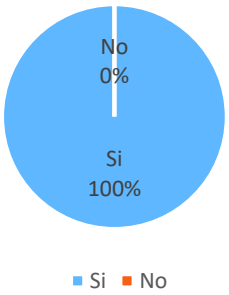


Figura 3.63.- Respuesta a la octava pregunta de la encuesta de validación

La novena pregunta permite confirmar si el usuario pudo consultar sus horas de salida. En la Figura 3.64 indica que todos los encuestados pudieron consultar sus horas de salida.

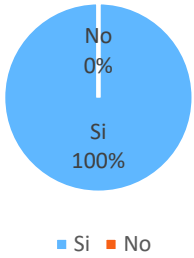


Figura 3.64.- Respuesta a la novena pregunta de la encuesta de validación

La décima pregunta permite confirmar si le pareció útil al usuario la opción para consultar su calendario de feriados. En la Figura 3.65 indica que la opción para consultar feriados les parecieron muy útil para todos los encuestados.

Figura 3.65.- Respuesta a la décima pregunta de la encuesta de validación

La onceava pregunta permite confirmar si el usuario pudo consultar sus atrasos. En la Figura 3.66 indica que todos los encuestados pudieron consultar sus atrasos.

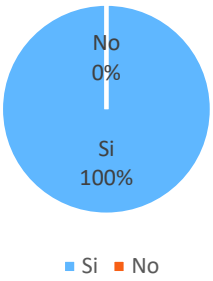


Figura 3.66.- Respuesta a la onceava pregunta de la encuesta de validación

La doceava pregunta permite al usuario calificar entre valores del 1 (valor más bajo) al 5 (valor más alto) su experiencia tras el uso de las opciones para solicitar permisos. En la Figura 3.67 indica que el 80% de los encuestados calificaron con el valor más alto la experiencia en cuanto a las diferentes opciones para el envío de peticiones de permisos.

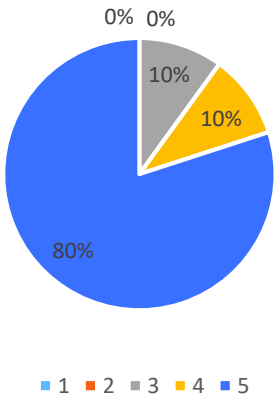


Figura 3.67.- Respuesta a la doceava pregunta de la encuesta de validación

La treceava pregunta permite confirmar si el usuario pudo consultar sus faltas, en caso de que los tuvieran. En la **Figura 3.68** indica que todos los encuestados pudieron consultar sus faltas en caso de que estos existiesen.

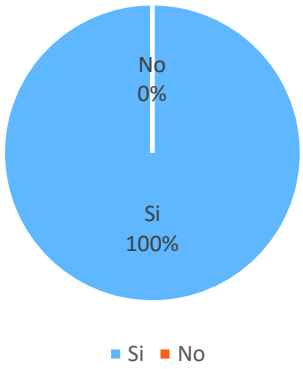


Figura 3.68.- Respuesta a la treceava pregunta de la encuesta de validación

La catorceava pregunta permite al usuario calificar entre valores del 1 (valor más bajo) al 5 (valor más alto) su experiencia tras el uso de la opción para consultar solicitudes de trabajo. En la Figura 3.69 indica que el 70% de los encuestados calificaron con el valor más alto sus experiencias en cuanto al uso de las consultas para ver sus solicitudes de trabajo. El 20% de los encuestados calificaron esta opción con el valor de 4, el cual representa una calificación bastante buena, mientras que solo el 10% de los encuestados calificaron esta opción con un valor medio bajo.

La quinceava y última pregunta permite confirmar si el usuario pudo consultar sus horas totales de trabajo. En la Figura 3.70 indica que todos los encuestados pudieron consultar sus horas de trabajo totales.

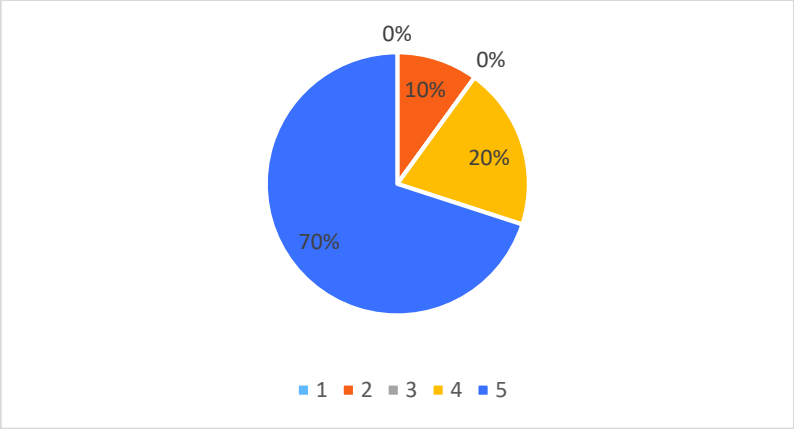


Figura 3.69.- Respuesta a la catorceava pregunta de la encuesta de validación

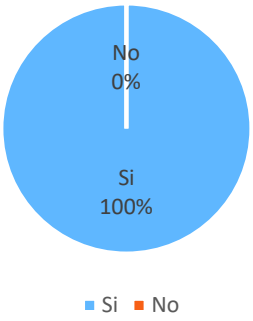


Figura 3.70.- Respuesta a la quinceava pregunta de la encuesta de validación

Correcciones Sprint Review

Después de cada *sprint* se presentaron los avances al Director del presente Trabajo de Titulación, quien revisó detalladamente los avances y especificó las correcciones requeridas a realizarse. Se realizaron las correcciones debidas en todas las iteraciones y se los recopiló en un conjunto de tablas que se presentan a continuación.

En la Tabla 3.6 se muestran las correcciones realizadas durante el primer *sprint*.

Tabla 3.6.- *Sprint Review* 1

Sprint 1	Correcciones
	Las faltas injustificadas se pueden justificar.
	Las configuraciones de horas del sistema no deben ser estáticos, deben de ser configurables.
	Se tiene que poder registrar asistencias en días fuera de jornada convencional, ya sea en sábado o domingo, cuando se lo requiera.
	Se debe de realizar una rutina cada noche para que se registre faltas de los empleados que no registraron sus asistencias.

En la Tabla 3.7 se muestran las correcciones realizadas en el segundo *sprint*.

Tabla 3.7.- *Sprint Review* 2

Sprint 2	Correcciones
	El empleado solo debe trabajar como máximo 40 horas semanales

En la Tabla 3.8 se muestran las correcciones realizadas en el tercer *sprint*.

Tabla 3.8.- *Sprint Review* 3

Sprint 3	Correcciones
	El usuario del Cliente Android al seleccionar la opción registro de salida automática, ya no tiene que presionar otro botón para empezar el proceso en cuestión.
	Para que el empleado solicite permiso, debe seleccionar de un `Spinner` el motivo el cual solicita el permiso.
	Para solicitar permisos se debe de validar que no sea en vacaciones o en días que no tenga que trabajar.
	Corrección de las flechas para regresar a la actividad anterior

En la Tabla 3.9 se muestran las correcciones realizadas en el cuarto *sprint*.

Tabla 3.9.- *Sprint Review* 4

Sprint 4	Correcciones
	Sin observaciones

En la Tabla 3.10 se muestran las correcciones realizadas en el quinto *sprint*.

Tabla 3.10.- *Sprint Review* 5

Sprint 5	Correcciones
	En el Cliente Gestor, se deben mostrar mensajes del `MessageBox` con sus respectivos iconos, títulos, etc.
	Limpiar el formulario para agregar empleado después de terminar de realizar el proceso en cuestión
	No mostrar en ninguna parte los identificadores únicos de los registros
	Para guardar cambios se debe aparecer un diálogo que pregunte si se desea guardar o no los cambios.
	No se debe eliminar empleados por motivo de auditoría. Por lo que se debe de agregar un nuevo atributo que muestre el estado de cuenta del usuario, si se encuentra habilitado o deshabilitado.
	Para agregar un nuevo horario, se debe de controlar que este no pase de 40 horas semanales.
	Para realizar la búsqueda de los diferentes registros, se debe emplear un `LIKE` en las consultas SQL en vez del =.
	Para borrar un horario de trabajo, primero se debe de verificar que nadie haga uso de esta para que se pueda realizar tal acción.
	Para modificar un registro, se debe de dar una razón para hacerlo.
	Por motivo de auditoría, no se debe de eliminar los registros de asistencias de los empleados. Se debe de crear un historial en el cual se guarden todos los cambios realizados.

En la Tabla 3.11 se muestran las correcciones realizadas en el sexto *sprint*.

Tabla 3.11.- *Sprint Review* 6

Sprint 6	Correcciones
	Las interfaces gráficas del Cliente Gestor no se encuentran alineados. Se deben usar herramientas de Visual Studio para alinear.
	Corrección de espacios de las interfaces gráficas
	Se debe verificar que todos los `MessageBox` tengan icono.
	Eliminación de imágenes de fondo de todos los formularios para evitar el *Flickering* de estos.

Por último, en la Tabla 3.12 se muestran las correcciones realizadas en el séptimo y último *sprint*.

Tabla 3.12.- *Sprint Review* 7

Sprint 7	Correcciones
	No mostrar todas las letras de las interfaces gráficas del Cliente Gestor en negrilla
	Poner el mismo formato de fechas para todos los controles `DateTimePicker`
	Mantener la lógica del orden de los componentes gráficos
	Poner mayor explicación en el `RadioButton` para que este busque todos los registros.
	Cambiar el título del formulario para ver notificaciones geovalla a “Información Entrada/Salida Geovalla”
	Corrección del `MenuStrip` del formulario principal.
	Cambio del título del formulario para configurar geovallas a “Datos de la Geovalla”

4. CONCLUSIONES

- El propósito de este Trabajo de Titulación era contar con un sistema que permita controlar la asistencia del personal administrativo de la Escuela Politécnica Nacional. Al concluir este trabajo se dispone de un prototipo que permite tener dicho control. El prototipo está conformado por: un Cliente Android, un Cliente Gestor, un servicio WCF y un controlador de asistencias.

- El Cliente Android permite al usuario registrar sus entradas y salidas por medio de una huella digital, un código de seguridad PIN, además, permite al usuario registrar su asistencia cuando este salga fuera del campus politécnico, de manera automática, sin que el usuario intervenga, y a su vez, permite enviar notificaciones al entrar o salir del campus. Así mismo, permite al usuario consultar las entradas, las salidas, las asistencias, los atrasos, las faltas, los días de feriado, las solicitudes de permiso, las solicitudes de trabajo en días fuera de jornada de trabajo y el total de horas trabajadas. Adicionalmente, permite al usuario solicitar permisos para faltar al trabajo por un determinado periodo de tiempo. Por tanto, el usuario puede registrar su asistencia y realizar todo lo antes mencionado desde cualquier punto con acceso a Internet, de manera descentralizada a través del dispositivo móvil.

- El Cliente Gestor permite al usuario administrador gestionar los empleados con la finalidad de crear, modificar, habilitar y deshabilitar empleados. También permite consultar las asistencias, las faltas, los atrasos y el tiempo total trabajado de todos los empleados. Así mismo, permite emitir solicitudes de trabajo y aprobar o rechazar permisos. Además permite al administrador observar notificaciones cuando un empleado entre o salga del campus durante el horario de trabajo. Por otra parte, permite realizar configuraciones del sistema, como el establecimiento del horario límite para registrar salida, del intervalo de tiempo en que se considere como atraso, la hora permitida para registrar entrada y el establecimiento de los detalles del área de monitoreo.

- Existen varios medios por los cuales se pueden obtener una imagen de una huella digital, ya sea por medio óptico, eléctrico o sonido, siendo este último el más fiable ya que es menos suceptible a la suciedad, grasa y sudor de los dedos. El sistema operativo Android guarda estas imágenes en un lugar seguro denominado `Keystore`, la cual se accede para realizar la autenticación por medio de huella. Por esto, el uso de huellas digitales en aplicaciones Android resulta ser para el

usuario un medio fácil y seguro para acceder a diferentes funcionalidades que, en el presente trabajo se focalizó en registrar asistencias.

- Se desarrolló el Cliente Android para la API versión 23 Android 6.0 “Marshmallow” o superior, puesto que desde este nivel de API, se tienen funcionalidades que permiten manejar el proceso de autenticación por medio de huellas digitales.

- Las geovallas son áreas circulares virtuales de monitoreo las cuales disparan eventos cuando un dispositivo Android entra o sale del área en cuestión, y para su implementación se empleó la API de Google denominado *Location Services*. Se empleó esta API ya que facilita la elección del proveedor de ubicación según la precisión que se desea, ya sea por medio de WiFi o GPS, además permite un uso eficiente de la batería del dispositivo móvil. Por todo esto, el manejo de monitoreo de geovallas puede resultar potencialmente atractivo para empleadores, dueños de empresas y personal de recursos humanos que desean controlar asistencias de sus trabajadores en las diferentes empresas e instituciones.

- Se separaron la lógica de consumo de operaciones web, la lógica de manejo de registros de base de datos y la lógica de presentación de datos con el fin de facilitar el trabajo de depuración y desarrollo del sistema. Se emplearon *proxies* tanto en la aplicación Android y Gestor que vienen a ser como una instancia local del servicio que ofrece todas las funcionalidades establecidas según el usuario que llame las operaciones. Se emplearon clases centralizadas que se encargan de manipular datos de las tablas de base de datos y se presentaron los resultados de las operaciones en los diferentes componentes de las interfaces gráficas.

- Se empleó la librería Volley el cual posee hilos para enviar peticiones y obtener respuestas del servicio web con el objetivo de evitar bloqueos de la aplicación Android. Por otro lado, permite guardar respuestas de peticiones en memoria caché para que no se tengan que volver a enviar estos a través de la red, mejorando así el rendimiento del uso de la red en cuestión.

- Los servicios web permiten compatibilidad entre diferentes plataformas, sistemas operativos, lenguajes de programación ya que emplean formatos web estandarizados como JSON. Para que ambos lados, cliente y servicio puedan entenderse y llevar a cabo la comunicación, el cliente debe implementar lo conformado en la interfaz o contrato del servicio, caso contrario no se podrán recibir respuestas adecuadas del servicio.

- Se empleó el tipo de formato JSON debido a que este tipo de notación resulta ser más ligero y menos verboso que otros formatos como XML, HTML, mensajes SOAP. Por esto, las aplicaciones Android manejan este tipo de formato debido a las limitaciones de recursos del dispositivo móvil.

- Se implementaron métodos para serializar y deserializar tipos de datos que denotan fechas, horas e intervalos de tiempo debido a que las diferentes instancias de clases para la serialización/deserialización no podían realizarlo de manera automática.

- Los métodos del código de control de asistencia retornan números enteros positivos y negativos los cuales representan códigos que indican si un empleado puede realizar o no sus registros de asistencias.

- La navegación entre las actividades del Cliente Android se encuentra conformada de manera jerárquica, categorizadas en diferentes niveles. El primer nivel se trata del nivel superior, el cual contiene lo más importante para el usuario, por ejemplo una lista de todos los módulos del aplicativo para registrar asistencia, consultas o para solicitar permiso. El segundo se trata del nivel de categorías, en el cual el usuario puede seleccionar una categoría y se despliegan los resultados de esto en una lista, por ejemplo, cuando el usuario consulta registros de asistencias, se muestran los registros de asistencias en una lista. El tercer y último nivel se trata de detalles o información, y permite desplegar por ejemplo toda la información de manera detallada sobre un registro de asistencia en particular.

- Se empleó el programador de tareas de Windows con el propósito de automatizar el proceso de controlar las faltas de todos los empleados del sistema. Puesto que se requiere controlar la asistencia todos los días.

- Según el Código de Trabajo vigente en el país, los artículos 47 y 50 establecen que la jornada máxima de trabajo no debe exceder de cuarenta horas semanales y que los días sábados y domingos son días de descanso forzosos, sin embargo cuando se requiera realizar trabajo en los días mencionados, se debe designar otro tiempo igual para el descanso, por lo que el prototipo de sistema del presente trabajo permite realizar tal pericia por medio de solicitudes de trabajo y solicitudes de permiso.

- Por motivo de auditoría, toda información referente a empleados resulta ser valiosa y no debe de ser desechadas, especialmente información referente a asistencias.

Por esta razón, se implementaron historiales de cambios que guardan un respaldo de todos los cambios que se realicen a estos registros.

Recomendaciones

- El servicio WCF no cuenta con medidas de seguridad debido a que no se lo especificó en el alcance del trabajo, todas las peticiones y respuestas se encuentran en texto plano y es legible por cualquier persona que analice el tráfico de la red, por lo que se recomienda a futuro incluir protocolos seguros.
- Es necesario emplear consultas parametrizadas en las cuales se especifiquen el tipo de dato y el valor para evitar SQL Injection.
- Para configurar información de latitud y longitud de la geovalla, se debe de ingresar varios dígitos decimales para tener mayor precisión. Además, se recomienda la implementación de una interfaz gráfica más amigable, en que el administrador pueda seleccionar en un mapa las coordenadas y el radio del área de monitoreo.
- Para la serialización y deserialización de objetos JSON en .NET, se recomienda el uso de la librería JSON.NET ya que resulta ser más rápido en realizar su función.
- El contrato de servicio WCF solo especifican métodos HTTP `GET` y `POST`, por lo que se recomienda explorar sobre la configuración del servicio para que se permitan los métodos HTTP `PUT` y `DELETE` ya que estos no se encuentran habilitados por defecto.
- Se recomienda que en futuros trabajos de titulación se desarrolle la integración del sistema de control de asistencia actual de la Escuela Politécnica Nacional con el prototipo desarrollado en el presente trabajo, para que los empleados tengan diferentes opciones para registrar sus asistencias y que exista congruencia de registros en todo el sistema.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] Ministerio de Relaciones Laborales, “Oficio Circular No. 1799”. 13-mar-2012.

[2] S. Klein, *Professional WCF Programming*. Wiley Publishing, Inc.

[3] L. Masinter, T. Berners-Lee, y R. T. Fielding, “Uniform Resource Identifier (URI): Generic Syntax”. [En línea]. Disponible en: https://tools.ietf.org/html/rfc3986. [Consultado: 06-jul-2018].

[4] J. Flanders, *RESTful .NET*. Sebastopol, CA: O’Reilly Media Inc.

[5] K. Evenepoel, “Introduction to WCF 3.5 and Rest”. .

[6] mcleblanc, “Configuration file schema for the .NET Framework”. [En línea]. Disponible en: https://docs.microsoft.com/en-us/dotnet/framework/configure-apps/file-schema/. [Consultado: 18-jul-2018].

[7] Microsoft Developer Network, “Información general sobre las cookies en ASP.NET”. [En línea]. Disponible en: https://msdn.microsoft.com/es-es/library/ms178194(v=vs.100).aspx. [Consultado: 19-jul-2018].

[8] Webcindario, “Definición de Servidor IIS”. [En línea]. Disponible en: http://2003server.webcindario.com/iis/definici.htm. [Consultado: 11-jun-2018].

[9] Microsoft Developer Network, “Introducción al servidor web (IIS)”. [En línea]. Disponible en: https://msdn.microsoft.com/es-es/library/hh831725(v=ws.11).aspx. [Consultado: 11-jun-2018].

[10] S. Li, “Maintain HTTP Session State in WCF REST Services with HttpWebRequest - CodeProject”. [En línea]. Disponible en: https://www.codeproject.com/Articles/322436/Maintain-HTTP-Session-State-in-WCF-REST-Services-w. [Consultado: 12-jul-2018].

[11] Android Developers, “Volley overview”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/training/volley/. [Consultado: 12-jun-2018].

[12] afzaln, “Volley Toolbox Overview”. [En línea]. Disponible en: https://afzaln.com/volley/. [Consultado: 13-jun-2018].

[13] “Gson User Guide - gson”. [En línea]. Disponible en: https://sites.google.com/site/gson/gson-user-guide. [Consultado: 28-jun-2018].

[14] Android Developers, “Arquitectura de la plataforma”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/guide/platform/?hl=es-419. [Consultado: 18-jul-2018].

[15] Android Developers, “Conoce Android Studio”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/studio/intro/?hl=es-419. [Consultado: 09-jul-2018].

[16] Android Developers, “Configurar tu compilación”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/studio/build/?hl=es-419. [Consultado: 10-jul-2018].

[17] Android Developers, “Manifiesto de la app”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419. [Consultado: 10-jul-2018].

[18] Android Developers, “Actividades”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/guide/components/activities?hl=es-419. [Consultado: 10-jul-2018].

[19] “Autenticación por huella dactilar en tu App. Sistemas de Identificación Biométrica.”, *Aplicaciones Móviles*, 21-mar-2018. .

[20] D. S. Zorita y J. Ortega, “Reconocimiento automático mediante patrones biométricos de huella dactilar”, Universidad Politécnica de Madrid, 2003.

[21] Techotopia, “An Android Fingerprint Authentication Tutorial”. [En línea]. Disponible en: https://www.techotopia.com/index.php/An_Android_Fingerprint_Authentication_Tutorial. [Consultado: 14-jun-2018].

[22] Android Developers, “Location and context overview”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/training/location/. [Consultado: 18-jun-2018].
[23] Android Developers, “Location and Maps”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/guide/topics/location/. [Consultado: 18-jun-2018].
[24] Oracle Mobile Platform, *MCS: 70. Working with Android Geofences*. .
[25], Android Developers, “Create and monitor geofences”, *Android Developers*. [En línea]. Disponible en: https://developer.android.com/training/location/geofencing. [Consultado: 12-jul-2018].
[26] T. Teorey, *Database Modeling & Desing*. USA: Morgan Kaufmann Publishers, 2011.
[27] A. Silberschatz, H. F. Korth, y S. Sudarshan, *Database system concepts*, 4th ed. Boston: McGraw-Hill, 2002.
[28] stevestein, “SQL Server Management Studio (SSMS)”. [En línea]. Disponible en: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms. [Consultado: 15-jul-2018].
[29] K. Schwaber, “La Guía de Scrum”. 2013.
[30] J. Calvache, “Análisis y diseño de un sistema que permita controlar el acceso y asistencia del personal para la empresa Human Trend”, Escuela Politécnica Nacional, Quito, Pichincha, Ecuador, 2010.
[31] L. Chicaiza, “Diseño e implementación de un sistema de control de asistencia de personal, mediante el uso de tecnología biométrica de huella dactilar”, Escuela Politécnica Nacional, Quito, Pichincha, Ecuador, 2014.
[32] J. Casas Anguita, J. R. Repullo Labrador, y J. Donado Campos, “La encuesta como técnica de investigación. Elaboración de cuestionarios y tratamiento estadístico de los datos (I)”, *Aten. Primaria*, vol. 31, núm. 8, pp. 527–538, 2003.
[33] dotnet-bot, “Binding Class (System.ServiceModel.Channels)”. [En línea]. Disponible en: https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel.channels.binding. [Consultado: 13-ene-2019].

6. ANEXOS

ANEXO I. Encuesta de requerimientos de usuario

ANEXO II. Historias de usuario

ANEXO III. *Sketches* y *Wireframes*

ANEXO IV. *Scripts* de la base de datos

ANEXO V. Solución del servicio WCF (Código Fuente)

ANEXO VI. Solución del Cliente Android (Código Fuente)

ANEXO VII. Solución del Cliente Gestor (Código Fuente)

ANEXO VIII. Instalador del Cliente Android

ANEXO IX. Instalador del Cliente Gestor

ANEXO X. Encuestas de validación

ANEXO XI. Manual de usuario

*Debido a la extensión, todos los anexos se han incluido en el disco compacto adjunto a este documento.

.

ORDEN DE EMPASTADO