ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE SISTEMA DISTRIBUIDO QUE MUESTRE LA DISPONIBILIDAD DE PARQUEO EN LA FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA MEDIANTE ENVÍO DE NOTIFICACIONES EN TIEMPO REAL

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

RICARDO FABIÁN SILVA POVEDA
ricardo.silva@epn.edu.ec

DIRECTOR: PhD. ANA MARÍA ZAMBRANO VIZUETE
ana.zambrano@epn.edu.ec

CODIRECTOR: MSc. FABIO GONZÁLEZ GONZÁLEZ
fabio.gonzalez@epn.edu.ec

Quito, Febrero 2019

AVAL

Certifico que el presente trabajo fue desarrollado por Ricardo Fabián Silva Poveda, bajo mi supervisión.

PhD. Ana María Zambrano
DIRECTORA DEL TRABAJO DE TITULACIÓN

MSc. Fabio González
CO-DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo Ricardo Fabián Silva Poveda, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado 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.

Ricardo Fabián Silva Poveda

DEDICATORIA

Este trabajo es dedicado a mis Abuelitos, Jamen y Angelita, además de mi madre Gisela y mi hermano Cristhian, quienes han sido mi apoyo incondicional a lo largo de este proceso y han sabido apoyarme e incentivarme para alcanzar este objetivo.

Ricardo

AGRADECIMIENTO

Gracias a Dios por permitirme conocer a mis abuelos Jamen y Angelita, gracias a la vida por permitirme disfrutar cada día a su lado, gracias Papitos por ser los mejores de todo el mundo, gracias, simplemente gracias.

Agradezco especialmente a mi madre Gisela quien ha sido mi pilar, mi fortaleza, mi refugio, que me ha apoyado en cada momento de mi vida junto a mi hermano Christian.

Agradezco a mis tíos, mis primos y toda mi familia que con sus palabras y muestras de cariño han sabido darme fuerza para continuar este camino.

Agradezco a la Dra. Ana Zambrano, quien me dio la oportunidad y la confianza para desarrollar este proyecto, además de ser quien supo guiarme y aconsejarme para realizar las cosas de una manera correcta.

Agradezco a la Facultad de Ingeniería Eléctrica y Electrónica y a sus profesores, que gracias a sus consejos y enseñanzas, me han permitido desarrollar habilidades y conocimientos que son de vital importancia en el día a día laboral.

Ricardo

ÍNDICE DE CONTENIDO

AVAL I
DECLARACIÓN DE AUTORÍA II
DEDICATORIA III
AGRADECIMIENTO IV
ÍNDICE DE CONTENIDO V
ÍNDICE DE FIGURAS VII
ÍNDICE DE TABLAS X
ÍNDICE DE SEGMENTOS DE CÓDIGOS XII
RESUMEN XIII
ABSTRACT XIV
1 INTRODUCCIÓN 1
1.1 Objetivos 2
1.1.1 Objetivo General 2
1.1.2 Objetivos Específicos 2
1.2 Alcance 2
1.3 Marco Teórico 4
1.3.1 Smartparking 4
1.3.2 Sistemas Operativos y Smartphones 5
1.3.3 Android 6
1.3.4 IPhone OS (iOS) 7
1.3.5 Metodologías Ágiles de Desarrollo de Software 8
1.3.6 Herramientas, Tecnologías y Lenguajes por utilizar Capa de Datos 13
1.3.7 Herramientas, Tecnologías y Lenguajes por utilizar Capa de Negocio .. 15
1.3.8 Herramientas, Tecnologías y lenguajes por utilizar Capa Presentación. 24
2. METODOLOGÍA 28
2.1 Estado Actual del Parqueadero de la FIEE 28
2.1.1 Componentes del Sistema de Ingreso a la FIEE 29
2.2 Diseño 32
2.2.1 Tablero Kanban 32
2.2.2 Diagrama General del Prototipo 33
2.2.3 Requerimientos Funcionales 34

2.2.4 Requerimientos No Funcionales 36
2.2.5 Diseño de la Capa de Datos.....37
2.2.6 Diseño de la Capa de Negocio.....38
2.2.7 Diseño de la Capa de Presentación 52
2.3 Implementación 55
2.3.1Instalación de Herramientas.....55
2.3.2 Codificación de la Capa de Datos 59
2.3.3 Codificación de la Capa de Negocio 61
2.3.4 Codificación Capa de Presentación 77
3. RESULTADOS Y DISCUSIÓN 82
3.1 Pruebas Funcionales 82
3.1.1 Prueba de Funcionamiento al Subsistema Arduino.....82
3.1.2 Prueba de Funcionamiento al Subsistema de Registro e Ingreso.....84
3.1.3 Prueba de Funcionamiento al Subsistema de Usuario.....86
3.2 Pruebas de Requerimientos No Funcionales.....88
3.2.1 Prueba de Requerimiento No Funcionamiento RNF01 88
3.2.2 Prueba de Requerimiento No Funcionamiento RNF02 90
3.2.3 Pruebas de Aceptación. Requerimiento No Funcional RNF03 y RNF04.91
3.3 Resumen.....92
4 CONCLUSIONES Y RECOMENDACIONES 94
4.1 Conclusiones.....94
4.2 Recomendaciones.....95
5 REFERENCIAS BIBLIOGRÁFICAS..... 96
6 ANEXOS 99
ANEXO I. Codificación de la Base de Datos. 100
ANEXO II. Codificación de los Servicios Web. 103
ANEXO III. Codificación de Notificaciones PUSH 107
ANEXO IV. Codificación del Módulo Arduino. 108
ANEXO V. Pruebas realizadas a los usuarios del prototipo. 111

ÍNDICE DE FIGURAS

Figura 1.1 Prototipo distribuido de disponibilidad de parqueo. 1
Figura 1.2 Estructura actual del parqueadero de la FIEE 2
Figura 1.3 Pila de componentes de Android 7
Figura 1.4 Arquitectura iOS 8
Figura 1.5 Pizarra o Tablero Kanban 11
Figura 1.6 Arduino UNO 16
Figura 1.7 *Sockets*. 17
Figura 1.8 Arquitectura REST 19
Figura 1.9 Microsoft Azure 21
Figura 1.10 Esquema de notificaciones *push*. 22
Figura 1.11 Notificaciones *push* en Android. 23
Figura 1.12 Notificaciones *push* en iOS 24
Figura 1.13 Arquitectura *Xamarin*. 25
Figura 2.1 Parqueadero FIEE. 28
Figura 2.2 Sistema de ingreso actual a la FIEE 29
Figura 2.3 Sistema RFID 29
Figura 2.4 Tablero Kanban 32
Figura 2.5 Diagrama general del prototipo de disponibilidad de parqueo 33
Figura 2.6 Subsistemas del prototipo de disponibilidad de parqueo 34
Figura 2.7 Diagrama relacional de bases de datos 37
Figura 2.8 Caso de uso del Subsistema Arduino 40
Figura 2.9 Caso de uso del Subsistema de Registro e Ingreso. 40
Figura 2.10 Caso de uso del Subsistema de Usuario. Estructura general 41
Figura 2.11 Caso de uso del Subsistema de Usuario. Proceso de notificación. 41
Figura 2.12 Caso de uso del Subsistema de Administración 42
Figura 2.13 Diagrama de secuencia subsistema Arduino 42
Figura 2.14 Diagrama de secuencia para el registro de un usuario. 43
Figura 2.15 Diagrama de secuencia para la autenticación de un usuario. 43
Figura 2.16 Diagrama de secuencia subsistema de usuario. 44
Figura 2.17 Diagrama de clases. 45
Figura 2.18 Arquitectura de los Servicios Web 45
Figura 2.19 Proceso envío de una notificación *push*. 46

Figura 2.20 Establecimiento de conexión entre el servidor local y el módulo Arduino 50
Figura 2.21 Obtención del número de espacios al ingreso de la FIEE 51
Figura 2.22 Obtención del número de espacios a la salida de la FIEE 51
Figura 2.23 Cálculo del número de espacios disponibles en la FIEE 52
Figura 2.24 Interfaz para la autenticación de un usuario 52
Figura 2.25 Interfaz del Subsistema de Registro 53
Figura 2.26 Interfaz del Subsistema de Usuario. Vista general 53
Figura 2.27 Interfaz Subsistema de Usuario. Programación de notificaciones 54
Figura 2.28 Interfaz Subsistema de Usuario. Agregar comentario. 54
Figura 2.29 Interfaz Subsistema de Administración. 54
Figura 2.30 Tablero Kanban. Implementación 55
Figura 2.31 Activación de las características del servidor Web IIS. 56
Figura 2.32 Visual Studio Community 2017. 56
Figura 2.33 Selección de características de instalación Visual Studio 2017. 57
Figura 2.34 Centro de instalación de *SQL Server* 2014. 58
Figura 2.35 Selección características para la instalación de *SQL Server 2014.* 58
Figura 2.36 Instalación IDE de Arduino. 59
Figura 2.37 Creación Base de Datos. 59
Figura 2.38 Registros ingresados en la tabla Usuario. 60
Figura 2.39 Creación Biblioteca de Clases. 61
Figura 2.40 Creación de un nuevo elemento ADO.NET *Entity Data Model.* 61
Figura 2.41 ADO.NET, configuración de la conexión. 62
Figura 2.42 Modelo .edmx. 63
Figura 2.43 Creación de un nuevo proyecto ASP.NET. 63
Figura 2.44 Carpetas del proyecto ASP.NET. 64
Figura 2.45 ADO.NET, agregar un nuevo controlador. 64
Figura 2.46 Publicar los servicios Web mediante servidor IIS. 66
Figura 2.47 Configuración parámetros para publicar un Servicios Web. 66
Figura 2.48 Portal Azure. Centro de Notificaciones. 67
Figura 2.49 Centro de Notificaciones. Creación del recurso notifications hub. 67
Figura 2.50 Configuración Centro de Notificaciones. 68
Figura 2.51 Firebase Google. 68

Figura 2.52 Firebase Google. 69
Figura 2.53 Propiedades *Proyecto Cross-Plataform Visual Studio.* 69
Figura 2.54 Firebase Google, Clave del Servidor 70
Figura 2.55 Azure, API Key. 70
Figura 2.56 Solicitud de firma de certificado. 71
Figura 2.57 Centro de Desarrolladores Apple. 72
Figura 2.58 Registro de la Aplicación. 72
Figura 2.59 Agregar certificado de iOS. 73
Figura 2.60 Certificado de la aplicación iOS 74
Figura 2.61 Configuración del servicio notifications hub de Azure para iOS. 74
Figura 2.62 Proyecto iOS en Visual Studio 74
Figura 2.63 Creación del proyecto Cross-Plataform 78
Figura 2.64 Página principal del prototipo de disponibilidad de parqueo 79
Figura 2.65 Página de Registro del prototipo de disponibilidad de parqueo 79
Figura 3.1 Tablero Kanban. Pruebas 82
Figura 3.2 Módulo Arduino 83
Figura 3.3 Lectura y envío de datos con Arduino. 83
Figura 3.4 Pruebas de funcionalidad. Registro de un usuario. 84
Figura 3.5 Pruebas de funcionalidad. Registro de usuario en la base de datos 84
Figura 3.6 Pruebas de funcionalidad. Ingreso de un usuario 85
Figura 3.7 Página principal de la aplicación 85
Figura 3.8 Pruebas de funcionalidad. Consulta de disponibilidad de parqueo. 86
Figura 3.9 Pruebas de funcionalidad. Configuración horario y recepción de notificación. 86
Figura 3.10 Estadísticas notificación en Cloud Messaging 87
Figura 3.11 Pruebas de funcionalidad. Almacenar un comentario. 87
Figura 3.12 Tiempo de respuesta. Subsistema de Registro 89
Figura 3.13 Tiempo de respuesta. Subsistema de Ingreso 89
Figura 3.14 Tiempo de respuesta. Subsistema de Usuario 89
Figura 3.15 Prueba de Fiabilidad 90
Figura 3.16 Tablero Kanban. Finalización de fase de pruebas 93

ÍNDICE DE TABLAS

Tabla 1.1 Ventas mundiales de smartphones por proveedor en el primer trimestre del 2017. 5
Tabla 1.2 Ventas mundiales de *smartphones* por sistema operativo en el primer trimestre del 2017. 6
Tabla 1.3 Postulados del Manifiesto Ágil. 9
Tabla 1.4 Principios respecto de metodologías de desarrollo ágil de software 9
Tabla 1.5 Principales características de la metodología Kanban 10
Tabla 1.6 Ventajas de la metodología Kanban. 12
Tabla 1.7 Scrum vs Kanban 13
Tabla 1.8 Tecnologías presentes en *SQL Server*. 14
Tabla 1.9 Instrucciones SQL 15
Tabla 1.10 Principales características de los *sockets* TCP y UDP 18
Tabla 1.11 Principales características de los servicios web REST 19
Tabla 1.12 Características del IDE de *Visual Studio* 20
Tabla 1.13 Características de las notificaciones *push*. 22
Tabla 1.14 Características de *Visual Studio Xamarin* 25
Tabla 2.1 Requerimientos funcionales del prototipo disponibilidad de parqueo. 34
Tabla 2.2 Requerimientos funcionales del Subsistema Arduino. 35
Tabla 2.3 Requerimientos funcionales del Subsistema de Registro e Ingreso. 35
Tabla 2.4 Requerimientos funcionales del Subsistema de Usuario. 35
Tabla 2.5 Requerimientos funcionales del Subsistema de Administración. 36
Tabla 2.6 Requerimientos no funcionales del prototipo de disponibilidad de parqueo. 37
Tabla 2.7 Permisos por tipos de usuarios en el prototipo de disponibilidad de parqueo. 39
Tabla 2.8 Opciones para el envío de una notificación *push* en Android. 47
Tabla 2.9 Atributos para la carga útil en notificaciones *push* Android. 48
Tabla 2.10 Atributos para la carga útil en notificaciones *push* iOS. 49
Tabla 3.1 Pruebas de Funcionamiento al Subsistema Arduino. 83
Tabla 3.2 Pruebas de Funcionamiento al Subsistema de Registro e Ingreso. 85
Tabla 3.3 SB03 Pruebas de Funcionamiento. 88

Tabla 3.4 Resumen Pruebas RNF01. 89
Tabla 3.5 Prueba RNF01. 90
Tabla 3.6 Prueba RNF02. 90
Tabla 3.7 Pruebas de Aceptación. 91
Tabla 3.8 Prueba RNF03 y RNF04. 91
Tabla 3.9 Resumen Pruebas. 92

ÍNDICE DE SEGMENTOS DE CÓDIGOS

Código 2.1 Carga útil de una notificación *push* Android. 47
Código 2.2 Carga útil de una notificación *push* iOS. 49
Código 2.3 Creación de la tabla USUARIO 60
Código 2.4 Archivo App.Config. Cadena de conexión para la base de datos 62
Código 2.5 Controlador para el registro de usuario 65
Código 2.6 AndroidManifest.xml 70
Código 2.7 Código para la recepción de notificaciones *push* en *Visual Studio* 75
Código 2.8 Código Servidor Escucha TCP 75
Código 2.9 Código Servidor TCP 76
Código 2.10 Implementación Módulo Arduino al Ingreso de la FIEE. 77
Código 2.11 Interfaz Gráfica de Inicio 78
Código 2.12 Registro de un nuevo usuario 80
Código 2.13 Archivo constants para almacenar las URL de los servicios web 81
Código 3.1 Tiempo respuesta de los servicios Web 88

RESUMEN

El presente trabajo se creó con la finalidad de ayudar a resolver la necesidad de encontrar un espacio de parqueo en el estacionamiento de la Facultad de Ingeniería Eléctrica y Electrónica (FIEE); complementando el sistema actual gestionado por la Dirección de Gestión de la Información y Procesos (DGIP) el cual permite el ingreso pero no permite a los usuarios conocer la disponibilidad del parqueadero en cualquier instante. El prototipo permite conocer la disponibilidad del parqueadero basando su funcionamiento en el concepto de sistema distribuido, mediante servicios web y una aplicación que consume dichos servicios, desarrollada para *smartphones* que trabajan con sistema operativo iOS y Android. Además, un usuario puede programar un horario en el cual pueda recibir una notificación con el número de espacios disponibles.

En el **Capítulo 1** se analiza la metodología de desarrollo de software Kanban que se utilizó para el desarrollo del presente Proyecto de Titulación; además se realiza una descripción de las herramientas, tecnologías y lenguajes utilizados. En el **Capítulo 2** se muestra el diseño e implementación de los diferentes componentes del prototipo. En el **Capítulo 3** se detallan las pruebas y el análisis de los resultados obtenidos. Y finalmente en el **Capítulo 4** se muestran las conclusiones y recomendaciones.

En el **Anexo I** se detalla el código utilizado para implementar la Base de Datos. En el **Anexo II** se muestra el código utilizado para la codificación de los Servicios Web, mientras que en el **Anexo III** se muestra el código para añadir notificaciones *push*. El **Anexo IV** muestra el código desarrollado para el módulo Arduino. Y finalmente se adjuntan las pruebas realizadas en el **Anexo V.**

Palabras clave: Sistema Distribuido. Android. iOS. Notificaciones *Push*. Servicios Web. Arduino.

ABSTRACT

The present work was created with the purpose of helping to solve the need to find a parking space in the parking lot of the Faculty of Electrical and Electronic Engineering (FIEE); complementing the current system managed by the Management of Information and Processes (DGIP) which allows entry but does not allow users to know the availability of parking at any time. The prototype will allow to know the availability of parking based on the concept of distributed system, through web services and an application that consumes these services, developed for smartphones that work with the iOS and Android operating system. In addition, a user can program a schedule in which he can receive a notification with the number of available spaces

In **Chapter 1** the Kanban software development methodology that was used for the development of the present Titling Project is analyzed; In addition, a description of the tools, technologies and languages used is made. **Chapter 2** shows the design and implementation of the different components of the prototype. **Chapter 3** details the tests and the analysis of the results obtained. And finally in **Chapter 4** the conclusions and recommendations are shown.

In **Annex I**, the code used to implement the Database is detailed. The code used for the coding of Web Services is shown in **Annex II**, while in **Annex III** the code to add push notifications is shown. **Annex IV** shows the code developed for the Arduino module. And finally the tests done are attached in **Annex V**.

Keywords: Distributed System. Android. iOS. Push Notifications. Web Services. Arduino

1. INTRODUCCIÓN

El parqueadero de la Escuela Politécnica Nacional (EPN) ubicado en la calle Bilbao corresponde al estacionamiento utilizado por la Facultad de Ingeniería Eléctrica y Electrónica (FIEE), Ingeniería Mecánica y por cualquier docente que trabaja en la EPN, para propósitos de este proyecto se lo denomina como parqueadero de la FIEE, el mismo que cuenta con un sistema gestionado por la Dirección de Gestión de la Información y Procesos (DGIP), el cual brinda acceso a las instalaciones mediante la tecnología RFID (del inglés Radio Frequency Identification), un minicomputador que valida la información del usuario en una bases de datos y un sensor de lazo magnético para detectar la presencia de vehículos, utilizado para la apertura de barreras (Véase Figura 1.1, módulo A).

Sin embargo, este sistema no brinda la posibilidad de informar a los usuarios en un instante determinado si el parqueadero dispone de espacios libres. En este sentido se desarrolla un prototipo de sistema distribuido (Véase Figura 1.1, módulo B) que, acoplado al sistema actual para ingresar a la institución, permita informar al usuario del número de espacios libres en un instante determinado mediante notificaciones en tiempo real hacia un *smartphone*.

El prototipo consta de algunos elementos necesarios para implementar las diferentes funcionalidades que permitan informar al usuario de la disponibilidad del parqueadero: un módulo Arduino que en conjunto con un servidor local contabiliza el número de vehículos que ingresan/salen del parqueadero de la FIEE; una base de datos que sirve para almacenar la información de los usuarios y del parqueadero; servicios web que permiten acceder y modificar dicha información; y una aplicación móvil que permite consumir los servicios web desarrollados.

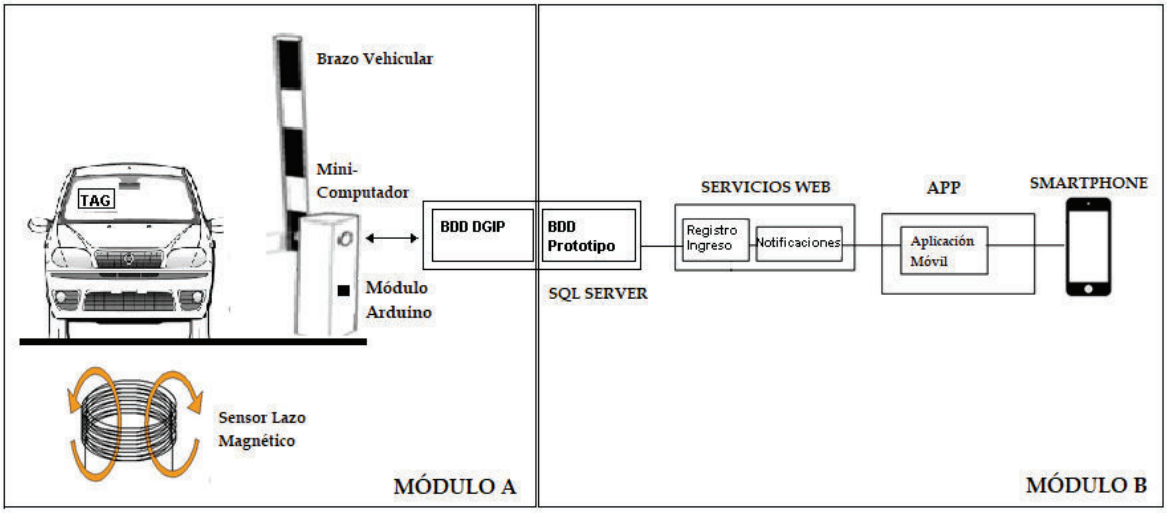

Figura 1.1 Prototipo distribuido de disponibilidad de parqueo.

Además, haciendo uso de herramientas en la nube, el prototipo cuenta con la posibilidad de enviar notificaciones *push* hacia el dispositivo móvil del usuario.

1.1 Objetivos

1.1.1 Objetivo General

Desarrollar un prototipo de sistema distribuido que muestre la disponibilidad de espacios de parqueo en la FIEE mediante el envío de notificaciones en tiempo real.

1.1.2 Objetivos Específicos

- Analizar el sistema actual de parqueadero vigente utilizado por la DGIP, sus componentes, requerimientos y funcionamiento.
- Analizar las tecnologías necesarias para el desarrollo de aplicaciones móviles y el protocolo para notificaciones en tiempo real.
- Diseñar e implementar los elementos y componentes que conformarán el prototipo de sistema distribuido.
- Analizar los resultados de las pruebas de funcionalidad, compatibilidad y aceptación realizadas al prototipo.

1.2 Alcance

La propuesta se realiza en el parqueadero de la FIEE; este acceso posee todos los elementos que componen el sistema de ingreso, el cual brinda acceso a las instalaciones haciendo uso de la tecnología RFID, un minicomputador y un sensor de lazo magnético que se encargan de la apertura y cierre del brazo vehicular luego de que la información del usuario es validada. Los elementos antes mencionados, se encuentran instalados en la FIEE, como se muestra en la Figura 1.2.

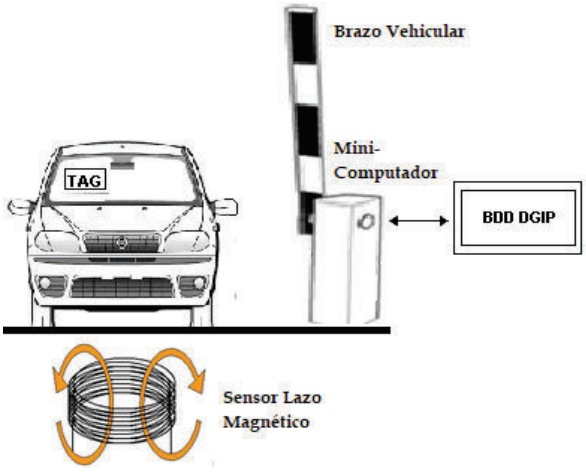

Figura 1.2 Estructura actual del parqueadero de la FIEE.

Además, el parqueadero de la FIEE posee un área delimitada en donde se puede encontrar un número preciso de lugares de parqueo. Esto a diferencia de otras zonas de parqueadero como la Facultad de Petróleos o Sistemas en donde el ingreso se lo realiza por el mismo acceso (Administración) y el área es demasiado extensa para notificar la disponibilidad.

Este Proyecto de Titulación aprovecha las características del módulo Arduino para monitorear la cantidad de vehículos que ingresan/salen del parqueadero de la FIEE. Un servidor local basado en *sockets* se comunica con el módulo Arduino, lee e interpreta la información enviada por el módulo y almacena en la base de datos el número de espacios disponibles.

El prototipo emplea un servidor almacenado en la nube para el envío de notificaciones, y una aplicación para dispositivos móviles que trabaja con los 2 sistemas operativos más populares en la actualidad como son iOS y Android.

La nube utilizada es *Microsoft Azure*, la cual brinda a sus usuarios el servicio de $Notification\ Hubs$[1] mediante el cual es posible enviar notificaciones de inserción (*push*) a cualquier plataforma.

Los requerimientos son agrupados mediante subsistemas los cuales son:

- **Subsistema Arduino**: Mediante Arduino el prototipo monitorea el número de autos que ingresan/salen de la FIEE, el servidor local se comunica con el módulo Arduino, recibe e interpreta los datos que envía el módulo y en ese momento actualiza el valor de los espacios disponibles en la base de datos.
- **Subsistemas de Registro e Ingreso**: Mediante la aplicación móvil los usuarios pueden registrarse en el prototipo agregando un usuario y una contraseña; luego permite al usuario "loguearse" en el prototipo (si el usuario posee un TAG, es decir, si este es un usuario del parqueadero, puede ingresar haciendo de su correo institucional y su número de cédula), caso contrario hace uso de los datos ingresados en el módulo de Registro.
- **Subsistema de Usuario**: Mediante la aplicación móvil los usuarios pueden consultar la disponibilidad del parqueadero, programar él envío de una notificación donde se muestre la disponibilidad y también tienen la opción de comentar el funcionamiento del prototipo.
- **Subsistema de Administración**: El administrador puede verificar la información de los usuarios del prototipo.

[1] Forma de comunicación en la que los usuarios de aplicaciones móviles reciben una notificación.

1.3 Marco Teórico

En el Ecuador el parque automotor crece a un ritmo acelerado tanto que, entre el 2014 y 2015 el número de vehículos aumentó en un 13,1% a nivel nacional, siendo Pichincha la provincia en donde se registra una mayor cantidad de vehículos matriculados [1]. Con este crecimiento se registra una mayor demanda de sitios de parqueo, tanto en espacios públicos como en zonas y parqueaderos privados, situación que se la vive también dentro de la FIEE, en donde usuarios tienen problemas para encontrar un sitio de parqueo especialmente en horas pico; problemas que se traducen en mayores tiempos de desplazamiento, contaminación atmosférica como acústica, problemas como estrés y ansiedad que pueden afectar el estado de ánimo y la salud de los usuarios, entre otros efectos colaterales [2]. Esto obliga en ocasiones a los usuarios a estacionar sus vehículos en lugares prohibidos, e incluso en ocasiones, el usuario se ve obligado a salir del parqueadero buscando otra opción en donde estacionar su automotor.

El presente Proyecto de Titulación desarrolla un prototipo de sistema distribuido de disponibilidad de parqueo, en el cual a través de una aplicación móvil el usuario puede recibir información útil al momento de buscar un espacio de parqueo.

En el presente capítulo se muestra el marco teórico fundamental que es la base para la realización de este proyecto. En primer lugar, se describe el concepto de "*smartcity*" y dentro de este concepto se incluye la definición de "*smartparking*", en donde se resalta la aplicación de las Tecnologías de la Información y Comunicación (TIC's) en problemas cotidianos de las personas como el hecho de encontrar un espacio de parqueo. Luego, se realiza una breve descripción del mercado de teléfonos inteligentes y los sistemas operativos más utilizados actualmente por los usuarios que muestra el por qué se ha escogido los sistemas operativos iOS y Android para el desarrollo del presente trabajo. Posteriormente se muestra una breve descripción de las metodologías ágiles de desarrollo de software y la metodología Kanban que sirve para organizar las tareas creadas. Por último, se describen las herramientas utilizadas para el desarrollo del Proyecto de Titulación agrupadas por capas, las cuales son Capa de Datos, Negocio y Presentación.

1.3.1 Smartparking

La sociedad de la información ha creado en los últimos años términos que empiezan con la palabra "*smart*", como "*smartphones*" o "*smartTV*", para teléfonos, televisores o cualquier otro dispositivo que incorpora la conexión a Internet como parte fundamental para su funcionamiento o para referirse a innovaciones basadas en las tecnologías de la información. Siguiendo esta tendencia aparece el término "*smartcity*" para definir a las

ciudades que hacen uso de la tecnología para mejorar su gestión. Dentro de este ámbito la industria del *parking* utiliza el término "*smartparking*" para englobar el uso de las TIC's en la mejora de la gestión de estacionamientos [3].

Ciudades como New York o Londres han desplegado sistemas basados en el concepto de "*smartparking*" para mejorar la gestión de sus estacionamientos, por ejemplo, en Londres en la ciudad de Westminster, se hace uso de sensores, para detectar si un automóvil está ocupando o no un espacio de parqueo [4]. Este sistema brinda la disponibilidad en tiempo real en más de 40000 espacios de estacionamiento en toda la ciudad, con la ayuda de una aplicación llamada *ParkRight*. Los conductores pueden ver la disponibilidad de un estacionamiento; además mediante esta aplicación un usuario puede acceder a información útil a la hora de hallar y pagar por el servicio de estacionamiento como [5].

- Ubicación.
- Capacidad.
- Longitud y latitud GPS.
- Tarifas.
- Horarios de funcionamiento.
- Detalles que le pueden ayudar al usuario a pagar por el servicio de estacionamiento.

1.3.2 Sistemas Operativos y Smartphones

Un factor a tomar en cuenta en el desarrollo de este prototipo es el creciente mercado de teléfonos móviles. En el primer trimestre del 2017, las ventas mundiales de *smartphones* alcanzaron 380 millones de unidades vendidas como se muestra en la Tabla 1.1.

Tabla 1.1 Ventas mundiales de *smartphones* por proveedor en el primer trimestre del 2017 [6].

Empresa	1er Trimestre 2017 (unidades)	1er Trimestre 2017 Porcentaje Mercado (%)	1er Trimestre 2016 (unidades)	1er Trimestre 2016 Porcentaje Mercado (%)
Samsung	78641	20.7	81186	23.3
Apple	51992	13.7	51629	14.8
Huawei	34181	9.0	28861	8.3
Otros	215131	56.6	186546	53.6
TOTAL	379977	100.0	348224	100.0

Las ventas de *smartphones* se incrementaron en un 9.1% respecto al primer trimestre del año 2016 [6]. Los compradores de teléfonos móviles gastan más dinero para tratar de obtener un teléfono inteligente de mejores características.

Adicionalmente, hoy el mercado de los sistemas operativos móviles es una batalla de 2 participantes, como se muestra en la Tabla 1.2. La plataforma Android de Google y la plataforma iOS de Apple acaparan el mercando con respecto a otras plataformas como Windows o BlackBerry. El crecimiento se debe en gran parte por una mayor aceptación de marcas chinas en los mercados, liderada por teléfonos inteligentes de alta calidad a menores precios.

Tabla 1.2 Ventas mundiales de *smartphones* por sistema operativo en el primer trimestre del 2017 [6].

Sistema Operativo	1er Trimestre 2017 (unidades)	1er Trimestre 2017 Porcentaje Mercado (%)	1er Trimestre 2016 (unidades)	1er Trimestre 2016 Porcentaje Mercado (%)
ANDROID	327163	86.1	292746	84.1
iOS	51992	13.7	51629	14.8
Otro S.O.	8212	0.2	3847	1.1
TOTAL	379977	100.0	348224	100.0

1.3.3 Android

Es un sistema operativo móvil introducido en el mercado por Google, que basa su funcionamiento en Linux y Java. Este sistema tiene como objetivo obtener un modelo estandarizado de programación que simplifica las labores de creación de aplicaciones móviles. Google ofrece una plataforma de desarrollo gratuita, flexible y simple que permite el fácil desarrollo de aplicaciones [7].

El sistema operativo Android se utiliza hoy en día en teléfonos inteligentes, ordenadores, tabletas, televisores, relojes, entre algunos otros dispositivos. Su popularidad es tan grande que se considera en la actualidad el sistema operativo móvil más utilizado en todo el mundo, tal y como reflejan los datos mostrados anteriormente.

Las capas de la arquitectura Android están diseñadas como una pila de componentes, en donde las aplicaciones forman la capa superior, mientras que, por otro lado, el núcleo de LINUX es la capa más baja, como se muestra en la Figura 1.3.

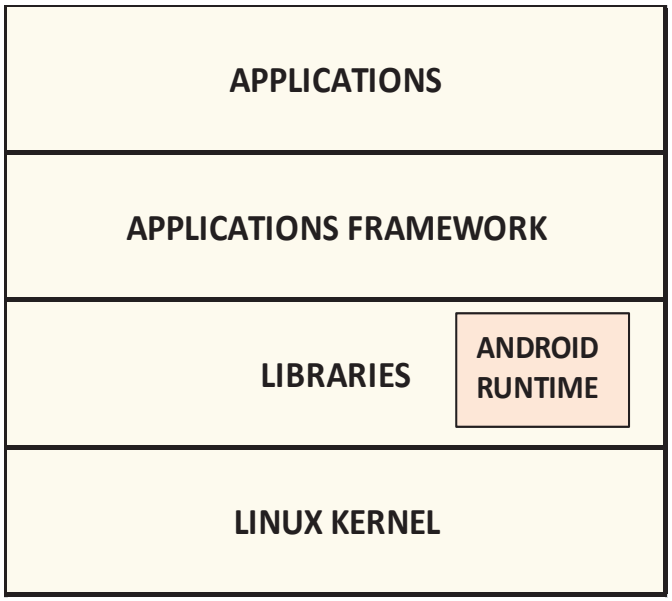

Figura 1.3 Pila de componentes de Android [8].

A continuación, se describe la pila de componentes del sistema operativo Android [8]:

- *Linux Kernel:* Actúa como una capa de abstracción entre el hardware y software; además de proveer los servicios centrales del sistema como gestión de memoria, gestión de procesos, seguridad, red y modelo de controlador.
- *Libraries*: Incluye un conjunto de librerías en C/C++, muchas de ellas de código abierto, librerías que son utilizadas por varios componentes de Android, como por ejemplo la librería *Media Framework* que soporta *codecs* utilizados para reproducción y grabación multimedia.
- *Application Framework:* Los desarrolladores poseen acceso completo al *framework* de las aplicaciones ya que la arquitectura está diseñada para reutilizar componentes. Las funcionalidades de una aplicación pueden ser publicadas y luego ser utilizadas por cualquier otra aplicación sin ningún problema o restricción.
- *Aplications:* Android incluye un conjunto de aplicaciones básicas desarrolladas con lenguaje JAVA como SMS, correo, navegador entre otras. Es en esta capa en la que los desarrolladores añaden sus aplicaciones.

1.3.4 IPhone OS (iOS)

Es el sistema operativo de Apple para dispositivos móviles, conocido más comúnmente como iOS. El sistema operativo de Apple está conformado por un modelo de 4 capas [9], como se muestra en la Figura 1.4.

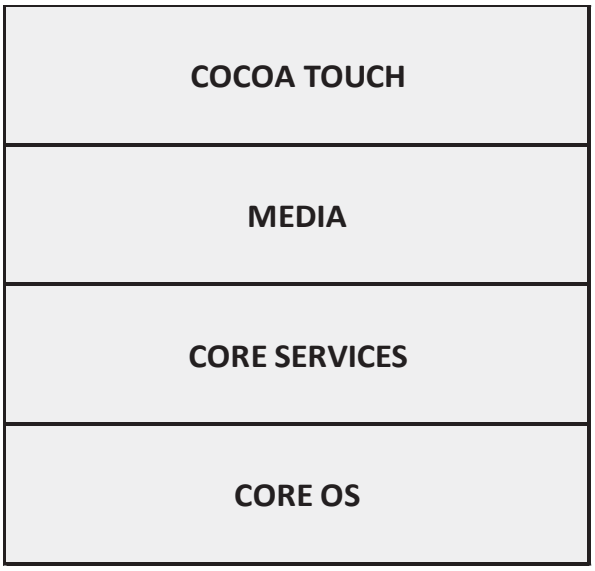

Figura 1.4 Arquitectura iOS [9].

A continuación, se describe brevemente las capas del sistema operativo iOS [10]:

- *Cocoa Touch:* Capa superior en donde el usuario interactúa con las aplicaciones. Es una capa de abstracción de los componentes visuales.
- *Media:* Capa que contiene tecnologías que brindan acceso a ficheros multimedia (audio, video, gráficos, etc.) basado en una mezcla del lenguaje C y *Objetive C*.
- *Core Services:* Capa compuesta a su vez por 2 subcapas:
  - *Core Fundation:* Subcapa que contiene librerías que facilitan el manejo de colecciones, flujos, entre otras.
  - *CFNetwork:* Subcapa que se centra en el uso de protocolos como http, ftp, etc. En el *Core Services* se ofrecen servicios principales como acceso a la red, base de datos, servicios que se encuentran disponibles para todas las aplicaciones.
- *Core OS*: Núcleo en donde se encuentran elementos de seguridad, procesos, entrada y salida estándar, memoria, manejo de ficheros, etc.

1.3.5 Metodologías Ágiles de Desarrollo de Software

En el año 2011 varios críticos de los modelos de mejora de desarrollo de software se reunieron para tratar sobre técnicas y procesos para el desarrollo. El término “Métodos Ágiles” se adoptó como resultado de esa reunión; dicho término define métodos que en ese entonces sugerían una alternativa de desarrollo a las metodologías tradicionales. Se obtuvieron cuatro postulados que describen los principios en los que los nuevos métodos basan su funcionamiento. Estos postulados se conocen como “Manifiesto Ágil” [11]. Dichos postulados se muestran en la Tabla 1.3.

Tabla 1.3 Postulados del Manifiesto Ágil [11].

Postulados del Manifiesto Ágil
Se valora a los individuos y las interacciones sobre procesos y herramientas.
Se valora las aplicaciones que funcionan sobre la documentación exhaustiva.
Se valora la colaboración del cliente sobre la negociación contractual.
Se valora la respuesta al cambio sobre el seguimiento del plan.

La alianza ágil elaboró además un conjunto basado en principios comunes respecto de las metodologías de desarrollo ágil de software [11], principios que dan una guía para el desarrollo ágil de software y que se muestran en la Tabla 1.4.

Tabla 1.4 Principios respecto de metodologías de desarrollo ágil de software [11].

Principios de las metodologías de desarrollo ágil de Software
La prioridad es satisfacer al cliente con una entrega temprana y continua del software.
Requisitos cambiantes son aceptados, los procesos ágiles se adaptan al cambio.
Entregar con frecuencia software que funcione, con preferencia en periodos breves.
Cliente y desarrollador trabajan conjuntamente de manera cotidiana a lo largo del proceso.
Construcción de proyectos que giren en torno a individuos motivados.
La mejor forma de comunicar información dentro de un equipo de trabajo es mediante la conversación cara a cara.
Software funcional es la principal medida del progreso.
Procesos ágiles promueven el desarrollo sostenido.
Atención continua a la excelencia técnica aumenta la agilidad.
Simplicidad como herramienta para maximizar la cantidad de trabajo.

Cabe destacar que estos enunciados no constituyen un proceso formal para el desarrollo de software, representan ideas o guías principales a seguir si se escoge esta metodología.

a) Metodología Kanban

Kanban (en Kanji, kan significa “visual”, y ban significa “tablero”) es una herramienta de desarrollo que proviene de la filosofía *Lean*, de tipo “*pull*”, lo que significa que los recursos deciden cuando y cuanto trabajo se comprometen a realizar [12].

Esta metodología de desarrollo se basa en la optimización de procesos y enfatiza la respuesta al cambio por sobre seguir un plan, permitiendo así obtener una respuesta más rápida sobre otras metodologías como Scrum. Además, Kanban es una metodología que se adapta a los requerimientos cambiantes del cliente y los avances del producto son entregados periódicamente de forma incremental [12].

Kanban al igual que otras metodologías de desarrollo ágil como Scrum, comparten la idea de crear un *backlog* (bloque de tareas) del producto mediante una serie de tareas priorizadas. Kanban visualiza el flujo de trabajo (*workflow*), es decir, divide el trabajo en tareas y controla el trabajo en progreso (*work in progress*) de tal manera que, cuando existe poco trabajo en progreso se añade la tarea con mayor prioridad del *backlog* a producción. Kanban mide el tiempo de ciclo (*lead time*) o tiempo que toma completar una tarea [12]. Las características de esta metodología de desarrollo se muestran en la Tabla 1.5.

Tabla 1.5 Principales características de la metodología Kanban [13].

CARACTERÍSTICA	DESCRIPCIÓN
Visualiza el “*Workflow*”	Escribe cada bloque en una tarjeta y se la coloca en el tablero. Utiliza columnas con nombres para ilustrar en qué etapa dentro del proceso se encuentra cada tarea.
Limita el trabajo en curso (*work In progress*)	Asigna límites referentes a cuantas tareas pueden ser procesadas a la vez en cada estado del flujo de trabajo.
Mide el “*Lead Time*”	Mide el tiempo promedio para completar una tarea o el tiempo en que la tarea pasa por todas las columnas del *workflow*. El objetivo de Kanban es lograr hacer lo más pequeño posible el *lead time*.

Esta metodología basa su funcionamiento en dos premisas esenciales [13]:

- Lograr un producto de calidad.
- Hacer que cada fase del proyecto termine correctamente.

Kanban es considerada una excelente herramienta de visualización; mediante el "Kanban *Board*" o tablero Kanban; esta metodología proporciona una visibilidad completa del proceso, en donde muestra el trabajo que va a ser organizado mediante tareas. Permite comunicar claramente las prioridades de las actividades y resalta posibles "cuellos de botella" que puedan presentarse durante los diferentes procesos dentro del desarrollo e implementación del prototipo de disponibilidad de parqueo. En la Figura 1.5 se muestra un ejemplo de un tablero Kanban.

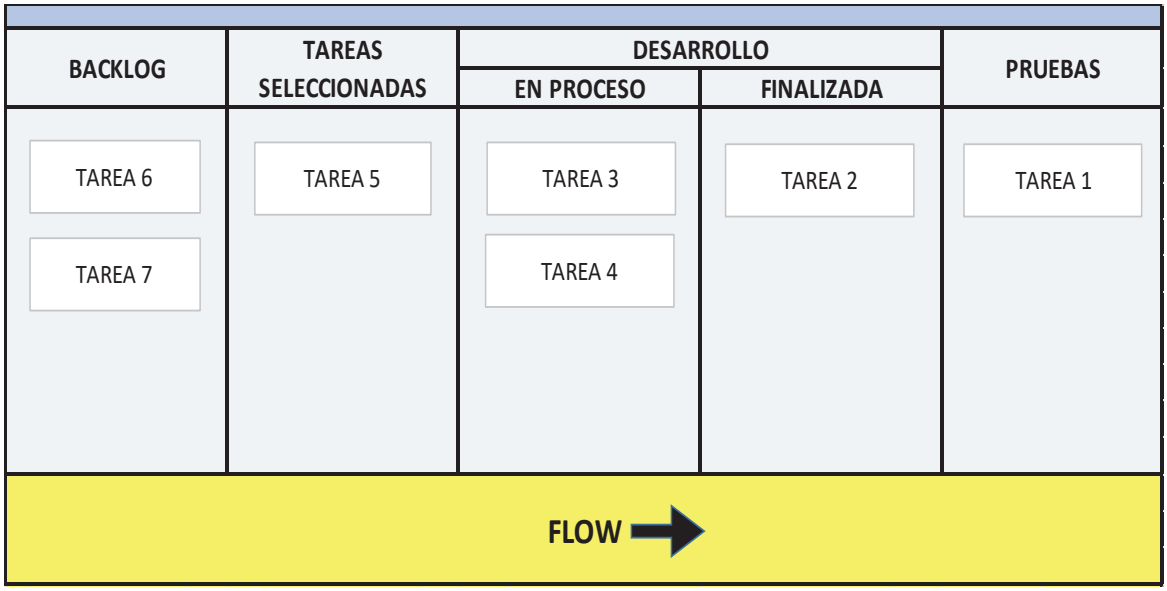

Figura 1.5 Pizarra o Tablero Kanban.

Cada columna representa un determinado estado dentro del *workflow*, la cantidad de columnas son determinadas de acuerdo a la experiencia del desarrollador y a las necesidades del proyecto; no hay un número de columnas ni estados determinados dentro de esta metodología [14].

En Kanban es fundamental colocar prioridades en las tareas y es necesario delimitar el límite de trabajo en curso (*work in progress*), es decir, se requiere conocer el número de tareas que se puede realizar en cada fase ya que existe una cantidad de trabajo óptima que se puede realizar sin llegar a sacrificar eficiencia. A esto adicional se le suma la idea de que, para empezar una nueva tarea, otra tarea previa debió haber sido finalizada.

Algunas de las ventajas de utilizar Kanban se muestran en la Tabla 1.6.

Tabla 1.6 Ventajas de la metodología Kanban.

Ventajas de la metodología Kanban
Genera tiempos de entrega más cortos.
Ayuda a priorizar los distintos procesos.
Permite la visibilidad de “cuellos de botella” en tiempo real, con esto se puede optimizar recursos para completar determinada tarea. En el tablero se puede visualizar un “cuello de botella” cuando la columna A está llena y la columna B se encuentra vacía.
Kanban es útil cuando los intervalos de tiempo para avanzar en la producción de un producto no hacen mucho sentido en proyectos con alto rango de variabilidad.
Es fácil ver cómo se desarrolla el proyecto con solo ver el tablero.
Permite detectar productos o tareas defectuosas

Es importante recalcar que Kanban representa un modelo de desarrollo orientado a manejar *workflow*, y aunque no remplaza a ninguna metodología existente, Kanban puede complementarlas.

b) Comparación entre metodología de desarrollo Scrum y Kanban

Entre las metodologías actuales más populares para desarrollo ágil de software se puede hallar Scrum, una metodología similar a Kanban en ciertos aspectos y que brinda un conjunto de buenas prácticas para desarrollar proyectos de una forma colaborativa. Siguiendo la Tabla 1.7, se comparan las metodologías Kanban y Scrum, y se detalla el por qué se ha seleccionado Kanban sobre otras metodologías para el desarrollo del prototipo. Esta comparación se realiza en base a la evaluación de tres parámetros [15]:

- Vista global del sistema como algo cambiante.
- Colaboración entre miembros del equipo.
- Otras más específicas como simplicidad, adaptabilidad, colaboración, entre otras.

Para analizar estas metodologías se puntúan los parámetros de evaluación con calificaciones del 1 al 5, donde 1 indica que la metodología no cumple con esa especificación y 5 indica que la metodología si cumple con el requisito establecido.

Tabla 1.7 Scrum vs Kanban [16].

PARÁMETRO	Scrum	Kanban
Sistema Cambiante	5	5
Resultados	5	5
Simplicidad	5	5
Adaptabilidad	4	5
Prácticas de Colaboración	4	4
TOTAL	23	24

Como se puede observar, se seleccionó Kanban gracias a su adaptabilidad, ya que a lo largo del desarrollo del prototipo se tuvo que ir acoplando los componentes entre sí y además incluir estos componentes al sistema vigente instalado ya en la FIEE. Kanban se adapta de mejor manera a los cambios e imprevistos que en el transcurso del desarrollo se presentaron.

Con la organización de tareas mediante el tablero Kanban se tiene una visión completa de las actividades, el estado de cada una dentro del desarrollo del prototipo, y los posibles problemas que se detectaron tanto en la fase de implementación como en las pruebas del producto. Kanban se adaptó perfectamente a las necesidades y requerimientos del prototipo por lo que fue la herramienta utilizada para el diseño e implementación de los componentes del prototipo.

1.3.6 Herramientas, Tecnologías y Lenguajes por utilizar en la Capa de Datos

a) SQL Server

Las bases de datos son el método preferido para el almacenamiento estructurado. Hoy en día teléfonos móviles, agendas electrónicas, aplicaciones, usan bases de datos para asegurar la integridad de la información. Por sus siglas en inglés *Structured Query Language (SQL); SQL Server* [17] es un sistema de administración y análisis de bases de datos de *Microsoft*, diseñado para la función de almacenar y consultar datos según sea necesario por otras aplicaciones. La Tabla 1.8, muestra varias tecnologías de administración y análisis de datos que *SQL Server* ofrece.

Tabla 1.8 Tecnologías presentes en *SQL Server* [17].

CARACTERÍSTICA	TAREA
Motor de base de datos	Servicio principal para almacenar, procesar y proteger datos. Brinda acceso controlado y rápido procesamiento de transacciones para cumplir con requisitos de aplicaciones consumidoras de datos.
Servicio de calidad de datos (DQS)	Proporciona una solución de limpieza de datos. Permite generar una base de conocimiento y usarla para realizar tareas de corrección y eliminación de datos duplicados.
Servicio de Análisis	Conjunto de herramientas que brindar modelado tabular y análisis en grandes volúmenes de datos,
Servicio de integración	Genera soluciones de integración de datos de alto rendimiento, incluye paquetes para procesamiento de extracción, transformación y cargar para almacenamiento de datos.
Replicación	Copia y distribución de datos y objetos de una base a otra.

SQL Server trabaja con una Interfaz Gráfica de Usuario (GUI) y con software basado en comandos brindando un entorno amigable para el usuario. Además, soporta el lenguaje SEQUEL, el cual es un lenguaje diseñado para administrar sistemas de gestión de bases de datos relacionales [18].

El lenguaje SEQUEL es un lenguaje de consulta estructurado, basado en el modelo relacional, en donde la estructura de almacenamiento de los datos son las relaciones, mientras que el modelo relacional brinda las bases teóricas de la base de datos; es el lenguaje SQL el que apoya y facilita la implementación física de la base.

SQL es el lenguaje estándar ANSI/ISO para la creación, manipulación y control de bases de datos relacionales mediante una serie de instrucciones propias del lenguaje.

Un método común para categorizar las instrucciones SQL es dividirlas según las funciones que realizan [19], estas funciones de muestran en la Tabla 1.9.

Tabla 1.9 Instrucciones SQL [19].

INSTRUCCIÓN	DEFINICIÓN
DDL *Data definition Language*	Se usan para crear, modificar o borrar registros de una base de datos como tablas, vistas, entre otras. Las palabras clave asociadas con las instrucciones DDL son: *CREATE, ALTER y DROP*
DCL *Data Control Language*	Estas Instrucciones permiten controlar quien o que tiene accesos a los registros en la base de datos. Mediante instrucciones puede otorgar (*GRANT*) y restringir (*REVOKE*) el acceso.
DML *Data Manipulation Language*	Estas instrucciones se usan para recuperar, agregar, modificar o borrar datos guardados en los registros de una base de datos mediante las instrucciones *SELECT, INSERT, UPDATE y DELETE.*

Dadas las opciones que brinda *SQL Server* y además, considerando el hecho de que *SQL Server* 2014 es la base de datos utilizada por la DGIP, se ha seleccionado esta base de datos para el desarrollo del presente Proyecto de Titulación. El lenguaje SQL es el mecanismo para realizar consultas y manipular registros de la base de datos.

1.3.7 Herramientas, Tecnologías y Lenguajes por utilizar en la Capa de Negocio

a) Arduino

El módulo Arduino es la placa utilizada dentro del prototipo para el conteo de vehículos que ingresan/salen de la FIEE trabajando en conjunto con el sensor de lazo magnético ya instalado en la FIEE. Arduino es una plataforma de prototipos electrónicos de código abierto basada en hardware y software de fácil uso. Mediante Arduino es posible “sentir” el entorno mediante la lectura de entradas provenientes de un sin número de sensores y puede afectar el exterior mediante el control de ciertos artefactos como luces, motores, etc.

Arduino permite simplificar el uso de los microcontroladores y ofrece ciertas ventajas que se listan a continuación [20]:

- **Multiplataforma:** Las placas de Arduino se ejecutan en dispositivos Windows/Mac/Linux.
- **Código Abierto:** El software de Arduino está publicado como herramientas de código abierto.
- **Hardware y Software extensible:** Basado en microcontroladores ATMEGA8 y ATMEGA168 de Atmel, sus planos están publicados bajo licencia *Creative Commons*, de manera que un desarrollador podría construir su propio módulo. De igual manera el lenguaje puede ser expandido mediante librerías C++.
- **Económico:** Las placas de Arduino son relativamente baratas en relación a otros microcontroladores. Las placas más costosas de Arduino no superan los 50 dólares.

La placa Arduino Uno es la placa utilizada en el presente Proyecto de Titulación como se muestra en la Figura 1.6. Esta placa [21] está basada en el controlador $\mathrm{ATmega328}$ [2] que ofrece:

- Voltaje de operación: 5V
- Voltaje Entrada (recomendado) 7-12V, Voltaje Entrada (límite) 6-20V
- 14 Pines digitales de entrada/salida (6 pueden usarse como salidas PWM).
- 6 entradas analógicas.
- Corriente DC por entrada/salida: 40 mA
- *Flash Memory*: 32 KB
- SRAM: 2 KB
- EEPROM: 1 KB
- Velocidad de reloj: 16 MHz

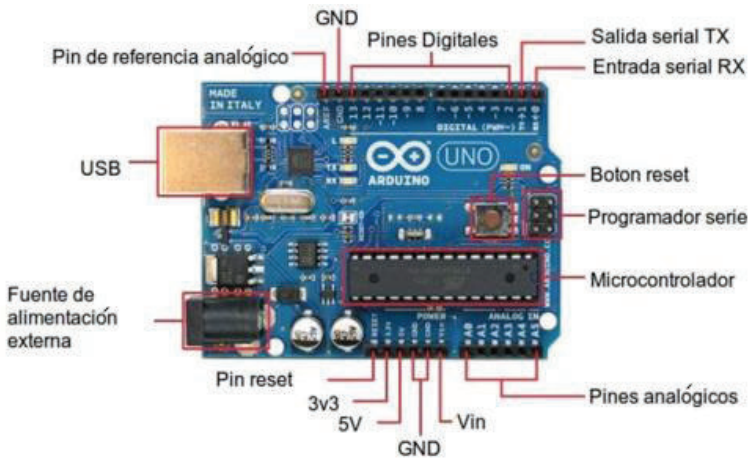

Figura 1.6 Arduino UNO.

[2] ATMEGA microcontroladores AVR de memoria *flash* programable.

b) Sockets

Los *sockets* son mecanismos de comunicación entre procesos que permiten que un proceso se comunique con otro, incluso si dichos procesos se encuentran en diferentes dispositivos. El paso de mensajes se basa en dos operaciones esenciales que son: send() y receive() [22]. Para que un mensaje pueda ser enviado a otro proceso es necesario especificar 2 parámetros fundamentales que son:

- Dirección IP
- Número de puerto

Un proceso no puede compartir un puerto con otros procesos. Un PC tiene 2^{16} posibles puertos que se los puede agrupar de la siguiente manera:

- Puertos bien conocidos (<1024)
- Puertos registrados (1024 - 49151)
- Puertos dinámicos (privados > 49151)

Los *sockets* basan su funcionamiento en la arquitectura Cliente/Servidor. En esta arquitectura uno de los procesos debe estar iniciado y a la espera de que alguien establezca conexión con él. A este proceso se le conoce como “servidor”. Por otro lado, el proceso “cliente” es iniciado por el usuario y este solicita conectarse con el “servidor” y hacerle alguna petición. La comunicación basada en *sockets* se muestra en la Figura 1.7.

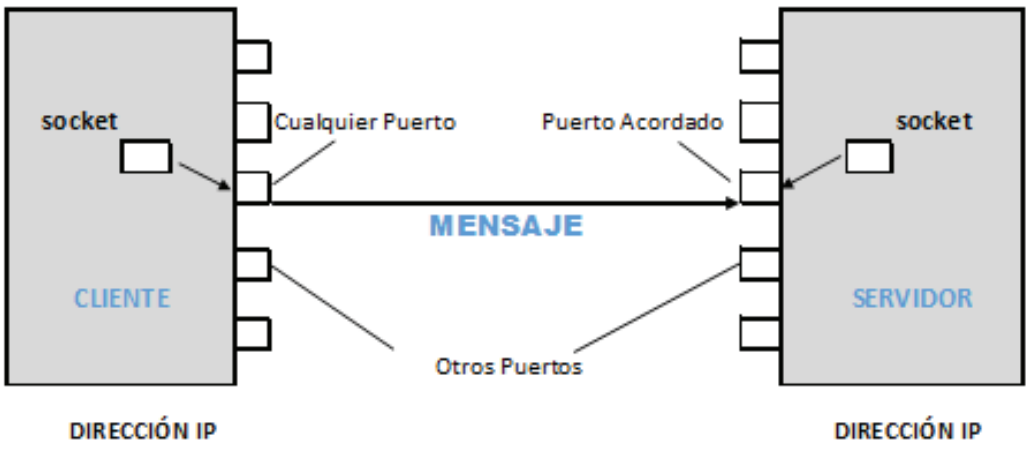

Figura 1.7 *Sockets*.

TCP como UDP hacen uso de abstracciones basadas en *sockets*. En la Tabla 1.4 se muestran las principales características de los *sockets* basados en TCP y UDP [23].

Tabla 1.10 Principales características de los *sockets* TCP y UDP

Sockets TCP	Sockets UDP
Son un servicio orientado a conexión. Requiere de un establecimiento de conexión entre *sockets*.	Son un servicio no orientado a conexión. Más eficiente que TCP pero no garantiza fiabilidad.
Una vez que los *sockets* están conectados es posible enviar un flujo de datos en ambas direcciones.	Los datos se envían y reciben en paquetes cuya entrega no está garantizada. Cada vez que se envíe un datagrama es necesario añadir información del *socket* local y remoto.
Usa acuses de recibo, y control de flujo.	Se envían datagramas sin acuse de recibo ni reintentos
Tiene sobrecargas ya que asegura la entrega confiable de datos.	No padece de sobrecargas asociadas a la entrega garantizada de mensajes.

De lo presentado anteriormente, los *sockets* basados en TCP son empleados en el presente Proyecto de Titulación para implementar un servidor local que permite establecer comunicación, recibir e interpretar la información que el módulo Arduino envíe; permite mediante un flujo de datos conocer el número de vehículos que ingresan/salen de la FIEE (evitando perder información) y obtener así el número de espacios disponibles.

c) Servicios Web

Un servicio web se define como una clase que permite que sus métodos sean llamados (invocados) por métodos en otras máquinas (desde una intranet o desde cualquier lugar de Internet) haciendo uso de varios formatos y protocolos (XML, JSON).

El mecanismo de invocación es totalmente independiente de la plataforma y el lenguaje que se utilice para implementar el servicio. Los servicios web ofrecen información con un formato estándar que puede ser entendido fácilmente por una aplicación.

Cada servicio tiene 3 componentes fundamentales [24]:

- Dirección: Ubicación (*end point*) del servicio.
- Asociación (*Binding*): Específica cómo se comunica el cliente con el servicio.
- Contrato: Es una Interfaz que representa los métodos del servicio y sus tipos de retornos. Mediante el contrato el cliente puede interactuar con el servicio.

Por sus siglas en inglés, SOAP (*Single Object Access Protocol*) y REST (*Representation State*) son siglas asociadas a estándares dedicados al diseño y desarrollo de servicios web. SOAP es un protocolo para el intercambio de mensajes sobre una red de computadoras que usa http y está basado en XML (lo que facilita la lectura, pero hace que los mensajes sean más largos). Por otro lado, REST es considerado un estilo de arquitectura de software que hace énfasis en los recursos del sistema, como acceder a ellos y cómo estos recursos se trasfieren hacia los clientes.

La arquitectura REST hace uso de los mecanismos web tradicionales de solicitud/respuesta y no envuelve los datos en un mensaje especial como lo hace SOAP. Se escogió la arquitectura REST puesto que brinda ciertas ventajas sobre SOAP [25], ventajas que se detallan en la Tabla 1.11.

Tabla 1.11 Principales características de los servicios web REST [25].

Servicios Web REST	
Características	• Mejores tiempos de respuesta y disminución de carga en el servidor. • Se integra mejor con HTTP ya que no requieren mensajes XML como SOAP. • Poseen una infraestructura ligera que permite que los servicios se implementen con el uso mínimo de herramientas. • El desarrollo REST resulta barato y su adopción es sencilla.

Por las ventajas antes mencionadas, los servicios web desarrollados para el prototipo distribuido de disponibilidad de parqueo se implementan mediante la arquitectura REST. El funcionamiento de la arquitectura REST se muestra en la Figura 1.8.

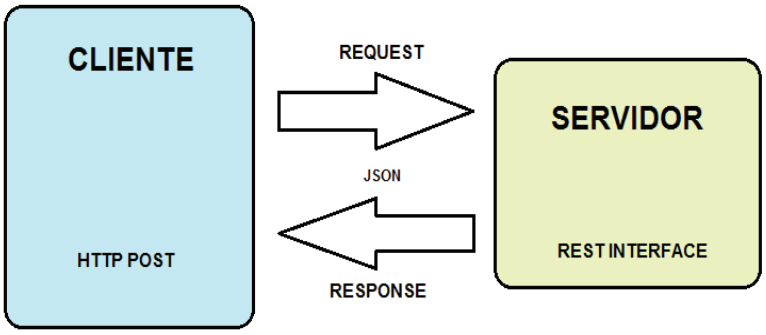

Figura 1.8 Arquitectura REST.

d) Microsoft *Visual Studio*

Visual Studio es un conjunto completo de herramientas de desarrollo que permite la generación de aplicaciones como: ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles.

Lenguajes de programación como *Visual Basic, Visual C# y Visual C++,* utilizan el mismo Entorno de Desarrollo Integrado (*IDE*), que permite el uso compartido de herramientas y facilita la creación de soluciones en diferentes lenguajes [26].

Visual Studio proporciona un *IDE* amigable con el usuario, y añade todas las características para Android, iOS, Windows, la Web y la nube [27]. Algunas de las características más relevantes de *Visual Studio* se muestran en la Tabla 1.12.

Tabla 1.12 Características del IDE de *Visual Studio* [27].

Característica	Detalle
Compilar aplicaciones inteligentes más rápido.	*Visual Studio* presenta características como navegación de código, *IntelliSense*, refactorización y corrección de código que permite ahorrar tiempo y esfuerzo.
Buscar y corregir errores más pronto.	Depuración y pruebas para detectar problemas y errores lo antes posible. Nuevas características han sido agregadas para ayudar a los programadores.
Integración con la nube.	Integración completa con aplicaciones de Azure, .NET Core, entre otras.
Colaboración más eficiente.	Permite administrar proyectos de equipo que hospede cualquier proveedor como Visual Studio *Team Services* o *GitHub*.
Entrega de aplicaciones móviles de calidad.	Se aceleran y simplifican los procesos de compilación, conexión y ajuste de aplicaciones.

Dichas características fueron aprovechadas para el desarrollo de los servicios web del presente Trabajo de Titulación.

e) Microsoft Azure, Notificaciones *Push*

Microsoft Azure es un conjunto de servicios y soluciones en la nube, en donde el gran éxito alcanzado es debido al enfoque de que todo estuviera en la nube, servicios para desarrolladores, productos de usuarios e inclusive su propia arquitectura interna [28].

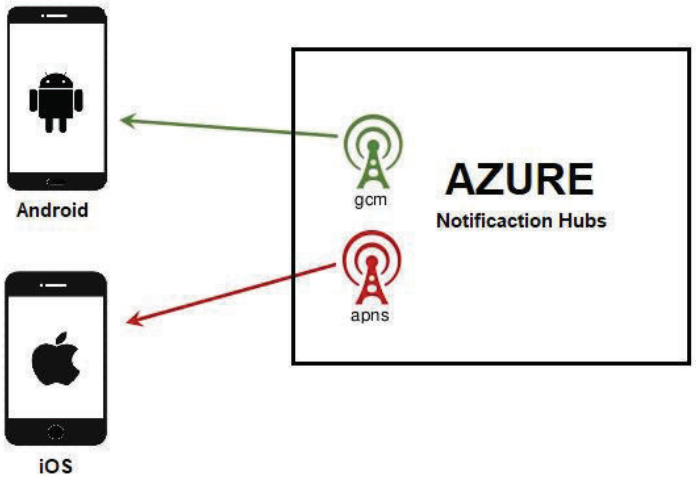

Figura 1.9 Microsoft Azure.

Dentro de la gama de servicios que *Azure* brinda a sus usuarios se encuentra el servicio de *Notification Hubs* mediante el cual es posible enviar notificaciones de inserción (*push*) a cualquier plataforma. A continuación, se muestran algunas de las características más relevantes al trabajar con el servicio de *Notification Hubs* de *Azure* [29].

- Permite llegar a plataformas importantes como iOS, Android, Windows, entre otras.
- Puede ser utilizado con cualquier *back-end* (servidor), en la nube o localmente.
- Permite llegar de una manera dinámica a un usuario o a un segmento de clientes.

Por lo antes descrito, el servicio *Notification Hubs* de *Azure* es el servicio utilizado para implementar las notificaciones acerca de la disponibilidad del parqueadero de la FIEE. Este es un servicio que permite, a través de mensajes cortos, la comunicación entre una aplicación para dispositivos móviles (*smatphones* y *tablets*) y los usuarios que tengan instalada dicha aplicación en sus dispositivos, haciendo uso de Internet como vía de comunicación [30].

Dentro de este concepto *Microsoft Azure* brinda un motor de notificaciones que permite enviar notificaciones hacia dispositivos móviles que funcionan con diferentes sistemas operativos. Además, los proveedores ofertan ciertos servicios para que sus plataformas puedan hacer uso de notificaciones *push*, por ejemplo, Google brinda el servicio GCM (*Google Cloud Messaging*) y Apple brinda el sistema *Apple Push Notification Services* (APNS).

En este sistema, es el emisor es quien inicia la comunicación tal y como se muestra en la Figura 1.10, evitando así que el usuario realice consultas al servidor para que éste responda, como se lo hace generalmente en los sistemas tradicionales Cliente-Servidor.

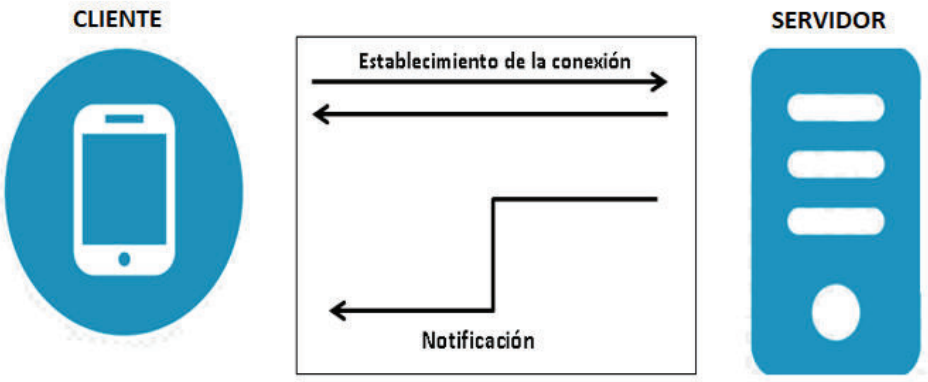

Figura 1.10 Esquema de notificaciones *push*.

Por otro lado, la recepción de la notificación es instantánea e inmediata y no requiere que la aplicación en el destino esté en ejecución, es el software del sistema operativo del dispositivo móvil quien "despierta a la aplicación", activando los mecanismos necesarios para que se realice la comunicación.

Sin embargo, en este sistema, para que se realice la comunicación se requiere que el dispositivo móvil del usuario se encuentre encendido, y debe tener conexión a Internet. Además, el dispositivo no debe tener bloqueadas las notificaciones en la configuración del sistema. Todos los dispositivos móviles, *smartphones* y *tablets*, proveen sistemas para realizar notificaciones *push*. En la Tabla 1.13 se detallan las características de las notificaciones *push* [30]:

Tabla 1.13 Características de las notificaciones *push*.

Características de las notificaciones *push*
Notificaciones instantáneas.
Notificaciones gratuitas. Es posible enviar notificaciones sin pagar ningún costo adicional.
Comunicación directa entre el desarrollador y los usuarios que han instalado la aplicación sin necesidad de conocer el número telefónico.
Selectividad de usuarios, las notificaciones llegan a los usuarios que tienen la app instalada y han aceptado los términos de uso.

f) Notificaciones Push en Sistemas Basados en Android

Para recibir notificaciones *push* en dispositivos basados en Android, Google brinda el servicio GCM (*Google Cloud Messaging*); servicio que permite enviar y recibir información desde el proveedor de una aplicación móvil a un dispositivo que trabaje con Android y que tenga instalada dicha aplicación [31].

El servicio GCM provee todos los mecanismos para enviar y recibir notificaciones *push*. Una vez llegada la notificación al dispositivo, es la misma aplicación la que se encarga de procesarla; en este caso no lo realiza el sistema operativo. Esto tiene como ventaja principal que al ser la misma aplicación la que tramita la notificación, el desarrollador tiene la libertad de decidir el tratamiento o la respuesta que se le da a esta notificación. Este proceso por otro lado es automático en iOS.

El envío de mensajes de texto es la respuesta más habitual en el uso de notificaciones *push*, este mensaje consiste de tres bloques principales: *token* del dispositivo, opciones de envío y carga útil. El *token* representa un identificador único del dispositivo; las opciones de envío son opciones configurables para la comunicación; y la carga útil es la información a comunicar al usuario. En todos los bloques la información se representa usando el formato JSON, formato para el intercambio de datos.

Al momento de recibir las notificaciones, estas adoptan dos estilos de presentación, pudiendo ser presentadas como tiras y alertas como se observa en la Figura 1.11. Una tira es una notificación que aparece en la parte superior del dispositivo a manera de noticia, mientras que las alertas brindan mayor interacción con el usuario, donde se tienen dos botones que permiten ver o cancelar la notificación.

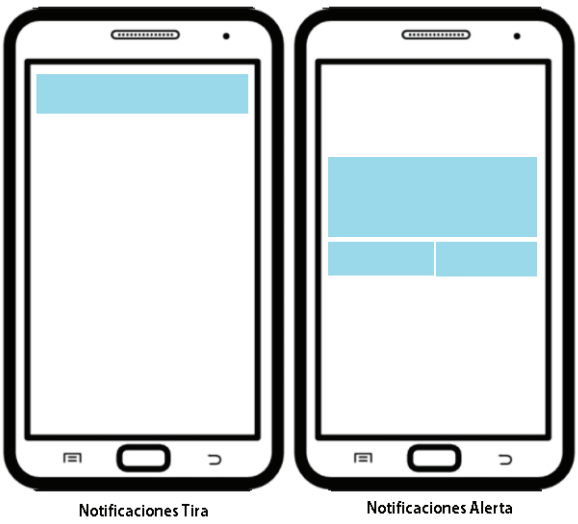

Figura 1.11 Notificaciones *push* en Android.

g) Notificaciones *Push* en Sistemas Basados en iOS

Para enviar notificaciones push en dispositivos basados en iOS, Apple brinda el sistema *Apple Push Notification Services* (APNS), el cual funciona como un servicio intermediario en la comunicación y el envío de notificaciones encargándose del transporte y enrutamiento de la notificación al dispositivo final. APNS ofrece un servicio robusto y eficiente con conexiones seguras [31]. En iOS una notificación *push* representa un mensaje corto que consta de dos bloques; el *token* que es similar a un número telefónico para poder ubicar al dispositivo y la carga útil de envío que es la información que será notificada al usuario.

Al momento de recibir las notificaciones, al igual que Android, éstas adoptan dos estilos de presentación: tiras y alertas como se observa en la Figura 1.12.

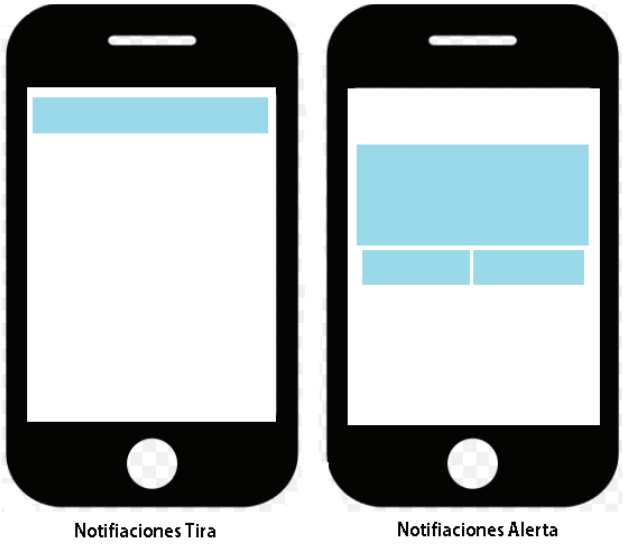

Figura 1.12 Notificaciones *push* en iOS.

Las notificaciones p*ush* fueron seleccionadas en el presente Proyecto de Titulación para enviar notificaciones al usuario e informar de la disponibilidad del parqueadero en la hora establecida por el usuario.

1.3.8 Herramientas, tecnologías y lenguajes por utilizar en la Capa de Presentación

a) Plataforma de Desarrollo de Aplicación Móvil

Microsoft Visual Studio desde el año 2015 ofrece *Xamarin*, que es una plataforma para el desarrollo de aplicaciones móviles, que permite compilar aplicaciones para sistemas operativos Android, iOS y Windows a partir de una base común de código C#.

La arquitectura de *Visual Studo Xamarin* se muestra en la Figura 1.13.

- ***Shared C# App Logic**:* Representa la lógica compartida desarrollada en el lenguaje C#. Esta capa representa un proyecto compartido, que contiene los componentes de las Capas de Negocio, Servicios y Datos.
- ***Shared C# User Interface Code*:** Capa utilizada para compartir la interfaz de usuario y el código de fondo entre plataformas, de esta manera se ofrece una experiencia de interfaz de usuario totalmente nativa.
- ***Specific Plataform Layer:*** Permite el acceso a los recursos específicos de cada plataforma. En esta sección es posible trabajar con diferentes sistemas operativos como Android, iOS y Windows.

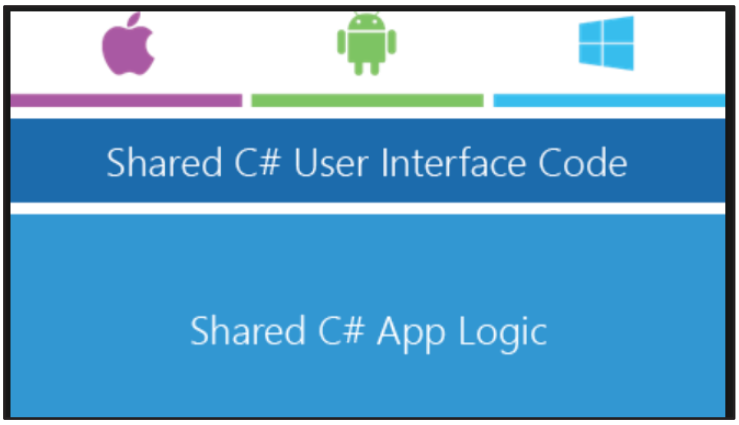

Figura 1.13 Arquitectura *Xamarin* [32].

Xamarin con *Visual Studio* permiten trabajar en un ambiente multiplataforma reutilizando código compartido basado en el lenguaje C#. Algunas de las ventajas y desventajas de utilizar *Xamarin*, se muestran en la Tabla 1.14.

Tabla 1.14 Características de *Visual Studio Xamarin* [32].

VENTAJAS	DESVENTAJAS
Combina la tecnología móvil con C#	Muestra cierta dificultad para ejecutar proyectos nuevos como *Xamarin Forms*
Plataforma compatible con Android, iOS y Windows.	Existen errores al descargar paquetes *Nuget* (extensiones que permite agregar referencias a librerías)
Es posible conseguir una reutilización entre 75-100 % de código.	Las actualizaciones pueden causar errores en los proyectos ya compilados.
Aplicaciones escritas con *Xamarin* disponen de acceso completo a las API de plataformas subyacentes.	Se requieren conocimientos sólidos para utilizar y agregar librerías

Xamarin ofrece muchos beneficios al momento de desarrollar aplicaciones, la documentación que se encuentra en línea es completa y permite a los desarrolladores conocer *Xamarin* de una manera sencilla; errores que pudieran presentarse son errores comunes y que pueden ser consultados en los foros de *Xamarin*.

Desarrolladores que han trabajado con C# no tienen la necesidad de aprender lenguajes nativos; mediante la capa de lógica compartida basada en lenguaje C# un desarrollador en *Xamarin* puede compilar aplicaciones para diferentes sistemas operativos [33].

Por otro lado, el lenguaje C# es un lenguaje de programación diseñado a mediados de los años 80. La intención de su creación fue el extender al exitoso lenguaje de programación C, con mecanismos que permitieran la manipulación de objetos. Este lenguaje toma las mejores características de lenguajes preexistentes como *Visual Basic*, *Java* o C++ combinándolos en un solo lenguaje. A continuación, se muestran de manera resumida algunas de las principales características del lenguaje C# [34]:

- **Orientado a Objetos:** C# soporta las características propias del paradigma de Programación Orientada a Objetos, como encapsulación, herencia, polimorfismo entre otras.
- **Gestión automática de memoria:** Incluye un recolector de basura que posibilita la destrucción de objetos que no sean utilizados, este recolector entra en funcionamiento cuando no se tiene memoria para la creación de nuevos objetos.
- **Seguridad de tipo:** C# brinda mecanismos que permiten asegurar que los accesos a tipos de datos se realicen correctamente como por ejemplo conversiones entre tipos de datos compatibles.
- **Instrucciones seguras:** Se han utilizado una serie de restricciones en el uso de instrucciones de control más comunes. Como por ejemplo la guarda de toda condición es una expresión condicional y no aritmética (==).
- **Sistema de tipo unificado:** Todos los tipos de datos se derivan de una clase (*System.Object*).
- **Extensibilidad de tipos básicos:** Permite definir a través de estructuras, tipos de datos.
- **Eficiente:** Permite manipular objetos a través de punteros.
- **Compatible:** C# mantiene una sintaxis similar a C o C++.

El lenguaje C# permite a los desarrolladores compilar aplicaciones sólidas y seguras que se ejecutan en .Net Framework, permitiendo crear [35]:

- Aplicaciones cliente de Windows.
- Servicios Web.
- Componentes distribuidos.
- Aplicaciones cliente-servidor, entre otras.

Por las razones descritas anteriormente, *Visual Studio 2017* es la herramienta utilizada para elaborar la aplicación que funciona para dispositivos que trabajan con sistemas operativos iOS y Android. Además, se utiliza para implementar el servidor escucha mediante *sockets* el cual permite establecer comunicación con los módulos Arduino.

El lenguaje C# es el lenguaje en el cual se desarrolla el código de la aplicación y el servidor escucha que forma parte del prototipo distribuido de disponibilidad de parqueo.

2. METODOLOGÍA

En este capítulo en primera instancia se muestra el estado actual del parqueadero de la FIEE y se detallan los elementos que componen el sistema de ingreso al parqueadero. Luego se muestra el diseño y posteriormente la implementación de los diferentes componentes que constituyen el prototipo distribuido de disponibilidad de parqueo. En el diseño primeramente se muestran los requisitos que el prototipo debe cumplir, después en la etapa de implementación se codifica el diseño presentado. Tanto en el diseño como en la implementación de los diferentes componentes que conforman el prototipo distribuido de disponibilidad de parqueo son separados en Capa de Datos, Capa de Negocio y Capa de Presentación.

2.1 Estado Actual del Parqueadero de la FIEE

El parqueadero ubicado en la calle Bilbao en Quito, como se muestra en la Figura 2.3, es uno de los parqueaderos utilizados por el personal que trabaja en la EPN.

Figura 2.1 Parqueadero FIEE.

Mediante un proceso de observación en el parqueadero FIEE se pudo constatar que el sitio, incluyendo la zona ubicada entre los edificios de Ingeniería Eléctrica y el edificio de Electrónica-Química dispone de un total de 75 espacios para aparcar vehículos. Este número de espacios están destinados en su mayor parte a los docentes que trabajan en la EPN, sin embargo, también está habilitado para personal administrativo y proveedores externos que en ciertas ocasiones requieren el ingreso vehicular.

2.1.1 Componentes del Sistema de Ingreso a la FIEE

El parqueadero de la FIEE cuenta con un sistema de ingreso basado en la tecnología RFID y conformado de varios elementos como se muestra en la Figura 2.2.

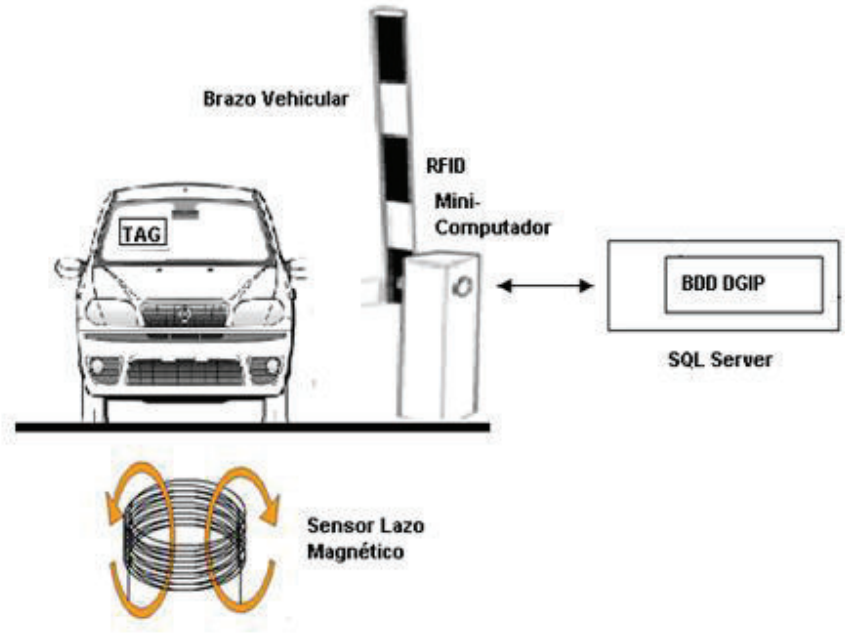

Figura 2.2 Sistema de ingreso actual a la FIEE.

a) Tecnología RFID

El sistema de ingreso utiliza la tecnología RFID o identificación por radiofrecuencia. Este es un sistema de auto identificación inalámbrica el cual consta de etiquetas (*tags*) que almacenan la información y lectores capaces de leer a estas etiquetas utilizando una frecuencia de onda electromagnética para realizar dicha lectura [36], como se muestra en la Figura 2.3.

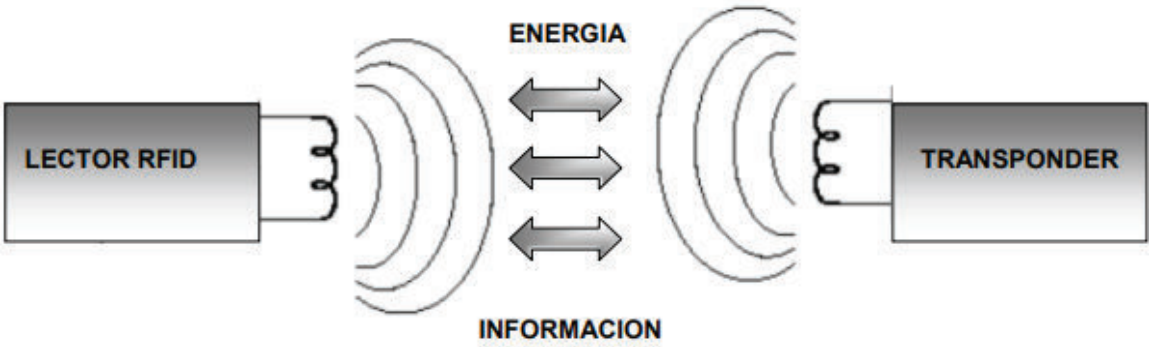

Figura 2.3 Sistema RFID.

Los componentes que conforman el sistema RFID y que son utilizados por la DGIP se muestran a continuación:

- **Un *sticker* / tarjeta RFID**: Es entregado al usuario final, ya sea para ser colocados en el vehículo (*sticker*) o para ser utilizado como tarjeta de ingreso (tarjeta RFID). Estos dispositivos poseen un código (*TAG*) que es utilizado para identificar al usuario y a su vehículo.
- **Lector RFID:** Ubicado en el ingreso y salida del parqueadero de la FIEE que permite leer la información del *sticker* / tarjeta, y consultar en una base de datos de la DGIP si el usuario está registrado en el sistema y tiene habilitado el permiso de ingreso.

b) Base de Datos

La DGIP utiliza una base de datos central para almacenar la información de los usuarios que hacen uso del parqueadero, esta base de datos se encuentra implementada mediante *SQL Server* 2014, anteriormente detallada en el **Capítulo 1 Apartado 1.3.6**.

A continuación, se muestra la información usada para registrar a los usuarios, sus vehículos y generar el TAG que sirve como credencial de ingreso a la FIEE.

- **Datos de Usuario**

Para los usuarios se manejan ciertos parámetros básicos, a continuación, se listan los parámetros que se tienen para almacenar la información de un usuario:

  - Nombres: NVARCHAR(40).
  - Apellidos: NVARCHAR(20).
  - Correo: Electrónico: NVARCHAR(20).
  - Teléfono: VARCHAR(25).
  - Dirección: NVARCHAR(70).

- **Datos de Auto**

La base guarda los datos de los automóviles que poseen los usuarios. Los datos que se manejan para registrar un vehículo se muestran a continuación:

  - Placa: NVARCHAR(20).
  - Marca: NVARCHAR(20).
  - Modelo: NVARCHAR(20).
  - Tipo: NVARCHAR(20).

- **Datos del TAG**

El *TAG* es el dispositivo RFID que identifica al usuario y a su vehículo, además proporciona el código de ingreso para que el usuario pueda acceder a la FIEE. La información relacionada con el *TAG* se muestra a continuación:

  - *TAG ID:* VARCHAR(8).
  - Fecha Inicio: DATETIME.
  - Fecha Fin: DATETIME.
  - Es Carnet: BIT.

El campo *TAG ID* representa el código del *TAG* que opera con la tecnología RFID. Cabe resaltar que la asignación y entrega del *TAG* se encuentra bajo la gestión y administración de la DGIP. La DGIP es el único departamento autorizado para la entrega de estos distintivos a los usuarios. El *TAG ID* es un identificador único el cual tiene habilitado el permiso de ingreso durante cierto periodo de tiempo, periodo comprendido entre los parámetros “Fecha Inicio” y “Fecha Fin”.

Por otro lado, el parámetro “Es Carnet” es utilizado para diferenciar el tipo de acceso que se le brinda al usuario, en caso de utilizar un *sticker* RFID o un carnet de acceso.

Para otorgar el acceso a las instalaciones de la FIEE el usuario debe tener habilitados 2 campos esenciales que se listan a continuación:

  - ***TAG ID***
  - Vigencia TAG (Fecha inicio – Fecha fin)

Si el periodo de vigencia del *TAG* finaliza, el usuario debe acercarse a la DGIP para ampliar este periodo en el cual su *TAG* se encuentra habilitado.

c) Sensor de Lazo Magnético

Permite verificar la presencia del vehículo en el ingreso de la FIEE. Una vez que se valida las credenciales del usuario, el brazo metálico se levanta y el sensor permite que el brazo permanezca arriba mientras exista la presencia de un vehículo.

d) Minicomputador

En el ingreso a la FIEE se tiene un pequeño minicomputador que es el encargado de procesar la solicitud, valida las credenciales de usuario y registra en la base de datos el ingreso y/o salida de los vehículos con la hora exacta. Si el *TAG* está registrado y habilitado, el brazo metálico se levanta permitiendo así el ingreso a los usuarios.

2.2 Diseño

En el siguiente apartado se muestra el diseño del prototipo distribuido de disponibilidad de parqueo; diseño realizado para las diferentes capas que conforman el prototipo que son: Capas de Datos, Capa de Negocio y Capa de Presentación. En primera instancia se presenta el tablero Kanban que tiene el detalle de las tareas realizadas, los requerimientos funcionales y no funcionales. En la Capa de Datos se realiza un diagrama relacional; mientras que la Capa de Negocio incluye el diagrama de casos de uso, diagrama de secuencia y diagrama de clases para mostrar el funcionamiento del prototipo de una forma sistemática. Además, incluye el diseño del servicio web, diseño del subsistema para el envío de notificaciones y el diseño del subsistema Arduino. Finalmente, en la Capa de Presentación se muestran las vistas de la aplicación.

2.2.1 Tablero Kanban

Kanbanflow [37] es una herramienta en línea que permite construir tableros Kanban, agregando columnas en base al diseño del prototipo. El tablero Kanban (Figura 2.4) es la herramienta que permite controlar el trabajo del programador.

Figura 2.4 Tablero Kanban.

2.2.2 Diagrama General del Prototipo

El diagrama general describe el funcionamiento del prototipo y muestra la interacción que los diferentes componentes del prototipo distribuido tienen, para verificar la disponibilidad del parqueadero según se muestra en la Figura 2.5.

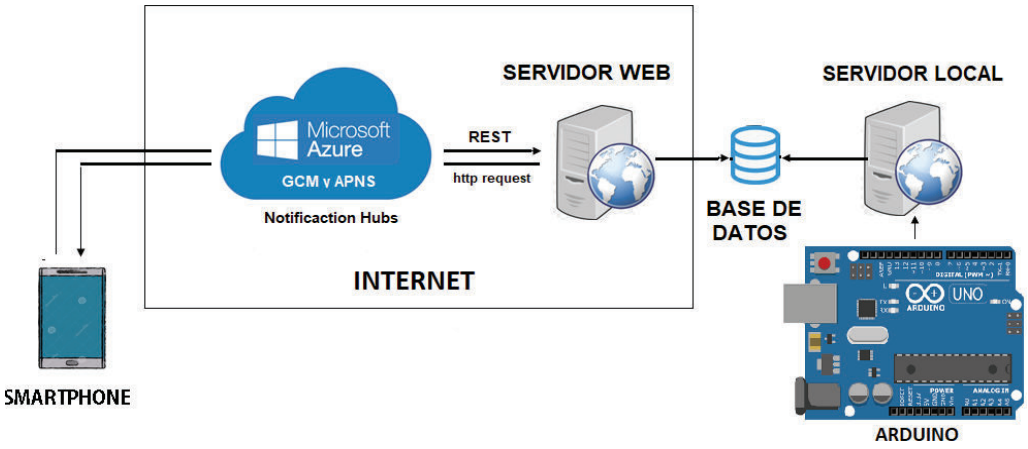

Figura 2.5 Diagrama general del prototipo de disponibilidad de parqueo.

En la Figura 2.6 se muestra el diagrama general del prototipo de disponibilidad de parqueo implementado mediante 4 módulos (subsistemas) dentro de los cuales los requerimientos funcionales son agrupados por afinidad. Los diferentes subsistemas del prototipo distribuido de disponibilidad de parqueo se listan a continuación:

- **Subsistema Arduino**: Mediante Arduino el prototipo monitorea el número de autos que ingresan/salen de la FIEE, el servidor local se comunica con el módulo Arduino, recibe e interpreta los datos que envíe el módulo y en ese momento actualiza el valor de los espacios disponibles en la base de datos.
- **Subsistemas de Registro e Ingreso**: Mediante la aplicación móvil los usuarios pueden registrarse en el prototipo agregando un usuario y una contraseña; luego permite al usuario “loguearse” en el prototipo (si el usuario posee un TAG, es decir, si este es un usuario del parqueadero, puede ingresar haciendo de su correo institucional y su número de cédula), caso contrario hace uso de los datos ingresados en el módulo de Registro.
- **Subsistema de Usuario**: Mediante la aplicación móvil los usuarios pueden consultar la disponibilidad del parqueadero, programar el envío de una notificación donde se muestre la disponibilidad y también tienen la opción de comentar el funcionamiento del prototipo.
- **Subsistema de Administración**: El administrador puede verificar la información de los usuarios del prototipo.

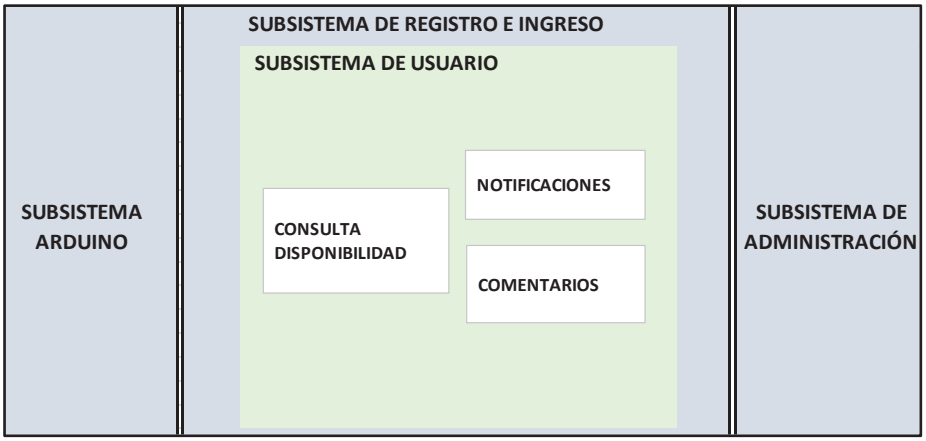

Figura 2.6 Subsistemas del prototipo de disponibilidad de parqueo.

2.2.3 Requerimientos Funcionales

Estos requerimientos fueron seleccionados en base a la información que se recolectó del estudio realizado en la DGIP. En la Tabla 2.1 se muestra un resumen de los requerimientos obtenidos.

Tabla 2.1 Requerimientos funcionales del prototipo de disponibilidad de parqueo.

SUBSISTEMA	RF	DESCRIPCIÓN
Subsistema Arduino	**RF01**	El módulo Arduino monitorea el sensor de lazo magnético, realiza la lectura y envío de datos.
	RF02	El prototipo permite la recepción e interpretación de los datos.
	RF03	El prototipo actualiza el valor de espacios disponibles en la base de datos.
Subsistema Registro e Ingreso	**RF04**	La aplicación móvil permite la creación de un nuevo usuario
	RF05	La aplicación móvil permite autenticarse a un usuario.
Subsistema de Usuario	**RF06**	La aplicación móvil permite visualizar la disponibilidad del parqueadero.
	RF07	La aplicación móvil permite programar un horario para las notificaciones.
	RF08	El prototipo envía la notificación al usuario en el horario establecido.
	RF09	La aplicación móvil permite visualizar la notificación.
	RF010	La aplicación móvil permite guardar un comentario del usuario.
Subsistema Administración	**RF011**	La aplicación de escritorio permite visualizar la información de los usuarios.

En la Tabla 2.2 se muestran los requerimientos funcionales referentes al Subsistema Arduino.

Tabla 2.2 Requerimientos funcionales del Subsistema Arduino.

SUBSISTEMA	RF	Número de horas
Subsistema Arduino	**RF01**	8
	RF02	8
	RF03	4
	TOTAL	20

En la Tabla 2.3 se muestran los requerimientos funcionales referentes al Subsistema de Registro e Ingreso.

Tabla 2.3 Requerimientos funcionales del Subsistema de Registro e Ingreso.

SUBSISTEMA	RF	Número de horas
Subsistema de Registro e Ingreso	**RF04**	10
	RF05	10
	TOTAL	20

En la Tabla 2.4 se muestran los requerimientos funcionales referentes al Subsistema de Usuario.

Tabla 2.4 Requerimientos funcionales del Subsistema de Usuario.

SUBSISTEMA	RF	Número de horas
Subsistema de Usuario	**RF06**	10
	RF07	25
	RF08	25
	RF09	10
	RF010	10
	TOTAL	80

En la Tabla 2.5 se muestran los requerimientos funcionales referentes al Subsistema de Administración.

Tabla 2.5 Requerimientos funcionales del Subsistema de Administración.

SUBSISTEMA	RF	Número de horas
Subsistema de Administración	RF011	10
	TOTAL	10

2.2.4 Requerimientos No Funcionales

Un requerimiento no funcional es aquel requerimiento que especifica criterios para evaluar la operación de un determinando servicio. Para el prototipo distribuido de disponibilidad de parqueo se han detallado varios requerimientos en función a las opciones que el prototipo brinda, por ejemplo: disponibilidad, notificaciones, entre otras. A continuación, se muestran los parámetros establecidos:

a) Rapidez

Todas las funcionalidades del prototipo como: *login*, creación de un usuario, consulta de disponibilidad, almacenamiento de comentarios; deben responder al usuario en un lapso no mayor a 5 segundos.

b) Fiabilidad

El prototipo es diseñado para que el proceso de conteo de vehículos sea continuo, es decir, el módulo Arduino debe estar siempre monitoreando el sensor de lazo magnético procurando así no perder información.

c) Usabilidad

Parámetro que se refiere a la cualidad del prototipo de ser simple para el usuario, se puede citar algunos casos:

- El prototipo posee interfaces gráficas bien definidas y simples para el usuario.
- El prototipo proporciona mensajes de error y mensajes informativos orientados al usuario final.
- El prototipo cuenta con un manual de usuario que describe el modo de operación y los objetivos del prototipo.

d) Satisfacción

Parámetro que indica qué tanto le satisface o no al usuario utilizar el prototipo (aplicación) y sus diferentes funcionalidades. En la Tabla 2.6 se muestra un resumen de los requerimientos no funcionales para el prototipo distribuido de disponibilidad de parqueo.

Tabla 2.6 Requerimientos no funcionales del prototipo de disponibilidad de parqueo.

RNF	DESCRIPCIÓN
RNF01	Rapidez
RNF02	Fiabilidad
RNF03	Usabilidad
RNF04	Satisfacción

2.2.5 Diseño de la Capa de Datos

a) Diseño del Modelo Relacional para la Capa de Datos

La Capa de Datos es mostrada a través del diseño relacional de datos; diseño que es empleado para la creación de la base de datos y sus tablas correspondientes en *SQL Server 2014*. Del estudio previo realizado se establecen tres entidades principales que son: Usuario, Vehículo y Tarjeta RFID. Para los propósitos de este prototipo se va a tomar la información de la base de datos de la DGIP como punto de partida para la creación de la base de datos. Para lograr esto se utiliza el modelo de datos mostrado en la Figura 2.7.

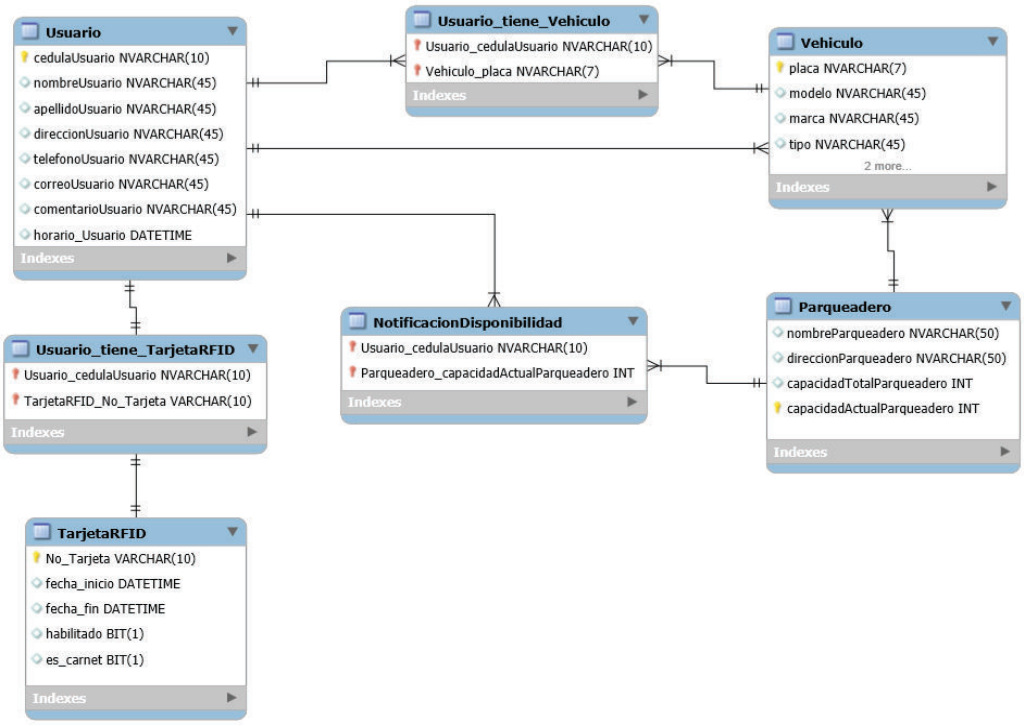

Figura 2.7 Diagrama relacional de bases de datos.

En la Figura 2.7 se puede observar las tablas que almacenan la información que genere el prototipo, por ejemplo, la tabla '*Vehículo'*: un usuario puede tener uno o varios vehículos debido a la cardinalidad '*uno a muchos'* entre las tablas '*Usuario'* y '*Vehículo'*. El diagrama mostrado permite a las entidades interactuar entre sí para cumplir con la lógica del prototipo.

2.2.6 Diseño de la Capa de Negocio

La Capa de Negocio muestra la lógica de los diferentes procesos que permiten implementar el prototipo distribuido de disponibilidad de parqueo. Para mostrar dicha lógica se hace uso de diagrama de casos de uso, diagrama de secuencia y diagrama de clases para mostrar el funcionamiento del prototipo de disponibilidad de parqueo de una forma sistemática; además incluye el diseño del servicio web, diseño del subsistema para el envío de notificaciones y el diseño del subsistema Arduino.

a) Diagramas de Casos de Uso

Un diagrama de caso de uso permite representar la forma cómo un actor opera e interactúa con el sistema. Mediante los diagramas de casos de uso presentados a continuación se muestra el comportamiento de los usuarios dentro del prototipo distribuido de disponibilidad de parqueo.

Los usuarios son aquellas personas que hacen uso del parqueadero en la FIEE como autoridades, profesores, alumnos que pueden descargar, registrarse e ingresar a la aplicación. Dentro de las opciones que el prototipo ofrece, los usuarios pueden realizar consultas y programar el horario en el cual desean recibir notificaciones del estado del parqueadero. Adicionalmente los usuarios tienen la posibilidad de evaluar y comentar el funcionamiento de la aplicación para retroalimentar el proceso.

El presente prototipo define usuarios del tipo Nuevos, Usuarios del Parqueadero, y Sistema. A continuación, se detalla el rol de cada uno de ellos.

- **Usuario Registrado:** Dentro del prototipo un usuario registrado corresponde a aquel que hace uso del parqueadero, es decir al usuario que posee una '*Tarjeta RFID*' y hace uso de las instalaciones de La FIEE. Estos usuarios tienen previamente cargada su información en la base de datos por lo que no requieren crear un *user* y *password* dentro del sistema.
- **Usuario Nuevo:** Este tipo de usuarios corresponde a la comunidad en general, es decir cualquier persona que desee hacer uso del prototipo. Ambos tipos de usuarios pueden acceder a las funcionalidades descritas anteriormente.

- **Usuario Administrador:** Este tipo de usuario corresponde al administrador del prototipo (DGIP). Este puede verificar la información de los usuarios registrados y evaluar los comentarios que los usuarios realicen sobre el funcionamiento de la aplicación.
- **Usuario Sistema:** Se define este usuario para relacionar las diferentes tareas que realiza el prototipo.

Tabla 2.7 Permisos por tipos de usuarios en el prototipo de disponibilidad de parqueo.

FUNCIONALIDADES PROTOTIPO DE DISPONIBILIDAD DE PARQUEO			
Actividad	**Usuario Registrado**	**Usuario Nuevo**	**Administrador**
Descargar aplicación	X	X	X
Crear perfil de usuario	X	X	X
Ingresar al sistema	X	X	X
Consultar disponibilidad	X	X	X
Programar horario	X	X	X
Visualizar Comentarios			X

La DGIP es el ente encargado de crear un nuevo usuario y asignar un *TAG,* toda la información referente a los usuarios y sus vehículos se encuentra en la base de datos de la DGIP. Para el prototipo se realiza una copia de la base de datos y esta es la base de datos que se utiliza como base para el prototipo. Un usuario del parqueadero puede "loguerase" haciendo uso de su correo institucional como *username* y su número de cédula como *password* para acceder al sistema.

Adicional, dentro del prototipo, cualquier persona que descargue la aplicación, sea o no usuario del parqueadero, puede registrarse, crear un perfil de usuario para así poder acceder al sistema, conocer la disponibilidad y acceder a las demás opciones que brinda el prototipo.

Los casos de uso representan los requerimientos funcionales. A continuación, se muestran los casos de uso dentro de su correspondiente subsistema.

La Figura 2.8 muestra el caso de uso referente al Subsistema Arduino; el módulo Arduino monitorea al sensor de lazo magnético y envía un aviso al prototipo cada vez que el sensor ha cambiado de estado. Luego, el prototipo recibe los datos enviados por el módulo Arduino.

El servidor local lee e interpreta la información recibida para determinar el número de espacios disponibles en la FIEE. Finalmente; el servidor local actualiza el valor de espacios disponibles en la base de datos, para lograr esto, se necesita que el prototipo haya recibido al menos una notificación del módulo Arduino (el valor se actualiza cada vez que el Arduino detecte que el brazo metálico se elevó y envíe una notificación al prototipo).

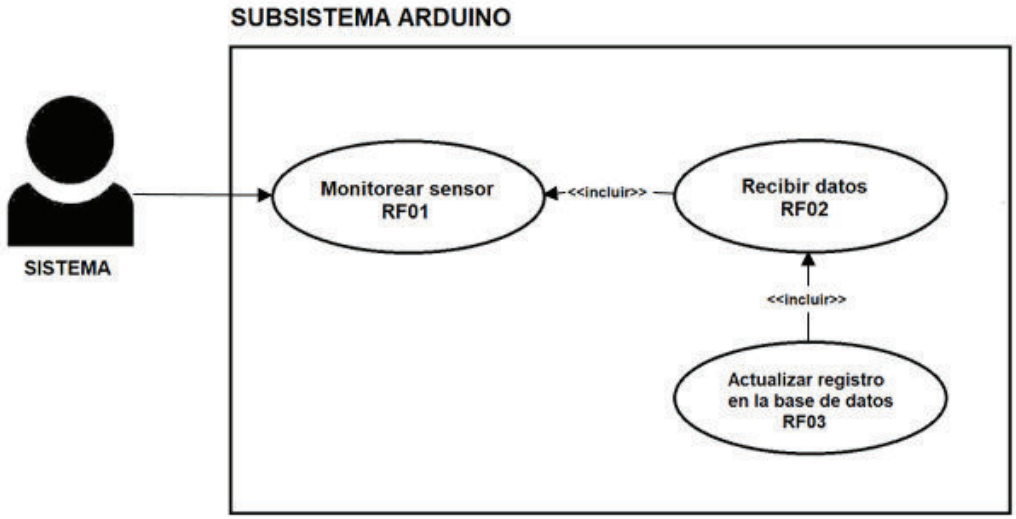

Figura 2.8 Caso de uso del Subsistema Arduino.

La Figura 2.9 muestra el caso de uso del Subsistema de Registro e Ingreso. Para la creación y autenticación de un nuevo usuario es un requisito previo instalar y acceder a la aplicación móvil. Se debe considerar que dentro del prototipo existen usuarios previamente registrados que son usuarios del parqueadero de la FIEE y usuarios nuevos que podrán crear un perfil de usuario. Luego es necesario el proceso de autenticación.

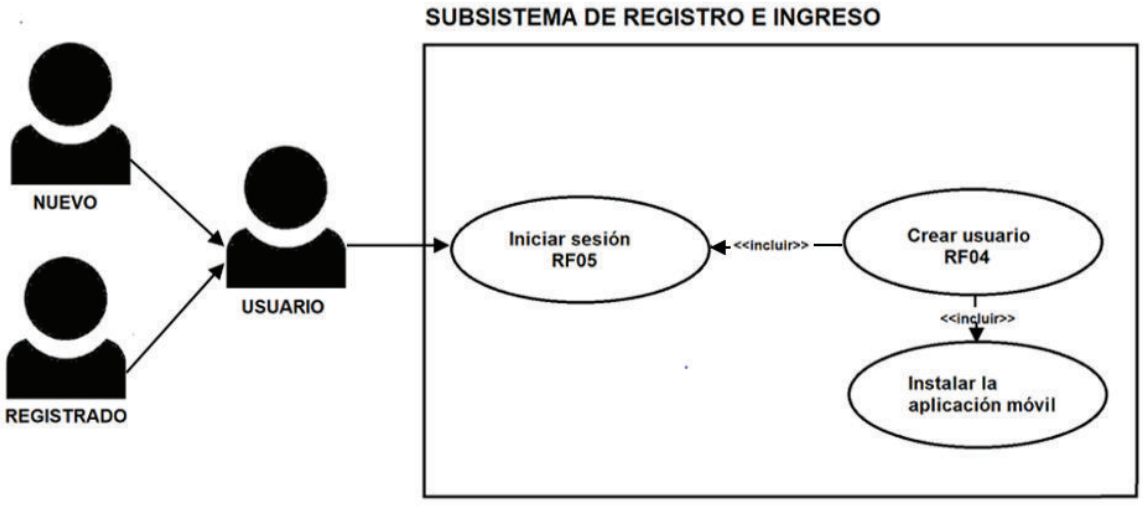

Figura 2.9 Caso de uso del Subsistema de Registro e Ingreso.

En la Figura 2.10 se observa el caso de uso correspondiente a los requerimientos funcionales RF06, RF07 Y RF010 del Subsistema de Usuario. El caso de uso describe el proceso para visualizar la disponibilidad del parqueadero en el dispositivo del usuario, usuarios nuevos y registrados pueden realizar esta acción. Luego se observa el proceso para programar un horario para recibir notificaciones sobre la disponibilidad del parqueadero de la FIEE. Finalmente se observa el proceso para guardar un comentario sobre el funcionamiento de la aplicación. El requisito previo para estos requerimientos es haber iniciado sesión en la aplicación móvil.

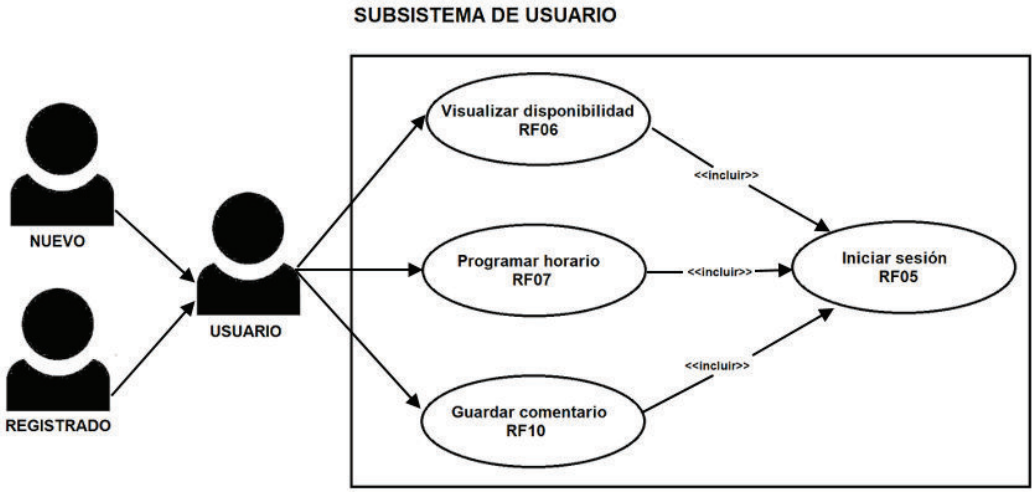

Figura 2.10 Caso de uso del Subsistema de Usuario. Estructura general.

La Figura 2.11 muestra el diagrama de caso de uso del Subsistema de Usuario en el que el prototipo va a enviar la notificación con el número de espacios disponibles en el parqueadero de la FIEE al usuario, como se puede observar la notificación se envía solo si el usuario tiene programado un horario. Para recibir notificaciones el sistema operativo del dispositivo móvil es el encargado de recibir la notificación por lo que no es necesario que la aplicación se encuentra abierta en el dispositivo.

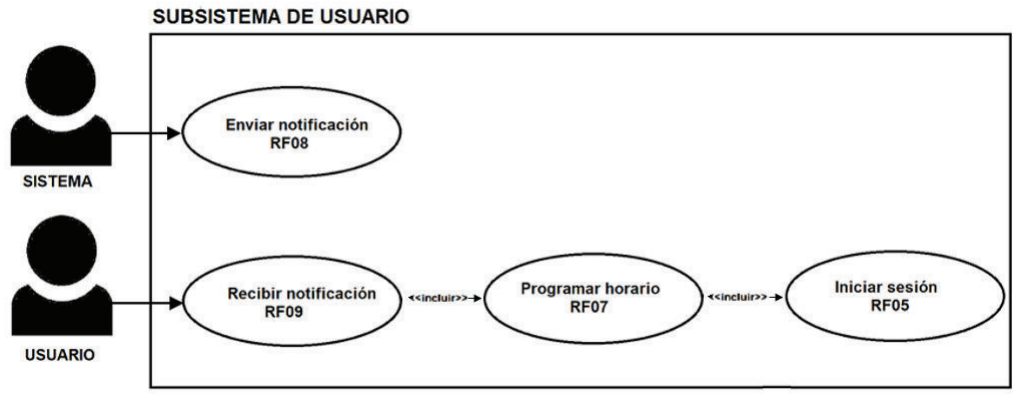

Figura 2.11 Caso de uso del Subsistema de Usuario. Proceso de notificación.

En la Figura 2.12 se observa el caso de uso correspondiente al Subsistema de Administración, en donde el usuario administrador puede visualizar la información de los usuarios registrados en el prototipo, previamente debe “loguearse” en la aplicación de escritorio.

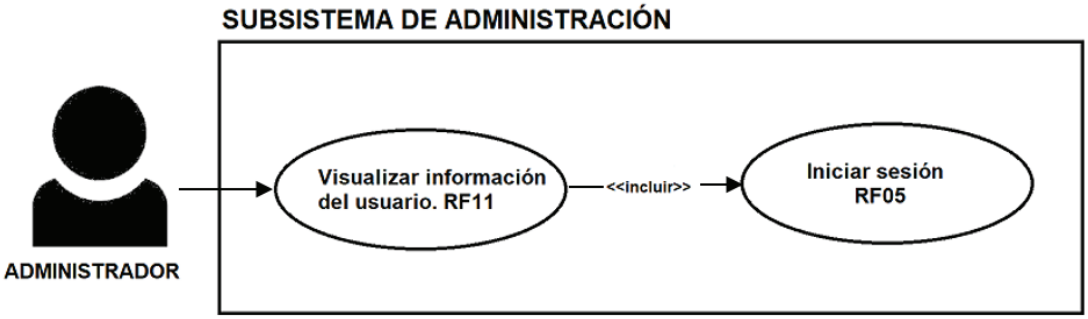

Figura 2.12 Caso de uso del Subsistema de Administración.

b) Diagrama de Secuencia

Un diagrama de secuencia es un tipo de diagrama que muestra la interacción de un grupo de objetos en un sistema a través del tiempo. Estos diagramas contienen detalles en donde se muestra el escenario presentado, incluye objetos, clases y mensajes intercambiados entre estos.

En la Figura 2.13 se observa el diagrama de secuencia correspondiente al RF01, RF02 y RF03 que conforman el Subsistema Arduino.

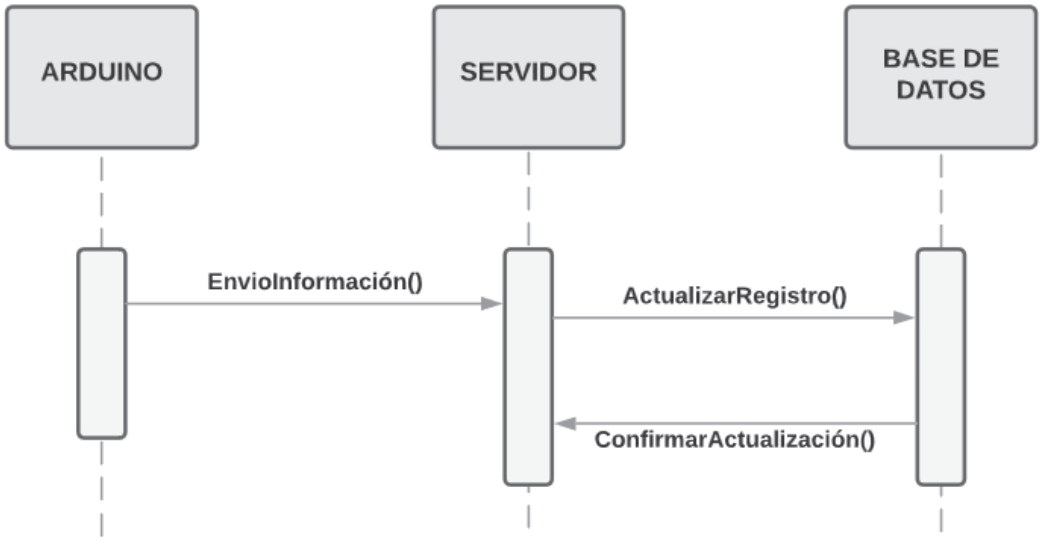

Figura 2.13 Diagrama de secuencia subsistema Arduino.

Posteriormente, se muestra en la Figura 2.14 el diagrama de secuencia correspondiente al proceso de registro de un usuario, que corresponde al RF04 del subsistema de registro e ingreso.

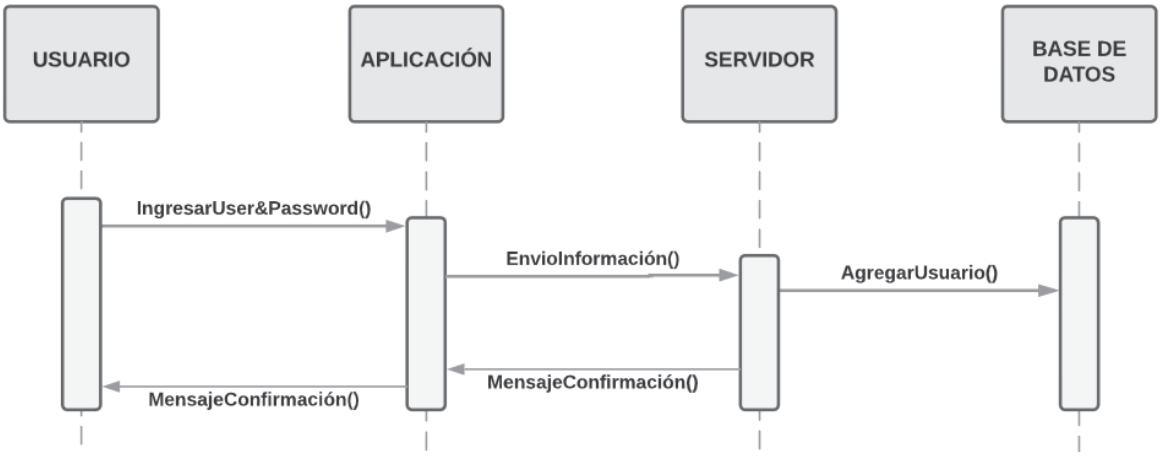

Figura 2.14 Diagrama de secuencia para el registro de un usuario.

En la Figura 2.15 se muestra el diagrama de secuencia del proceso de autenticación de un usuario en el sistema, que corresponde al RF05 del subsistema de registro e ingreso.

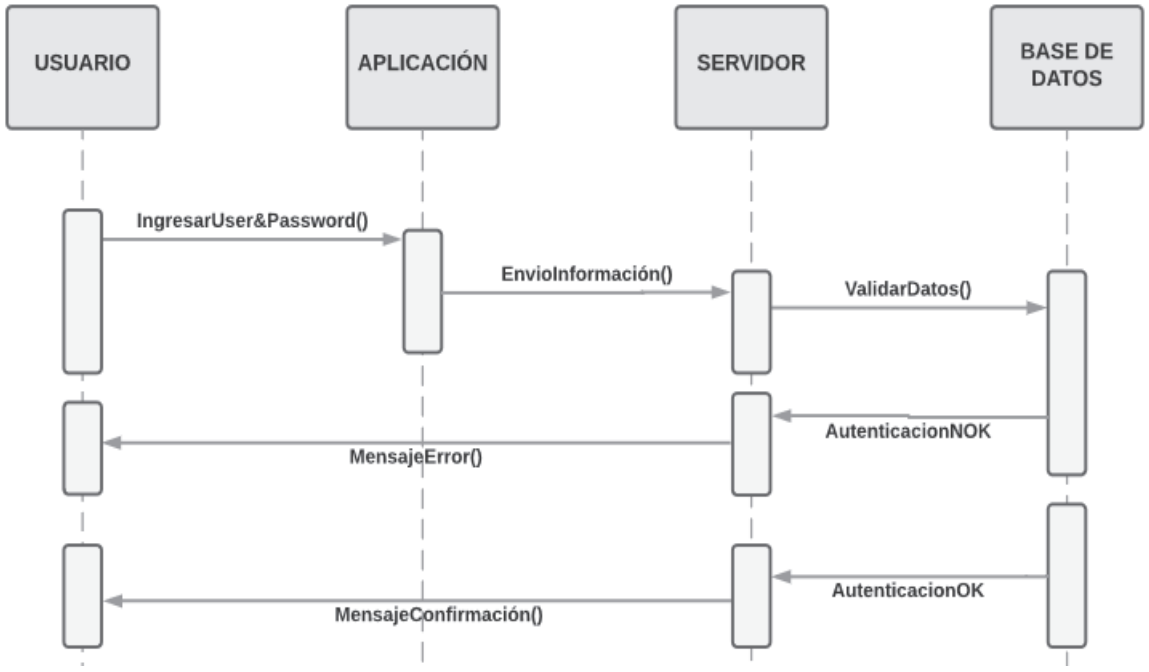

Figura 2.15 Diagrama de secuencia para la autenticación de un usuario.

En la Figura 2.16 se muestra el diagrama de secuencia correspondiente al subsistema de usuario en donde se engloban los RF06-10.

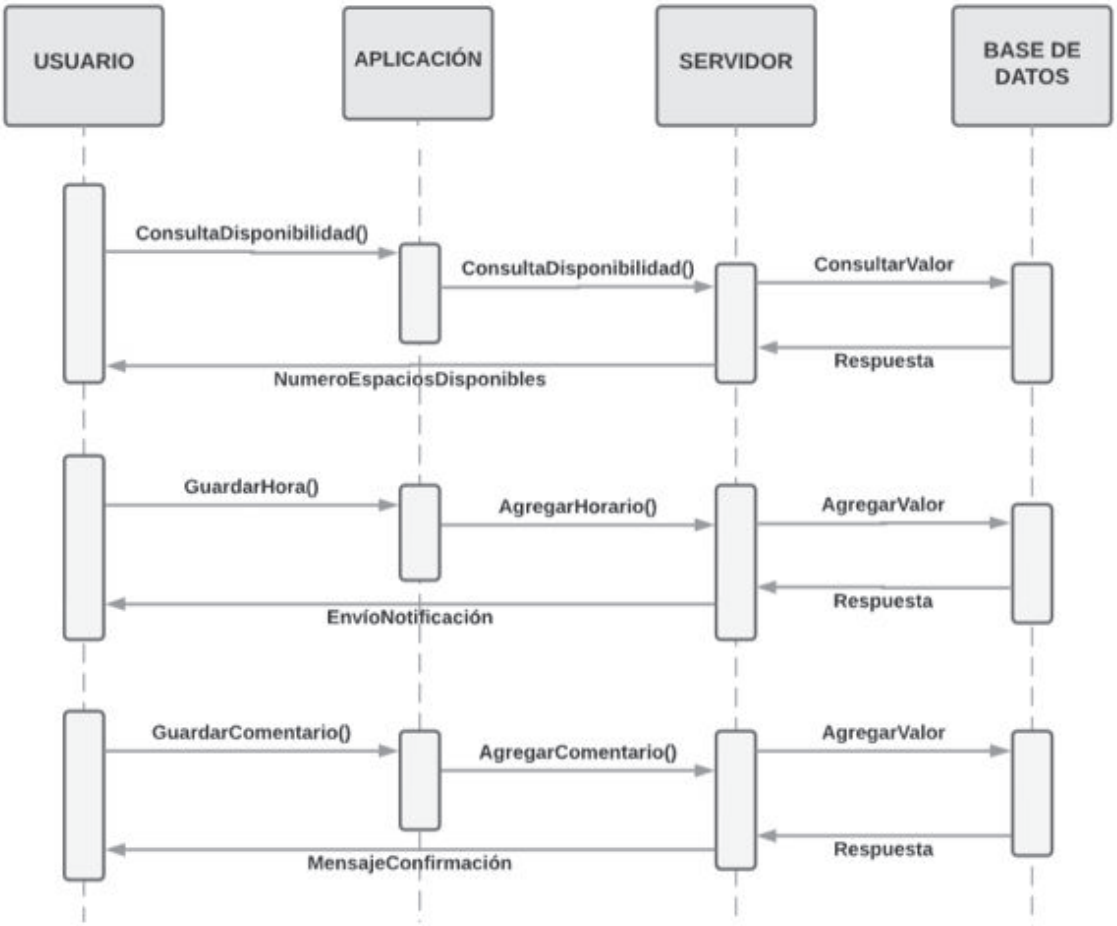

Figura 2.16 Diagrama de secuencia subsistema de usuario.

c) Diseño del Diagrama de Clases

En la Figura 2.17 se puede observar el diagrama de clases que implementa la lógica del negocio; adicional en el diagrama se puede apreciar los atributos de cada clase y las cardinalidades que existe entre las diferentes clases que se tiene en el prototipo.

Por ejemplo se observa que la clase '*Usuario'* y '*Vehículo'* tienen la cardinalidad *'uno'* a *'n'*, la cual implica que un usuario puede tener varios vehículos. Por otro lado la clase '*Usuario'* y '*TarjetaRFID'* tienen la cardinalidad *'uno'* a *'uno'* pues a cada usuario del parqueadero se le proporciona una sola tarjeta RFID.

La clase '*Parqueadero'* por otro lado es la clase que almacena el número de espacios disponibles que dispone el parqueadero; este es el campo que el prototipo consulta cada vez que el usuario requiera conocer la disponibilidad de espacios de parqueo en la FIEE.

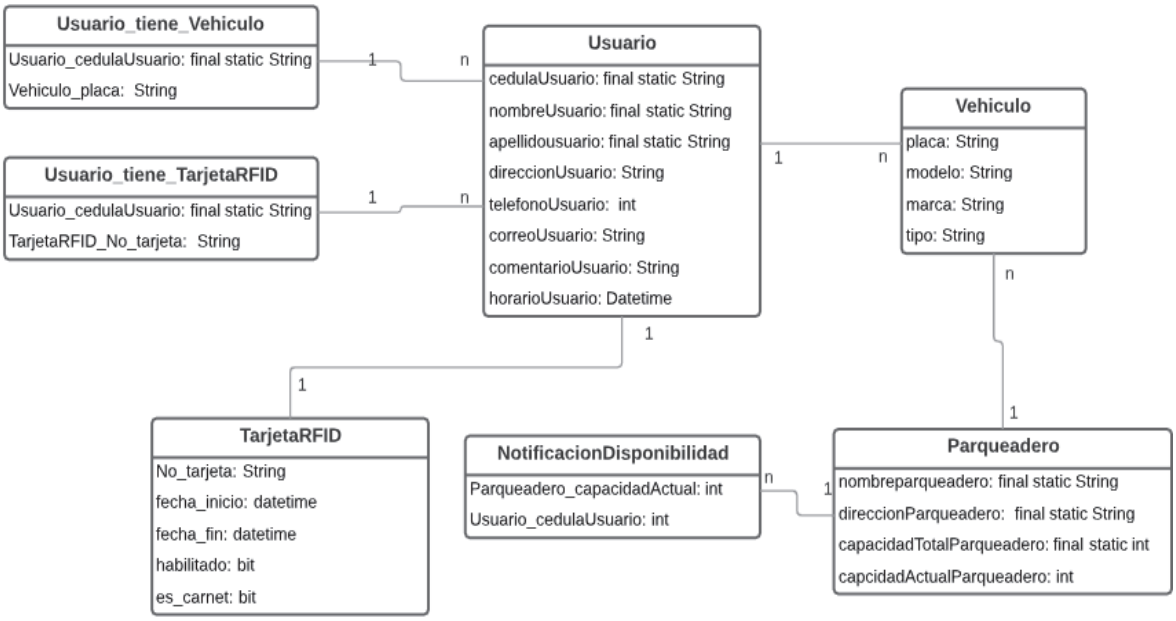

Figura 2.17 Diagrama de clases.

d) Diseño del Servicio Web

Se realiza la programación de los servicios web utilizando el lenguaje C# en *Visual Studio* para así ofrecer a los clientes de la aplicación las diferentes funcionalidades descritas en el **Capítulo 2 Aparatado 2.2.2**.

La arquitectura REST define principios en los cuales se diseñan los servicios web haciendo énfasis en los recursos del sistema, como acceder a dichos recursos y cómo se transfieren mediante HTTP hacia los clientes. Los servicios web REST siguen la lógica mostrada en la Figura 2.18.

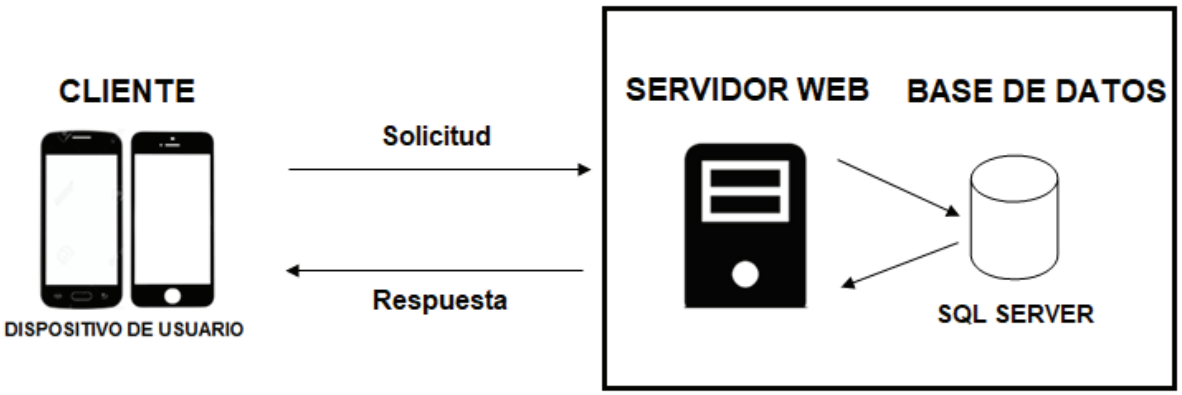

Figura 2.18 Arquitectura de los Servicios Web.

Para acceder a los servicios web es necesario conocer la IP donde se aloja el servidor web y concatenar con el nombre del servicio. Dentro de *Visual Studio* en el proyecto ASP.NET (.NET Framework) se añade la lógica necesaria para implementar los servicios web que permiten cumplir con los requerimientos funcionales del prototipo distribuido de disponibilidad de parqueo.

Las consultas hacia la base de datos incluyen leer, crear, actualizar registros como por ejemplo crear un nuevo usuario dentro del prototipo para acceder a la aplicación.

e) Diseño del Sistema para el envío de notificaciones

Las notificaciones *push* permiten el envío de mensajes cortos hacia los usuarios que tienen instalada la aplicación en sus dispositivos móviles. Basado en este concepto se presenta el diseño del subsistema de notificaciones que permite a los usuarios recibir notificaciones sobre la disponibilidad de espacios en el parqueadero de la FIEE. Se implementa un centro de gestión de las notificaciones (*back end)*, el cual permite gestionar el envío de notificaciones hacia los dispositivos que tengan la aplicación instalada. En este contexto es posible enviar notificaciones instantáneas o programar envíos en horas específicas.

Como infraestructura se utiliza *Visual Studio*, C#, IIS, *SQL Server* para el desarrollo del subsistema de notificaciones; Azure es el motor de notificaciones y hace uso de los servicios GCM (*Google Cloud Messaging*) y APNS (*Apple Push Notification Services*) para enviar notificaciones hacia el dispositivo del usuario. En la Figura 2.19 se muestra el proceso para el envío y recepción de una notificación *push*.

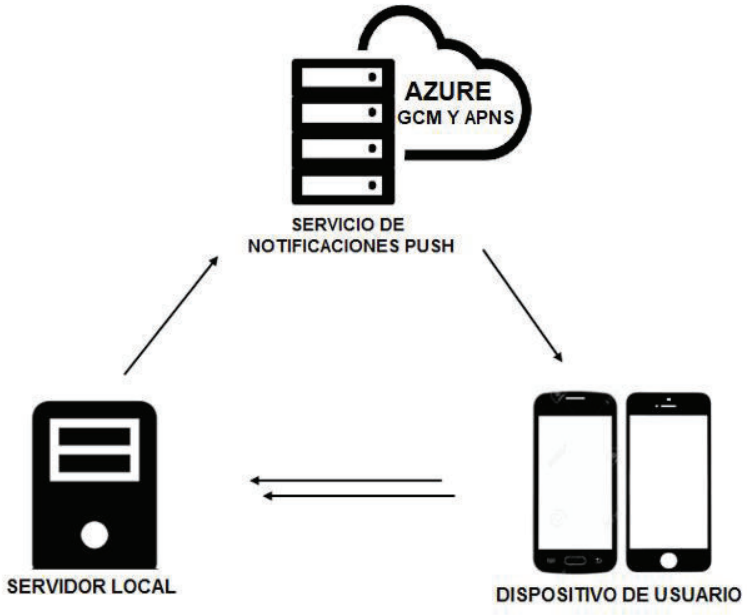

Figura 2.19 Proceso envío de una notificación *push*.

- **Android**

Para el envío de notificaciones hacia dispositivos Android como se mencionó en el **Capítulo 1 Apartado 1.3.7** se hace uso del servicio GCM (*Google Cloud Messaging*) ofertado por Google, permitiendo enviar y recibir información desde el proveedor de una aplicación a un dispositivo que tenga la aplicación instalada [38]. Una notificación *push* consiste en el envío de un mensaje hacia el dispositivo destinatario. Dicho mensaje consta de 3 partes:

- **El *token* del dispositivo:** Es un identificador único del dispositivo. Para que un dispositivo reciba una notificación *push*, el dispositivo debe registrarse en el servicio GCM. Al ingresar a la aplicación el dispositivo solicita el registro en el servicio, en respuesta, el servicio GCM responde con el *token* del dispositivo, un identificador denominado como "*Registration ID*"; el *token* identifica a la aplicación y al dispositivo de manera única.
- **Opciones de envío:** Opciones configurables para la comunicación. A continuación, se muestran algunas de las opciones de envío disponibles:

Tabla 2.8 Opciones para el envío de una notificación *push* en Android [38].

ATRIBUTO	TIPO DE VALOR	DESCRIPCIÓN
message_id	String	Identificador único del mensaje
priority	String	Prioridad del mensaje entre 0-10, siendo 0 la mayor prioridad.
time_to_live	Integer	Tiempo máximo para la retransmisión en caso de que el dispositivo no esté disponible.

- **Carga útil**. Listado de propiedades que contienen la información que se va a mostrar al usuario, información que es enviada en la notificación de inserción. La carga útil presenta la siguiente estructura:

```
{
        "data":
`       {
                "mensaje": "Disponibilidad del parqueadero:",
                "espacios": "23"
        }
}
```

Código 2.1 Carga útil de una notificación *push* Android.

El *token* del dispositivo es proporcionado por el servicio GCM mientras que la carga útil consta de un mensaje de texto más el valor de espacios disponibles que se tiene en la base de datos. En la Tabla 2.9 se muestra el listado de las opciones que se tiene para construir la carga útil de una notificación [38]:

Tabla 2.9 Atributos para la carga útil en notificaciones *push* Android [38].

ATRIBUTO	TIPO DE VALOR	DESCRIPCIÓN
title	String	Especifica el título que lleva la notificación
body	String	Especifica el cuerpo de la notificación.
icon	String	Especifica un ícono para la notificación.
sound	String	Especifica un sonido que se reproduce cuando la notificación llegue.
click_action	String	Se puede especificar la acción o pantalla a mostrar cuando el usuario da clic en la notificación.

El envío de la notificación va desde el servidor Azure (Servicio GCM) hasta el dispositivo del usuario con la aplicación instalada. El proveedor arma la notificación incluyendo los datos de autenticación de GCM, el *token* del dispositivo, las opciones de envío y la carga útil; el envío de la notificación se realiza haciendo uso de la solicitud POST.

- **iOS**

Para el envío de notificaciones *push* a dispositivos que trabajan con sistema operativo iOS se hace uso del servicio APNS (*Apple Push Notification Services*) proporcionado por Apple. Este servicio funciona como intermediario en el envío de notificaciones. El servicio es seguro, robusto y eficiente y se encarga del transporte y enrutamiento de la notificación proporcionando conexiones seguras [38].

En iOS una notificación es un mensaje corto compuesto de 2 partes:

- ***Token* del dispositivo**: Es un número o identificador que permite al servicio APNS localizar y autenticar al dispositivo que tiene instalada la aplicación. Para recibir una notificación *push* el usuario debe obtener el *token*; para lo cual el dispositivo debe estar registrado en el servicio APNS. Este procese se realiza de manera transparente al usuario al momento de que la aplicación es instalada en el dispositivo. Este *token* debe ser almacenado para enviar las notificaciones a determinados usuarios.

- **Carga útil.** Propiedades que tienen la información final que es mostrada al usuario. El tamaño máximo de la carga es 256 bytes. La estructura básica de la carga útil está compuesta por:

Tabla 2.10 Atributos para la carga útil en notificaciones *push* iOS [38].

ATRIBUTO	TIPO DE VALOR	DESCRIPCIÓN
alert	String/Estructura JSON	Mensaje en texto plano.
badge	Integer	Número que se muestra como insignia en el ícono de la aplicación.
sound	String	Especifica un sonido que se reproduce cuando la notificación llegue.

La carga útil tiene la siguiente estructura:

```
{
        “aps”:
        {
                “alert”  : “Disponibilidad del parqueadero 23”,
                “badge” : 1,
        }
}
```

Código 2.2 Carga útil de una notificación *push* iOS.

Para el envío de una notificación *push*, el desarrollador compone la notificación en donde se incluye el *token* del dispositivo y la carga útil; el *token* es proporcionado por el APNS mientras que la carga útil consta de un mensaje de texto más el valor de espacios disponibles que se tiene en la base de datos. Luego se autentica en el servidor APNS; si la autenticación es correcta el servidor APNS se encarga de reenviar la notificación al dispositivo destino.

Dentro del proceso de autenticación se hace uso del llamado “certificado digital” que el desarrollador obtiene al crear su cuenta con permisos de administrador en Apple.

El servidor APNS almacena la notificación en los servidores de Apple si el dispositivo final no está disponible para recibir la notificación, en caso de encolamiento. La última notificación es almacenada y reenviada al usuario.

f) Diseño del Subsistema Arduino

El ingreso y salida en el parqueadero de la FIEE son independientes por lo que cada uno cuenta con un sensor de lazo magnético; para conocer el número de vehículos que ingresan/salen del parqueadero, se coloca un módulo Arduino para la entrada y un segundo módulo Arduino para la salida de la FIEE. Para obtener el valor del número de espacios disponibles en la FIEE como se mencionó en el **Capítulo 2 Apartado 2.2.2**, se implementa un servidor basado en *sockets* para enlazar y obtener la información que constantemente los módulos Arduino emiten.

El servidor mediante *sockets* TCP es el encargado de enlazar los módulos Arduino. Para esto hace uso de las siguientes funciones:

- **Bind():** Es una llamada al sistema en la cual se asigna una dirección al *socket*.
- **Listen():** Esta función alerta al sistema operativo que empiece a atender la conexión, el sistema registra la conexión de cualquier cliente.
- **Accept():** Acepta las conexiones de clientes.

Por otro lado, el cliente (módulo Arduino) mediante su módulo Ethernet, solicita establecer la conexión con el servidor y una vez que el servidor acepte la conexión se establece el flujo de datos en los cuales los módulos Arduino notifican el ingreso y salida de vehículos en la FIEE. El diagrama mostrado en la Figura 2.20 muestra el proceso para establecer la conexión entre el servidor y el cliente y el envío/recepción de la información.

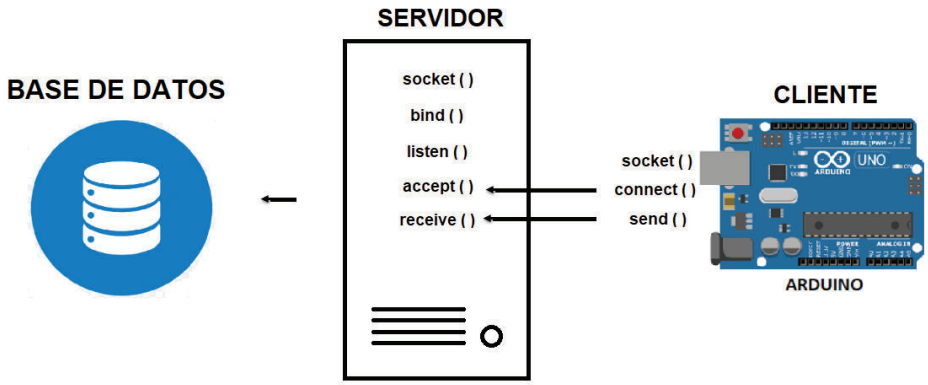

Figura 2.20 Establecimiento de conexión entre el servidor local y el módulo Arduino.

Una vez que el microcontrolador detecta un cambio de estado del sensor de lazo magnético este envía la palabra “in” al servidor local, que a su vez interpreta la información y aumenta el valor del contador de ingreso. Mediante el diagrama de flujo mostrado en la Figura 2.21 se muestra el proceso detallado de cómo se obtiene el valor del registro una vez que el usuario ingrese a la FIEE.

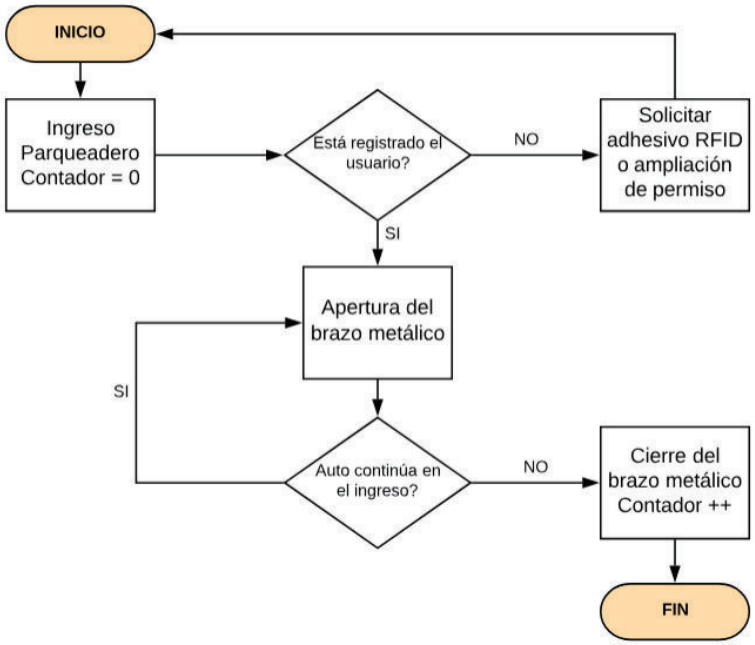

Figura 2.21 Obtención del número de espacios al ingreso de la FIEE.

De igual manera, una vez que el microcontrolador detecta un cambio de estado del sensor de lazo magnético este envía la palabra “out” al servidor local cada vez que un auto salga de la FIEE, el servidor local interpreta la información y aumenta el valor del contador de salida. El diagrama de flujo mostrado en la Figura 2.22 muestra cómo se obtiene el valor del registro una vez que el usuario sale de la FIEE.

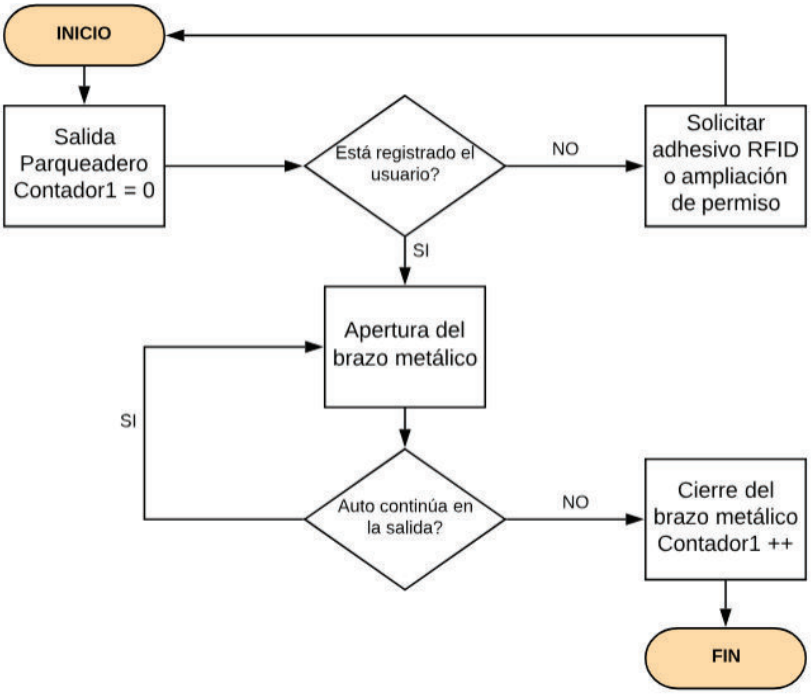

Figura 2.22 Obtención del número de espacios a la salida de la FIEE.

El servidor local procesa esta información; la diferencia del valor de los contadores restado de la capacidad del parqueadero entrega el valor de los espacios disponibles como se muestra en la Figura 2.23.

Disponibilidad = Capacidad Total - (Contador - Contador1)

Figura 2.23 Cálculo del número de espacios disponibles en la FIEE.

Este registro es actualizado en la base de datos cada vez que un vehículo ingrese/salga de la FIEE, esto con la finalidad que el usuario mediante la aplicación y el uso de servicios web puedan visualizar esta información en su dispositivo móvil.

2.2.7 Diseño de la Capa de Presentación

El diseño de las interfaces para el prototipo de disponibilidad de parqueo se realiza en función de los requerimientos obtenidos en el **Capítulo 2 Apartado 2.2.2** en donde se obtuvo información del sistema de ingreso de la DGIP.

a) Interfaz del Subsistema de Registro e Ingreso

Una vez descargada la aplicación en el *smartphone* del usuario, este puede acceder a la página principal de la aplicación en donde encuentra un inicio de sesión en donde el usuario puede registrarse e iniciar sesión como se muestra en la Figura 2.24.

Figura 2.24 Interfaz para la autenticación de un usuario.

La interfaz diseñada para el registro al prototipo en donde el usuario proporciona sus datos se muestra en la Figura 2.25.

Figura 2.25 Interfaz del Subsistema de Registro.

b) Interfaz del Subsistema de Usuario

El diseño propuesto para la interfaz de usuario se muestra siguiendo la Figura 2.26.

Figura 2.26 Interfaz del Subsistema de Usuario. Vista general.

El usuario dentro del prototipo tiene la posibilidad de realizar las siguientes opciones:

- Consultar la disponibilidad del parqueadero en cualquier instante.
- Programar un horario en el cual puede recibir notificaciones acerca de la disponibilidad que tiene el parqueadero de la FIEE.
- Comentar y valorar el uso del prototipo.

Para que el usuario pueda agregar la hora en la que desea recibir notificaciones sobre la disponibilidad, se tendrá la interfaz mostrada en la Figura 2.27.

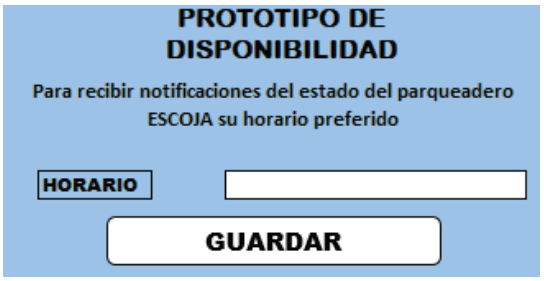

Figura 2.27 Interfaz Subsistema de Usuario. Programación de notificaciones.

Para que el usuario pueda agregar un comentario acerca del funcionamiento del prototipo de disponibilidad de parqueo, se implementará la interfaz mostrada en la Figura 2.28.

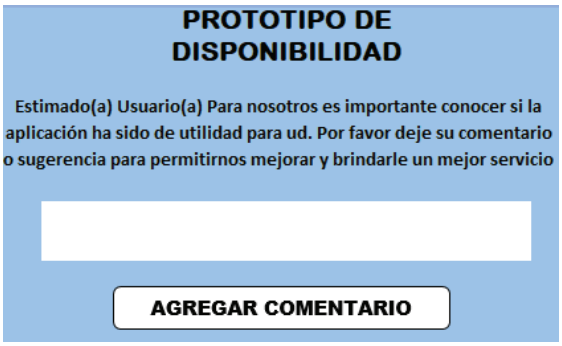

Figura 2.28 Interfaz Subsistema de Usuario. Agregar comentario.

c) Interfaz Subsistema de Administración

En la Figura 2.29 se muestra la interfaz en la cual el administrador puede verificar la información del usuario, puede visualizar de igual manera los comentarios realizados por los diferentes usuarios para constatar la acogida que puede llegar a tener el prototipo. Esta interfaz está definida por una aplicación de escritorio en la cual el administrador debe ingresar para visualizar toda la información.

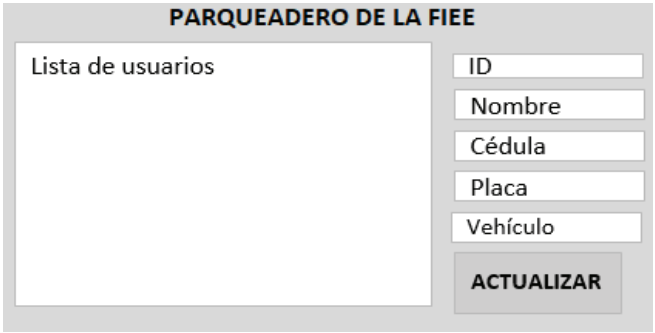

Figura 2.29 Interfaz Subsistema de Administración.

Una vez detallado el diseño de los subsistemas se inicia con la fase de implementación de los mismos.

2.3 Implementación

En esta sección se presentan detalles correspondientes a la implementación de los componentes del prototipo distribuido de disponibilidad de parqueo; así como la integración de estos entre sí. En primer lugar, se muestra la instalación y configuración del software necesario para el desarrollo de los componentes y luego según el diseño propuesto en el **Capítulo 2 Apartado 2.2** y según el tablero Kanban mostrado en la Figura 2.30, se detalla la codificación de la aplicación y los diferentes subsistemas que conforman el prototipo. En paralelo se implementa la base de datos para almacenar la información y el módulo Arduino para la obtención del valor de espacios disponibles en el parqueadero de la FIEE. Por último, se detalla la implementación de notificaciones *push*.

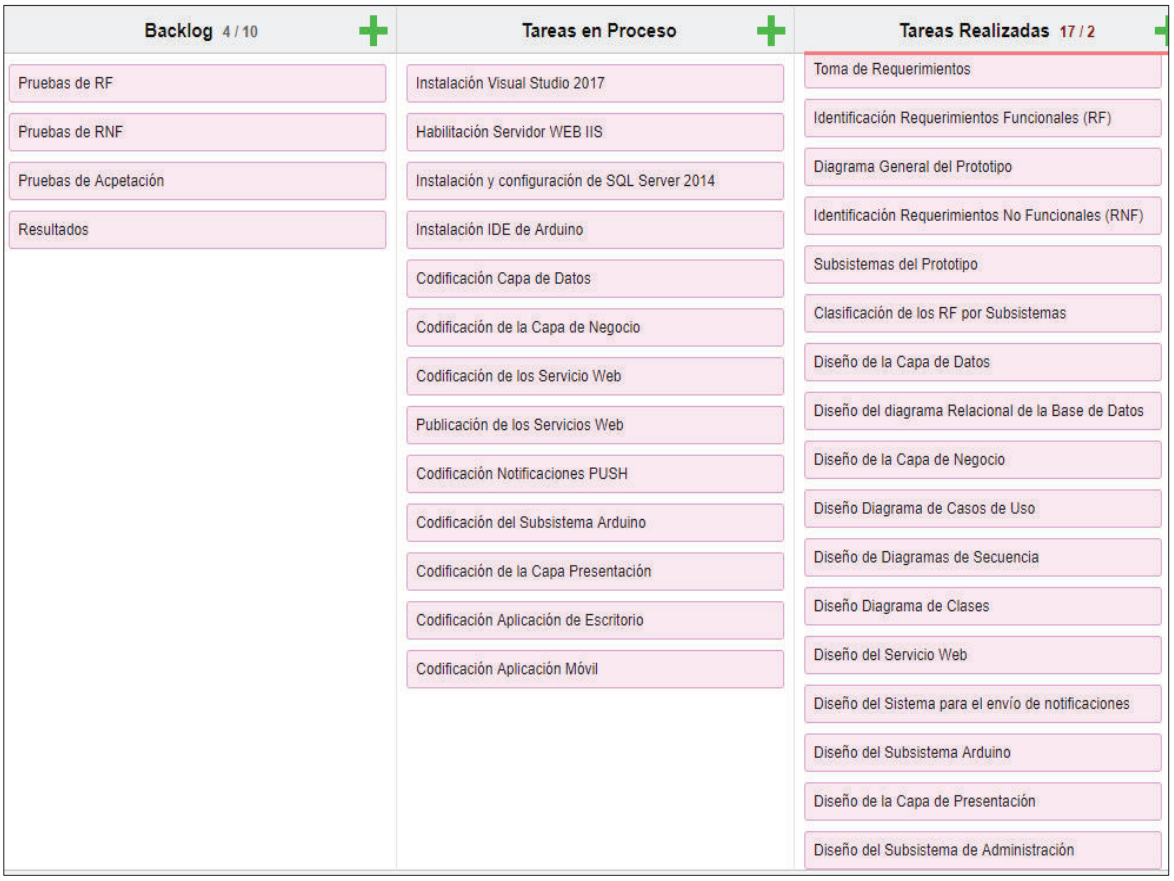

Figura 2.30 Tablero Kanban. Implementación.

2.3.1 Instalación de Herramientas

a) Habilitación del Servidor Web IIS

Internet Information Services (IIS) es un servidor web y un conjunto de servicios para el sistema operativo *Windows*, que permite albergar sitios, páginas web y aplicaciones proporcionando una manera fácil y segura de compartir información con otros usuarios [39].

IIS alberga el sitio web para el prototipo. El servidor IIS es un complemento de *Windows* por lo que no requiere ser instalado, solo necesita ser habilitado. Para habilitar el servidor Web IIS se debe abrir el Panel de Control, opción “Programas y Características” y luego se debe seleccionar “Activar o Desactivar las características de Windows”; como se muestra en la Figura 2.31 se debe marcar la opción IIS (*Internet Information Services*). Hecho esto ya se encuentra habilitado el servidor IIS.

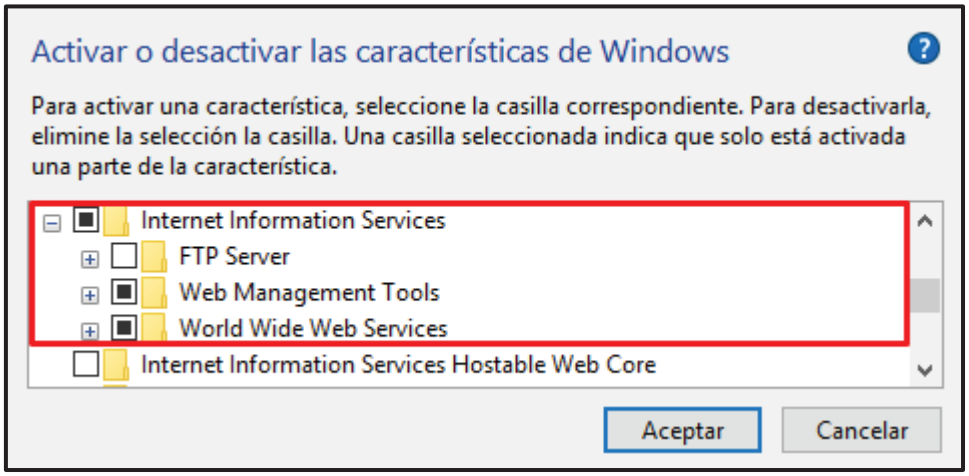

Figura 2.31 Activación de las características del servidor Web IIS.

b) Instalación de *Visual Studio* 2017

La versión utilizada es la versión sin costo que ofrece *Visual Studio,* conocida como *Visual Studio Community 2017* [40] como se puede observar en la Figura 2.32. Esta versión incluye características para el desarrollo de aplicaciones móviles por lo que ha sido el software seleccionado para el despliegue del prototipo distribuido de disponibilidad de parqueo. En la página oficial de *Microsoft* [41] es posible encontrar la versión antes mencionada.

Figura 2.32 Visual Studio Community 2017.

Una vez que el instalador sea descargado y ejecutado, aparece una interfaz la cual permite seleccionar los componentes que deseamos incluir en la instalación, como se muestra en la Figura 2.33.

Figura 2.33 Selección de características de instalación Visual Studio 2017.

Durante la selección existen 2 parámetros esenciales que deben ser incluidos en la instalación:

- **Desarrollo de escritorio de .NET:** Incluye el editor principal predeterminado de *Visual Studio*.
- **Desarrollo móvil con .NET:** Permite construir soluciones móviles multiplataforma, para dispositivos Andorid, iOS y Windows.

Una vez seleccionadas las características, se selecciona la opción ***instalar***. Los componentes antes seleccionados se instalan y están disponibles para para construir el proyecto.

c) Instalación y Configuración de SQL Server 2014

Microsoft SQL Server 2014 [42] es el motor de base de datos utilizado para almacenar los datos de las diferentes entidades del prototipo. El instalador al igual que *Visual Studio 2017*, puede ser encontrado en las descargas de la página oficial de *Microsoft* [43].

En la Figura 2.34 se muestra el centro de instalación para *SQL Server*. Primeramente, es necesario crear una nueva instancia de la base de datos añadiendo las características necesarias para el proyecto.

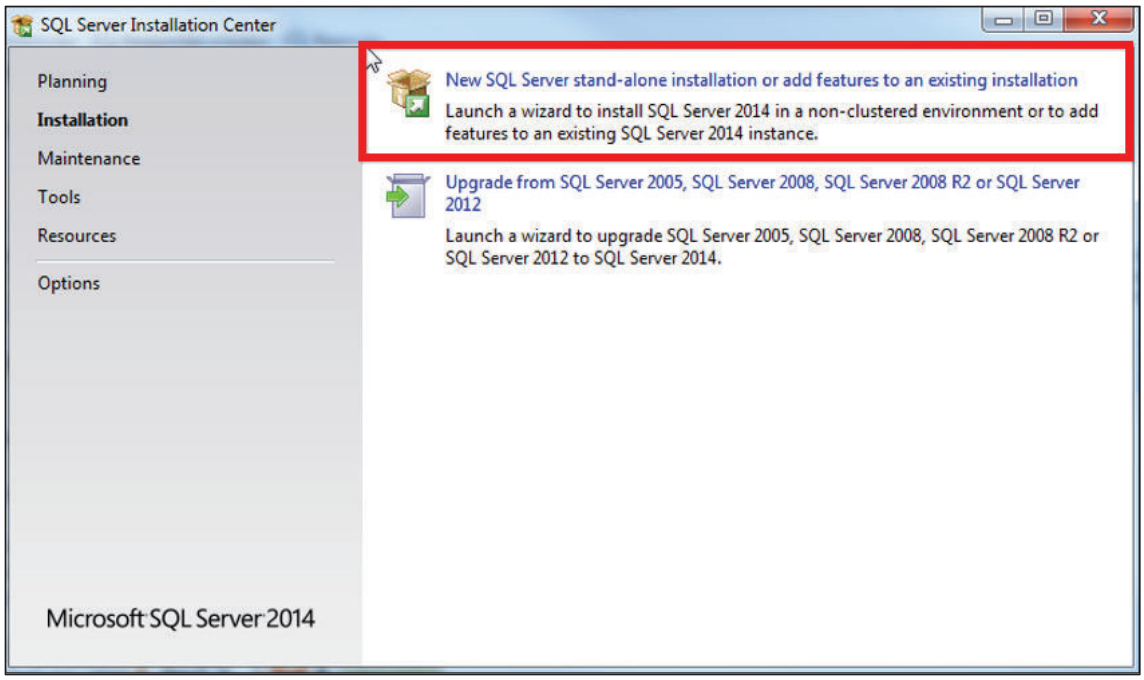

Figura 2.34 Centro de instalación de *SQL Server* 2014.

Se han agregado las características que por defecto brinda *SQL Server*. En la Figura 2.35 se muestra las características seleccionadas para la instalación y una vez instalados los componentes, el motor de base de datos está listo para registrar la información del prototipo.

Finalmente se instala el motor de base de datos con todos los componentes seleccionados.

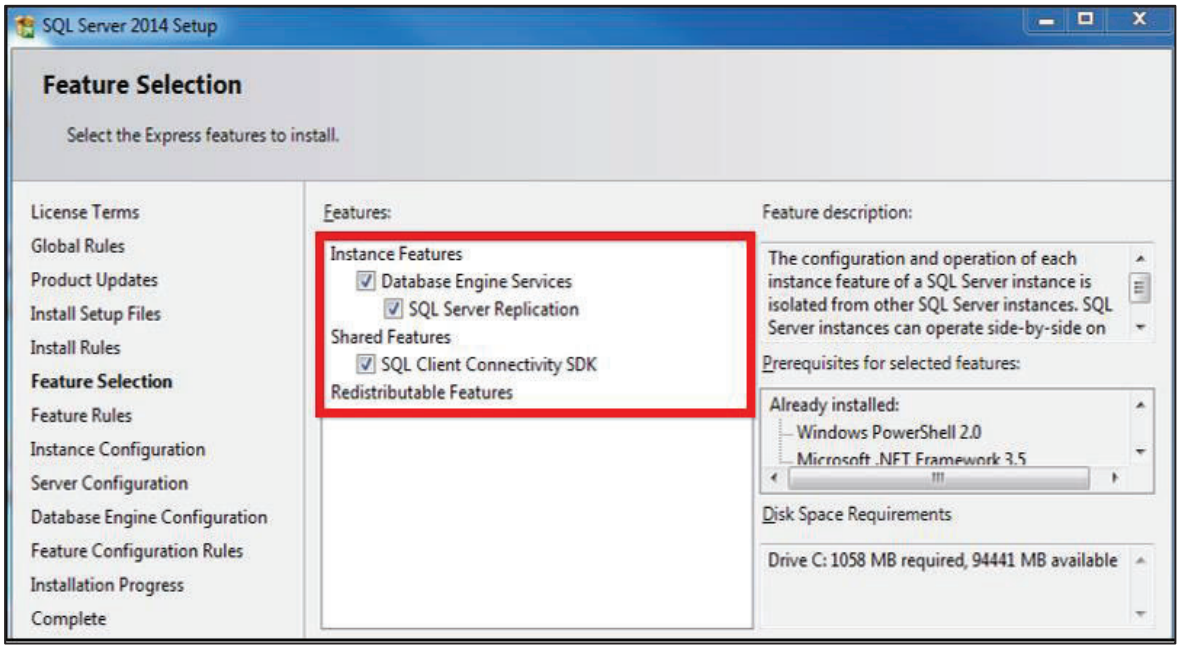

Figura 2.35 Selección características para la instalación de *SQL Server 2014.*

d) Instalación del IDE Arduino

Arduino es el hardware/software seleccionado para interactuar con el sensor de lazo magnético. El instalador de este IDE se lo puede encontrar y descargar en la página oficial de Arduino [44]. Los componentes seleccionados para la instalación del IDE de Arduino se muestran en la Figura 2.36.

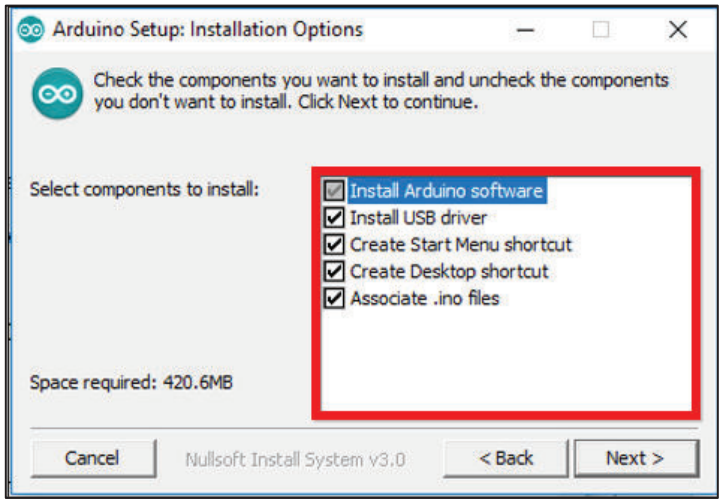

Figura 2.36 Instalación IDE de Arduino.

Una vez instaladas las herramientas se muestra a continuación la implementación de los diferentes componentes que conforman el prototipo.

2.3.2 Codificación de la Capa de Datos

Se ha creado la nueva base de datos, dando c*lic* derecho sobre "*Databases"* y seleccionando la opción "*New Database",* como se muestra en la Figura 2.37. Se despliega una segunda ventana donde se ingresa el nombre de la base de datos y se selecciona aceptar para completar la creación de la base.

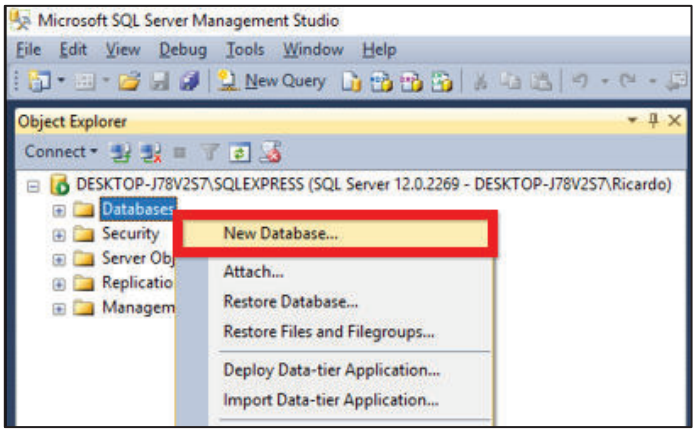

Figura 2.37 Creación Base de Datos.

La implementación de la base de datos se realizó en base al modelo presentado en el **Capítulo 2**, **Apartado 2.2.5**. La implementación consta de las siguientes tablas: USUARIO, VEHICULO, TARJETA_RFID, PARQUEADERO, COMENTARIO y NOTIFICACION.

En el Código 2.3 se detalla la creación de la tabla USUARIO. En el **Anexo I** se detalla la codificación de las demás tablas requeridas en el prototipo.

```
Create Table [Usuario]
(
            [IdUsuario] int NOT NULL,
            [Nombre]      nvarchar(45) NOT NULL,
            [Apellido]    nvarchar(45) NOT NULL,
            [Cedula]      nvarchar(10) NOT NULL,
            [Direccion]   nvarchar(45) NOT NULL,
            [Telefono]    nvarchar(45) NOT NULL,
            [Comentario]  nvarchar(45) NOT NULL,
            [Correo]      nvarchar(45) NOT NULL,
            [Horario]     Datetime NOT NULL,

            Constraint [pk_USUARIO] Primary Key ([Cedula])
)
go
```

Código 2.3 Creación de la tabla USUARIO.

Como se observa en el Código 2.3, el comando *Create Table* crea la tabla USUARIO. En el código se detalla los campos de la tabla, así como el tipo de dato de cada campo, como por ejemplo el campo **Nombre** corresponde a un tipo de dato *Varchar* de longitud 20. Adicionalmente el comando *Constraint* codifica la clave primaria para esta tabla, en este caso la clave primaria es el campo **IdUsuario** (Not Null).

La Figura 2.38 muestra una consulta de la tabla USUARIO dentro de la base de datos una vez que algunos valores han sido ingresados para verificar la funcionalidad de la misma.

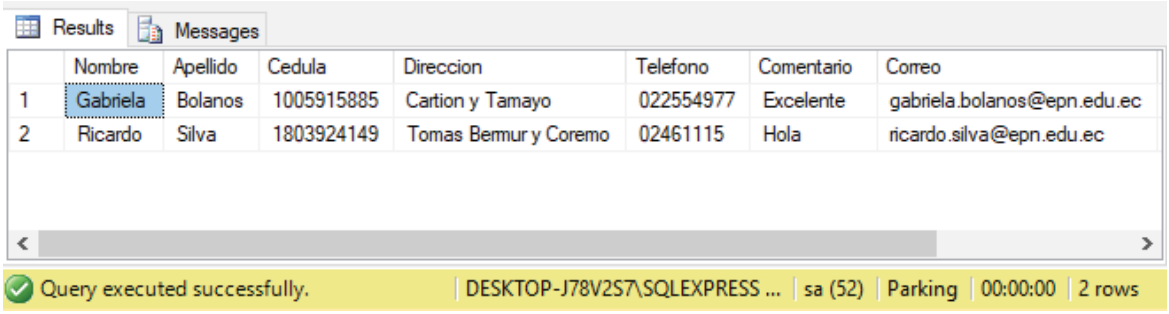

Figura 2.38 Registros ingresados en la tabla Usuario.

Una vez implementada la base de datos el siguiente paso es establecer la conexión entre *Visual Studio* y *SQL Server*.

2.3.3 Codificación de la Capa de Negocio

a) Codificación de Clases

Las clases implementadas en *SQL Server* deben ser codificadas en *Visual Studio* para así acceder a la información que estas almacenan. Para facilitar el proceso manual de codificación, se utilizó en *Visual Studio* un nuevo proyecto de Biblioteca de Clases, y dentro de este proyecto se hace uso del elemento *ADO.NET Entity Data Mode*l [45], elemento que permite acceder, crear, modificar o eliminar elementos de la base de datos. En primer lugar, se debe crear un nuevo proyecto de Biblioteca de clases (.NET framework) como se muestra en la Figura 2.39.

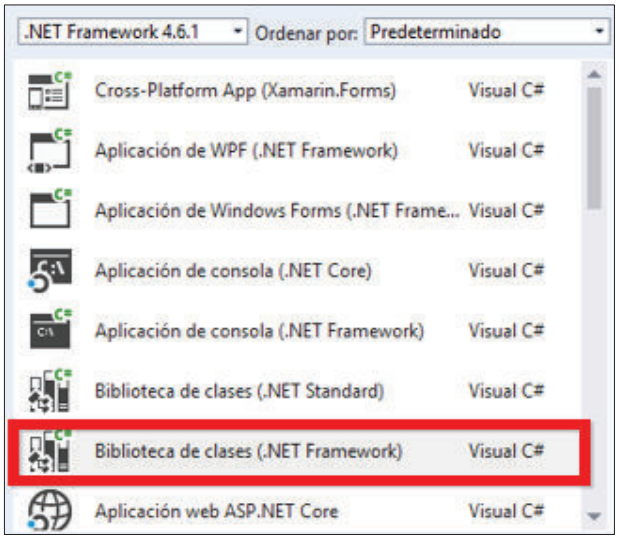

Figura 2.39 Creación Biblioteca de Clases.

En el proyecto Biblioteca de Clases se va a agregar el elemento *ADO.NET Entity Data Mode*l como se muestra en la Figura 2.40.

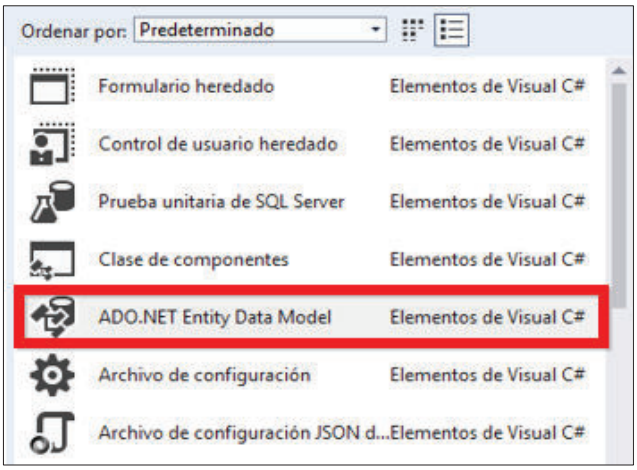

Figura 2.40 Creación de un nuevo elemento ADO.NET *Entity Data Model.*

Para crear la conexión hacia la base de datos se debe elegir la opción "*agregar contenido desde una base de datos existente*", siguiente, se selecciona el origen de datos que es *SQL Server* y se añade el nombre de la base de datos como se muestra en la Figura 2.41. El asistente despliega automáticamente la base de datos creada para el prototipo. Cabe recalcar que es importante seleccionar la opción "*guardar configuración de conexión en App.Config"*, caso contrario se debe añadir la cadena de conexión manualmente en el archivo ***App.config*** como se muestra en el Código 2.4.

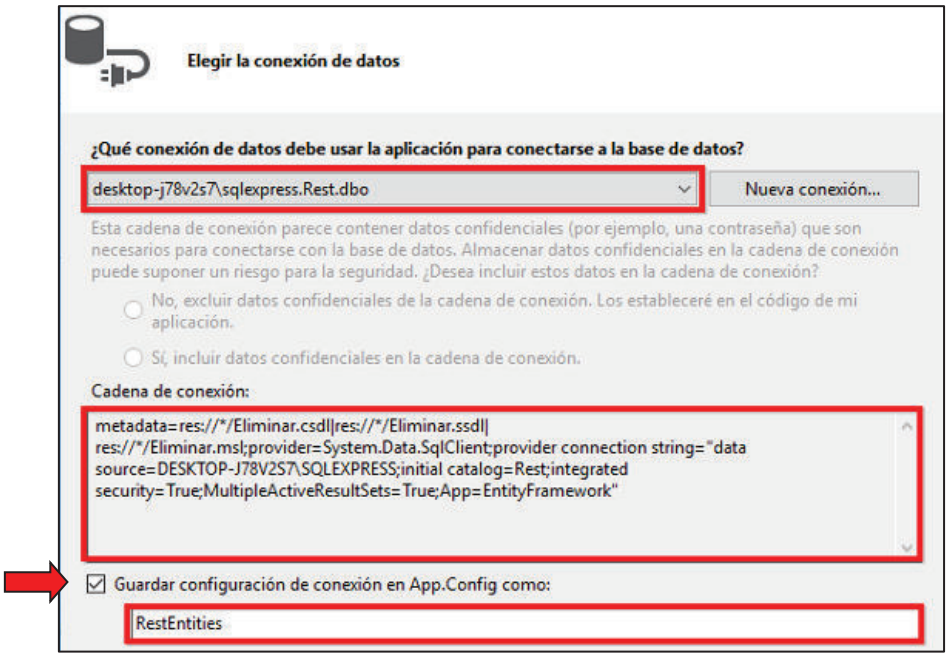

Figura 2.41 ADO.NET, configuración de la conexión.

```
<connectionStrings>
 <add name="RestEntities"
 connectionString="metadata=res://*/ModelREST1.csdl|res://*/ModelREST1.
       ssdl|res://*/ModelREST1.msl;provider=System.Data.SqlClient;provider
       connection string=&quot;data source=DESKTOP-J78V2S7\SQLEXPRESS;
       initial catalog=Rest;user id sa;password=Rikrkaka1989$;
       MultipleActiveResultSets=True;App=EntityFramework&quot;"providerName=
       "System.Data.EntityClient" />
</connectionStrings>
```

Código 2.4 Archivo App.Config. Cadena de conexión para la base de datos.

Una vez establecida la conexión con la base de datos se despliega en *Visual Studio* un archivo ***.emdx*** en donde se pueden visualizar las tablas creadas como se muestra en la Figura 2.42. El archivo ***.emdx*** encapsula el modelo de almacenamiento, el modelo conceptual y las asignaciones de la base de datos. Así, de esta manera, una vez creado el modelo es posible desde *Visual Studio* acceder a la información que se dispone en la base de datos.

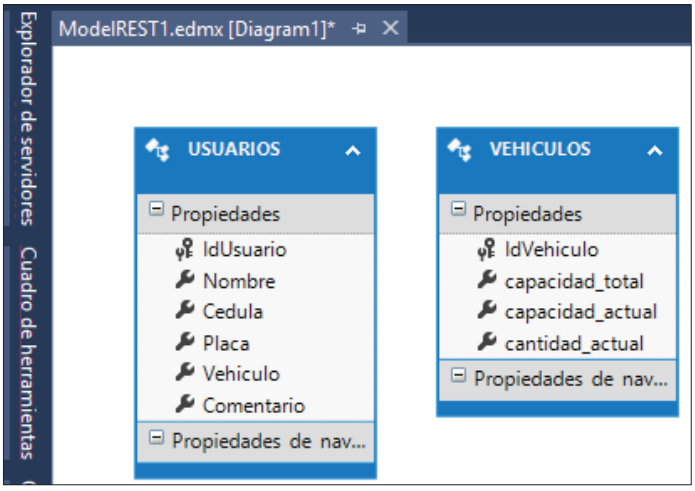

Figura 2.42 Modelo .edmx.

Posteriormente se codifican los servicios web que permiten al usuario interactuar con la Capa de Datos.

b) Codificación de los Servicios WEB

En este apartado se utiliza *Visual Studio 2017* para la creación y publicación de los servicios web. Los servicios web representan la conexión entre la Capa de Datos y la Capa de Presentación. Una vez dentro de *Visual Studio* el primer paso es crear el servicio web siguiendo el proceso detallado en [46]. Se creó un nuevo proyecto del tipo ***Aplicación web ASP.NET (.NET Framework)*** como se muestra en la Figura 2.43.

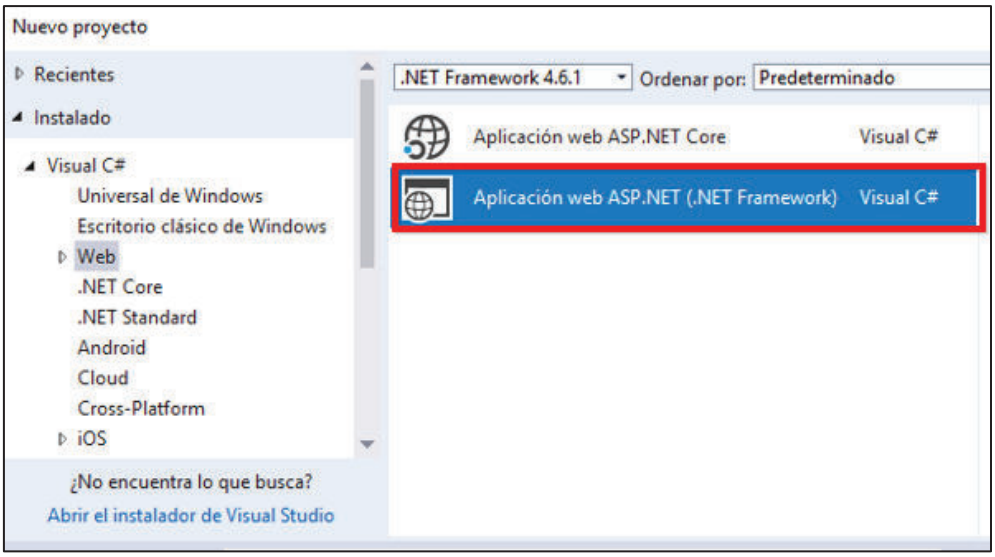

Figura 2.43 Creación de un nuevo proyecto ASP.NET.

Una vez creado el proyecto se generan varias carpetas y archivos como se muestra en la Figura 2.44.

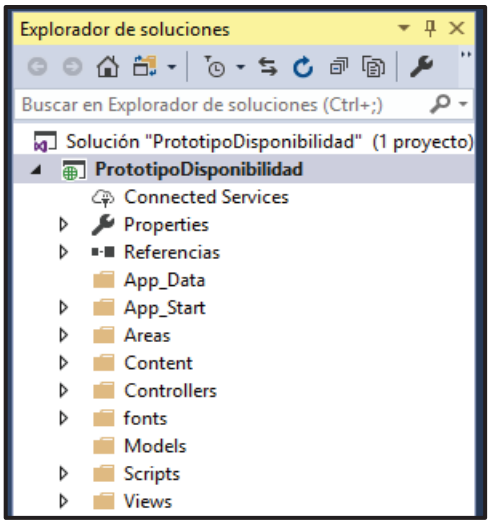

Figura 2.44 Carpetas del proyecto ASP.NET.

A continuación, se describen ciertos componentes que son de gran relevancia para la implementación del prototipo [47]:

- ***Content:*** Contiene los archivos de compatibilidad del contenido.
- ***Controllers:*** Contiene los archivos de controlador. Un controlador permite procesar solicitudes entrantes, gestionar datos proporcionados por el usuario y ejecuta la lógica de la aplicación.
- ***Global.asax:*** Proporciona una manera de responder a eventos de nivel de aplicación en una ubicación central;
- ***Web.config:*** Archivo que define la configuración de la aplicación, almacenado en formato XML.

Dentro del proyecto se implementaron varios controladores *(Controllers)* para desplegar las diferentes funcionalidades del prototipo según lo detallado en el **Capítulo 2 Apartado 2.2.2**. Para agregar un nuevo controlador se selecciona la carpeta ***Controllers***, *clic* derecho, agregar nuevo controlador del tipo *WEB API 2* como se muestra en la Figura 2.45.

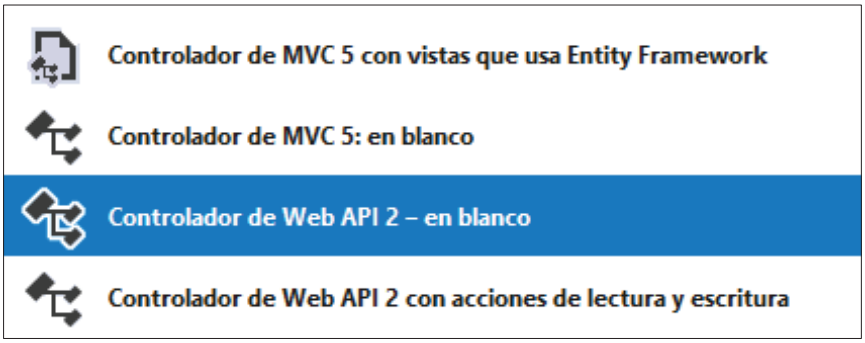

Figura 2.45 ADO.NET, agregar un nuevo controlador**.**

En el Código 2.5 se muestra la implementación del controlador encargado del proceso de registro de un usuario. El código completo de los controladores de autenticación y servicios de usuario que han sido implementados dentro del prototipo se detallan en el **Anexo II**.

```
1. namespace REST1.Controllers
2. {
3.     public class RegisterController : ApiController
4.     {
5.         public String Get(String id)
6.         {
7.             if (id.Contains("user=") && id.Contains(";pass="))
8.             {
9.              using (PruebaRestEntities entities = new   PruebaRestEntities())
10.                 {
11.
12.                     string pass = id.Substring(auxindex, auxint - uxindex);
13.                     pass = pass.Substring(1, pass.Length - 1);
14.                     USUARIOS AuxUser = new USUARIOS();
15.                     AuxUser.Correo = user;
16.                     AuxUser.Cedula = pass;
17.                     AuxUser.IdUsuario = (int)cantidad_actual + 1;
18.                    USUARIOS auxfind = entities.USUARIOS.FirstOrDefault(u =>
19.                    u.Correo == user);
20.                       if (auxfind == null)
21.                       {
22.                          USUARIOS aux = entities.USUARIOS.Add(AuxUser);
23.                          entities.SaveChanges();
24.                          return "OK";
25.                       }
26.                      else
27.                       {   return "error";   }
```

Código 2.5 Controlador para el registro de usuario.

Este controlador implementa un método público *GET* con un parámetro *String* como entrada como se puede ver en la línea 5 del código; luego en la línea 7 se define el *String* que contiene tanto el nombre de usuario y la contraseña para almacenar los datos del usuario. En las líneas 15 y 16 se lee la información agregada por el usuario, para luego en la línea 22 del código se añada la información del nuevo usuario mediante la sentencia *add.* Finalmente, en la línea 23 se guardan los cambios en la base de datos mediante la sentencia *SaveChanges.* Si el usuario no ingresa los datos en cualquiera de los 2 campos, el prototipo arroja un mensaje de error como se observa en las líneas 32 y 38.

c) Publicación de los Servicios WEB

Una vez que los controladores han sido implementados es necesario publicar el proyecto ASP.NET siguiendo los pasos descritos en [48]; de esta manera el usuario puede realizar las solicitudes haciendo uso de los servicios web. Dentro de *Visual Studio*, en la barra de menús, pestaña "*Compilar*", se selecciona la opción "*Publicar*" mediante IIS como se indica en la Figura 2.46.

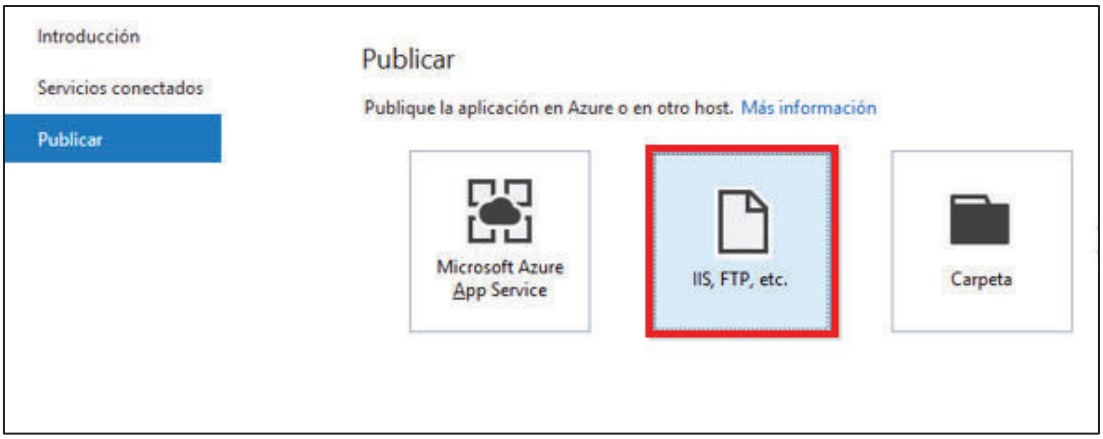

Figura 2.46 Publicar los servicios Web mediante servidor IIS.

Visual Studio despliega una segunda ventana en donde es necesario establecer varios parámetros de configuración como se indica en la Figura 2.47. Los parámetros que deben ser agregados se detallan a continuación:

- Método de publicación: *Microsoft Visual Studio* proporciona una extensión de IIS llamada *web deploy*; este método facilita la implementación de un proyecto de aplicación web.
- Nombre del servidor: Nombre o dirección IP del servidor que aloja el proyecto ASP.NET.
- Nombre del sitio web: Se debe colocar un nombre para el sitio Web.
- Dirección URL: Dirección a la cual se realizan las consultas y peticiones.

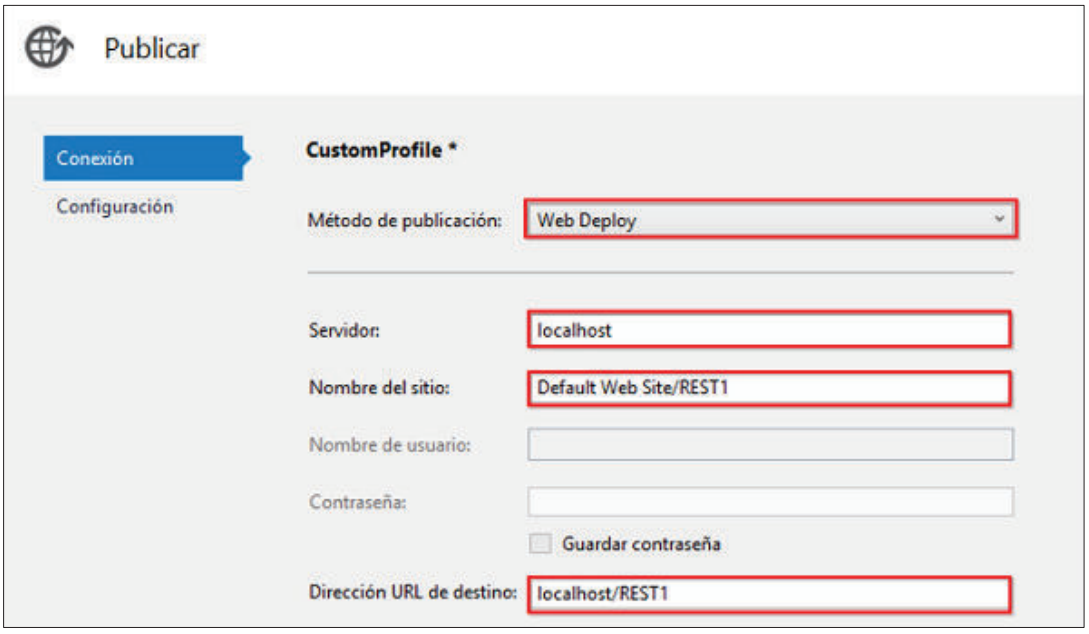

Figura 2.47 Configuración parámetros para publicar un Servicios Web.

d) Codificación de Notificaciones *Push*

Para la codificación de las notificaciones *push* es necesario crear el centro de notificaciones en el portal Azure y registrarse en los servicios GCM y APNS según lo detallado en el **Capítulo 2 Apartado 2.2.6**. Para la creación del centro se notificaciones se siguieron los pasos detallados en [49]. Se ha creado un recurso móvil del tipo *notifications hub*, según como se muestra en la Figura 2.48.

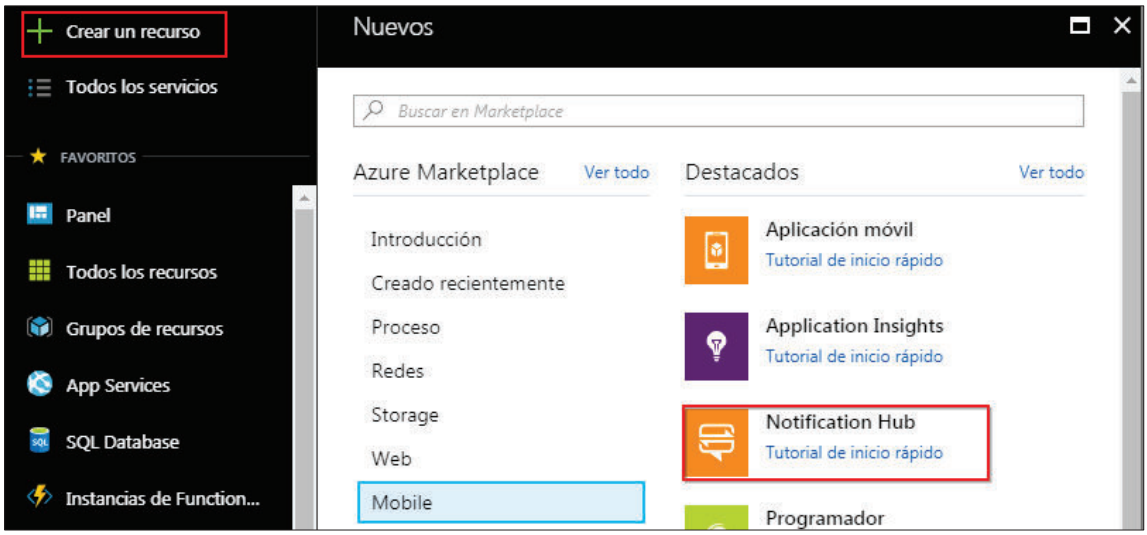

Figura 2.48 Portal Azure. Centro de Notificaciones.

Se ha colocado un nombre para el proyecto y en la parte de subscripción se ha seleccionado la opción free (permite el envío de hasta 1M notificaciones sin costo y permite trabajar hasta con 500 dispositivos) como se muestra en la Figura 2.49.

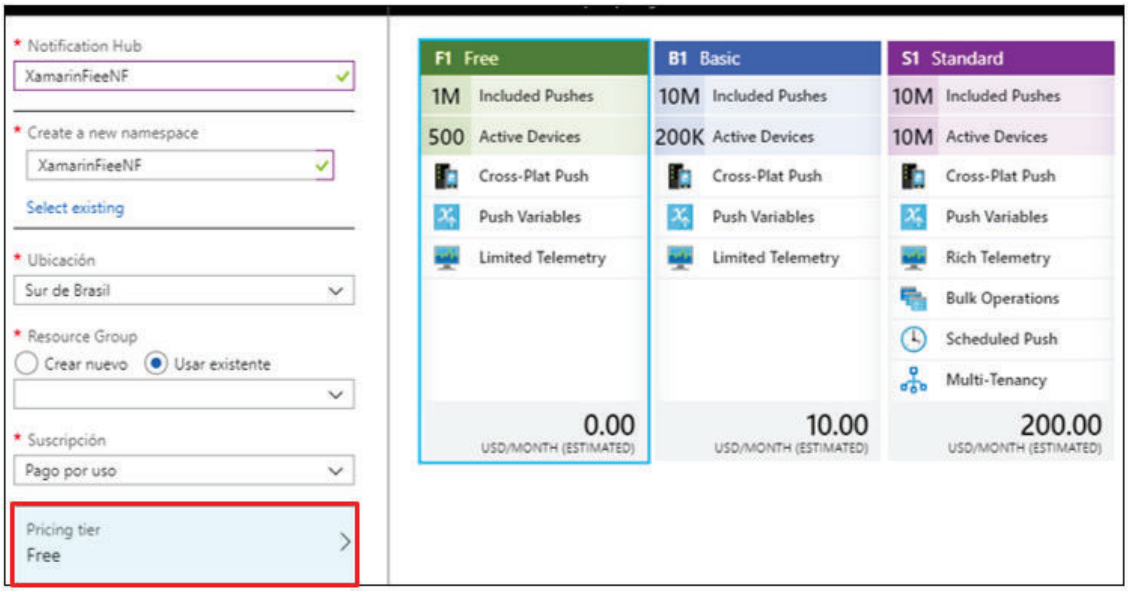

Figura 2.49 Centro de Notificaciones. Creación del recurso *notifications hub*.

Luego, una vez creado el centro de notificaciones es necesario configurar los parámetros para trabajar con dispositivos iOS y Android, en la pantalla principal del centro de notificaciones se muestran los diferentes sistemas operativos que se dispone para trabajar con *Azure* como se indica en la Figura 2.50.

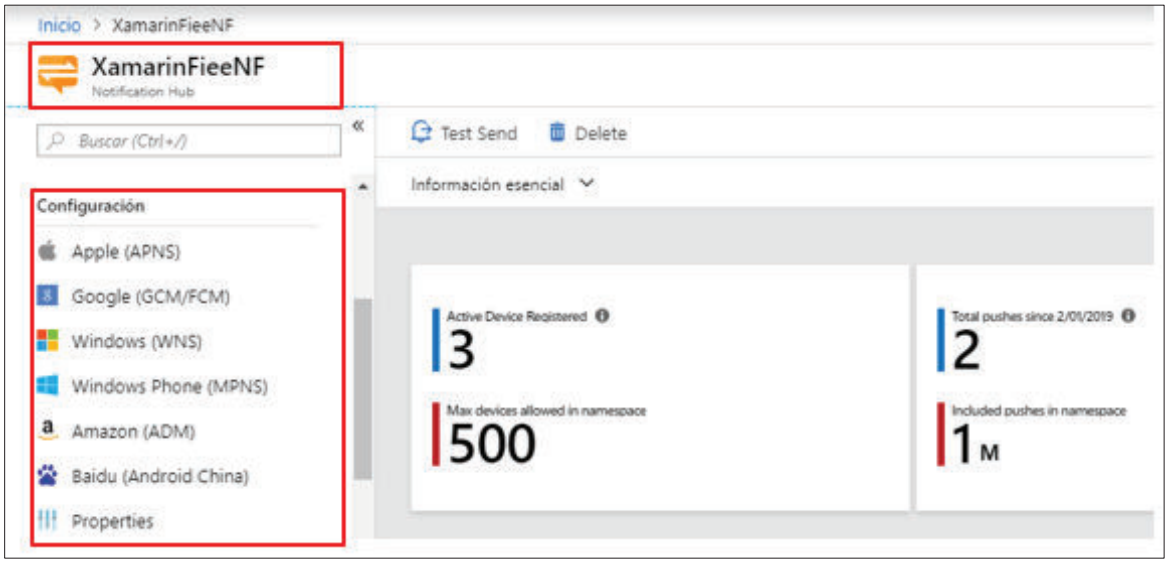

Figura 2.50 Configuración Centro de Notificaciones.

e) Android

Para configurar el envío de notificaciones a dispositivos Android según lo detallado en el **Capítulo 1 Apartado 1.3.9**, se requiere del centro de notificaciones *Firebase Cloud Messaging* (FCM), para difundir las notificaciones a los dispositivos que ejecuten la aplicación. Se ha creado un nuevo proyecto *Firebase* como se muestra en la Figura 2.51 siguiendo los pasos detallados en [49].

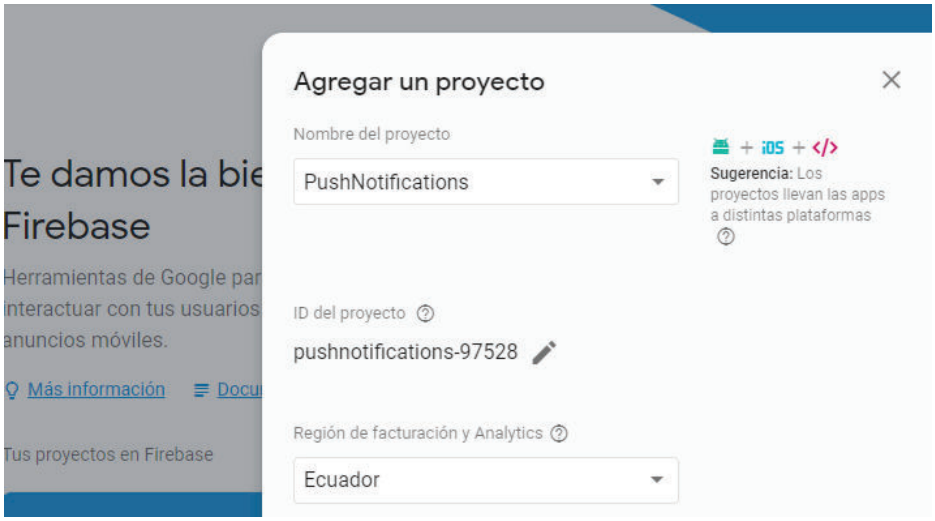

Figura 2.51 Firebase Google.

Una vez creado el proyecto es necesario seleccionar la opción "*agregar Firebase a tu Android app"* como se muestra en la Figura 2.52. Es de vital importancia añadir el nombre del paquete de Android que corresponde al campo *Company* detallado en las propiedades del proyecto *CrossPlataform* de *Visual Studio* como se muestra en la Figura 2.53.

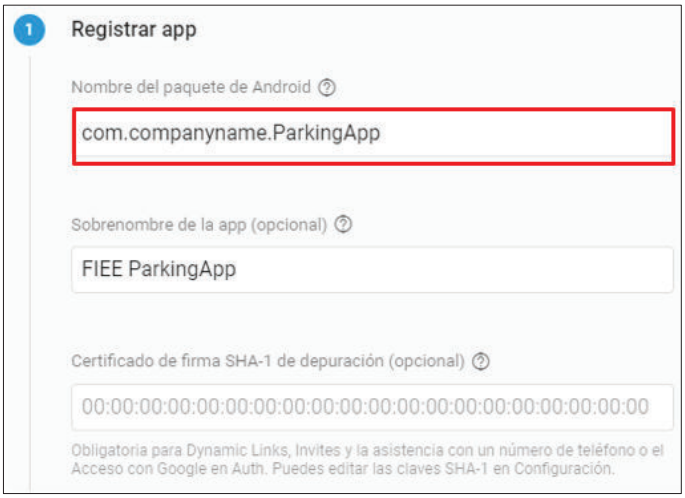

Figura 2.52 Firebase Google.

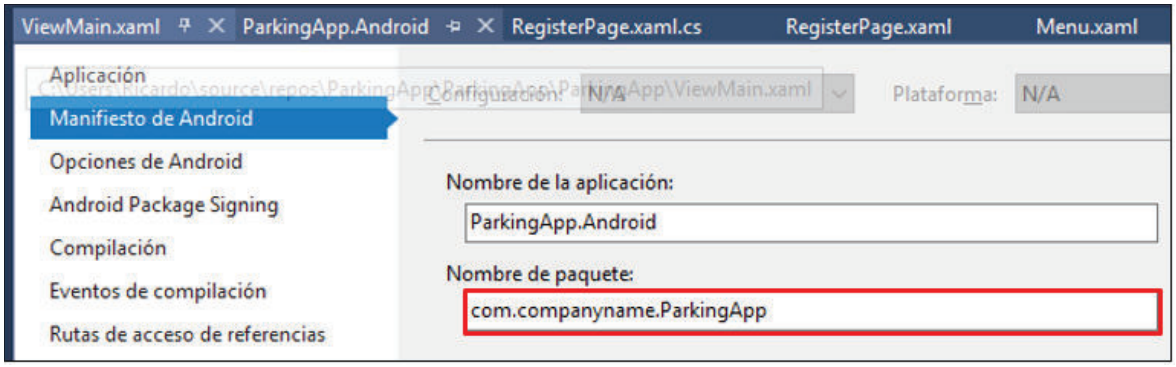

Figura 2.53 Propiedades *Proyecto Cross-Plataform Visual Studio.*

Una vez registrada la App, en la pestaña general del proyecto es necesario descargar el archivo *google-services.json* que contiene la clave de la API del servidor y el identificador de cliente como se indica en [49]; este archivo ha sido añadido en la carpeta ***resources*** del proyecto Android en *Visual Studio.*

Por último en la pestaña *Cloud Messaging* del *Firebase* de Google se toma el valor de *Legacy Server key* (Clave del servidor heredada) como se muestra en la Figura 2.54 [49].

Este valor es utilizado para configurar la directiva de acceso al centro de notificaciones, esta llave es colocada en portal de *Azure, Notification Service* en la sección GCM como se indica en la Figura 2.55.

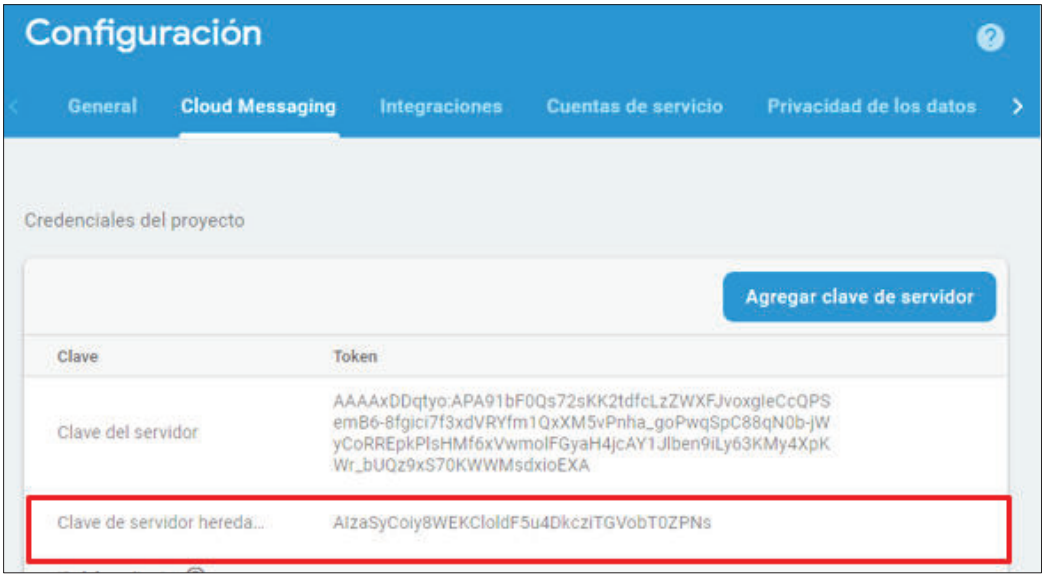

Figura 2.54 Firebase Google, Clave del Servidor

Figura 2.55 Azure, API Key.

Una vez configurado el centro de notificaciones es necesario agregar el código en el proyecto *Xamarin Forms* de *Visual Studio*. En el archivo **AndroidManifest.xml** del proyecto es necesario agregar la configuración mostrada en la Código 2.6. Esta configuración es necesaria para cualquier proyecto al que se le agrega notificaciones *push*.

```
<receiver android:name="com.google.firebase.iid.FirebaseInstanceIdInternalReceiver"
          android:exported="false" />
         <receiver android:name="com.google.firebase.iid.FirebaseInstanceIdReceiver"
          android:exported="true"
          android:permission="com.google.android.c2dm.permission.SEND">
             <intent-filter>
              <action android:name="com.google.android.c2dm.intent.RECEIVE" />
              <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
              <category android:name="${applicationId}" />
             </intent-filter
          </receiver>
```

Código 2.6 AndroidManifest.xml

En el código se puede apreciar la llamada a la instancia de *Firebase* de Google asociada a la aplicación desarrollada, y se registran las acciones que de desarrollan una vez

instalada la aplicación que son *REGISTRATION* para solicitar el registro a Firebase y *RECEIVE* que habilita al dispositivo que tiene instalada la aplicación a recibir notificaciones. El código completo para la inserción de notificaciones *push* en el proyecto Android se muestra en el **Anexo III**.

f) iOS

Para agregar notificaciones *push* hacia dispositivos con sistema operativo iOS es necesario cumplir con ciertos pasos que se detallan a continuación:

- **Generar un archivo de solicitud de firma de certificado.**

  Apple dispone de un servicio de notificaciones denominado *Apple Push Notification Service (APNS)* que utiliza un certificado para autenticar las notificaciones. Para generar el certificado es necesario tener un computador MAC y ejecutar la herramienta Acceso a llaves (*Keychain Access)* ubicada en la carpeta utilidades como se muestra en la Figura 2.56. Se requiere agregar el correo electrónico del usuario y agregarle un nombre a la solicitud [49].

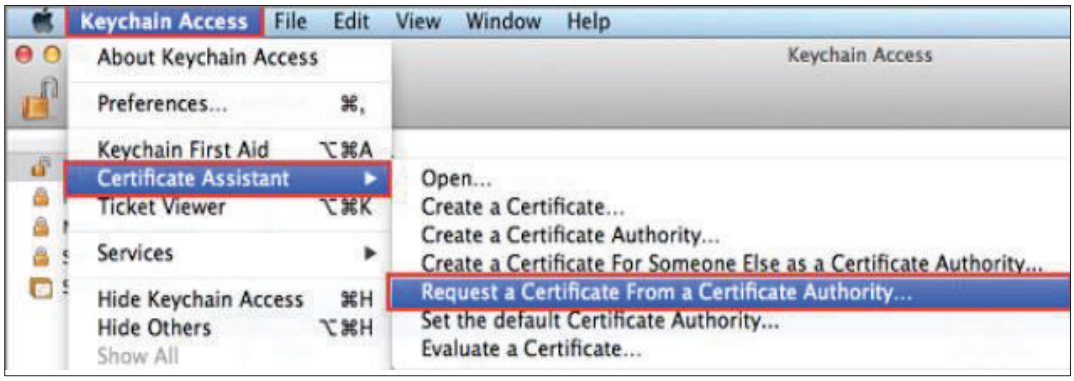

Figura 2.56 Solicitud de firma de certificado.

Se genera un archivo de solicitud de firma de certificado CSR (*Certificate Signing Request*). Una vez generado el archivo se requiere registrar la aplicación en Apple y habilitar el uso de notificaciones *push*.

- **Registro de la aplicación para notificaciones *Push***

Para poder enviar una notificación *push* a un dispositivo iOS se debe registrar la aplicación en Apple; esto se lo realiza mediante el portal de aprovisionamiento de iOS, en el centro de desarrolladores de Apple mostrado en la Figura 2.57, ingresando en la siguiente dirección [50].

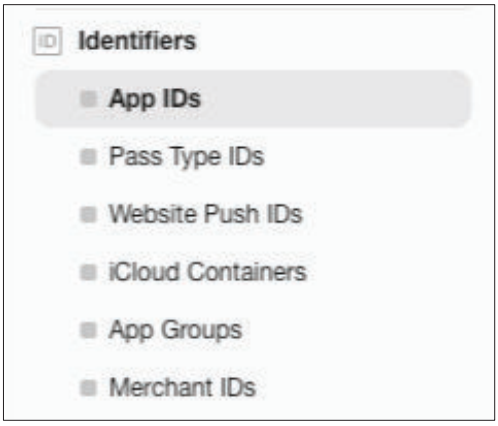

Figura 2.57 Centro de Desarrolladores Apple**.**

En la sección *Identificadores*, opción *App IDs*, se registra la aplicación como se muestra en la Figura 2.58 [50].

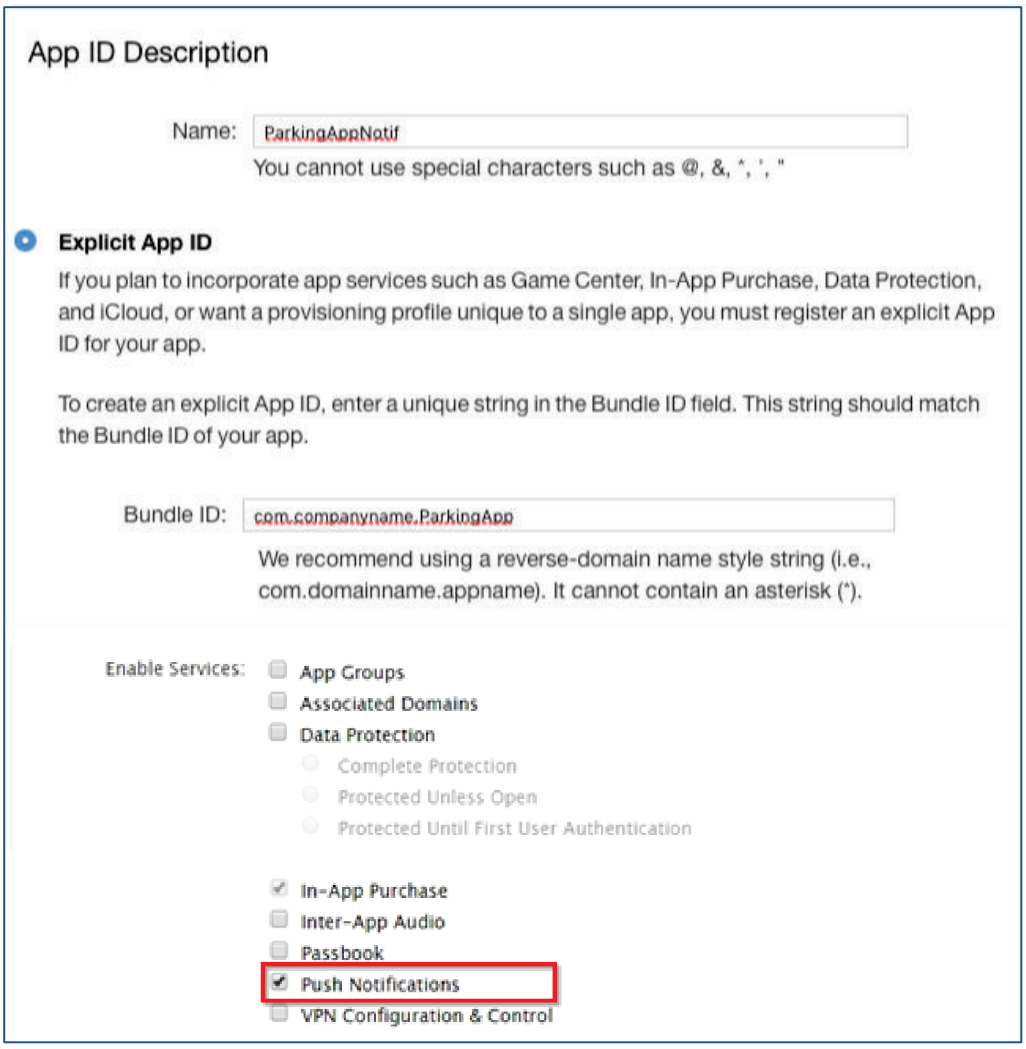

Figura 2.58 Registro de la Aplicación.

Es necesario llenar 3 campos obligatorios que se describen a continuación [50].

- *Name*: Describe el nombre para la aplicación.
- *Bundle Identifier*: En la opción Explicit App ID (Identificador explícito de aplicación), se añade el identificador de paquete, este debe coincidir con el nombre del producto de *Visual Studio* y que se agregó en las notificaciones Android, para el proyecto ***com.companyname.ParkingApp.***
- *Notificaciones push:* Se selecciona la opción Notificaciones *push*.

Una vez completa la información y seleccionada la opción de notificaciones se registra la aplicación. Luego, en el identificador *AppID* creado en el punto anterior, seleccionamos la opción "*edit"*, opción "*Create Certicate"*, ya que se requiere generar un certificado SSL [50], que representa un certificado de inserción para las notificaciones, como se muestra en la Figura 2.59.

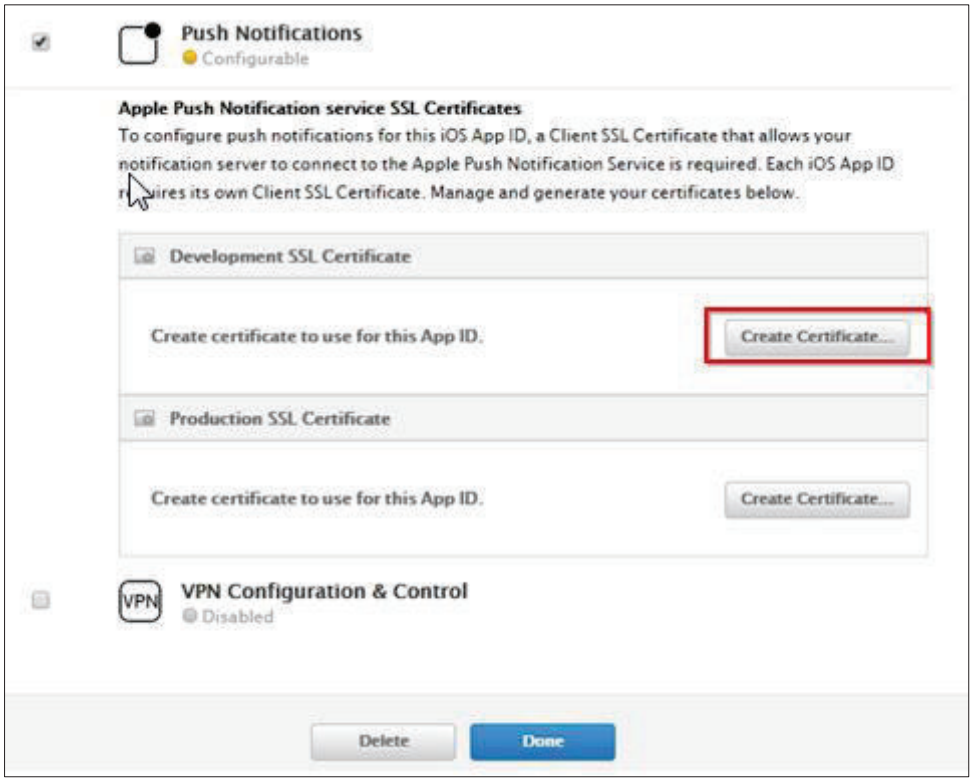

Figura 2.59 Agregar certificado de iOS.

En esta sección se requiere agregar en el centro de desarrolladores el archivo CSR que se generó en el PC MAC [50]. Se carga el archivo como se muestra en la Figura 2.60 y de esta manera se genera el certificado de iOS para el envío/recepción de notificaciones *push* en la aplicación desarrollada en el **Capítulo 2 Apartado 2.6.**

Figura 2.60 Certificado de la aplicación iOS

Este certificado generado en el centro de desarrolladores de Apple debe ser agregado como parámetro de autenticación en el portal *Azure* como se muestra la Figura 2.61.

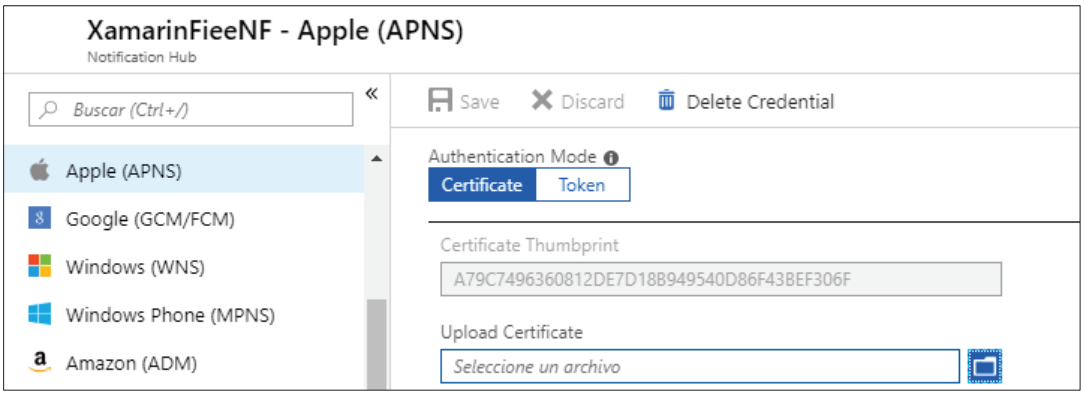

Figura 2.61 Configuración del servicio notifications hub de Azure para iOS.

Finalmente se requiere agregar el código necesario en *Visual Studio* para que la aplicación iOS pueda recibir notificaciones *push*. En el explorador de soluciones de *Visual Studio* en el proyecto para iOS como se muestra en la Figura 2.62, se modifica el archivo *AppDelegate.cs y* se añade el fragmento de Código 2.7. El código completo para la inserción de notificaciones *push* en el proyecto iOS se muestra en el **Anexo III**.

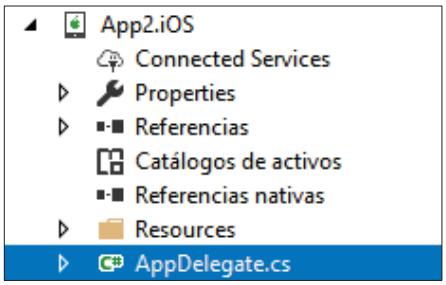

Figura 2.62 Proyecto iOS en Visual Studio

Cabe recalcar que para iOS y Android se modifican archivos propios de cada proyecto ya que cada sistema operativo utiliza servicios de mensajería (claves y certificados) diferentes.

```
1. using WindowsAzure.Messaging;
2.  void ProcessNotification(NSDictionary options, bool fromFinishedLaunching)
3.   {
4.    if (null != options && options.ContainsKey(new NSString("aps")))
5.    {
6.     NSDictionary aps = options.ObjectForKey(new NSString("aps")) as NSDictionary;
7.      string alert = string.Empty;
8.      if (aps.ContainsKey(new NSString("alert")))
9.      alert = (aps [new NSString("alert")] as NSString).ToString();
10.          if (!fromFinishedLaunching)
11.          {
12.            if (!string.IsNullOrEmpty(alert))
13.                  {
14.                        UIAlertView avAlert = new UIAlertView("Notification",
                          alert, null, "OK", null);
15.                        avAlert.Show();    }
```

Código 2.7 Código para la recepción de notificaciones *push* en *Visual Studio*.

g) Codificación del Subsistema Arduino

En este apartado se describe la implementación del servidor local que es el encargado de enlazar los módulos Arduino, como se muestra en el Código 2.8.

```
1. using System.Net;
2. using System.Net.Sockets;
3. namespace Chat_TCP
4. {
5.    public partial class Form1 : Form
6.    {
7.        int capacidad_total = 75; int numeroingreso = 0; int numerosalida = 0;
8.        public Form1()
9.        {
10.           InitializeComponent();
11.       }
12.
13.          Socket socket_escucha = new Socket(AddressFamily.InterNetwork,
             SocketType.Stream, ProtocolType.Tcp);
14.          Socket socket_cliente;
15.       private void Form1_Load(object sender, EventArgs e)
16.       {
17.           IPEndPoint ipservidor = new IPEndPoint(direccionServidor, 8888);
18.           socket_escucha.Bind(ipservidor);
19.           System.Console.WriteLine("El servidor enlazo el socket...");
20.           Thread hiloescucha = new Thread(new ThreadStart(Escuchar));
21.       }
22.       private void Escuchar()
23.       {
24.           while (true)
25.           {   socket_escucha.Listen(-1);
26.               System.Console.WriteLine("El servidor espera conexiones");
27.               socket_cliente = socket_escucha.Accept();
28.               System.Console.WriteLine("El servidor ha aceptado un cliente");
29.               if (socket_cliente.Connected)                {
30.                   Thread hilocliente = new Thread(new ThreadStart(Recibir));
```

Código 2.8 Código Servidor Escucha TCP.

En *Visual Studio* se añade un nuevo proyecto de *Windows Forms*. Es necesario agregar la referencia a la biblioteca *System.Net* y *System.Net.Sockets* para invocar los métodos para trabajar con *sockets* como se muestra en las líneas 1 y 2 del código. En la línea 13 se define el tipo de *socket* (tcp); luego en la línea 17 se define el *IP end point* en donde se define la ip del servidor y el puerto en donde el servidor local espera a los clientes; mediante el método ***bind*** mostrado en la línea 18, el servidor enlaza el *socket* y en la línea 26 mediante el método ***listen*** el *socket* escucha las solicitudes de los clientes. Si el servidor acepta un cliente se invoca al método ***Recibir*** en donde el servidor recibe los datos enviados por el módulo Arduino.

Posteriormente el servidor lee e interpreta la información que los módulos envían. En el segmento de Código 2.9 se muestra la recepción y el procesamiento de la información.

```
1.      private void Recibir()
2.      {
3.          Socket socketC = new Socket(AddressFamily.InterNetwork,
4.          SocketType.Stream, ProtocolType.Tcp);
5.          System.Console.WriteLine("Recibiendo datos...");
6.          while (true)
7.          {
8.              int cantidadbytesRecibidos = 0;
9.              byte[] bytesRecibidos = new byte[128];
10.              try
11              {
12.                  cantidadbytesRecibidos = socketC.Receive(bytesRecibidos);
13.                 if (cantidadbytesRecibidos != 0)
14.                  {   string datos = Encoding.ASCII.GetString(bytesRecibidos);
15.                      string texto = datos.ToString();
16.                  }
17.              }
18.              catch (Exception ex)
19.              {      System.Console.WriteLine("Error" + ex);
20.              }
21.          }
22.      }
23.      public void ActualizarItemLblConexions(string texto)
24.      {
26.          if (lblresul.Text == "IN")
27.          {   numeroingreso = numeroingreso + 1;
28.              System.Console.WriteLine(numeroingreso);   }
29.          else if (lblresul.Text == "OUT")
30.          {   numerosalida = numerosalida + 1;
31.              System.Console.WriteLine(numerosalida);    }
32.          else
33.                {   Console.WriteLine("Valor desconocido");    }
34.          int capactual = capacidad_total - numeroingreso + numerosalida;
35.          System.Console.WriteLine(capactual);   }
```

Código 2.9 Código Servidor TCP

Luego el servidor actualiza los valores de los contadores de ingreso y salida y obtiene el valor de los espacios disponibles en la FIEE, como se puede observar entre las líneas 34 y 35 del Código 2.9.

Por otro lado, en el Código 2.10 se muestra una porción del código empleado para la configuración del módulo Arduino, este envía una solicitud al servidor y se establece la conexión para él envío de información. El código completo se adjunta en el **Anexo IV**.

```
1. #include <Dhcp.h>
2. #include <Ethernet.h>
3. #include <EthernetClient.h>
4. #include <EthernetServer.h>
5. #include <EthernetUdp.h>
6. unsigned long tiempo; unsigned long tiempo_max= 1000;
7. int ledpin= 6;
8. int buttonPin = 7;
9. boolean estado= false;
10. int led = 7 ;
12. byte mac [ ] = { 0x00,0xAA,0xBB,0xCC,0xDE,0x03 } ;
13. byte ip [ ] = { 192, 168, 0, 81 } ;
14. byte server[ ] = { 192, 168, 0, 80 } ;  //ip del servidor
15. EthernetClient client;
16. void setup() {
17.    Ethernet.begin(mac, ip);
18.    Serial.begin(9600);
19.    delay (1000);
20.    Serial.println("Conectando...");
21.    Serial.println();
22.         if (client.connect( server , 8888 ))
23.         {    Serial.println("Conectado");  }
24.         else
25.         {    Serial.println("Conexion Fallida");  }
26.              }
27.  void loop() {
28.    if (client.connected())
29.   {
30.      while(digitalRead(buttonPin)==LOW)
31.      {    if(millis()-tiempo >= tiempo_max)
32.                {      estado = true;  } }
33.    if (estado == true)
34.    {    ejecuta();  //Envío de información  (client.println("IN");) }
35.    digitalWrite(buttonPin,HIGH); //Activa PullUp resistor
36.    digitalWrite(ledpin,LOW);
37.    tiempo= millis();
```

Código 2.10 Implementación Módulo Arduino al Ingreso de la FIEE.

En las líneas 8 y 9 del Código 2.10 se definen la dirección MAC y la dirección IP del cliente, mientras que en la línea 14 se define la dirección IP del servidor; luego en la línea 17 se realiza la solicitud del cliente hacia la dirección y puerto del servidor establecido. Si la conexión es exitosa el módulo Arduino envía la información de la entrada analógica que está siendo censada y refleja el número de veces que el sensor cambia su estado al detectar la presencia de un vehículo que ingresa/sale de la FIEE.

2.3.4 Codificación Capa de Presentación

En esta sección se presenta la implementación de las interfaces gráficas de los subsistemas según lo expuesto en el **Capítulo 2 Apartado 2.2.7**. Se ha creado en *Visual*

Studio un nuevo proyecto del tipo *Cross-Plataform* (proyecto para generar aplicaciones en diferentes sistemas operativos*)* como se muestra en la Figura 2.63, proyecto que permite desarrollar una aplicación que corre sobre dispositivos iOS y Android.

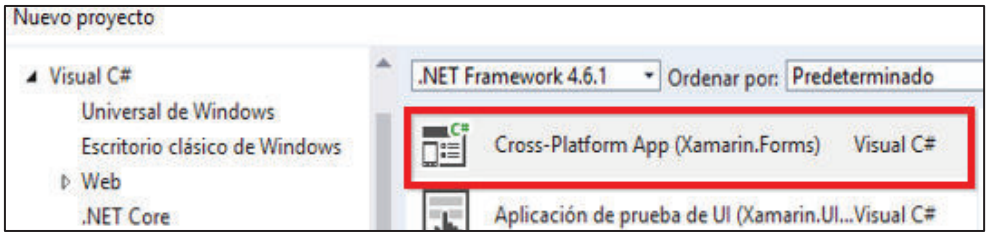

Figura 2.63 Creación del proyecto Cross-Plataform.

Dentro del proyecto *cross-plataform* se agrega un archivo *.XAML* que permite describir gráficamente interfaces de usuario siguiendo el Diseño de Interfaces mostrado en el **Capítulo 2 Apartado 2.2.7**. En el Código 2.11 se muestra el código implementado para la interfaz inicial de usuario que despliega 3 opciones que son *registro, ingreso y términos de uso*. A continuación, se describen algunos de los elementos utilizados:

a) ***Stacklayout***: Elemento que permite organizar vistas de manera unidimensional (como una pila), permitiendo colocar elementos secundarios de forma horizontal o vertical en función de la propiedad orientación.
b) ***Entry***: Permite ingresar texto que es solo una línea.
c) ***Button***: Permite controlar un evento o generar acciones dentro del proyecto.

```
1.  <StackLayout
2.     VerticalOptions="StartAndExpand" HorizontalOptions="CenterAndExpand">
3.          <Label x:Name="info1"
4.                      Font="32"
5.                      XAlign="Center"
6.                      Text="PARQUEADERO FIEE"
7.                      TextColor="DarkRed"/>
8.          <Image Source="EPN.png" WidthRequest="300" HeightRequest="200"/>
9.          <Entry x:Name="Ecorreo"
10.                Placeholder="Correo Institucional/Nickname"
11.                PlaceholderColor="LightBlue"
12.                Text="{Binding password}" />
13.         <Entry x:Name="Ecedula"
14.                Placeholder="Contraseña"
15.                PlaceholderColor="LightBlue"
16.                Text="{Binding user}"  />
17.         <Button Text="INGRESAR" Clicked="ButtonIngresar"></Button>
18.         <Button Text="REGISTAR" Clicked="ButtonRegistrar"></Button>
19. </StackLayout>
```

Código 2.11 Interfaz Gráfica de Inicio.

Todos estos elementos poseen atributos que el programador puede modificar como la posición, el color de fondo, o color de texto. En la Figura 2.64 se muestra el resultado final para la interfaz gráfica de inicio de la aplicación.

Figura 2.64 Página principal del prototipo de disponibilidad de parqueo.

Luego de implementar la interfaz gráfica de inicio se necesita una segunda página para el registro de usuarios en prototipo de disponibilidad de parqueo según lo expuesto en el **Capítulo 2 Apartado 2.2.7**. Se crea una segunda página similar a la página principal con los componentes gráficos necesarios para almacenar la información ingresada por el usuario. El diseño de la interfaz se muestra en la Figura 2.65.

Figura 2.65 Página de Registro del prototipo de disponibilidad de parqueo.

En el Código 2.12 se muestra el código implementado para realizar el registro de un nuevo usuario en la base de datos. El código completo en donde se detallan las clases implementadas para los demás servicios web se adjunta en el **Anexo III**.

```
1. public partial class RegisterPage : ContentPage
2.   {
3.        private string auxStr = "";
4.        Entry Epass = new Entry();
5.        Entry Euser = new Entry();
6.        public static HttpClient client;
7.
8.        public RegisterPage ()
9.          {
10.                InitializeComponent ();
11.                Euser = this.FindByName<Entry>("Ecorreo");
12.                Epass = this.FindByName<Entry>("Ecedula");
13.                client = new HttpClient();
14.               client.MaxResponseContentBufferSize = 256000;
15.         }
16.        async void ButtonRegistro(object sender, EventArgs e)
17.        {
18.            Button boton = sender as Button;
19.            password = Epass.Text; username = Euser.Text;
20.          if (password != "" && username != "" && password != null &&
21.                username != null)
22.              intentConnection();
23.                else
24.                 {
25.                await DisplayAlert("Faltan parámetros", "Denegado", `
26.                "Aceptar");
27.            }
29.        public async void intentConnection()
30.     {
31.          var uri = new Uri(string.Format(constants.RestUrlRegistro,
32.          String.Empty) + "user=" + username + ";pass=" + password);
33.
34.          try
35.             {
36.                var response = await MainPage.client.GetAsync(uri);
37.                if (response.IsSuccessStatusCode)
38.                {
39.                String content = await response.Content.ReadAsStringAsync();
40.                    String dat = content;
41.                    auxStr = dat;
42.                    if (auxStr.Contains("OK"))
43.                    {
44.                        await DisplayAlert("Bienvenido nuevo usuario", "OK");
45.                    }
46.            }
```

Código 2.12 Registro de un nuevo usuario.

Se define un *httpclient* (línea 6) que proporciona una clase base para enviar solicitudes HTTP y recibir respuestas HTTP de un recurso identificado por una URI (línea 31 y 32). Esta URI está definida es en la interfaz llamada “constants” en donde se muestra la estructura de la URI; en la solicitud se incluye la información ingresada con el usuario en donde a la URI se le añade “username” y “password” ingressados por el usuario.

Luego se realiza la petición, si falta cualquier parámetro se despliega un mensaje de error, caso contrário se llama al método "*intentConnection*" (línea 22) este método realiza la solicitud mediante el servicio web. Si la respuesta es correcta el usuario es agregado a la base de datos y se envía un mensaje de bienvenida.

La URI mostrada en la línea 31 y 32 del código se obtiene de una la interfaz llamada "constants", como se muestra en el Código 2.13. Esta variable contiene la dirección IP del servidor que publicó los servicios web y la URL mediante la cual se accede a los servicios.

```
interface constants
{
   public const string
   RestUrlRegistro="http://192.168.0.80/REST1/api/Register/";
   public const string
   RestUrlServicio="http://192.168.0.80/REST1/api/Servicios/";
}
```

Código 2.13 Archivo constants para almacenar las URL de los servicios web.

3. RESULTADOS Y DISCUSIÓN

En esta sección se muestran las pruebas realizadas al prototipo distribuido de disponibilidad de parqueo, así como el análisis de los resultados obtenidos siguiendo el tablero Kanban presentado en la Figura 3.1. El prototipo es sometido a pruebas de funcionalidad y aceptación realizadas en conjunto con funcionarios de la DGIP; todo esto para demostrar las características y funcionalidades del mismo.

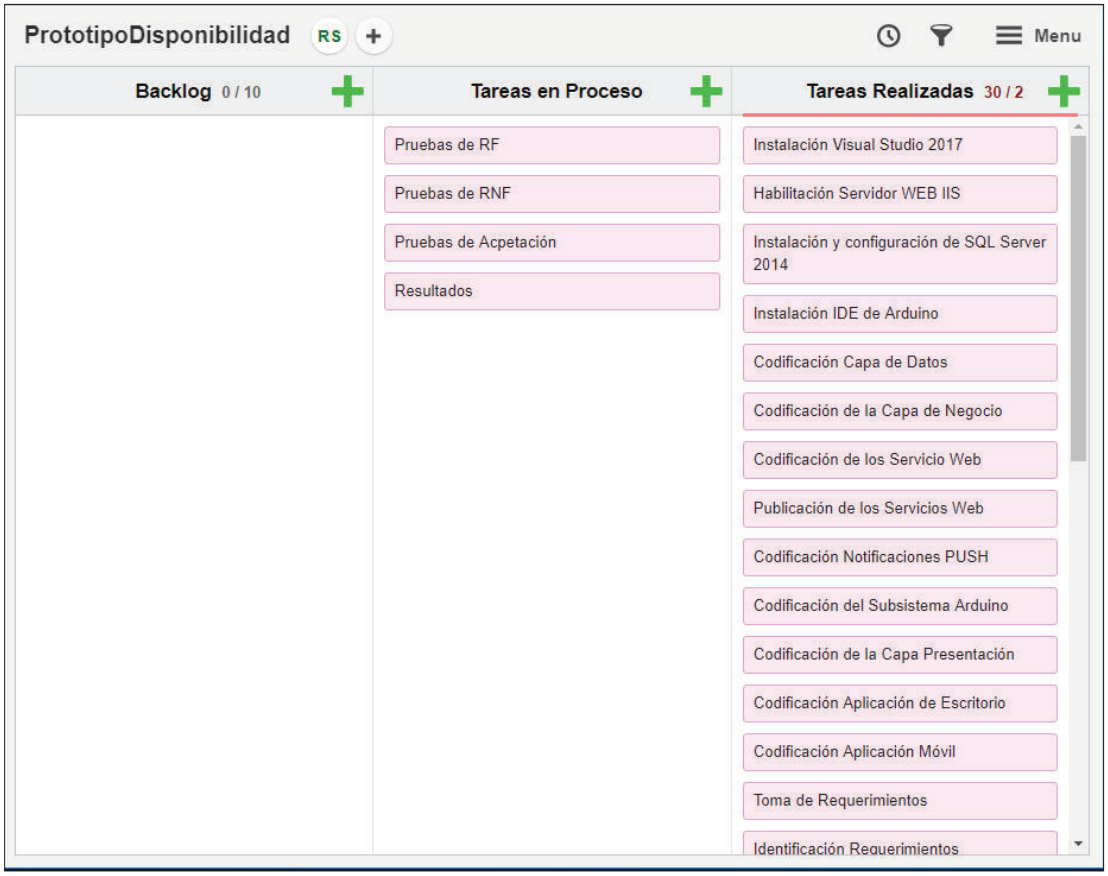

Figura 3.1 Tablero Kanban. Pruebas.

3.1 Pruebas Funcionales

Para la realización de las pruebas funcionales se va a evaluar cada uno de los requerimientos funcionales descritos en el **Capítulo 2 Apartado 2.2.2**, especificando el requerimiento asociado a cada subsistema descrito para el prototipo.

3.1.1 Prueba de Funcionamiento al Subsistema Arduino

En este subsistema se verifica la lectura y envío de información del sensor de lazo magnético para registrar y contabilizar la cantidad de vehículos que ingresan y salen del parqueadero de la FIEE.

Se coloca el módulo Arduino en el cajetín de entrada al parqueadero de la FIEE como se muestra en la Figura 3.2. El módulo Arduino lee constantemente la información del sensor y envía la información al servidor escucha encargado de interpretar los datos.

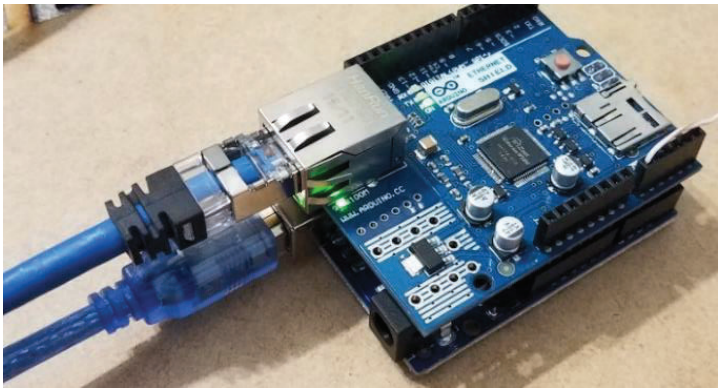

Figura 3.2 Módulo Arduino.

Los datos que son obtenidos a través del módulo Arduino y enviados a través de la red son receptados mediante el servidor local implementado en *Visual Studio,* los datos son interpretados e instantáneamente esa información se ve actualizada en la base de datos del prototipo. La Figura 3.3 muestra el cumplimiento de: a) RF01, b) RF02 y c) RF03.

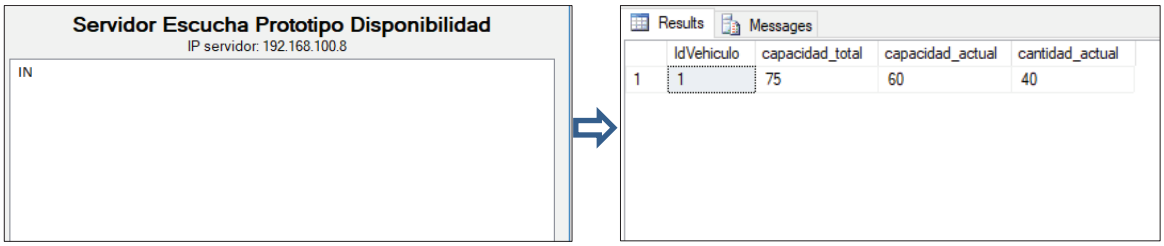

Figura 3.3 Lectura y envío de datos con Arduino.

Tabla 3.1 Pruebas de Funcionamiento al Subsistema Arduino.

SUBSISTEMA	RF ASOCIADO	DESCRIPCIÓN	STATUS
Subsistema Arduino	RF01	El módulo Arduino monitorea el sensor de lazo magnético, realiza la lectura y envío de datos.	OK
Subsistema Arduino	RF02	El prototipo permite la recepción e interpretación de los datos.	OK
Subsistema Arduino	RF03	El prototipo actualiza el valor de espacios disponibles en la base de datos.	OK

3.1.2 Prueba de Funcionamiento al Subsistema de Registro e Ingreso

Para este subsistema, las pruebas de funcionalidad se han realizado mediante el registro e ingreso de un usuario al prototipo distribuido de disponibilidad de parqueo.

El prototipo despliega el formulario para que cualquier usuario pueda registrarse en el sistema. Una vez que el formulario se encuentre completo y se guarden los cambios se visualiza un mensaje que indica que el usuario ha sido creado. La Figura 3.4 muestra el cumplimiento del RF04.

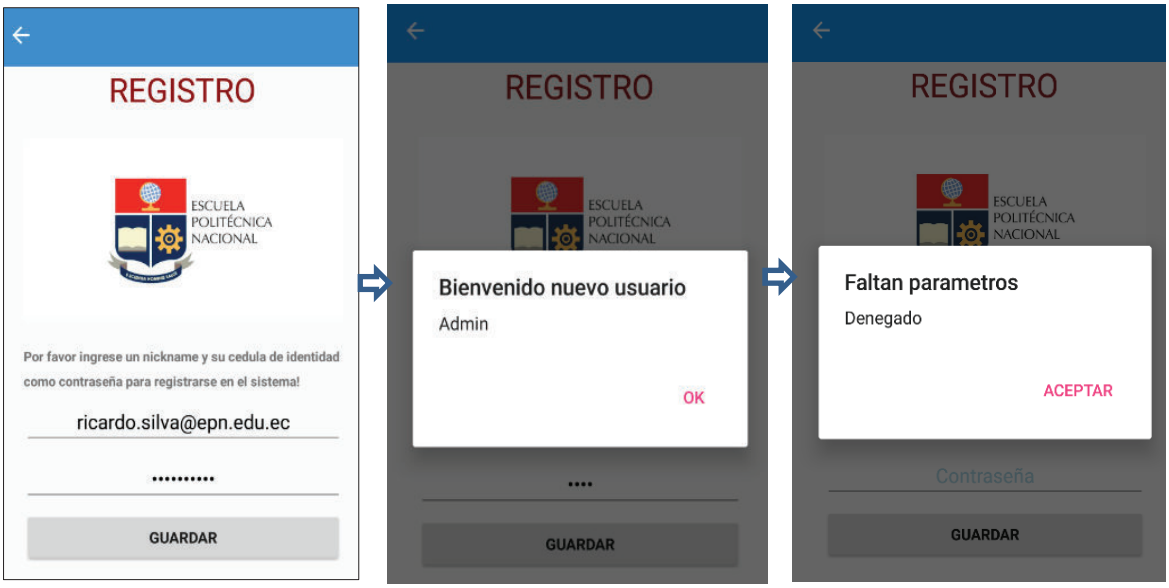

Figura 3.4 Pruebas de funcionalidad. Registro de un usuario.

La creación del usuario puede ser observada en la tabla USUARIO de la base de datos, como se muestra en la Figura 3.5.

Results | Messages

	Nombre	Apellido	Cedula	Direccion	Telefono	Comentario	Correo
1	Ricardo	Silva	1803924149	Tomas Bermur y Coremo	02461115	Hola	ricardo.silva@epn.edu.ec

Figura 3.5 Pruebas de funcionalidad. Registro de usuario en la base de datos.

Para el Requerimiento Funcional RF03 que corresponde a la autenticación, el usuario debe ingresar sus credenciales en la página inicial del prototipo.

Si la información coincide con los datos guardados en la base, el prototipo despliega un mensaje de bienvenida, caso contrario despliega un mensaje de error. La Figura 3.6 muestra el cumplimiento del RF05.

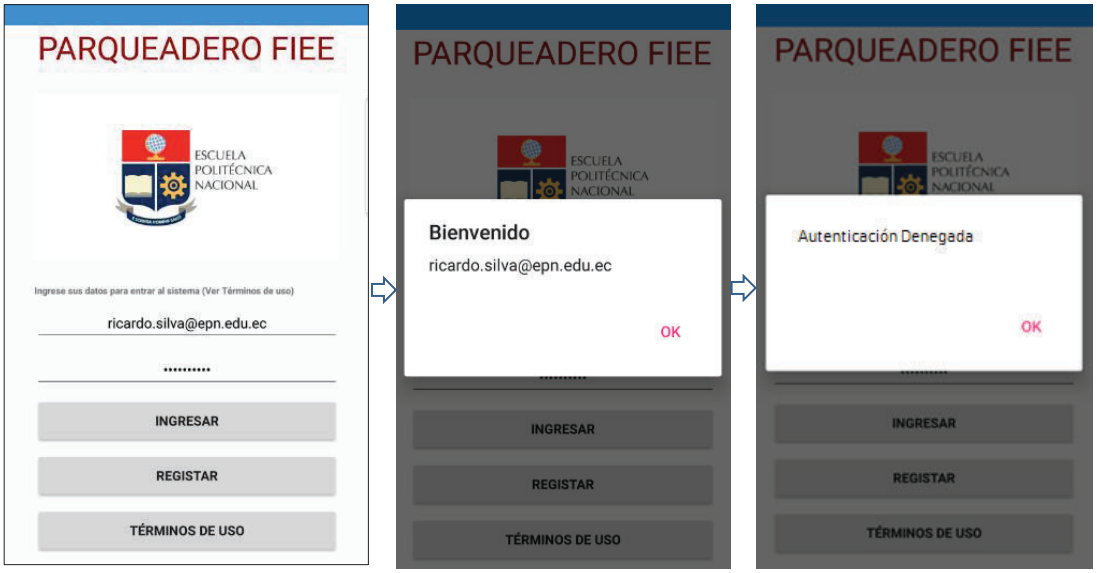

Figura 3.6 Pruebas de funcionalidad. Ingreso de un usuario.

Una vez que las credenciales del usuario han sido ingresadas y validadas en el sistema se despliega la interfaz de usuario como se muestra en la Figura 3.7

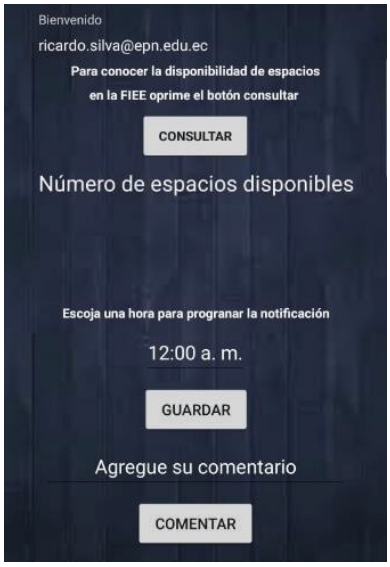

Figura 3.7 Página principal de la aplicación.

Tabla 3.2 Pruebas de Funcionamiento al Subsistema de Registro e Ingreso.

SUBSISTEMA	RF ASOCIADO	DESCRIPCIÓN	STATUS
Subsistema de Registro e Ingreso	RF04	La aplicación móvil permite la creación de un nuevo usuario	OK
Subsistema de Registro e Ingreso	RF05	La aplicación móvil permite autenticarse a un usuario.	OK

3.1.3 Prueba de Funcionamiento al Subsistema de Usuario

En el subsistema de usuario, la aplicación cuenta con varias funcionalidades. Dentro de la página principal el usuario dispone de la opción para consultar la disponibilidad del parqueadero de la FIEE en cualquier momento. La Figura 3.8 muestra el cumplimiento del RF06.

Figura 3.8 Pruebas de funcionalidad. Consulta de disponibilidad de parqueo.

Dentro del prototipo se tiene la posibilidad de programar un horario para recibir notificaciones. Una vez configurada la hora esta es almacenada en la base de datos y el usuario puede recibir la notificación del estado del parqueadero en su dispositivo móvil a la hora indicada. La Figura 3.8 muestra el cumplimiento del RF07 y RF09

Figura 3.9 Pruebas de funcionalidad. Configuración horario y recepción de la notificación.

La Figura 3.10 muestra el cumplimiento del RF08.

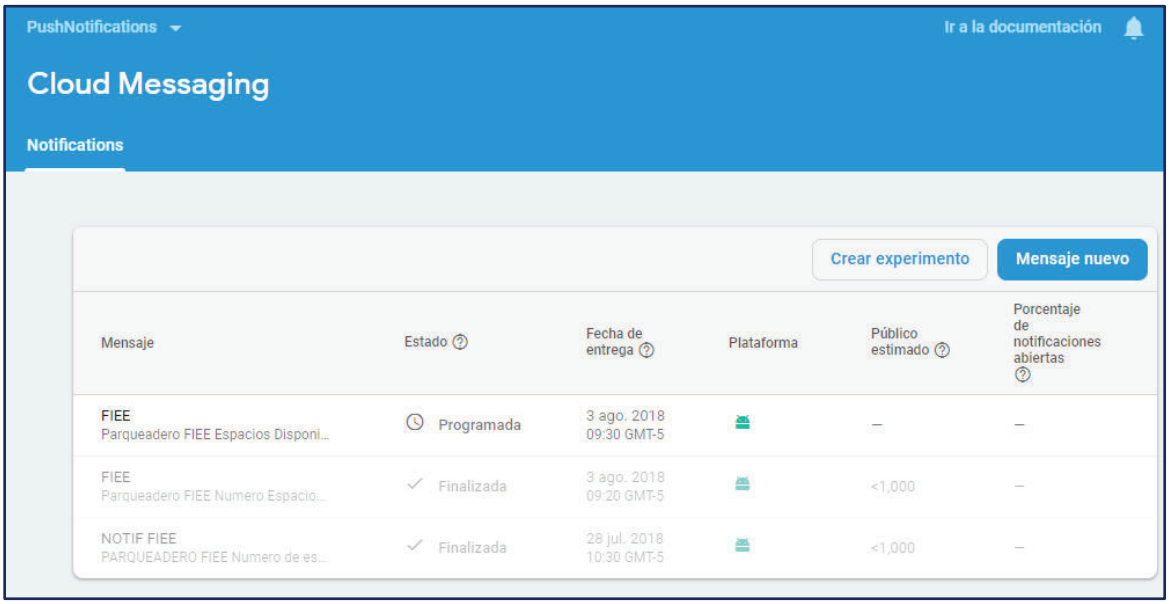

Figura 3.10 Estadísticas notificación en Cloud Messaging.

La Figura 3.11 muestra el cumplimiento del RF010.

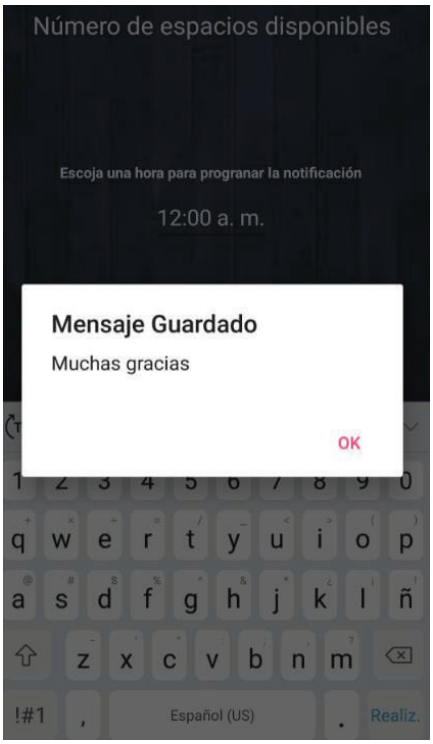

Figura 3.11 Pruebas de funcionalidad. Almacenar un comentario.

En la Tabla 3.3 se muestra el detalle de los requerimientos funcionales que se debieron cumplir en este subsistema.

Tabla 3.3 SB03 Pruebas de Funcionamiento.

SUBSISTEMA	RF ASOCIADO	DESCRIPCIÓN	STATUS
Subsistema de Usuario	RF06	La aplicación móvil permite visualizar la disponibilidad del parqueadero.	OK
Subsistema de Usuario	RF07	La aplicación móvil permite programar un horario para las notificaciones.	OK
Subsistema de Usuario	RF08	El prototipo envía la notificación al usuario en el horario establecido.	OK
Subsistema de Usuario	RF09	La aplicación móvil permite visualizar la notificación.	OK
Subsistema de Usuario	RF010	La aplicación móvil permite guardar un comentario del usuario.	OK

3.2 Pruebas de Requerimientos No Funcionales

Los requerimientos no funcionales detallados han sido evaluados según las especificaciones descritas en el **Capítulo 2, Apartado 2.2.3**. Los subsistemas han sido desarrollados para cumplir con los requerimientos no funcionales y brindar al usuario un prototipo fácil de manejar y utilizar.

3.2.1 Prueba de Requerimiento No Funcionamiento RNF01

El RNF01 hace referencia a la rapidez que se tiene en el prototipo. El tiempo que el prototipo tarda en contestar una solicitud al usuario no debe ser mayor a 5 segundos. Para realizar las pruebas de rapidez se utilizó un objeto *TimeSpam* que representa un intervalo de tiempo para completar una o varias instrucciones como se muestra en el Código 3.1.

```
TimeSpan stop;
TimeSpan start = new TimeSpan(DateTime.Now.Ticks);
          {
             Código a evaluar.
          }

 stop = new TimeSpan(DateTime.Now.Ticks);
 Console.WriteLine("Tiempo:" + stop.Subtract(start).TotalMilliseconds);
```

Código 3.1 Tiempo respuesta de los servicios Web.

A continuación se muestra el tiempo de respuesta (milisegundos) de los servicios web correspondientes al registro, login y consulta de disponibilidad.

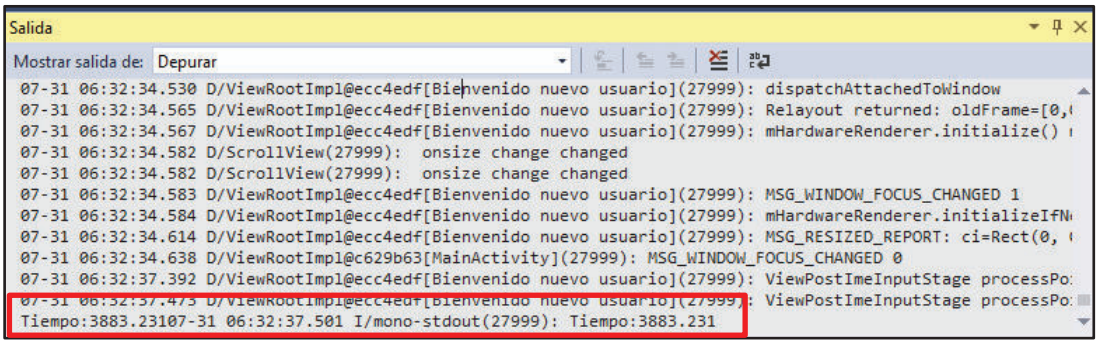

Figura 3.12 Tiempo de respuesta. Subsistema de Registro (mseg).

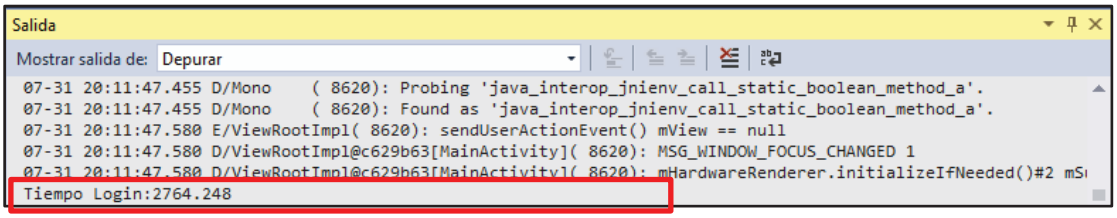

Figura 3.13 Tiempo de respuesta. Subsistema de Ingreso (mseg).

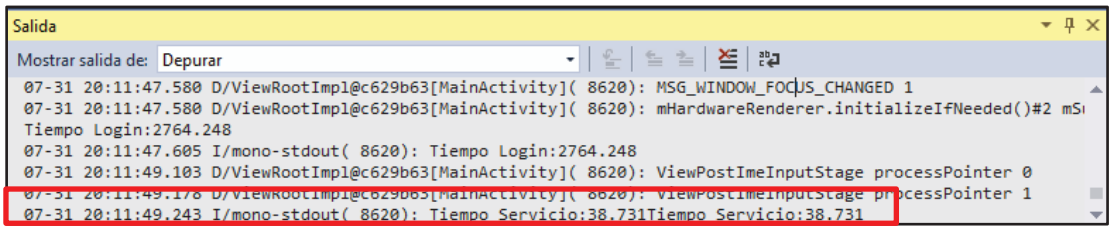

Figura 3.14 Tiempo de respuesta. Subsistema de Usuario (mseg).

Tabla 3.4 Resumen Pruebas RNF01.

Subsistema	Tiempo (ms)
Registro	3883
Ingreso	2764
Consulta Disponibilidad	38
Promedio	2228

Los tiempos de respuesta no sobrepasan los 5 segundos, por lo que se cumple con lo estipulado en el diseño del prototipo en el **Capítulo 2 Apartado 2.2.3**.

Tabla 3.5 Prueba RNF01.

REQUERIMIENTO	DESCRIPCIÓN	STATUS
RNF01	Rapidez	OK

3.2.2 Prueba de Requerimiento No Funcionamiento RNF02

Dentro del prototipo, la información que se muestra al momento de desplegar el número de espacios disponibles debe ser fiable y reflejar de forma precisa la disponibilidad que se tiene en el parqueadero de la FIEE. Para asegurar esta fiabilidad se implementó el proceso en el módulo Arduino como lo muestra el **Capítulo 2 apartado 2.3.3.**

El módulo trabaja constantemente y en caso de fallo de energía posee un respaldo de hasta 25 minutos para seguir operando. En una prueba realizada en el parqueadero de la FIEE, la disponibilidad del parqueadero actual es 45 espacios, luego del ingreso de 10 vehículos se verifica que la capacidad llega a 35 espacios. En la Figura 3.15, se puede visualizar la lectura de información y el valor reflejado en la base de datos luego del ingreso de los vehículos.

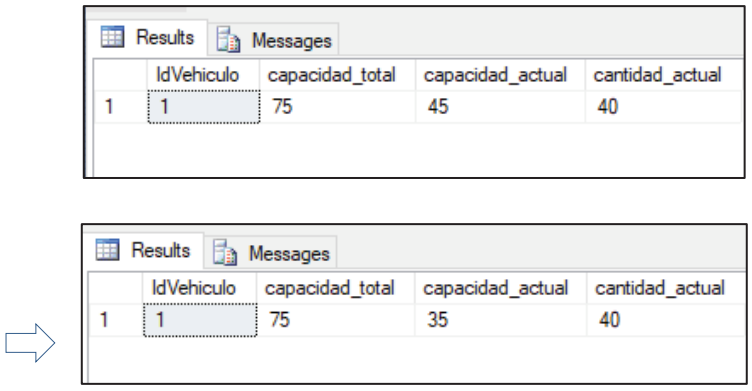

Figura 3.15 Prueba de Fiabilidad.

En la Tabla 3.6 se muestra el cumplimiento del RNF02.

Tabla 3.6 Prueba RNF02.

REQUERIMIENTO	DESCRIPCIÓN	STATUS
RNF02	Fiabilidad	OK

3.2.3 Pruebas de Aceptación. Requerimiento No Funcionamiento RNF03 y RNF04

Para verificar la facilidad de uso y la aceptación del prototipo que corresponden a los RNF03 Y RFN04, se realizó una encuesta a una muestra de 10 usuarios que utilizaron la aplicación. Los resultados obtenidos se muestran tabulados según la Tabla 3.8.

Las encuestas individuales de los usuarios se adjuntan en el **Anexo V**. Cada usuario que realizó la prueba la realizó varias veces, para que de esta manera el usuario pueda familiarizarse con el funcionamiento del prototipo, y así verificar el tiempo promedio en el cual un usuario puede ejecutar las opciones que brinda el prototipo.

Tabla 3.7 Pruebas de Aceptación.

PRUEBA	FACILIDAD DE USO [1-5]	TIEMPO (Seg)
Creación de usuario	5	20
Consulta disponibilidad	5	8
Guardar horario	5	12
Guardar comentario	5	15

Como se puede apreciar en los resultados obtenidos, el prototipo es calificado con un buen puntaje de aceptación en relación a su facilidad de uso. En la Tabla 3.7 se muestra el resultado de las pruebas referentes al RN03 Y RNF04

Tabla 3.8 Prueba RNF03 y RNF04.

REQUERIMIENTO	DESCRIPCIÓN	STATUS
RNF03	Usabilidad	OK
RNF04	Satisfacción	OK

3.3 Resumen

A manera de resumen en la Tabla 3.9 se muestra el detalle del cumplimiento de los requerimientos funcionales y los requerimientos no funcionales detallados en el desarrollo del presente Proyecto de Titulación. La Figura 3.16 muestra la organización de las tareas mediante el tablero Kanban en donde se observa el cumplimiento de las tareas definidas relacionadas con la fase de pruebas.

Tabla 3.9 Resumen Pruebas.

REQUERIMIENTOS FUNCIONALES DEL PROTOTIPO DISTRIBUIDO DE DISPONIBILIDAD DE PARQUEO		
REQUERIMIENTO	**DESCRIPCIÓN**	**CUMPLE**
RF01	El módulo Arduino monitorea el sensor de lazo magnético, realiza la lectura y envío de datos.	OK
RF02	El prototipo permite la recepción e interpretación de los datos.	OK
RF03	El prototipo actualiza el valor de espacios disponibles en la base de datos.	OK
RF04	La aplicación móvil permite la creación de un nuevo usuario	OK
RF05	La aplicación móvil permite autenticarse a un usuario.	OK
RF06	La aplicación móvil permite visualizar la disponibilidad del parqueadero.	OK
RF07	La aplicación móvil permite programar un horario para las notificaciones.	OK
RF08	El prototipo envía la notificación al usuario en el horario establecido.	OK
RF09	La aplicación móvil permite visualizar la notificación.	OK
RF010	La aplicación móvil permite guardar un comentario del usuario.	OK
RF011	La aplicación de escritorio permite visualizar la información de los usuarios.	OK

REQUERIMIENTOS NO FUNCIONALES DEL PROTOTIPO DISTRIBUIDO DE DISPONIBILIDAD DE PARQUEO		
REQUERIMIENTO	**DESCRIPCIÓN**	**CUMPLE**
RNF01	Rapidez	OK
RNF02	Fiabilidad	OK
RNF03	Usabilidad	OK
RNF04	Satisfacción	OK

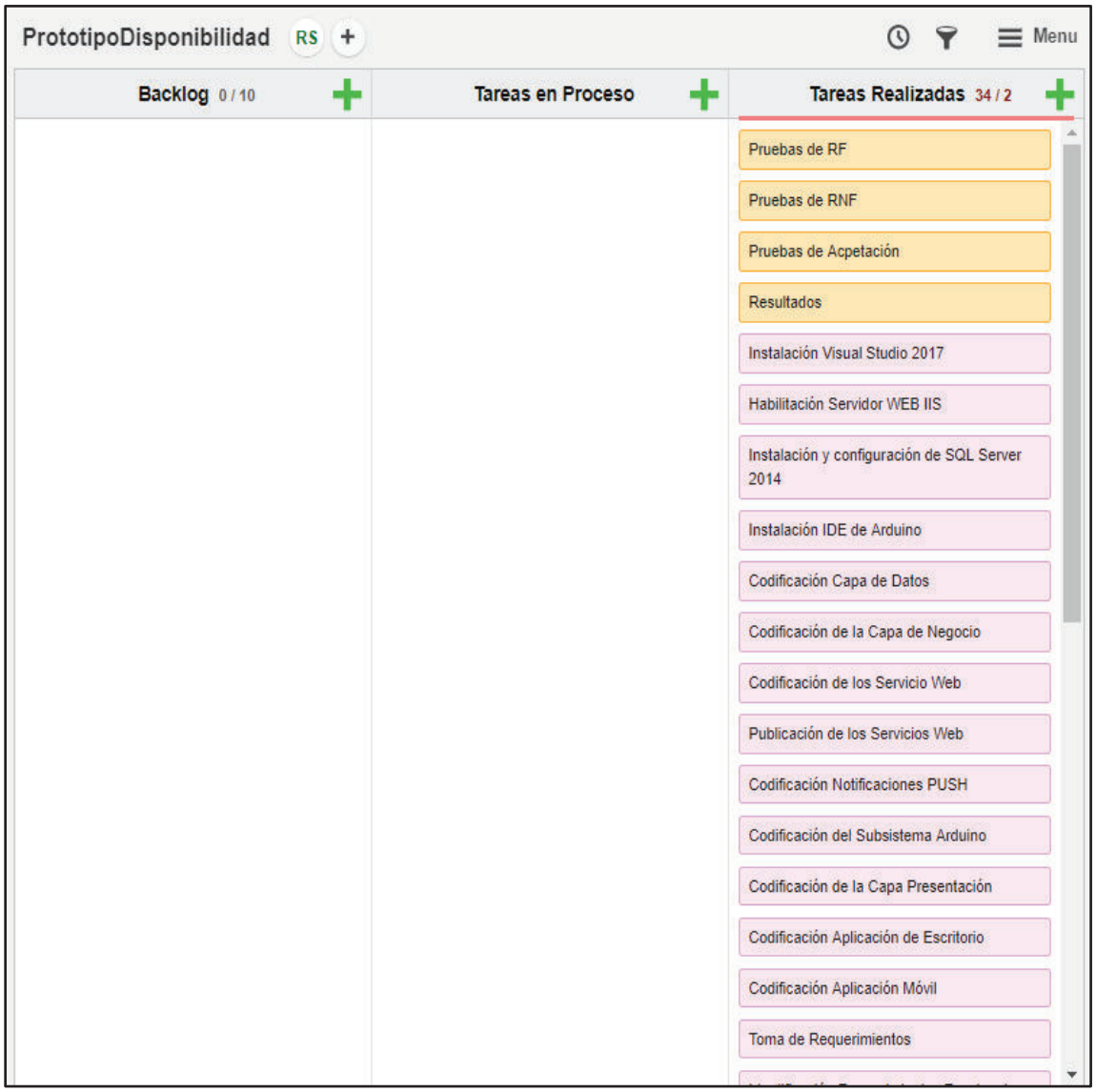

Figura 3.16 Tablero Kanban. Finalización de fase de pruebas.

4 CONCLUSIONES Y RECOMENDACIONES

En este apartado, en base a todo el proceso seguido en el desarrollo de este prototipo y en base a los resultados obtenidos se muestran las conclusiones y recomendaciones del prototipo distribuido de disponibilidad de parqueo de la FIEE.

4.1 Conclusiones

- El presente Proyecto de Titulación plantea una opción para complementar el actual sistema de ingreso al parqueadero de la FIEE, permitiendo consultar la disponibilidad de espacios del parqueadero desde un dispositivo móvil lo cual antes no se ofrecía con el sistema gestionado por la DGIP, permitiendo así brindar información oportuna al usuario mejorando la movilidad. Mediante la Capa de Datos, Capa de Negocio y Capa de Presentación y la interacción entre estas, ha sido posible implementar todas las funcionalidades necesarias para satisfacer los requerimientos solicitados en el diseño del prototipo.
- El prototipo hace uso de *Visual Studio 2017* que incluye *Xamarin* para el desarrollo de aplicaciones móviles multiplataforma partiendo de una capa de código común en C#, lo que ha sido una buena opción permitiendo así llegar a una mayor cantidad de usuarios ya que la aplicación corre sobre dispositivos que operan con sistema operativo iOS y Android.
- Mediante una plataforma informática en la nube como *Azure*, se implementó el envío de notificaciones *push* hacia los dispositivos móviles de los usuarios finales, en específico dispositivos que trabaja con sistema operativo iOS y Android; haciendo uso del servicio GCM (*Google Cloud Messaging*) brindado por Google y el sistema APNS (*Apple Push Notification Services*) ofertado por Apple. Las notificaciones permiten informar al usuario del número de espacios disponibles en la FIEE teniendo como únicos requisitos tener el *smartphone* encendido sin bloqueos para recibir notificaciones y tener instalada la aplicación en el dispositivo.
- El módulo Arduino permite monitorear continuamente el sensor de lazo magnético permitiendo detectar el ingreso/salida de vehículos cuando existe una modificación en el campo magnético del sensor. Por otro lado el módulo se conecta con el servidor local quien procesa la información obtenida y actualiza en la base de datos el registro que identifica el número de espacios disponibles.
- Las pruebas realizadas logran mostrar en detalle la funcionalidad y compatibilidad de los diferentes componentes del prototipo. Estas pruebas fueron satisfactorias y fueron realizadas por docentes de la EPN y funcionarios de la DGIP.

4.2 Recomendaciones

- Se recomienda el uso de la placa Arduino UNO para proyectos similares, en conjunto con el *Ethernet Shield* W5100 R3 ya que este módulo es compatible con Arduino Uno y Mega (Pines de la placa son comunes para ambos casos), que son las placas con las cuales se trabajó en el desarrollo de este proyecto. El *Ethernet Shield* provee un conector Ethernet estándar RJ45 que permite el envío de la información a través de la red de una manera rápida y sencilla.
- Un paquete *Nuget* permite a los desarrolladores crear, compartir y consumir código útil, permitiendo agregar referencias y consumir recursos al momento de escribir el código de programación. Varios paquetes *Nuget* vienen agregados al momento de ***instalar*** las características de *Visual Studio* sin embargo se recomienda actualizar los paquetes *Nuget* de la solución en especial el paquete *Entity Framework* que permite trabajar con las entidades disponibles en la base de datos. Es recomendable descargar la versión más estable que en ciertas ocasiones no corresponde a la última versión disponible.
- Para simular la aplicación en dispositivos móviles con sistema operativo Android es recomendable descargar el driver USB para dispositivos móviles ya que este driver es actualizado periódicamente para nuevos modelos. Es posible que *Visual Studio* no reconozca el dispositivo si este driver no está actualizado. El controlador puede ser descargado de la página oficial de SAMSUNG con el nombre SAMSUNG_USB_Driver_for_Mobile_Phones.
- Se recomienda habilitar los permisos del servidor WEB IIS para todos los usuarios, ya que al momento de realizar las solicitudes mediante servicios web se requiere tener permisos habilitados de lectura y escritura.
- Se recomienda configurar en *SQL Server* la autenticación mediante “usuario y contraseña”, esto debido a que al momento de agregar la cadena de conexión desde *Visual Studio* se requiere especificar estas credenciales, ya que al trabajar con las credenciales de Windows no se podrá establecer la conexión.

5 REFERENCIAS BIBLIOGRÁFICAS

[1] I. N. d. E. y. C. INEC, «Instituto Nacional de Estadística y Censos INEC. Anuario de transportes,» 2015. [En línea]. Available: http://www.ecuadorencifras.gob.ec//transporte. [Último acceso: 11 05 2017].

[2] I. T. y. A. Bull., «La congestión del tránsito urbano: causas y consecuencias económicas y sociales,» Revista de la CEPAL, nº 76, 2002. [En línea]. Available: http://www.cepal.org/publicaciones/xml/6/19336/lcg2175e_bull.pdf.

[3] CONVI, «Servicio y equipamineto. Smarth Parking.,» CONVI , [En línea]. Available: https://www.convi.net/que-es-el-smart-parking/.

[4] S. Curtis, «Smart parking app begins rollout in London's West End,» The Telegraph, 2014. [En línea]. Available: http://www.telegraph.co.uk/technology/news/10573651/Smart-parking-app-begins-rollout-in-Londons-West-End.html.

[5] P. Right, «Park Right, Smart Parking API.,» Park Right, [En línea]. Available: https://api.parkright.io/.

[6] I. F. A. M. Á. R. R. G. B. Arturo Baz Alonso, «Dispositivos móviles,» [En línea]. Available: http://isa.uniovi.es/docencia/SIGC/pdf/telefonia_movil.pdf.

[7] A. K. Saha, «A Developer's First Look At Android,» [En línea]. Available: https://s3.amazonaws.com/academia.edu.documents/30551848/andoid--tech.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1507299340&Signature=1PFpMUl6lq%2BOx9LlaBZj6WBjX0Q%3D&response-content-disposition=inline%3B%20filename%3DWhat_is_Android.pdf.

[8] J. C. A. F. A. W. P. R. Paco Blanco, «Metodología de desarrollo ágil para sistemas móviles Introducción al desarrollo con Android y el iPhone,» [En línea]. Available: https://www.researchgate.net/profile/Antonio_Fumero/publication/267795011_Metodologia_de_desarrollo_agil_para_sistemas_moviles_Introduccion_al_desarrollo_con_Android_y_el_iPhone/links/577009d108ae842225aa444b/Metodologia-de-desarrollo-agil-para-sistemas-m.

[9] Apple, «iOS Technology Overview,» Apple, 2010. [En línea]. Available: http://developer.apple.com/library/ios/navigation/.

[10] J. E. R. C. A. C. A. A. Karen Melissa Rojas Lizarazo, «Mobile Apps Development on the Iphone's Platform.,» 2011. [En línea]. Available: https://www.redalyc.org/pdf/4139/413940770007.pdf.

[11] E. Herrera Uribe y L. E. Valencia Ayala, «Del manifiesto ágil sus valores y principios.,» *Scientia Et Technica,* vol. vol. XIII, nº 34, pp. 381-386, mayo 2007.

[12] N. Figuerola, «Kanban Su uso en el desarrollo de software,» [En línea]. Available: https://articulosit.files.wordpress.com/2011/11/kanban.pdf.

[13] F. R. C. E. S. L. Edmundo Bonilla Huerta, «Advances in Intelligent Information Technologies,» [En línea]. Available: http://www.rcs.cic.ipn.mx/rcs/2014_79/RCS_79_2014.pdf#page=97.

[14] D. L. Arias, «Metodologías de desarrollo de software,» [En línea]. Available: http://www.alfarosolis.com/content/PDFs/IF7100/semana8/Metodologias.pdf.

[15] J. D. F. M. J. M. V. Andrés Navarro Cadavid, «A review of agile methodologies for software development,» [En línea]. Available: https://www.redalyc.org/pdf/4962/496250736004.pdf.

[16] J. Highsmith, Agile Software Development Ecosystems, Addison-Wesley, 2002.

[17] J. S. y. J. Hernández, «SQL SERVER VS MySQL,» [En línea]. Available: https://iessanvicente.com/colaboraciones/sqlserver.pdf. [Último acceso: 14 06 2017].

[18] Microsoft, «Microsoft SQL SERVER Microsoft 2014,» Microsoft, 2014. [En línea]. Available: https://msdn.microsoft.com/es-es/library/ms130214(v=sql.120).aspx.

[19] R. S. Andy Oppel, «Fundamentos de SQL,» [En línea]. Available: http://pedrobeltrancanessa-biblioteca.weebly.com/uploads/1/2/4/0/12405072/fundamentos_de_sql_3edi_oppel.pdf.

[20] R. E. Herrador, «Guía de Usuario de Arduino,» [En línea]. Available: http://www.uco.es/aulasoftwarelibre/wp-content/uploads/2010/05/Arduino_user_manual_es.pdf.

[21] Arduino, «Arduino UNO Datasheet,» Arduino , [En línea]. Available: https://www.farnell.com/datasheets/1682209.pdf.

[22] A. Q. A. F. J. S. P. Oscar Déniz Suárez, «Comunicación mediante sockets,» Universidad de las Palmas de Gran Canaria, [En línea]. Available: http://sopa.dis.ulpgc.es/progsis/material-didactico-teorico/tema7_1transporpagina.pdf.

[23] F. G. Carbelleira, «Sistemas Distribuidos, comunicación con sockets,» [En línea]. Available: https://www.arcos.inf.uc3m.es/fgarcia/wp-content/uploads/sites/4/2018/02/s.pdf.

[24] M. R. Marco Besteiro, «Web Services,» [En línea]. Available: http://www.ehu.eus/mrodriguez/archivos/csharppdf/ServiciosWeb/WebServices.pdf.

[25] D. V. J. S. Sosa, «Servicios Web: SOAP y REST,» [En línea]. Available: https://www.tamps.cinvestav.mx/~vjsosa/clases/sd/ServiciosWeb_Axis_REST.pdf.

[26] Microsoft, «Introducción a Visual Studio,» Microsoft, [En línea]. Available: https://msdn.microsoft.com/es-es/library/fx6bk1f4(v=vs.100).aspx. [Último acceso: 03 10 2017].

[27] Microsoft, «Novedades de Visual Studio 2017,» Microsoft Developer Network, [En línea]. Available: https://www.visualstudio.com/es/vs/whatsnew/. [Último acceso: 05 10 2017].

[28] Microsoft, «Microsoft. Microsoft Azure,» Microsoft, [En línea]. Available: https://azure.microsoft.com/es-es/.

[29] Microsoft, «Microsoft. Microsoft Azure. Notification Hubs,» Microsoft, [En línea]. Available: https://azure.microsoft.com/es-es/services/notification-hubs/.

[30] D. T. V. Ruiz, «Centro de gestión de notificaciones Push para dispositivos móviles basados en IOS y Android,» [En línea]. Available: https://addi.ehu.es/handle/10810/15912.

[31] T. V. Ruiz, «Centro de gestión de notificaciones Push para dispositivos móviles basados en IOS y Android,» 2015. [En línea]. Available: https://addi.ehu.es/handle/10810/15912.

[32] XAMARIN, «Deliver native Android, iOS, and Windows apps, using existing skills, teams, and code.,» XAMARIN, [En línea]. Available: https://www.xamarin.com/platform.

[33] Microsoft, «Microsoft Visual Studio Microsoft 2017,» Microsoft, 05 10 2017. [En línea]. Available: https://docs.microsoft.com/es-es/visualstudio/cross-platform/visual-studio-and-xamarin.

[34] J. A. G. Seco, «El lenguaje de programación C#,» [En línea]. Available: http://users.dsic.upv.es/~jlinares/csharp/lenguajeCsharp.pdf.

[35] Microsoft, «Microsoft. Introducción al lenguaje C# y .NET Framework,» Microsoft, [En línea]. Available: https://msdn.microsoft.com/es-es/library/z1zx9t92(v=vs.100).aspx.

[36] A. R. Hernandez, «Análisis y descripción de identificación por radio frecuencia: Tecnología, Aplicaciones Seguridad y Privacidad.,» [En línea]. Available: https://tesis.ipn.mx/bitstream/handle/123456789/5441/C2.302.pdf?sequence=1&isAllowed=y.

[37] K. Flow, «Kanban Flow,» [En línea]. Available: https://kanbanflow.com/.

[38] T. V. Ruiz, «Centro de gestión de notificaciones Push para dispositivos móviles basados en IOS y Android. 2015,» 2015. [En línea]. Available: https://addi.ehu.es/handle/10810/15912. [Último acceso: 12 12 2017].

[39] Microsoft., «Soporte técnico de Microsoft. Configurar su primer sitio Web de IIS.,» [En línea]. Available: https://support.microsoft.com/es-ec/help/323972/how-to-set-up-your-first-iis-web-site.

[40] Microsoft, «Microsoft. Visual Studio Documentation,» [En línea]. Available: https://docs.microsoft.com/en-us/visualstudio/.

[41] Microsoft, «Microsoft Visual Studio,» [En línea]. Available: https://www.visualstudio.com/es/downloads/?rr=https%3A%2F%2Fwww.google.com.ec%2F.

[42] Microsoft, «Microsoft. Libros en pantalla de SQL Server 2014,» [En línea]. Available: https://docs.microsoft.com/es-es/sql/2014-toc/books-online-for-sql-server-2014?view=sql-server-2014.

[43] Microsoft, «Microsoft® SQL Server® 2014 Express. Descargas,» [En línea]. Available: https://www.microsoft.com/es-mx/download/details.aspx?id=42299.

[44] Arduino, «Arduino. Software,» [En línea]. Available: https://www.arduino.cc/en/Main/Software.

[45] Microsoft, «Microsoft. ADO.NET Entity Data Model,» [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/framework/data/wcf/create-a-data-service-using-an-adonet-ef-data-wcf.

[46] Microsoft, «Microsoft. ASP.NET crear un nuevo sitio.,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/dd410269(v=vs.100).aspx.

[47] Microsoft, «Controladores y métodos de acción en aplicaciones ASP.NET MVC,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/dd410269(v=vs.100).aspx.

[48] Microsoft, «Microsfot Visual Studio. Implementar un proyecto de sitio web mediante la herramienta Publicar sitio web,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/1y1404zt(v=vs.100).aspx.

[49] M. Azure, «Microsoft Azure. Envío de notificaciones push a aplicaciones de Xamarin.Android mediante Azure Notification Hubs,» [En línea]. Available: https://docs.microsoft.com/es-es/azure/notification-hubs/xamarin-notification-hubs-push-notifications-android-gcm.

[50] Apple, «Apple developer,» [En línea]. Available: https://developer.apple.com/.

6 ANEXOS

ANEXO I. Codificación de la Base de Datos.

ANEXO II. Codificación de los Servicios Web.

ANEXO III. Codificación de Notificaciones *Push*

ANEXO IV. Codificación del Módulo Arduino.

ANEXO V. Pruebas realizadas a los usuarios del prototipo.

ANEXO I

CODIFICACIÓN DE LA BASE DE DATOS.

```
CREATE DATABASE [PruebaRest] ON  PRIMARY
( NAME = N'PruebaRest',
 FILENAME = N'C:\Program Files\Microsoft SQL Server\parking.mdf' ,
 SIZE = 4096KB ,
 MAXSIZE = 10240KB ,
 FILEGROWTH = 10%),
( NAME = N'PruebaRest1',
 FILENAME = N'C:\Program Files\Microsoft SQL Server\parking.ndf' ,
 SIZE = 4096KB ,
 MAXSIZE = 10240KB ,
 FILEGROWTH = 10% )
 LOG ON
( NAME = N'PruebaRest_log',
FILENAME = N'C:\Program Files\Microsoft SQL Server\parking_log.ldf' ,
SIZE = 1024KB ,
MAXSIZE = 3072KB ,
FILEGROWTH = 10%)
Go
-- -----------------------------------------------------
-- Table `parqueadero`
-- -----------------------------------------------------
Create Table [Parqueadero]
(
       [IdParqueadero]    int NOT NULL,
       [Nombre]           nvarchar (45) NOT NULL,
       [Direccion]        nvarchar(45) NOT NULL,
       [capacidadTotal]   int NOT NULL,
       [capacidadActual]  int NOT NULL,

       Constraint [pk_PARQUEADERO] Primary Key ([Nombre])

) go
-- -----------------------------------------------------
-- Table `tarjeta_rfid`
-- -----------------------------------------------------
```

```
Create Table [TarjetaRFID]
(
	[TAGID]			varchar(8) NOT NULL,
	[FechaInicio]		Datetime NOT NULL,
	[FechaFin]		Datetime NOT NULL,
	[EsCarnet]		Bit NOT NULL,

	Constraint [pk_TarjetaRFID] Primary Key ([TAGID])
) go
-- ----------------------------------------------------
-- Table `usuario`
-- ----------------------------------------------------
Create Table [Usuario]
(
	[IdUsuario] int NOT NULL,
	[Nombre]	nvarchar(45) NOT NULL,
	[Apellido]	nvarchar(45) NOT NULL,
	[Cedula]	nvarchar(10) NOT NULL,
	[Direccion]	nvarchar(45) NOT NULL,
	[Telefono]	nvarchar(45) NOT NULL,
	[Comentario] nvarchar(45) NOT NULL,
	[Correo]	nvarchar(45) NOT NULL,
	[Horario]	Datetime NOT NULL,

	Constraint [pk_USUARIO] Primary Key ([Cedula])
) go
-- ----------------------------------------------------
-- Table `vehiculo`
-- ----------------------------------------------------
Create Table [Vehiculo]
(
	[Placa]			nvarchar(20) NOT NULL,
	[Modelo]		nvarchar(20) NOT NULL,
	[Marca]			nvarchar(20) NOT NULL,
	[Tipo]			nvarchar(20) NOT NULL,

	Constraint [pk_VEHICULO] Primary Key ([Placa])
) go
```

```
-- -----------------------------------------------------
-- Table `Usuario_tiene_Vehiculo`
-- -----------------------------------------------------
Create Table [Usuario_tiene_Vehiculo]
(
	[Placa]	nvarchar(20) NOT NULL,
	[Cedula]	nvarchar(10) NOT NULL,

	Constraint [pk_PARQUEADERO] Primary Key ([Placa],[Cedula])

) go
-- -----------------------------------------------------
-- Table `Usuario_tiene_TarjetaRFID`
-- -----------------------------------------------------
Create Table [Usuario_tiene_TarjetaRFID]
(
	[TAGID]	 varchar(8) NOT NULL,
	[Cedula]	nvarchar(10) NOT NULL,

	Constraint [pk_PARQUEADERO] Primary Key ([TAGID],[Cedula])

) go
-- -----------------------------------------------------
-- Table `NotificacionDisponibilidad`
-- -----------------------------------------------------
Create Table [Notificacion]
(
	[capacidadActual]	int NOT NULL,
	[Cedula] nvarchar(10) NOT NULL,

	Constraint [pk_PARQUEADERO] Primary Key ([capacidadActual],[Cedula])
) go
```

ANEXO II.

CODIFICACIÓN DE LOS SERVICIOS WEB

```
using System;
using System.Linq;
using System.Net.Http;
using System.Web.Http;
using System.Data.Entity;
using System.Runtime.Serialization;
using System.Data;
using System.Data.Common;
using System.Data.Objects;
using System.Data.Objects.DataClasses;

namespace REST1.Controllers
{
    public class AutenticationController : ApiController
    {
        private PruebaRestEntities datacon;
        public IEnumerable<USUARIOS> Get()
        {
            using (datacon = new PruebaRestEntities())
            {
                return datacon.USUARIOS.ToList();
            }
        }
        public String Get(String id)
        {
            using (datacon = new PruebaRestEntities())
            {
                    return datacon.USUARIOS.FirstOrDefault(e => e.Correo ==
                   id).Cedula;
            }
        }
    }
}
```

```
namespace REST1.Controllers
{
    public class RegisterController : ApiController
    {
        public String Get(String id)
        {
            if (id.Contains("user=") && id.Contains(";pass="))
            {
                using (PruebaRestEntities entities = new PruebaRestEntities())
                {
                    long cantidad_actual = entities.USUARIOS.LongCount();
                    int auxint = id.Length;
                    int auxindex = id.IndexOf("=");

                    string user = id.Substring(auxindex + 1, id.IndexOf(";") -
                    auxindex - 1);

                    auxindex = id.LastIndexOf("=");
                    string pass = id.Substring(auxindex, auxint - auxindex);
                    pass = pass.Substring(1, pass.Length - 1);
                    USUARIOS AuxUser = new USUARIOS();
                    AuxUser.Correo = user;
                    AuxUser.Cedula = pass;

                    AuxUser.IdUsuario = (int)cantidad_actual + 1;
                    USUARIOS auxfind = entities.USUARIOS.FirstOrDefault(u =>
                    u.Correo == user);

                    if (auxfind == null)
                    {
                        USUARIOS aux = entities.USUARIOS.Add(AuxUser);
                        entities.SaveChanges();

                        return "OK";
                    }
                    else
                    {
                        return "error";
                    }
```

```
                }
            }
            else
            {
                return "error";
            }
        }
    }
}

namespace REST1.Controllers
{
    public class ServiciosController : ApiController
    {
        public String Get()
        {
            using (PruebaRestEntities entities = new PruebaRestEntities())
            {
                    Parqueadero aux = entities.Parqueadero.First(e =>
                    e.IdParqueadero == 1);
                    return aux.capacidad_actual.ToString();
            }
        }

        public string Get(String id)
        {
            if (id.Contains("user=") && id.Contains(";comment="))
            {
                using (PruebaRestEntities entities = new PruebaRestEntities())
                {
                    long cantidad_actual = entities.USUARIOS.LongCount();
                    int auxint = id.Length;
                    int auxindex = id.IndexOf("=");

                    string user = id.Substring(auxindex + 1, id.IndexOf(";") -
                    auxindex - 1);

                    auxindex = id.LastIndexOf("=");
                    string comment = id.Substring(auxindex, auxint - auxindex);
```

```
                comment = comment.Substring(1, comment.Length - 1);

                USUARIOS aux = entities.USUARIOS.First(e => e.Cedula ==
                user);
                aux.Comentario = comment;
                entities.SaveChanges();
            }
        }

        //GUARDAR LA HORA
        else if (id.Contains("user=") && id.Contains("hour="))
        {
            using (PruebaRestEntities entities = new PruebaRestEntities())
            {
                long cantidad_actual = entities.USUARIOS.LongCount();
                int auxint = id.Length;
                int auxindex = id.IndexOf("=");

                string user = id.Substring(auxindex + 1, id.IndexOf(";") -
                auxindex - 1);

                auxindex = id.LastIndexOf("=");
                string hour = id.Substring(auxindex, auxint - auxindex);
                hour = hour.Substring(1, hour.Length - 1);

                USUARIOS aux = entities.USUARIOS.First(e => e.Cedula ==
                user);
                aux.Hora = Convert.ToInt32(hour);
                entities.SaveChanges();
            }
        }
        return "OK";
    }
}
}
```

ANEXO III.

CODIFICACIÓN DE NOTIFICACIONES *PUSH*

El código del prototipo de notificaciones *push* está disponible en el CD adjunto.

ANEXO IV.

CODIFICACIÓN MÓDULO ARDUINO

```
//Modulo Arduino
#include <Dhcp.h>
#include <Dns.h>
#include <Ethernet.h>
#include <EthernetClient.h>
#include <EthernetServer.h>
#include <SPI.h>

unsigned long tiempo; //Variable para el control
unsigned long tiempo_max= 1000; //Variable para el control
int ledpin= 6;
int buttonPin = 7;
boolean estado= false;

byte mac [ ] = { 0x00,0xAA,0xBB,0xCC,0xDE,0x03 } ;
byte ip [ ] = { 192, 168, 0, 81 } ;
byte server[ ] = { 192, 168, 0, 80 } ;  //ip del servidor
EthernetClient client;

void setup()
{
  Ethernet.begin(mac,ip);
  Serial.begin(9600);
  pinMode(ledpin, OUTPUT);
  pinMode(buttonPin, INPUT);
  digitalWrite(buttonPin,HIGH); //Activa PullUp resistor
  digitalWrite(ledpin,LOW);
  delay (1000);

  Serial.println("Conectando...");
  Serial.print("Mi direccion IP es: ");
  for (byte B = 0; B < 4; B++)
   {
       Serial.print(Ethernet.localIP()[B], DEC);
       Serial.print(".");
    }
```

```
  Serial.println();
  if (client.connect( server , 8888 ))
  {
    Serial.println("Conectado");
  }
  else
  {
    Serial.println("Conexion Fallida");
  }
}


void loop()
{
   if (client.connected())
   {
      while(digitalRead(buttonPin)==LOW)
      {
      if(millis()-tiempo >= tiempo_max){
      estado = true;
      }
  }
    if (estado == true)
    {
      ejecuta();
    }
    digitalWrite(buttonPin,HIGH); //Activa PullUp resistor
    digitalWrite(ledpin,LOW);

    tiempo= millis();
   }
   else
   {
      client.connect( server , 8888 );
    }
}
```

```
void ejecuta()
{

  digitalWrite(ledpin,HIGH);
  delay(1000);
  digitalWrite(ledpin,LOW);
  delay(1000);
  Serial.println("Ingreso un vehículo");
  client.println("IN");
  estado= false;
}
```

ANEXO V.

PRUEBAS REALIZADAS A LOS USUARIOS DEL PROTOTIPO.

Las pruebas realizadas están disponibles en el CD adjunto.