

# **ESCUELA POLITÉCNICA NACIONAL**

## **FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA**

**DESARROLLO DE UN PROTOTIPO DE SISTEMA DE CONTROL  
EN LA NUBE PARA GESTIONAR EL PRÉSTAMO DE BICICLETAS  
EMPLEANDO ANDROID.**

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

**MARÍA BELÉN CUESTA PLÚA**

**maria.cuesta@epn.edu.ec**

**DIRECTOR: MSc. GABRIEL ROBERTO LÓPEZ FONSECA**

**gabriel.lopez@epn.edu.ec**

**CODIRECTOR: MSc. FRANKLIN LEONEL SÁNCHEZ CATOTA**

**franklin.sanchez@epn.edu.ec**

**Quito, enero 2020**

## **AVAL**

Certifico que el presente trabajo fue desarrollado por María Belén Cuesta Plúa, bajo nuestra supervisión.

---

**MSc. GABRIEL ROBERTO LÓPEZ FONSECA**  
**DIRECTOR DEL TRABAJO DE TITULACIÓN**

---

**MSc. FRANKLIN LEONEL SÁNCHEZ CATOTA**  
**CODIRECTOR DEL TRABAJO DE TITULACIÓN**

## **DECLARACIÓN DE AUTORÍA**

Yo María Belén Cuesta Plúa, 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 dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

---

MARÍA BELÉN CUESTA PLÚA

## **DEDICATORIA**

Le dedico a Dios, a mi madre querida, Cristina, a mi gemela Cris, a Chary, a Virgy y a Shirley.

# AGRADECIMIENTO

La constancia alcanza lo que la dicha no alcanza.

Agradezco a Dios por esta vida, a mi madre Cristina por ser mi inspiración, a mi gemela Cris y a mis tías por su apoyo incondicional que me han permitido llegar a este punto de la vida.

Esta tesis fue realizada gracias a la imprescindible guía, comprensión, apoyo, impulso y paciencia de mis queridos tutores Gabriel López y Franklin Sánchez.

Agradezco a mi mejor amiga Sandra Aguilar por su amistad y apoyo a lo largo de la carrera, fue y es de las amistades más lindas que atesoro. También a mis amigas del colegio por estar siempre ahí y animarme: Kat, Dome y Maju las quiero mucho.

A todas las personas que me apoyaron e hicieron posible este proyecto y dejaron su huella de una u otra forma: Josue Burbano por tu tiempo y sabiduría, a Adrián Balcázar por tu paciencia y cariño para procurar el orden en todo, a Angel del Castillo por su apertura y ayuda siempre, es un verdadero ángelito y a mis amigos de la poli que me enseñaron tanto y estuvieron en el momento preciso para encender una luz en la oscuridad: Edison Gómez, Chris Arroyo, Andrés Alberca, Alex Montenegro, David Torres, Dario Lema, Santiago Pilicita y Dario Xavier Vargas.

A mis amigos Redes: Jona Pinzón, Vane Chenaz, Rob Pozo, Quila, Liss Angulo, Galo Rubio, Javi Armas, Christian Padilla y Neto Gangotena fue y es un gusto encontrarnos y compartir bellos momentos dentro y fuera de la poli.

A mis queridos amigos Alejandra Arciniegas y Celso Monteros por su comprensión, ánimo y apoyo. A los boys gracias por su gran apoyo, por alegrarme y sus consejos tan acertados: Juan David Canache, Santi Andrade, Javi Imbaquingo, Marqui Taco y Alejandro Morales gracias por ser el único sponsor de este proyecto.

# ÍNDICE DE CONTENIDO

AVAL .....	I
DECLARACIÓN DE AUTORÍA.....	II
DEDICATORIA.....	III
AGRADECIMIENTO.....	IV
ÍNDICE DE CONTENIDO.....	V
ÍNDICE DE FIGURAS .....	VIII
ÍNDICE DE TABLAS .....	XI
ÍNDICE DE CODIGO.....	XII
RESUMEN .....	XIV
ABSTRACT .....	XV
1 INTRODUCCIÓN .....	1
1.1 OBJETIVOS.....	1
1.2 ALCANCE .....	2
1.3 MARCO TEÓRICO .....	3
1.3.1 RAD.....	3
1.3.2 SISTEMA DISTRIBUIDO.....	4
1.3.3 MODELO CLIENTE SERVIDOR.....	4
1.3.4 SERVICIOS WEB.....	5
1.3.5 FORMATOS DE DATOS.....	6
1.3.6 PROTOCOLO HTTP .....	7
1.3.7 WINDOWS COMMUNICATION FOUNDATION .....	7
1.3.8 REST.....	10
1.3.9 ANDROID STUDIO.....	12

1.3.10	FIREBASE.....	15
1.3.11	CONSULTAS NO RELACIONALES .....	16
1.3.12	PYTHON .....	16
1.3.13	RASPBERRY PI MODELO 3B+ .....	17
1.3.14	LECTOR RFID.....	18
1.3.15	RELÉ.....	20
1.3.16	CERRADURA ELECTROMAGNÉTICA 12V .....	21
2	METODOLOGÍA.....	22
2.1.	DISEÑO DE SISTEMA Y PROTOTIPO .....	22
2.1.1	ANÁLISIS DE REQUERIMIENTOS .....	23
2.1.2	Historias de usuario .....	25
2.1.3	Diagrama de casos de uso .....	30
2.1.4	Diagramas de secuencia .....	34
2.1.5	Diagrama de clases servicio web WCF REST .....	37
2.1.6	Estructura del árbol de objetos JSON.....	43
2.1.7	Sketchs de las pantallas para programar la interfaz de usuario .....	44
2.1.8	Circuito de control de la cerradura electromagnética .....	52
2.1.9	Diagramas de flujo.....	52
2.2	IMPLEMENTACIÓN .....	54
2.2.1	Módulo principal e interfaz de usuario.....	55
2.2.2	Módulo de control .....	81
3	RESULTADOS Y DISCUSIÓN.....	85
3.1	Prueba de Interfaz de Usuario, módulo principal y módulo de control .....	85
4	CONCLUSIONES Y RECOMENDACIONES .....	98
4.1	CONCLUSIONES .....	98
4.2	RECOMENDACIONES .....	98
5	REFERENCIAS BIBLIOGRÁFICAS .....	100
	ANEXOS.....	103

ORDEN DE EMPASTADO.....104



# ÍNDICE DE FIGURAS

Figura 1.1. Esquema de sistema de préstamo de bicicletas.....	2
Figura 1.2. Fases de desarrollo de la metodología RAD[7] .....	4
Figura 1.3. Ejemplo dirección base .....	5
Figura 1.4. Arquitectura WCF[18].....	8
Figura 1.5. Diagrama arquitectura Retrofit .....	12
Figura 1.6. Ejemplo de nodo en Firebase .....	15
Figura 1.7. Componentes de Raspberry Pi 3 B+ .....	17
Figura 1.8. Puertos GPIO [34].....	18
Figura 1.9. Pines de Interfaz SPI[35] .....	18
Figura 1.10. Modo MFRC522.....	19
Figura 1.11. Conexión SPI a Host.....	20
Figura 1.12. Símbolo relé.....	20
Figura 1.13. Diagrama de conexión interno de relé [38].....	21
Figura 1.14. Cerradura electromagnética [40].....	22
Figura 2.1. Caso de uso registro de usuario .....	30
Figura 2.2. Caso de uso login de usuario.....	31
Figura 2.3. Caso de uso de estaciones del sistema .....	31
Figura 2.4. Caso de uso de reserva de bicicleta .....	32
Figura 2.5. Caso de uso de retiro de bicicleta .....	32
Figura 2.6. Caso de uso de entrega de bicicleta .....	33
Figura 2.7. Actores y funciones del sistema.....	34
Figura 2.8. Diagrama de secuencia de reserva de una bicicleta .....	35
Figura 2.9. Diagrama de secuencia retiro de una bicicleta .....	36
Figura 2.10. Diagrama de secuencia entrega de una bicicleta .....	37
Figura 2.11. Diagrama de clases servicio web .....	38
Figura 2.12. Árbol JSON base de datos.....	43
Figura 2.13. Activity de registro.....	44
Figura 2.14. Activity de Login.....	45
Figura 2.15. Activity pantalla principal.....	45
Figura 2.16. Menu de aplicación móvil.....	46
Figura 2.17. Activity reporte de reservas.....	46
Figura 2.18. Activity de bicicletas disponibles de una estación .....	47
Figura 2.19. Activity de reserva.....	47
Figura 2.20. Activity instrucciones Retiro .....	48

Figura 2.21. Activity procesando retiro .....	48
Figura 2.22. Activity que indica bicicleta rodando.....	49
Figura 2.23. Activity de entrega de bicicleta .....	49
Figura 2.24. Activity de procesando Entrega.....	50
Figura 2.25. Activity de login Administrador .....	50
Figura 2.26. Activity principal Administrador .....	51
Figura 2.27. Activities para cambiar duración de reserva .....	51
Figura 2.28. Esquema circuito .....	52
Figura 2.29. Algoritmo de bloqueo y desbloqueo de cerradura electromagnética .....	53
Figura 2.30. Algoritmo que verifica si bicicleta está en estación.....	54
Figura 2.31. Diseño de Menu.....	61
Figura 2.32. Sección APIs y servicios .....	62
Figura 2.33. Selección credenciales .....	62
Figura 2.34. Consola de EC2.....	69
Figura 2.35. Instancias del usuario .....	69
Figura 2.36. Plantilla AMI.....	70
Figura 2.37. Tipo de instancia.....	70
Figura 2.38. Enviar instancia – botón Launch .....	70
Figura 2.39. Clave de acceso a máquina virtual.....	71
Figura 2.40. IP pública de servicio .....	71
Figura 2.41. Conectar a instancia .....	72
Figura 2.42. Credenciales.....	72
Figura 2.43. Activar Web Server IIS.....	73
Figura 2.44. Activar características HTTP.....	73
Figura 2.45. Publicar servicio.....	74
Figura 2.46. Ubicación de servicio .....	74
Figura 2.47. Carpeta wwwroot .....	74
Figura 2.48. Agregar sitio en IIS .....	75
Figura 2.49. Nombre para sitio web .....	75
Figura 2.50. Prueba localhost .....	76
Figura 2.51. Grupo de seguridad de instancia .....	76
Figura 2.52. Lista de acceso -ACLs .....	76
Figura 2.53. IP pública del servicio .....	76
Figura 2.54. Pantalla de Inicio de Firebase .....	77
Figura 2.55. Interfaz para crear proyecto .....	77
Figura 2.56. Ingreso de título de base de datos a crear .....	78

Figura 2.57. Activar análisis de Google.....	78
Figura 2.58. Paso final de creación de base de datos.....	78
Figura 2.59. Proyecto listo .....	79
Figura 2.60. Interfaz de bienvenida.....	79
Figura 2.61. Realtime Database .....	80
Figura 2.62. Habilitar modo de prueba.....	80
Figura 2.63. Base de datos creada .....	80
Figura 2.64. Reglas de la base de datos.....	81
Figura 2.65. Conexión de lectores RFID a Raspberry Pi.....	83
Figura 3.1. Autenticación de cliente .....	86
Figura 3.2. Pantalla principal de sistema de reservas .....	86
Figura 3.3. Bicicletas disponibles de estación 12 de octubre .....	87
Figura 3.4. Pantalla de reserva – Elegir fecha de reserva.....	87
Figura 3.5. Mensaje de confirmación de reserva.....	88
Figura 3.6. Activity de instrucciones de retiro .....	88
Figura 3.7. Estación con bicicleta lista para retiro .....	89
Figura 3.8. Lectura de tag cliente enciende led.....	89
Figura 3.9. Procesando retiro.....	90
Figura 3.10. Candado libre .....	90
Figura 3.11. Retiro con éxito .....	91
Figura 3.12. Candado libre en estación elegida .....	91
Figura 3.13. Usuario en estación con bicicleta .....	92
Figura 3.14. Activity procesando entrega .....	92
Figura 3.15. Lector sensa bicicleta en estación.....	93
Figura 3.16. Servicio web en nube de AWS .....	93
Figura 3.17. Registro de reserva.....	94
Figura 3.18. Registro de retiro .....	94
Figura 3.19. Registro de entrega .....	94
Figura 3.20. Autenticación app administrador .....	95
Figura 3.21. Dashboard .....	95
Figura 3.22. Administración clientes .....	96
Figura 3.23. Administración bicicletas.....	96
Figura 3.24. Administración de estaciones.....	97
Figura 3.25. Menú - Crear elementos del sistema.....	97

## ÍNDICE DE TABLAS

Tabla 1.1. Características de Plan Spark de Firebase [24] .....	16
Tabla 1.2. Especificaciones cerradura electromagnética .....	21
Tabla 2.1. Tabla comparativa de aplicaciones móviles y web de renta de bicicletas .....	23
Tabla 2.2. Elementos de historias de usuario .....	25
Tabla 2.3. Resumen historias de usuario .....	26
Tabla 2.4. Historia de usuario Visualización de la página principal .....	27
Tabla 2.5. Historia de usuario Login en la aplicación móvil .....	27
Tabla 2.6. Historia de usuario visualización de la página principal .....	27
Tabla 2.7. Historia de usuario reservar bicicleta .....	27
Tabla 2.8. Historia de usuario retirar bicicleta .....	28
Tabla 2.9. Historia de usuario entregar bicicleta .....	28
Tabla 2.10. Historia de usuario visualización de reporte de reservas .....	28
Tabla 2.11. Historia de usuario administración de reservas .....	29
Tabla 2.12. Historia de usuario administración de usuarios .....	29
Tabla 2.13. Historia de usuario administración de bicicletas .....	29
Tabla 2.14. Historia de usuario administración de estaciones .....	29
Tabla 2.15. Historia de usuario administración de duración de reserva .....	30
Tabla 2.16. URI y métodos de servicio de bicicletas .....	41
Tabla 2.17. URI y métodos de servicio de clientes .....	41
Tabla 2.18. URI y métodos de servicio de clientes .....	42
Tabla 2.19. URI y métodos de servicio de estaciones .....	42
Tabla 2.20. URI y métodos de servicio de Bicicandados .....	42
Tabla 2.21. URI y métodos de servicio de Candados .....	43

## ÍNDICE DE CODIGO

Código 1.1. Ejemplo de XML .....	6
Código 1.2. Ejemplo de lista de objetos JSON.....	7
Código 2.1. Dependencias de Gradle .....	55
Código 2.2. Clase Bicicleta .....	56
Código 2.3. Método que obtiene bicicletas según estación .....	57
Código 2.4. Método onCreate .....	58
Código 2.5. Interface IServicioBicicletas .....	59
Código 2.6. Fragmento de código del archivo activity_login.xml .....	60
Código 2.7. Código XML del archivo drawer_menu.xml .....	60
Código 2.8. Fragmento para mapa de layout .....	61
Código 2.9. Actividad de mapa .....	61
Código 2.10. Permisos de la aplicación .....	61
Código 2.11. Credenciales para acceder al mapa.....	62
Código 2.12. Clave de API de Google .....	63
Código 2.13. Componente FrameLayout .....	64
Código 2.14. Fragmento de código del archivo MainActivity.java.....	64
Código 2.15. Fragmento de código del Archivo web.config.....	65
Código 2.16. Fragmento del archivo ICientesServicio.cs (ServiceContract) .....	66
Código 2.17. Fragmento del archivo ICientesServicio.cs (DataContract) .....	67
Código 2.18. Fragmento del archivo ClientesServicio.svc.cs .....	68
Código 2.19. Fragmento del archivo BicisCandadosServicio.svc.cs .....	68
Código 2.20. Pin 22 - pin Reset de cliente .....	82
Código 2.21. Ruta de pin 24 - primera interfaz SPI de Raspberry Pi.....	82
Código 2.22. Pin 11 - pin Reset de cliente .....	82
Código 2.23. Ruta de pin 26 – segunda interfaz SPI de Raspberry Pi .....	82
Código 2.24. Importe de librerías json, urllib y urllib2.....	83
Código 2.25. Función que implementa solicitud HTTP GET.....	84
Código 2.26. Función que implementa solicitud POST .....	84
Código 2.27. Deserializar objetos JSON.....	84
Código 2.28. Importe de librería de pines de Raspberry Pi .....	84

Código 2.29. Encendido pin 16 de Raspberry Pi.....	85
--	----

## RESUMEN

Este proyecto de titulación tiene por objeto desarrollar un prototipo de sistema en la nube que gestiona el préstamo de bicicletas a través de una aplicación móvil. Este prototipo utiliza lectura de RFID para identificar la presencia o ausencia de bicicletas y una aplicación móvil para la reserva, retiro y devolución del sistema de bicicletas. Además, incluye el desarrollo de un servicio WCF REST que permite la gestión del sistema y habilita el boqueo o desbloqueo de bicicletas.

El prototipo está formado por un módulo principal, un módulo de control y una interfaz de usuario para el cliente y otra para el administrador.

Primero, el módulo principal comprende un servicio WCF REST que maneja la lógica del negocio y la base de datos Firebase que almacena los atributos y transacciones inmersas en el sistema. Segundo, el módulo de control contiene la circuitería necesaria para comunicar dos lectores RFID, una cerradura electromagnética y una Raspberry Pi que controla el hardware. Tercero, la aplicación móvil permite al cliente realizar las reservas, retiros y devoluciones de bicicletas mientras que al administrador le permite gestionar la duración de los préstamos y manejo de usuarios, bicicletas, estaciones, candados y reservas del sistema.

En conclusión, los resultados de las pruebas de uso de la aplicación móvil confirman la funcionalidad del prototipo como producto final que prototipa una estación y el sistema de bicicletas como autoservicio de reserva, retiro y entrega de bicicletas que emplea el almacenamiento de datos en la nube.

**PALABRAS CLAVE:** aplicación móvil, bicicletas, Firebase, reserva, servicio WCF REST.

## **ABSTRACT**

The purpose of this project is to develop a cloud system prototype for free rental of bikes through a mobile application. This prototype uses RFID reading to identify the presence or absence of bicycles and a mobile application for the reservation, removal and return of bicycles. Also, it includes the development of a WCF REST for the rental management.

The prototype consists of a main module, a control module and a user interface for the client and another for the administrator.

First, the main module is composed of a WCF REST service that handles business logic and a database called Firebase that stores the attributes and transactions immersed in the system. Second, the control module contains the circuitry needed to communicate two RFID readers, an electromagnetic lock and a Raspberry Pi that controls this hardware. Third, the mobile application allows the client to reserve, remove and return bicycles while for the administrator it allows to manage the duration of the rentals and to control users, bicycles, stations, locks and reservations of the system.

In conclusion, the results of the tests confirm the functionality of the project as a final product that prototypes a station and the bicycle system as a self-service for reservation, withdrawal and return of bicycles that use data storage in the cloud.

**KEYWORDS:** bikes, mobile application, Firebase, web WCF REST service, reservation.



# 1 INTRODUCCIÓN

Alrededor del mundo existen sistemas automáticos de préstamo de bicicletas que funcionan con éxito. Por ejemplo, Santander Cycles[1] en Londres utiliza una aplicación móvil que permite el préstamo oportuno al usuario. Spokes Bicycle Rentals[2] en Vancouver emplea una aplicación en la que se puede reservar la bicicleta en línea. Donkey Republic[3] es un autoservicio de préstamo de bicicletas en Copenhague que utiliza una aplicación móvil.

En Quito existen servicios como BiciQuito[4], CiclóPolis[5] y comerciantes informales que ofrecen el préstamo de bicicletas, con procesos en los cuales se requiere la presencia del proveedor del servicio y del cliente simultáneamente.

Este modo de operación de renta de bicicletas no automatizado implica dificultades para el cliente, quien debe acercarse al lugar del alquiler donde en muchos casos encuentra que no hay bicicletas disponibles y está limitado al horario de atención del establecimiento.

Se puede aprovechar el gran potencial de la tecnología para automatizar este servicio. Y darles rapidez a los préstamos de bicicletas, además, permitirle al cliente disponer de información en tiempo real de la disponibilidad de bicicletas, realizar sus reservas desde cualquier lugar, retirar y entregar la bicicleta sin supervisión del encargado.

Debido a que en el Ecuador no existe un sistema de control en la nube para gestionar el préstamo de bicicletas y con el fin de ofrecer una alternativa que permita solventar esta problemática, este proyecto de titulación desarrolla un prototipo de sistema de control para gestionar reservas en línea y entrega – recepción automática de bicicletas a través de una aplicación móvil.

El sistema está formado por software y hardware. El software comprende a: la aplicación móvil como frontend, el servicio WCF REST y la base de datos en la nube denominada Firebase[6] como backend que maneja la lógica de datos del sistema. El hardware está integrado por una Raspberry Pi y la circuitería para controlar el bloqueo y desbloqueo de una cerradura de bicicleta.

## 1.1 OBJETIVOS

El objetivo general de este proyecto de titulación es desarrollar un prototipo de sistema de control en la nube para gestionar el préstamo de bicicletas empleando Android.

Los objetivos específicos de este proyecto de titulación son:

- Analizar los fundamentos teóricos necesarios para el desarrollo del proyecto.
- Diseñar los módulos necesarios para la implementación del sistema.
- Implementar los módulos que son parte del proyecto.
- Analizar los resultados de las pruebas de funcionamiento realizadas.

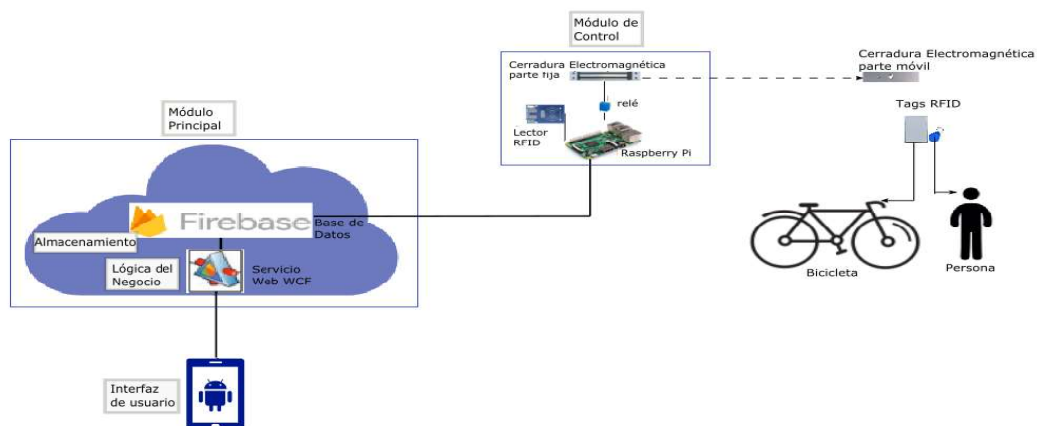
## 1.2 ALCANCE

Este proyecto de titulación implementa el desarrollo de un prototipo de control de un sistema de préstamo de bicicletas automático mediante una aplicación móvil en base a la metodología Rapid Application Development[7].

El sistema de bicicletas maneja una arquitectura cliente-servidor en donde los clientes a través de una interfaz de usuario en Android interactúan con un servicio web que gestiona las solicitudes a la base de datos y las respuestas de la base de datos hacia los clientes.

El sistema está formado por el módulo principal y el módulo de control como se aprecia en la Figura 1.1. El primer módulo contempla: la base de datos Firebase, y un servicio web Windows Communication Foundation (WCF) de tipo Representational State Transfer (REST) que alberga la lógica del negocio. El segundo módulo constituye el hardware que se coloca en la estación y está formado por una Raspberry Pi, sensores RFID, un lector RFID RC522, un relé y una cerradura electromagnética. A través de scripts alojados en la Raspberry Pi, se evalúa si la cerradura electromagnética se abre o no, y los estados de ésta se almacenan en Firebase.

De forma general, el sistema permite: reserva en línea, retiro y entrega de una bicicleta.



**Figura 1.1.** Esquema de sistema de préstamo de bicicletas

Para tener acceso a las secciones de la aplicación móvil se debe registrar o iniciar sesión. Acto seguido, se distinguen dos usuarios: administrador que gestiona los usuarios,

bicicletas, estaciones y préstamos; y el cliente que se le muestra en un mapa las estaciones disponibles, la disponibilidad de bicicletas para realizar una reserva en línea, y el estado de devolución de la bicicleta prestada.

Para el desarrollo de este proyecto se asumirá una empresa que dispone de tres estaciones de alquiler en la ciudad de Quito y para fines de pruebas, se implementará el sistema de entrega de una bicicleta la cual estará ubicada en una de las estaciones antes mencionadas.

### **1.3 MARCO TEÓRICO**

Primero, se describe la metodología Rapid Application Development (RAD), el modelo cliente servidor y el sistema distribuido empleado. Segundo, se revisan los servicios web WCF y REST, el formato de datos y el protocolo HTTP. Luego, se describe la plataforma de desarrollo Android Studio y la base de datos Firebase con las consultas no relacionales que utiliza. Después, se detallan las características de Raspberry Pi y el lenguaje empleado para programarla conocido como Python. Finalmente, se presentan aspectos importantes de un lector RFID, un relé y una cerradura electromagnética utilizados en el proyecto.

#### **1.3.1 RAD**

Rapid Application Development (RAD) es una metodología de diseño de software que presenta flexibilidad a cambios y acepta nuevas características en cada paso del proceso de desarrollo.

Las principales características de la metodología RAD son la alta velocidad y la calidad de desarrollo ya que los usuarios están involucrados en todo el proceso y pueden ver una prueba tangible del producto en la fase de desarrollo y participar en la mejora del mismo antes del término del proyecto. El enfoque de RAD es hacer lo que se necesita en el momento para cumplir las fechas de entrega. Rapid Application Development es adecuada para proyectos en donde el alcance es limitado o donde el trabajo puede dividirse en partes manejables. [8]

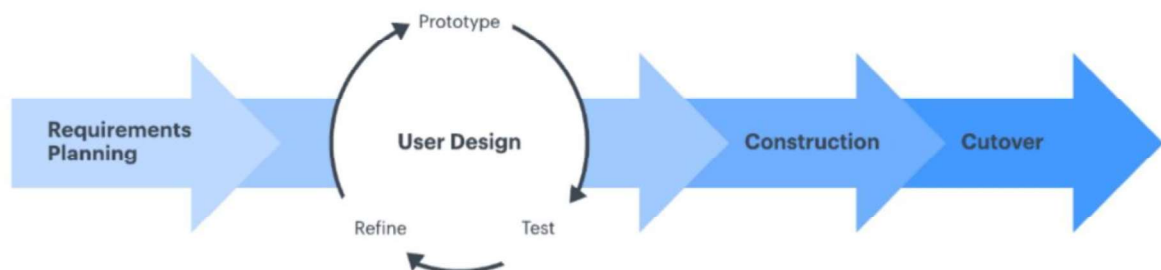
Es el caso de este proyecto de titulación en el que el sistema está compuesto por módulos y el alcance está definido a un prototipo de prueba.

La metodología consta de las siguientes fases[9] como se observa en la Figura 1.2:

- **Planificación de requerimientos:** Se establecen los objetivos, la funcionalidad y el alcance. El equipo de desarrollo se reúne para desarrollar la lista de los requerimientos

y el alcance del proyecto. Al finalizar esta fase, el equipo tiene claro la información que se genera, quien la genera y quién la procesa.

- **Diseño de usuario:** El equipo de desarrollo constituido por los involucrados del proyecto se reúne para planear cómo las partes esenciales del sistema deben trabajar. El resultado final es un documento que muestra los diseños del sistema como reglas del negocio y planes de prueba.
- **Construcción:** El prototipo se convierte en una aplicación funcional. En esta fase, los desarrolladores programan las funcionalidades del prototipo. Esto se realiza en ciclos iterativos de desarrollo, prueba, redefinición de requerimientos y desarrollo hasta que la aplicación esté completa.
- **Fase de corte:** En esta fase el usuario final prueba la aplicación del sistema. Este paso implica una revisión del sistema construido por los involucrados para determinar si cumple con las expectativas. Las características que cumplen las expectativas se mantienen en el prototipo, mientras que las características que no cumplen entran en un lazo iterativo.



**Figura 1.2.** Fases de desarrollo de la metodología RAD[7]

### 1.3.2 SISTEMA DISTRIBUIDO

Un sistema distribuido es una aplicación que ejecuta una colección de protocolos para coordinar las acciones de múltiples procesos en una red, a fin de que todos los componentes cooperen para desempeñar un conjunto de tareas relacionadas.[10]

### 1.3.3 MODELO CLIENTE SERVIDOR

En el modelo cliente servidor se establece un diálogo entre dos aplicaciones. La aplicación que solicita un servicio y la aplicación que devuelve los resultados a la aplicación que la llamó. En este modelo la primera se llama cliente y la segunda servidor.

En aplicaciones cliente servidor, el servidor provee un servicio al cliente. El cliente siempre inicia el diálogo y consume el servicio provisto por el servidor.[11]

### 1.3.4 SERVICIOS WEB

Un servicio web es una unidad de funcionalidad expuesta al mundo que puede ser local o remoto y está asociado a una única dirección o URL (Universal Resource Location) que indica la ubicación del servicio y el protocolo de transporte utilizado para comunicar el servicio. La dirección indica el nombre de la máquina objetivo, sitio o red, un puerto de comunicación y una Universal Resource Identifier (URI) o ruta opcional que es una dirección para identificar recursos y puede ser cualquier string como el nombre del servicio o un GUID<sup>1</sup>.

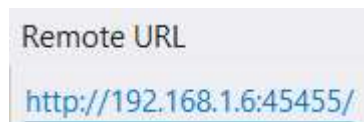
Las direcciones siguen este formato:

[dirección base]/[URI opcional]

Las direcciones base siguen este formato:

[transporte]://[máquina o dominio][:puerto opcional]

Se observa un ejemplo en la Figura 1.3:



**Figura 1.3.** Ejemplo dirección base

http://192.168.1.6:45455/ se lee así: emplear el protocolo HTTP, ir a la máquina 192.168.1.6, en el puerto 45455.

Cuando no se especifica un puerto, las direcciones HTTP usan por defecto el puerto 80.[12]

El grupo de actividad de servicios web del consorcio World Wide Web define a los servicios web como un sistema de software identificado por una URI, cuyas interfaces y bindings<sup>2</sup> son capaces de ser definidos, descritos y descubiertos a través de formatos de datos comunes como XML, JSON y el protocolo HTTP.[13]

En el contexto de servicios web se distingue un frontend y un backend. El frontend es la parte que el usuario visualiza y con la que interactúa, tales como: menús, formularios, entre otros. El backend usualmente consiste en tres partes: un servidor, una aplicación y una base de datos.[14]

---

<sup>1</sup> Globally Unique Identifier (GUID) es un número entero de 128 bits usado para identificar recursos, usado por desarrolladores que trabajan con Microsoft. Es también conocido de forma general por UUID(Universally Unique Identifier) [43]

<sup>2</sup> Una asociación entre una interfaz, un protocolo en concreto y un formato de datos. Especifica el protocolo y formato de datos a ser usados en los mensajes de transmisión definidos por la interfaz asociada.[44]

En el entorno de los servicios web se diferencian dos actores: el consumidor y productor. El primero es el proceso que invoca el servicio y obtiene el resultado. El segundo es el propietario del servicio que maneja la solicitud. [15]

Una ventaja que se le atribuye a los servicios web es independizar la lógica del negocio del dispositivo del usuario final.

### 1.3.5 FORMATOS DE DATOS

Al hablar de sistemas web se utilizan formatos de datos para expresar y almacenar contenido. Se tienen dos tipos:[16]

- **XML:** Extensible Markup Language (XML) es un lenguaje que almacena y transporta información estructurada. No tiene una sintaxis de programación, utiliza únicamente tags o marcas para convertir un archivo de texto en algo que sea manejable por un computador y que sea comprensible para un humano. A continuación, en el Código 1.1. se observa un ejemplo en el que Bel le envía una nota a Cris cuyo encabezado es Recordatorio y el cuerpo del texto indica No olvidar las llaves:

```
<nota>
  <para>Cris</para>
  <de>Bel</de>
  <encabezado>Recordatorio</encabezado>
  <cuerpo>No olvidar las llaves</cuerpo>
</nota>
```

**Código 1.1.** Ejemplo de XML

- **JSON:** JavaScript Object Notation es un formato de intercambio de información ligero utilizado en aplicaciones web. Fue diseñado para que sea manejado por un computador y comprendido por humanos, permite convertir objetos JSON en objetos Javascript y viceversa. Los objetos JSON tienen propiedades que se leen como pares de nombre-valor. A continuación, en el Código 1.2. se observa un ejemplo de una lista de objetos JSON con dos estudiantes que tienen las siguientes propiedades: primer objeto: nombre: Cristina, nacionalidad: ecuatoriana y edad: 25 años; segundo objeto: nombre: Sandra, nacionalidad: ecuatoriana y edad: 25 años.

```
{
  "estudiantes":[
    {
      "nombre":"Cristina",
      "nacionalidad":"Ecuatoriana"
      "edad":"25"
    },
    {
      "nombre":"Sandra",
      "nacionalidad":"Ecuatoriana"
      "edad":"25"
    },
  ]
}
```

**Código 1.2.** Ejemplo de lista de objetos JSON

### 1.3.6 PROTOCOLO HTTP

Hypertext Transport Protocol (HTTP) es un protocolo que determina cómo la información es transferida a través de Internet.

Hypertext es un documento embebido con documentos que tienen hipervínculos. HTTP fue creado para facilitar la comunicación de documentos y recursos multimedia tales como imágenes, videos, GIFs, texto, audio y documentos que incluyan una combinación de los antes mencionados.

Dependiendo de las solicitudes que realice el cliente, el protocolo HTTP tiene los siguientes métodos: GET, POST, PUT, DELETE para recuperar, crear, actualizar y eliminar los recursos. [17]

### 1.3.7 WINDOWS COMMUNICATION FOUNDATION

WCF es una implementación de Microsoft de un conjunto de estándares que definen interacciones de servicios, conversiones de tipo de datos, marshalling<sup>3</sup> y manejo de diferentes protocolos por lo que provee interoperabilidad entre servicios. La mayor parte de la funcionalidad de WCF está incluida en un ensamblado llamado System.ServiceModel.dll en el espacio de nombre (namespace) System.ServiceModel.[12]

Cada servicio está asociado a una dirección que define donde está el servicio, un binding que define cómo comunicarse con el servicio (REST o SOAP<sup>4</sup>) y un contrato que define lo que el servicio realiza.

---

<sup>3</sup> Conversión de datos para presentarlos y transferirlos por la red en el sistema distribuido.[45]

<sup>4</sup> Simplified Object Access Protocol (SOAP) es una especificación que permite la comunicación entre aplicaciones de diferentes sistemas a través de mensajes con formato XML que se transportan mediante el protocolo HTTP.[46]

### 1.3.7.1 ARQUITECTURA

La arquitectura consiste en cinco capas: Aplicación, Contratos, Servicio de Ejecución, Mensajería, Activación y Hosting. Como se aprecia en la Figura 1.4:

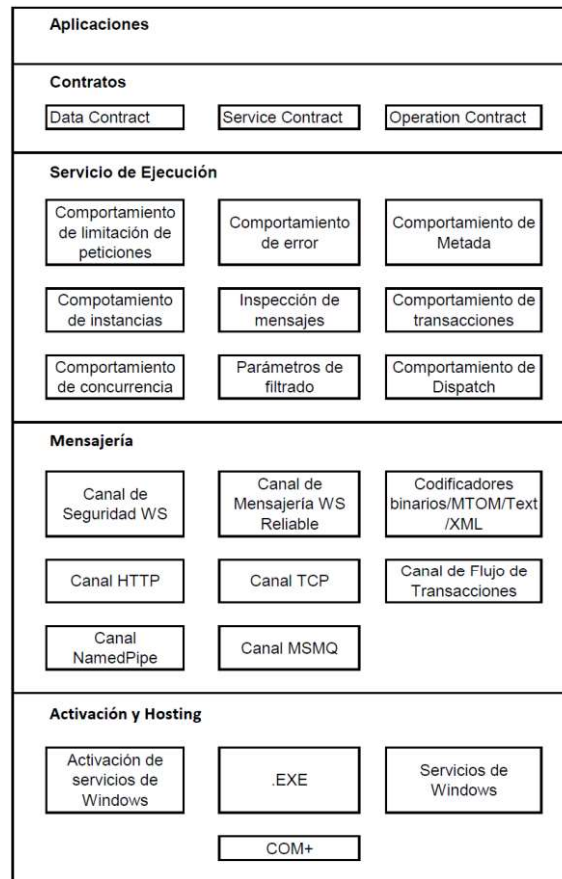


Figura 1.4. Arquitectura WCF[18]

- **Contratos:** Los contratos definen una lista de acuerdos entre un cliente y un servicio, es decir, son formas estándar para describir lo que el servicio realiza y define los métodos que serán expuestos para los clientes. Se diferencian los siguientes tipos de contrato:
  - **Contratos de Data:** Definen qué tipos de datos se pueden consumir, es decir, los que pasan hacia y desde el servicio. Es un atributo que serializa para ser usado en un contrato de operación.
  - **Contratos de servicio:** Describe las operaciones o métodos que están disponibles en el endpoint del servicio y que están expuestos al mundo externo. Un contrato de servicio describe las operaciones o funciones que pueden ser llamadas por el cliente. Para crear un contrato de servicio, se



define una interfaz con los métodos relacionados correspondientes a una colección de operaciones de servicio y se decora la interfaz/clase con el atributo **ServiceContract** para indicar que es un servicio de contrato. Es en donde el atributo ServiceContract hace visible la interfaz o la clase a los clientes. Las interfaces que están decoradas con el atributo ServiceContract son consideradas como contratos WCF.

- **Contratos de operación:** El atributo OperationContract determina los límites de las operaciones del servicio y se lo aplica únicamente en los métodos de la interfaz ya que WCF entiende únicamente operaciones lógicas para desempeñarlas como parte del servicio.
  
- **Servicio de tiempo de ejecución:** El servicio de tiempo de ejecución se compone de un conjunto de comportamientos que definen el funcionamiento del servicio en tiempo de ejecución que gestiona la carga de mensajes en el servicio. Formado por los siguientes:
  - **Comportamientos de peticiones:** Limita las peticiones y controla el número máximo de sesiones.
  - **Comportamientos de error:** Se gestiona información de error que se envía al cliente.
  - **Comportamientos de metada:** Se decide si está disponible o no el servicio para el cliente.
  - **Comportamientos de instancias:** Se gestiona el número de instancias de servicio para manejar las solicitudes del cliente.
  - **Inspección de mensajes:** Manejo de mensajes antes de recibir o enviar la solicitud.
  - **Comportamiento de transacciones:** Se gestiona el flujo de transacciones para la operación del servicio.
  - **Comportamiento de concurrencia:** Se controla el número de hilos activos en el tiempo de operación del servicio.
  - **Parámetros de filtrado:** Se aplican filtros en los parámetros de operación.
  - **Comportamiento de Dispatch:** Se despachan comportamientos que son responsables de la codificación, filtrado, selección e invocación de la operación en el cliente y el lado del servicio.

- **Mensajería:** La capa de mensajería se encarga de enviar el mensaje desde el cliente hasta el servicio. El cliente y el servidor tienen componentes que procesan un mensaje. Estos componentes se denominan canales. Una combinación de uno o más canales es llamado pila o stack de canales. Los canales funcionan en los mensajes y sus encabezados.

Los canales pueden ser de dos tipos:

- **Canales de Transporte:** Funciona en el nivel de red. Lee y escribe los mensajes de la red. Algunos transportes son HTTP, TCP y MSMQ. Estos canales serializan el mensaje saliente y lo pasan a la red. En el lado de entrada, se deserializa el mensaje entrante y lo pasa al protocolo del canal.
  - **Protocolo de canal:** Lee y escribe encabezados adicionales al mensaje. Canal de WS-Security proporciona seguridad mediante la adición de encabezados de mensaje. El canal de Mensajería WS-Reliable admite mensajes por garantía y en orden. Los codificadores ofrecen varios formatos de codificación como binario, texto, entre otros. El canal HTTP indica que el protocolo HTTP es utilizado para la entrega de mensajes. El canal TCP indica que TCP es el protocolo utilizado para la entrega de mensajes. El canal de flujo de transacciones permite patrones de mensajes transaccionales.
- **Activación y Hosting:** Un servicio debe ser alojado o hosteado en un proceso. WCF Runtime provee varias opciones:
    - **Windows Activation Service (WAS):** Es un sistema de servicio que automáticamente activa servicios WCF cuando el cliente solicita el servicio y WAS no puede alojar servicios en múltiples protocolos de transporte.
    - **.EXE:** Es un servicio puede auto alojarse que significa que un desarrollador es el responsable de iniciar y detener el servicio.
    - **Servicios de Windows:** Es un sistema que se ejecuta en segundo plano, y que aloja el servicio WCF. [18]

### 1.3.8 REST

REST se refiere a Representational State Transfer que significa que cada URL es una representación de un objeto.[19]

Según Roy Fielding<sup>5</sup> "Representational State Transfer tiene la intención de evocar una imagen de cómo se comporta una aplicación web bien diseñada: una red de páginas web (una máquina de estado virtual), donde el usuario avanza a través de una aplicación mediante la selección de enlaces (transiciones de estado), que transfieren al usuario a la siguiente página (que representa el siguiente estado de la aplicación) y se renderizan para su uso".

El estilo de arquitectura REST es una arquitectura cliente servidor en la que el cliente envía una solicitud al servidor y luego el servidor procesa la solicitud y devuelve respuestas.

Estas solicitudes y respuestas se construyen a través de la transferencia de representaciones de recursos. Un recurso es algo que se identifica por una URI.

La representación de un recurso es típicamente un documento que captura el estado actual de un recurso. El lenguaje REST está basado en el uso de sustantivos y verbos.

REST no requiere formatos de mensaje con encabezados, lo que implica un bajo ancho de banda.

El principio de diseño de REST es direccionable porque modela los conjuntos de datos para operar como recursos que están marcados con una URI. REST es de interfaz uniforme, ya que los recursos son manejados por métodos HTTP. REST no tiene estado debido a que todas las solicitudes del cliente al servidor son independientes y no tienen relación con las solicitudes anteriores, pues todos los datos necesarios para procesar la solicitud están contenidos en la misma.

Además, los datos de la sesión del cliente no se mantienen en el lado del servidor, por lo tanto, las respuestas del servidor son independientes.

Estos principios hacen que la aplicación REST sea simple y liviana. Una aplicación web que sigue la arquitectura REST se denomina servicio web RESTful.[20]

Un servicio WCF REST puede ser categorizado como un servicio basado en recursos, que implica que un cliente REST envía una solicitud HTTP. Los parámetros que determinan que el servicio es REST son: anidar el endpoint **webHttp** en el elemento **endpointBehaviors** que indica que los clientes se comunicarán con el servicio usando el estándar HTTP y su mecanismo solicitud/respuesta. Y en el elemento protocolMapping cambiar el protocolo por defecto para comunicarse con el servicio web (normalmente es SOAP) a webHttpBinding, que es usado por solicitudes HTTP basadas en REST.

---

<sup>5</sup> Doctor en ciencias de Computación que define Transferencia de Estado Representacional (REST) para desarrollo de servicios web.[19]

### 1.3.9 ANDROID STUDIO

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android (apps). Se basa en un entorno de desarrollo integrado de Java para software llamado IntelliJ IDEA.

El IDE realiza las compilaciones de código a través de Gradle que es un sistema de compilación que se basa en la máquina virtual de Java (Java Virtual Machine JVM), lo que implica que es posible programar un script en java y el IDE lo va a comprender.

Android Studio utiliza la característica Instant Push que permite lanzar cambios de código fuente a la aplicación Android en tiempo de ejecución sin necesidad de reiniciar la app.

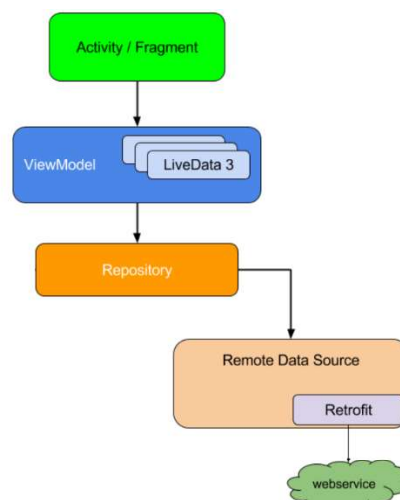
Android Studio tiene una arquitectura recomendada, Retrofit, que hace uso del principio de diseño Separation of Concerns SoC o separación de problemas y del modelo de persistencia que permite:

- Mantener los datos en caso de que el sistema operativo Android del dispositivo móvil finalice la app con el fin de liberar recursos.
- Garantizar la continuidad del funcionamiento de la app, cuando haya intermitencia o se pierda la conexión a Internet. [21].

En el presente proyecto de titulación, se va a utilizar la arquitectura Retrofit para consumir servicios web REST.

#### 1.3.9.1 RETROFIT

Para explicar la arquitectura, se tiene como ejemplo crear una interfaz de usuario (User Interface UI) para desplegar atributos de una clase. Se emplea como backend a una API REST para extraer los datos de la instancia de la clase seleccionada. A continuación, en la Figura 1.5, se observa el diagrama de arquitectura donde todos los módulos se comunican entre ellos:



**Figura 1.5.** Diagrama arquitectura Retrofit

Retrofit convierte la API REST en una interfaz Java. En la arquitectura intervienen:

- **Activity:** Es responsable de dibujar y mostrar la información en la pantalla y recibir la interacción del usuario, pero no de procesarla.
- **ViewModel:** Son objetos que proveen datos para mostrar en componentes de la UI y sobreviven a cambios de configuración. Como rotar la pantalla del celular en donde se pierden datos debido a que se destruye y recrea la actividad. Se utiliza la clase LiveData que es una clase que contiene datos observables, es decir, almacena actualiza y notifica los cambios a los observadores activos de los componentes de la app.
- **Repository:** A través de Retrofit recupera los datos, de este modo la app guarda y carga datos.
- **Web service:** Es la interfaz que hace las solicitudes hacia el backend.

### 1.3.9.2 Componentes de un proyecto en Android Studio

Android Studio y Android SDK proporcionan un conjunto de herramientas permiten crear aplicaciones en un corto periodo de tiempo. El proceso de construir una aplicación en Android con las herramientas del SDK implica los siguientes pasos:

- Configuración del entorno de desarrollo.
- Creación de un proyecto nuevo en Android Studio y escritura del código.
- Ejecución de la aplicación en un emulador o en un dispositivo.
- Depuración y generación de perfiles de la aplicación.[22]

Un proyecto Android posee los siguientes elementos clave en el directorio raíz:

- **AndroidManifest.xml:** Es un archivo XML que describe la aplicación que se está construyendo, qué componentes, actividades, servicios, etc., están siendo suministrados por la aplicación.
- **Build.xml:** Es un script de para compilar la aplicación e instalarla en el dispositivo.
- **bin/:** Contiene la aplicación una vez que se compila.
- **src/:** Contiene el código fuente de Java de la aplicación.
- **res/:** Contiene recursos como iconos, capas de GUI<sup>6</sup>, y similares, que se empaquetan con la compilación de Java en la aplicación.
- **assets/:** Contiene otros archivos estáticos que se incluye con la aplicación para la implementación en el dispositivo. [22]

Los componentes de una app normalmente comprenden una serie de diferentes archivos que se pueden clasificar en 4 categorías:

- Clases de Java
- Archivos Layout

---

<sup>6</sup> GUI Graphic User Interface es Interfaz gráfica de usuario.

- Recursos
- Archivos de configuración

Un proyecto de Android tiene la siguiente estructura:

Los archivos del proyecto casi en su totalidad se localizan en el directorio src/ junto con los recursos y el archivo AndroidManifest.

El archivo build.gradle se ubica fuera del directorio src/, pero es en si el directorio raíz del proyecto. Es un archivo de configuración que controla el proceso de construcción y se editará en el transcurso de la creación de la app.

El directorio src contiene todos los archivos de Java.

Los archivos AndroidManifest.xml y build.gradle son archivos de configuración que deben ser incluidos en toda aplicación móvil. El archivo AndroidManifest.xml contiene la metadata de la aplicación móvil que es requerida para el sistema Android y para que el proyecto opere. Además, es la descripción central de los datos de la aplicación móvil. AndroidManifest.xml contiene componentes esenciales de una aplicación de Android:

- El nombre de paquete Java
- Componentes de la aplicación:
- Nombre del icono
- Actividades con sus atributos
- Servicios
- Permisos (Autorizaciones)
- Nivel mínimo API

El archivo build.gradle contiene toda la información requerida para construir una aplicación móvil, por ejemplo para crear una o más APKs de los archivos naturales o raw. Es relevante notar que las mínimas versiones de Android ya no son leídas desde el archivo AndroidManifest.xml sino desde el archivo build.gradle. Además, en el archivo build.gradle se incluyen todas la librerías requeridas por la aplicación móvil. [23]

### **1.3.9.3 Componentes de Android Studio**

Se tiene los siguientes componentes:

Actividades o activities es el bloque de construcción de la interfaz del usuario. Es un análogo de Android para la ventana o cuadro de diálogo en una aplicación de escritorio.

Intent o intenciones: son mensajes del sistema, corriendo alrededor del interior del dispositivo, notificando a aplicaciones de diversos eventos, desde cambios en el estado del hardware (como una tarjeta SD insertada), hasta la entrada de datos (como un mensaje SMS llegó), hacia eventos de aplicación (como actividades lanzadas desde el menú principal del dispositivo).

Servicios: Actividades, proveedores de contenido, y todos los receptores de intents son de corta duración y pueden cerrarse en cualquier momento. Mientras que los servicios, están diseñados para seguir funcionando si es necesario, independiente de cualquier actividad.[24]

### 1.3.10 FIREBASE

Firebase fue fundada por Andrew Lee y James Tamplin en septiembre de 2011. Es una plataforma completa que tiene las herramientas para desarrollar aplicaciones web y móviles. Está construido para operaciones complejas entre múltiples plataformas de aplicaciones para Android, iOS y la Web con una alta calidad y libres de errores. [25]

Firebase Realtime Database es una base de datos NoSQL que almacena los datos en una estructura JSON en forma de árbol a través de pares nombre-valor, ver figura 1.6, donde cada fragmento de datos JSON se convierte en un nodo dentro de un árbol JSON, al cual se puede acceder mediante una clave.

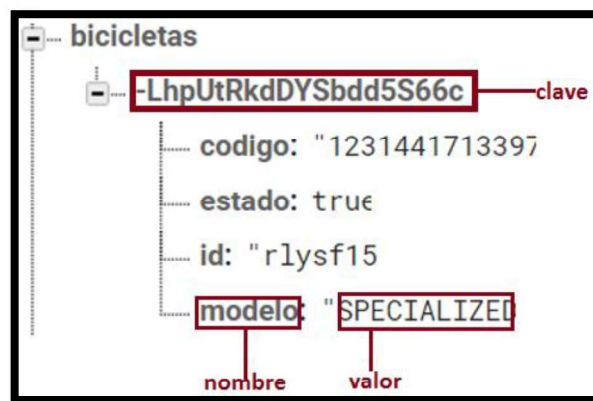


Figura 1.6. Ejemplo de nodo en Firebase

Firebase permite 10 MB como longitud máxima de string en sus nodos, 768 bytes como longitud límite de una clave y permite anidar hasta 32 niveles de profundidad. Es importante procurar un diseño plano de la base de datos, ya que, al realizar consultas de la información de un nodo, se recuperan datos de todos los nodos secundarios. [26]

Firebase ofrece el plan Spark, ver Tabla 1.1, que no tiene costo, el cual permite 100 conexiones simultáneas, almacenamiento de 1 GB y descarga de 10 GB de datos. [27]

**Tabla 1.1.** Características de Plan Spark de Firebase [24]

<b>Incluido sin cargo</b>	
<b>Firestore Realtime Database</b>	
Conexiones simultáneas	100
GB almacenados	1 GB
GB descargados	10 GB/mes

La información de la base de datos de Firebase es escrita y leída directamente por los dispositivos front-end y el servicio web. Firebase es una combinación de muchos servicios de Google en la nube.[28]

### **1.3.11 CONSULTAS NO RELACIONALES**

Una base de datos no-relacional es una base de datos que no utiliza el esquema tabular de las filas y las columnas que se encuentran en la mayoría de los sistemas de bases de datos tradicionales. En su lugar, las bases de datos no relacionales utilizan un modelo de almacenamiento en donde los datos se guardan como simples pares clave/valor.

La mayoría de las bases de datos que almacenan la información en forma de clave/valor suelen admitir consultas simples tales como: insertar y eliminar. Estas implementaciones de lectura y escritura son operaciones atómicas que no pueden ser separadas o reducidas en partes. [29]

En el caso del presente proyecto de titulación, se utiliza una biblioteca: `FirestoreDatabase.Net` y una API REST de Firebase: `FirestoreSharp`.

**FirestoreDatabase.Net:** biblioteca compleja de C# para Firestore Realtime Database construida sobre Firestore API REST. Admite el almacenamiento offline, consultas mediante filtros disponibles y permite agregar y modificar datos.[30]

**FirestoreSharp:** es una API desarrollada para .NET que permite realizar consultas y notificar automáticamente los cambios a los clientes suscritos. [29]

### **1.3.12 PYTHON**

Python es un lenguaje de programación orientado a objetos que utiliza un amplio conjunto de tipos de datos primitivos. Tales como números: enteros, punto flotante, números complejos y números racionales. También tiene poderosos strings, listas de tamaño variable y arreglos asociativos muy flexibles denominados diccionarios.

Python es útil para interoperar con estructuras de múltiples lenguajes mediante una sintaxis accesible.

Las bibliotecas de Python incluidas en el lenguaje y de proyectos externos, permiten interactuar con servidores web, bases de datos para procesar texto y enviar datos.[31]

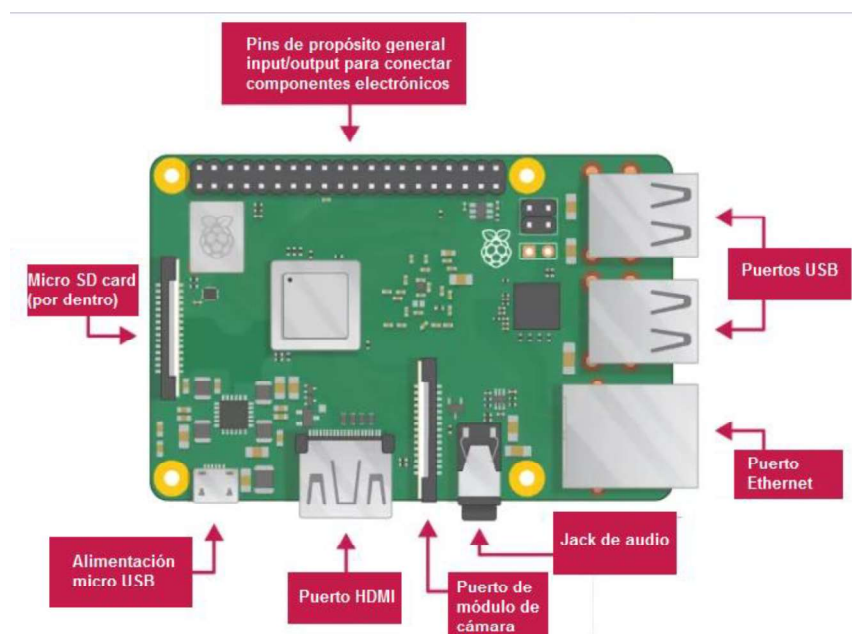


### 1.3.13 RASPBERRY PI MODELO 3B+

La Raspberry Pi 3B + es un equivalente a un completo ordenador personal en miniatura. Las dimensiones externas de la microcomputadora sólo son 86 x 54 x 17 mm.[32] Esta placa en miniatura contiene todos los componentes necesarios para utilizar el microcomputador como una simple computadora personal.[33] El componente más importante del dispositivo es el procesador Broadcom BC2837 ARM-8, 4 núcleos de 64 bits integrado con el Cortex A53 graphics. La memoria de funcionamiento en el modelo 3B + se ha aumentado a 1 GB y una tarjeta MicroSD es usada como el disco duro, que se inserta en el lector del dispositivo.

La Raspberry Pi tiene un sistema operativo Linux llamado Raspbian que es gratuito y de código abierto. Además, cuenta con 40 pines de propósito general (GPIO General Purpose Input Output).

En la Figura 1.7 se muestran las características que tiene la Raspberry Pi:



**Figura 1.7.** Componentes de Raspberry Pi 3 B+

La Raspberry posee pines de voltaje y tierra (Ground) como se observa en figura 1.8:

- pines de 3.3V
- pines de 5V
- 8 pines de Ground

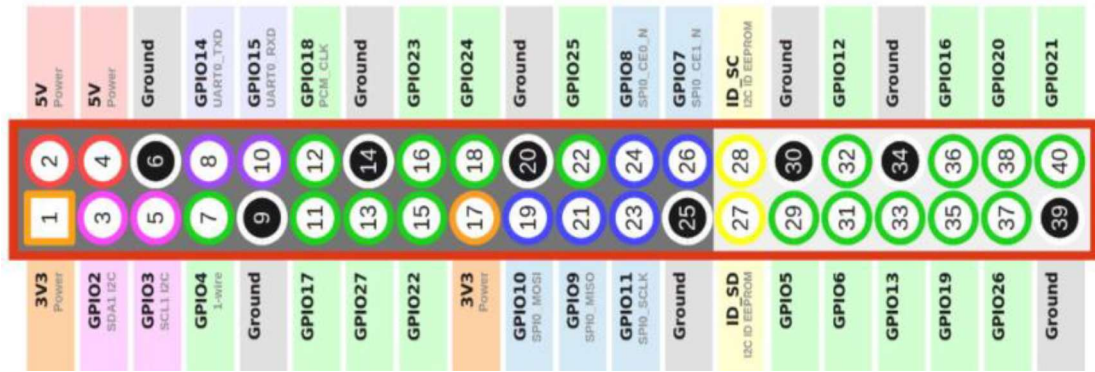


Figura 1.8. Puertos GPIO [34]

- Los puertos GPIO de la Raspberry son la principal forma de conectar con otras placas electrónicas utilizando diferentes protocolos. En el presente proyecto se utiliza Serial Peripheral Interface (SPI) que es una conexión serial síncrona de full dúplex (dos vías) que emplea los siguientes pines como se muestra en figura 1.9:

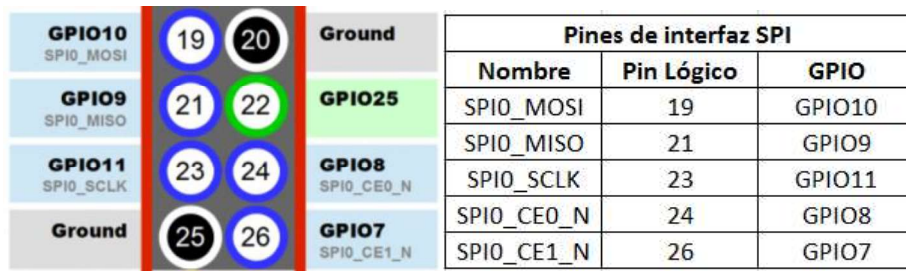


Figura 1.9. Pines de Interfaz SPI[35]

### 1.3.14 LECTOR RFID

El sistema de identificación por radiofrecuencia o RFID, ver figura 1.10, consta de dos componentes principales, un transpondedor o etiqueta adherida a un objeto a identificar y un transceptor también conocido como lector.

Un lector está constituido por un módulo de radiofrecuencia y una antena que genera un campo electromagnético de alta frecuencia. Por otro lado, la etiqueta suele ser un dispositivo pasivo, que tiene un microchip donde almacena y procesa información, y una antena para recibir y transmitir una señal.

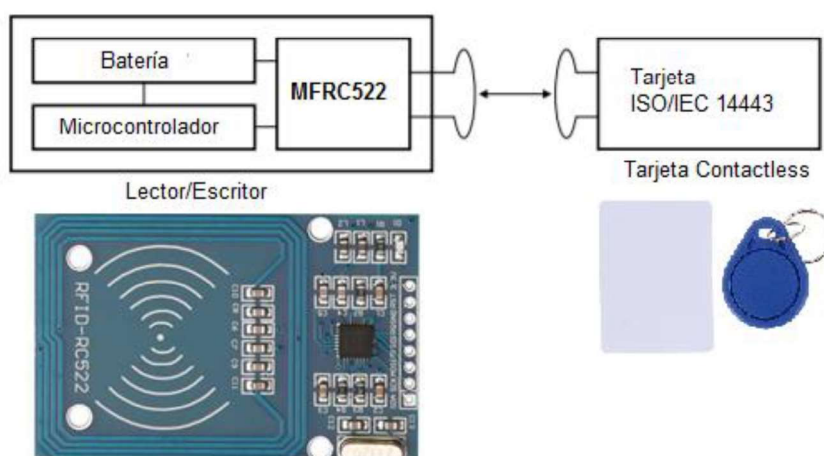
Para leer la información codificada en una etiqueta, se la coloca cerca del lector. Un lector genera un campo electromagnético que hace que los electrones se muevan a través de la antena de la etiqueta y posteriormente alimenten el chip.

El chip alimentado dentro de la etiqueta responde enviando su información almacenada al lector en forma de otra señal de radio, lo que se denomina como retrodifusión. Es decir, la retrodispersión, o cambio en la onda electromagnética / RF, es detectada e interpretada por el lector que luego envía los datos a una computadora o microcontrolador.

El MFRC522 es un lector/escritor altamente integrado para la comunicación RFID a 13,56 MHz. El lector/escritor MFRC522 está soportado por ISO 14443A<sup>7</sup> modo Mifare®.

Tiene las siguientes características:

- Circuito analógico altamente integrado para demodular y decodificar respuestas.
- Distancia de funcionamiento de lectura máxima de 50 mm que depende de las características de la tarjeta.
- Admite comunicación de velocidad de transferencia máxima de 848 kbit / s
- Permite suministro de alimentación de 2.5V a 3.3V
- Pines a nivel lógico toleran voltaje de 5V.



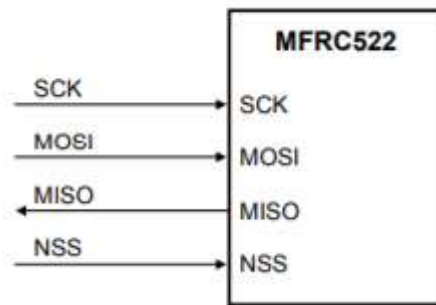
**Figura 1.10.** Modo MFRC522

Se admite una interfaz SPI (Serial peripheral Interface) que permite la comunicación de alta velocidad con un host como una Raspberry Pi o un Arduino. La interfaz puede manejar velocidades de datos de hasta 10 Mbit / s. Cuando se comunica con un host, el MFRC522 actúa como esclavo, recibe datos del host externo para la configuración del registro, envía y recibe datos relevantes para la comunicación de la interfaz de RF.

Una interfaz compatible con SPI permite la comunicación en serie de alta velocidad entre el MFRC522 y un microcontrolador. La interfaz implementada está de acuerdo con el estándar SPI.

---

<sup>7</sup> Estándar de tarjeta de identificación sin contacto (contactless) creada por la ISO



**Figura 1.11.** Conexión SPI a Host

El MFRC522 actúa como esclavo durante la comunicación SPI. En la figura 1.11, se observa un pin que representa la señal de reloj SPI SCK que debe ser generada por el maestro. La comunicación de datos del maestro al esclavo usa la línea MOSI. La línea MISO se utiliza para enviar datos desde el MFRC522 al maestro. Y el pin NSS habilita hacia donde se envían los datos [36]

### 1.3.15 RELÉ

Un relé es un interruptor que es operado eléctricamente. El relé SPDT – Single Pole Double Throw posee 5 pines, ver figura 1.12, y tiene una configuración en la que cambia entre un polo común a otros dos polos. Se puede considerar un relé SPDT con un polo común 'C' y dos polos 'A' y 'B'. Cuando la bobina no está alimentada (inactivo), el polo común 'C' está conectado al polo 'A' (NO normalmente abierto) y está en posición de reposo. Pero cuando el relé está alimentado, el polo común 'C' está conectado al polo 'B' (NC normalmente cerrado) y está activo. [37]



**Figura 1.12.** Símbolo relé

Son utilizados principalmente para manejar diferentes circuitos que tienen la misma señal de control. A continuación, en la figura 1.13 se observa un diagrama interno de conexión:

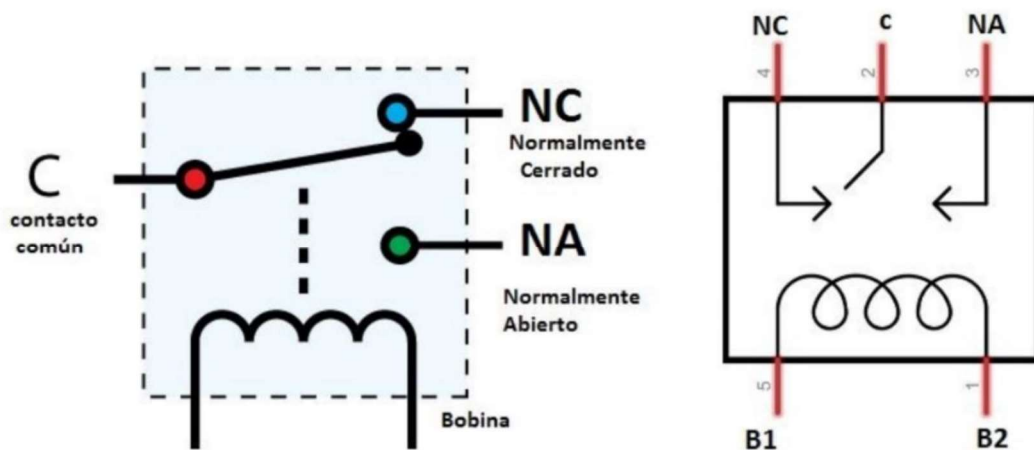


Figura 1.13. Diagrama de conexión interno de relé [38]

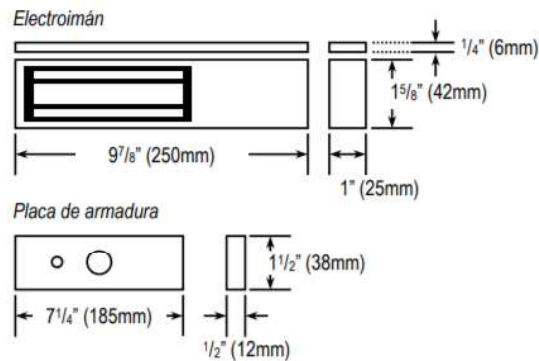
### 1.3.16 CERRADURA ELECTROMAGNÉTICA 12V

Una cerradura electromagnética crea un campo magnético a través de una corriente eléctrica para producir la fuerza magnética. De este modo el electroimán atrae a la placa de armadura con la suficiente fuerza para evitar su separación y mantener el bloqueo. [39] La cerradura electromagnética E-941SA-60PQ tiene una fuerza de retención de 600 libras o 272 kg, presenta una caja de aluminio anodizado, puede ser alimentada con 12V o 24V. En la tabla 1.2 se observan las especificaciones:

Tabla 1.2. Especificaciones cerradura electromagnética

Modelo	E-941SA-600PQ
Fuerza de retención	600-lb (272kg)
Voltaje de operación	12/24 V C.C. $\pm$ 10%
Consumo de corriente 12V C.C.	500mA@V C.C.
Consumo de corriente 24V C.C.	250mA@V C.C.
Temperatura de operación	14°~131° F (-10°~55° C)
Peso	4-lb, 6-oz (2kg)
Estatus LED	Verde: cerradura asegurada Rojo: cerradura no asegurada o bloqueada Apagado: cerradura abierta

En la Figura 1.14 se observa un esquema de la cerradura electromagnética:



**Figura 1.14.** Cerradura electromagnética [40]

Lo que se ha descrito en este capítulo resume la teoría utilizada para el desarrollo del proyecto y constituyen el marco de referencia para continuar con la metodología, diseño e implementación del sistema.

## 2 METODOLOGÍA

En este capítulo se desarrolla el diseño, construcción e implementación del sistema de préstamo de bicicletas y su prototipo. El sistema consta de dos módulos:

- Módulo Principal
- Módulo de Control

La implementación del módulo principal se enfoca en el desarrollo de un servicio web WCF que tiene clases e interfaces y alberga la lógica del negocio. Contempla la estructura de la base de datos Firebase que almacena los datos del sistema de bicicletas a manera de árbol de objetos JSON. En cuanto a la interfaz de usuario se toma en cuenta la definición de los requerimientos: funcionales y no funcionales. Adicionalmente, se realiza la conexión de la interfaz de usuario y el módulo principal con el fin de realizar el consumo del servicio web y se utilizan los sketches para implementar la app móvil.

Para el módulo de control se arma la circuitería necesaria para la construcción de la estación de bicicletas y se codifican los scripts con los algoritmos de bloqueo / desbloqueo de la cerradura, la lectura de los tags RFID y envío / recepción de datos entre la Raspberry Pi y Firebase.

### 2.1. DISEÑO DE SISTEMA Y PROTOTIPO

El diseño se realiza a través de diagramas de secuencia y de casos de uso que utiliza la interfaz de usuario. Además, se usan los sketches de las pantallas para programar la interfaz de usuario y la estructura del árbol de objetos JSON empleados en la base de datos.

Para el módulo de control se dimensiona el circuito de control de la cerradura electromagnética y se realizan diagramas de flujo que definen el comportamiento del

algoritmo empleado, tanto para la lectura de los tags RFID del cliente y la bicicleta como para bloquear y desbloquear la cerradura electromagnética.

### 2.1.1 ANÁLISIS DE REQUERIMIENTOS

Los requerimientos funcionales y no funcionales se obtienen en base a las características recopiladas de la Tabla 2.1 y al alcance planteado del presente proyecto de titulación.

En la Tabla 2.1 se realiza una comparación de las características de las aplicaciones móviles [1],[3] y la aplicación web [2] de renta de bicicletas.

**Tabla 2.1.** Tabla comparativa de aplicaciones móviles y web de renta de bicicletas

<b>Característica</b>	<b>Aplicación</b>	<b>Santander Cycles</b>	<b>Spokes Bicycle rentals</b>	<b>Donkey Republic</b>
Aplicación móvil		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Aplicación web		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<b>Registro de usuario</b>				
Solicita nombre, apellido, email		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Solicita teléfono celular		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Solicita número de de tarjeta		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Solicita foto de tarjeta		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Envío de número de verificación a celular		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<b>Reserva de bicicleta</b>				
Permite realizar una reserva de bicicleta online		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Utiliza mapa interactivo para indicar estaciones del sistema		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Muestra disponibilidad de bicicletas libres		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Para finalizar reserva solicita número de tarjeta		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<b>Retiro de bicicleta</b>				
Para retirar una bicicleta se genera un código		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Aplicación móvil tiene botón para desbloquear candado		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Estación tiene teclado para tipear código de retiro		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
El costo de la renta de bicicleta se calcula a partir de la hora de retiro de la bicicleta		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<b>Entrega de bicicleta</b>				
Muestra disponibilidad de candados libres		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Aplicación móvil tiene botón para bloquear candado		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Para entregar se empuja la bicicleta en un candado libre		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Muestra actividad reciente de entregas y retiros		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

## Requerimientos funcionales

Se detallan los requerimientos funcionales del sistema:

- Hay 2 perfiles de cuentas de usuario: cliente y administrador.
- Para acceder a las secciones de la aplicación móvil se debe iniciar sesión con un usuario y contraseña. Se comparan los datos con los de la base de datos, en caso positivo se accede al sistema y en caso negativo se muestra un error.
- En caso de no tener cuenta de usuario se debe registrar una con los siguientes datos: nombre, apellido, nombre de usuario, contraseña, cédula, email, dirección y teléfono.
- El administrador tiene una cuenta registrada en la base de datos desde el inicio.
- El rol de cliente concierne a un usuario que desea reservar una bicicleta.
- La aplicación móvil muestra un mapa con la ubicación actual del cliente y las estaciones de bicicletas cercanas del sistema con la cantidad de bicicletas disponibles como pantalla principal.
- El sistema permite realizar una reserva que contiene una estación, una bicicleta, fecha, hora inicio y hora fin.
- El sistema expira la reserva cuando la hora fin se cumple.
- El sistema permite cancelar la reserva de una bicicleta.
- El sistema registra un retiro fallido cuando el lector RFID de la bicicleta sensa que la misma sigue colocada en la estación por lo que mantiene la bicicleta ocupada y el candado bloqueado.
- Una vez elegida la estación donde se desea entregar la bicicleta, el sistema asigna un candado disponible en la misma.
- El administrador puede crear, modificar y leer los usuarios del sistema.
- El administrador puede crear, modificar y leer las estaciones del sistema.
- El administrador puede crear, modificar y leer las bicicletas del sistema.
- El administrador puede crear, modificar y leer las reservas del sistema.
- El cliente visualiza el reporte de sus reservas.
- La hora fin de reserva se auto calcula según la duración establecida por el administrador.
- Un cliente no puede hacer 2 reservas al mismo tiempo.
- No se permiten reservas en fechas anteriores a la fecha actual.
- Para realizar el retiro de una bicicleta se debe tener una reserva activa.
- Para entregar la bicicleta, el cliente debe colocar la tarjeta RFID en el lector y ubicar la bicicleta en el candado.



- El sistema registra una entrega fallida cuando el lector RFID no sensa la bicicleta y entonces mantiene la cerradura abierta y disponible.

### Requerimientos no funcionales

Definen las limitaciones de las funciones que ofrece el sistema en cuanto a tiempo, a proceso de desarrollo, es decir, IDE utilizado y lenguaje de programación. Estos requerimientos son más críticos que los requerimientos funcionales ya que si no se los toma en cuenta, el sistema puede no ser útil. Los requerimientos no funcionales se enfocan a todo el sistema, más no a las funciones individuales. [41]

Se listan los siguientes:

- Los dispositivos móviles deben tener Internet y activado el GPS para utilizar la app.
- La aplicación móvil es desarrollada para Android.
- Los datos inmersos en el sistema se almacenan en Firebase, una base de datos no relacional.
- La lógica del negocio se encuentra en un api REST WCF desarrollada en C#.
- La aplicación móvil y la Raspberry Pi SON clientes consumidores del servicio REST WCF.
- Las validaciones de retiro y entrega se realizan mediante la lectura del tag RFID del cliente.
- El sistema cuenta únicamente con 3 estaciones de prueba.

### 2.1.2 Historias de usuario

Las historias de usuario son definiciones de alto nivel de los requerimientos de un sistema, y captan las características de un software observado desde la perspectiva del usuario final.

En la Tabla 2.2 se observan los elementos que manejan las historias de usuario:

**Tabla 2.2.** Elementos de historias de usuario

Elemento	Detalle
Número	Identificador de historia de usuario
Usuario	Persona, subsistema o entidad que interactúa con el sistema a quién se oriente la historia
Nombre	Descripción de la historia
Prioridad	Puede ser alta, media o baja
Descripción	Se listan los objetivos
Validación	Forma en que usuario valida cumplimiento de historia de usuario

En la Tabla 2.3 se resumen las historias de usuario del proyecto:

**Tabla 2.3.** Resumen historias de usuario

<b>Id</b>	<b>Título</b>	<b>Descripción</b>
HU01	Registro	El usuario, para usar la aplicación, debe registrarse.
HU02	Login	Para iniciar sesión, el usuario debe completar su usuario y contraseña con la que se registró
HU03	Visualización de la página principal	El usuario, al hacer el login, visualiza un mapa con las estaciones y el número de bicicletas en cada una
HU04	Reservar bicicleta	Realizar la reserva de una bicicleta en un periodo
HU05	Retirar bicicleta	El usuario, al tener una reserva, podrá retirar la bicicleta
HU06	Entregar bicicleta	El usuario, al tener un retiro, podrá entregar la bicicleta
HU07	Visualización de reporte de reservas	El cliente puede observar la lista de reservas realizadas
HU08	Administración de reservas	El administrador podrá crear, editar y eliminar reservas.
HU09	Administración de usuarios	El administrador podrá crear, editar y eliminar usuarios.
HU10	Administración de bicicletas	El administrador podrá crear, editar y eliminar bicicletas.
HU11	Administración de estaciones	El administrador podrá crear, editar y eliminar estaciones.
HU12	Administración de duración de reserva	El administrador podrá cambiar la duración del préstamo del sistema

A continuación, de la Tabla 2.4 a la Tabla 2.15, se detallan las historias de usuario:

**Tabla 2.4.** Historia de usuario Visualización de la página principal

Historia de Usuario					
<b>Número</b>	01	<b>Usuario</b>	Cliente	<b>Prioridad</b>	Alta
<b>Nombre historia</b>		Registro			
<b>Descripción</b>		El cliente que no tenga una cuenta en el sistema, puede crearlo a través del ingreso de los siguientes datos: nombre, apellido, nombre de usuario, contraseña, cédula, email, dirección y teléfono.			
<b>Validación</b>		Validación de credenciales y acceso a la aplicación			

**Tabla 2.5.** Historia de usuario Login en la aplicación móvil

Historia de Usuario					
<b>Número</b>	02	<b>Usuario</b>	Administrador Cliente	<b>Prioridad</b>	Alta
<b>Nombre historia</b>		Login			
<b>Descripción</b>		El usuario, ya sea administrador o cliente podrá ingresar o hacer login mediante un usuario y contraseña registrado en la base de datos			
<b>Validación</b>		Validación de credenciales y acceso a la aplicación			

**Tabla 2.6.** Historia de usuario visualización de la página principal

Historia de Usuario					
<b>Número</b>	03	<b>Usuario</b>	Cliente	<b>Prioridad</b>	Alta
<b>Nombre historia</b>		Visualización de la página principal			
<b>Descripción</b>		El usuario, al hacer el login, visualiza un mapa con las estaciones.			
<b>Validación</b>		Usuario observa mapa con marcadores que corresponden a las estaciones			

**Tabla 2.7.** Historia de usuario reservar bicicleta

Historia de Usuario					
<b>Número</b>	04	<b>Usuario</b>	Cliente	<b>Prioridad</b>	Alta
<b>Nombre historia</b>		Reservar bicicleta			
<b>Descripción</b>		El cliente elige una estación y en ella escoge una bicicleta y asigna fecha y hora de préstamo.			
<b>Validación</b>		Realizar la reserva de una bicicleta en un periodo			

**Tabla 2.8.** Historia de usuario retirar bicicleta

Historia de Usuario					
<b>Número</b>	05	<b>Usuario</b>	Cliente	<b>Prioridad</b>	Alta
<b>Nombre historia</b>		Retirar bicicleta			
<b>Descripción</b>		El cliente se acerca a la estación dentro del tiempo de reserva y al acercarse su tarjeta el candado se desbloquea.			
<b>Validación</b>		Cliente retira bicicleta de estación y pantalla de aplicación se visualiza una imagen que dice bicicleta Rodando.			

**Tabla 2.9.** Historia de usuario entregar bicicleta

Historia de Usuario					
<b>Número</b>	06	<b>Usuario</b>	Cliente	<b>Prioridad</b>	Alta
<b>Nombre historia</b>		Entregar bicicleta			
<b>Descripción</b>		El cliente se acerca a la estación dentro del tiempo de reserva y al acercarse su tarjeta el candado se bloquea.			
<b>Validación</b>		Cliente entrega bicicleta de estación y aplicación móvil muestra pantalla de inicio que permite realizar otra reserva			

**Tabla 2.10.** Historia de usuario visualización de reporte de reservas

Historia de Usuario					
<b>Número</b>	07	<b>Usuario</b>	Cliente	<b>Prioridad</b>	Alta
<b>Nombre historia</b>		Visualización de reporte de reservas			
<b>Descripción</b>		En la aplicación móvil, el cliente puede observar el resumen de sus reservas			
<b>Validación</b>		Cliente elige Mis reservas y observa la lista de reservaciones realizadas			

**Tabla 2.11.** Historia de usuario administración de reservas

<b>Historia de Usuario</b>					
<b>Número</b>	08	<b>Usuario</b>	Administrador	<b>Prioridad</b>	Alta
<b>Nombre historia</b>	Administración de reservas				
<b>Descripción</b>	El administrador podrá manejar las reservas. Es decir, está facultado a crear, editar y leer reservas.				
<b>Validación</b>	El administrador podrá crear, editar y eliminar reservas.				

**Tabla 2.12.** Historia de usuario administración de usuarios

<b>Historia de Usuario</b>					
<b>Número</b>	09	<b>Usuario</b>	Administrador	<b>Prioridad</b>	Alta
<b>Nombre historia</b>	Administración de usuarios				
<b>Descripción</b>	El administrador podrá manejar las reservas. Es decir, está facultado a crear, editar y leer usuarios.				
<b>Validación</b>	El administrador podrá crear, editar y eliminar usuarios.				

**Tabla 2.13.** Historia de usuario administración de bicicletas

<b>Historia de Usuario</b>					
<b>Número</b>	10	<b>Usuario</b>	Administrador	<b>Prioridad</b>	Alta
<b>Nombre historia</b>	Administración de bicicletas				
<b>Descripción</b>	El administrador podrá manejar las reservas. Es decir, está facultado a crear, editar y leer bicicletas.				
<b>Validación</b>	El administrador podrá crear, editar y eliminar bicicletas.				

**Tabla 2.14.** Historia de usuario administración de estaciones

<b>Historia de Usuario</b>					
<b>Número</b>	11	<b>Usuario</b>	Administrador	<b>Prioridad</b>	Alta
<b>Nombre historia</b>	Administración de estaciones				
<b>Descripción</b>	El administrador podrá manejar las reservas. Es decir, está facultado a crear, editar y leer estaciones.				
<b>Validación</b>	El administrador podrá crear, editar y eliminar estaciones.				

**Tabla 2.15.** Historia de usuario administración de duración de reserva

Historia de Usuario					
<b>Número</b>	12	<b>Usuario</b>	Administrador	<b>Prioridad</b>	Alta
<b>Nombre historia</b>	Administración de duración de reserva				
<b>Descripción</b>	El administrador podrá cambiar la duración del préstamo de una bicicleta. Inicialmente está configurado en 1 hora				
<b>Validación</b>	El administrador podrá cambiar la duración del préstamo del sistema				

### 2.1.3 Diagrama de casos de uso

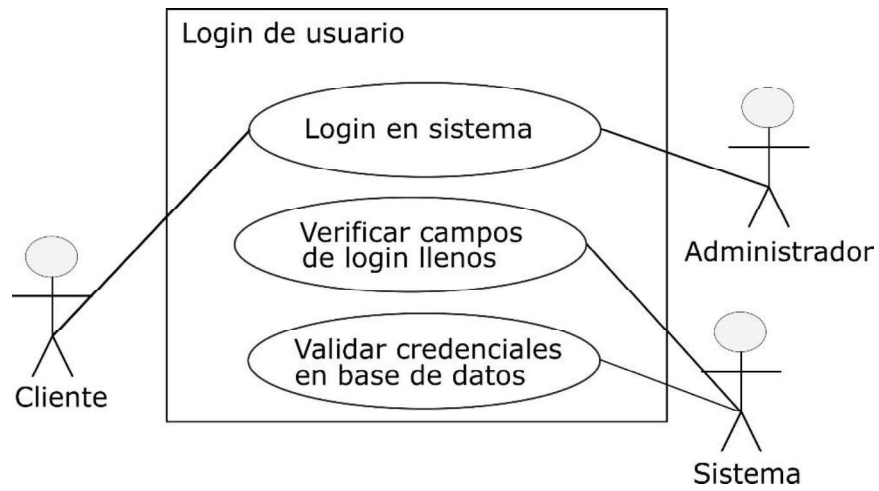
Los diagramas de casos de uso son diagramas de comportamiento que modelan la funcionalidad de un sistema usando actores y las funciones que un sistema desempeña. Se tienen los siguientes actores: el cliente y el administrador. El cliente tiene acceso a registrarse, realizar login, reservar una bicicleta, retirarla y devolverla. Desde la figura 2.1 a la figura 2.6, se muestran los diagramas de casos de uso:

- Registro de usuario



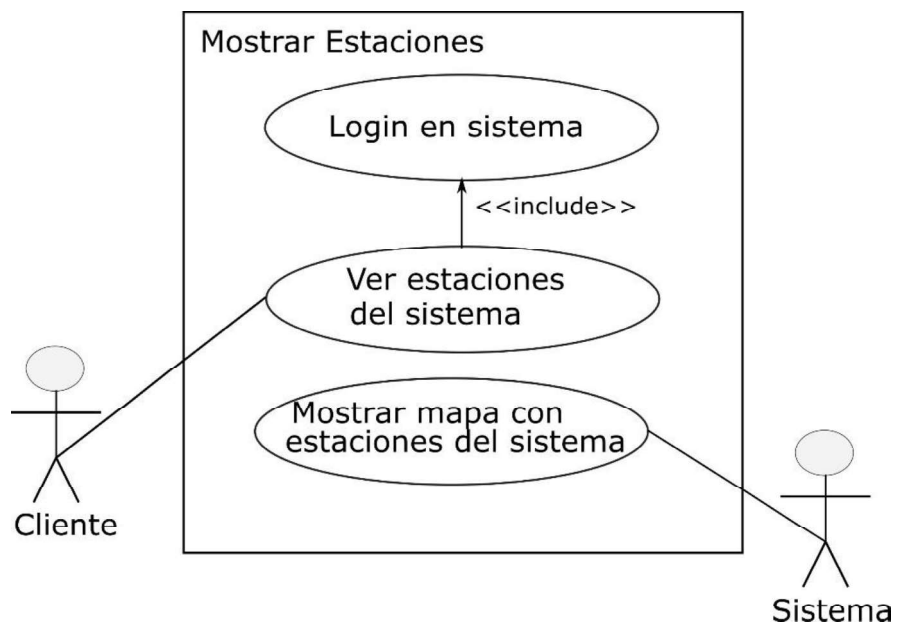
**Figura 2.1.** Caso de uso registro de usuario

- Login de usuario



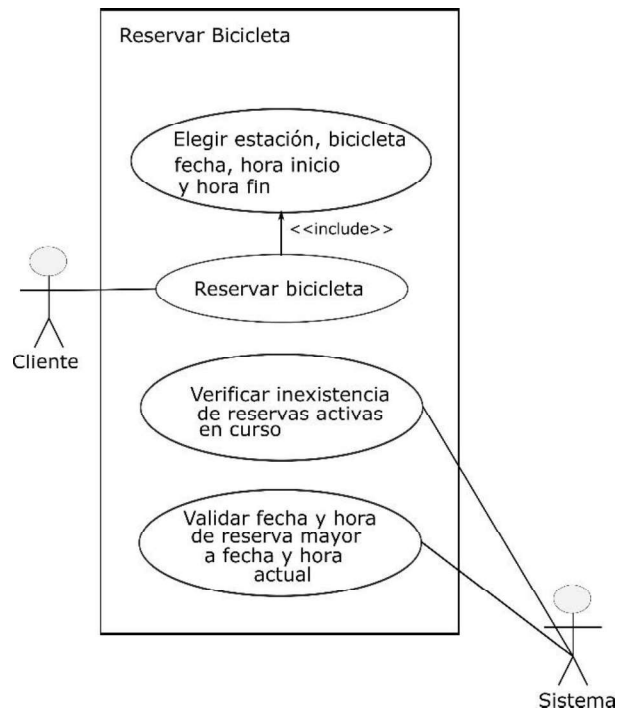
**Figura 2.2.** Caso de uso login de usuario

- Estaciones del sistema



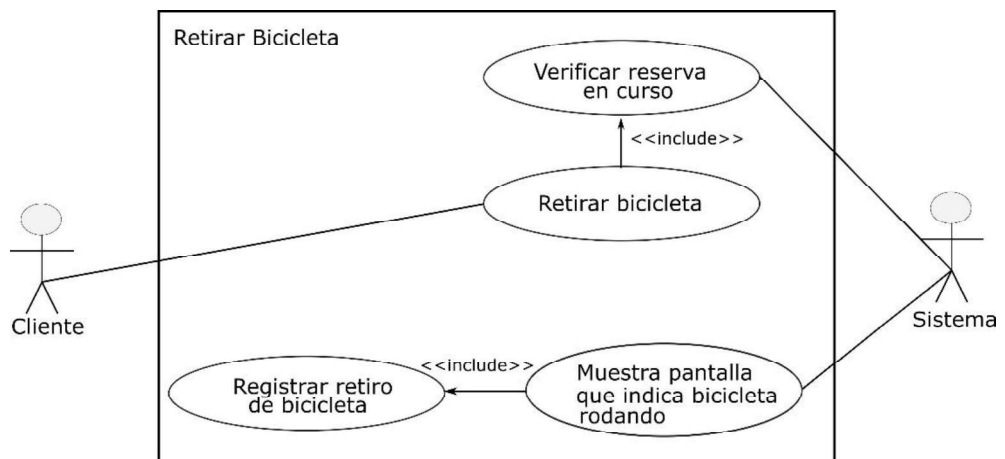
**Figura 2.3.** Caso de uso de estaciones del sistema

- Reserva de bicicleta



**Figura 2.4.** Caso de uso de reserva de bicicleta

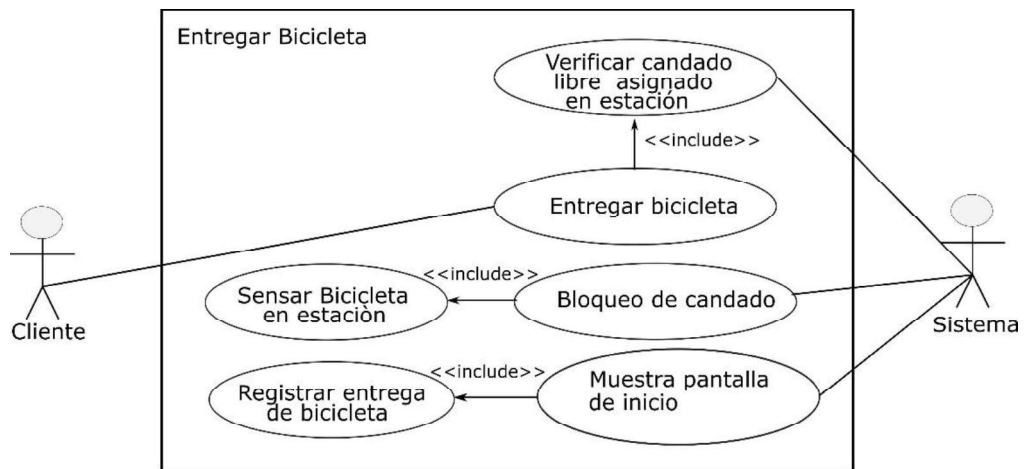
- Retiro de bicicleta



**Figura 2.5.** Caso de uso de retiro de bicicleta



- Entrega de bicicleta



**Figura 2.6.** Caso de uso de entrega de bicicleta

En el siguiente caso de uso, se visualizan las funcionalidades del sistema. Se diferencian los permisos de acceso del cliente y del administrador. El primero puede ver estaciones, bicicletas, reporte de reservas y crear reservas. El segundo puede administrar las reservas, los usuarios, las bicicletas, las estaciones y la configuración de la duración de reserva del sistema.

El control del administrador incluye la creación, lectura y edición, como se muestra en la Figura 2.7:



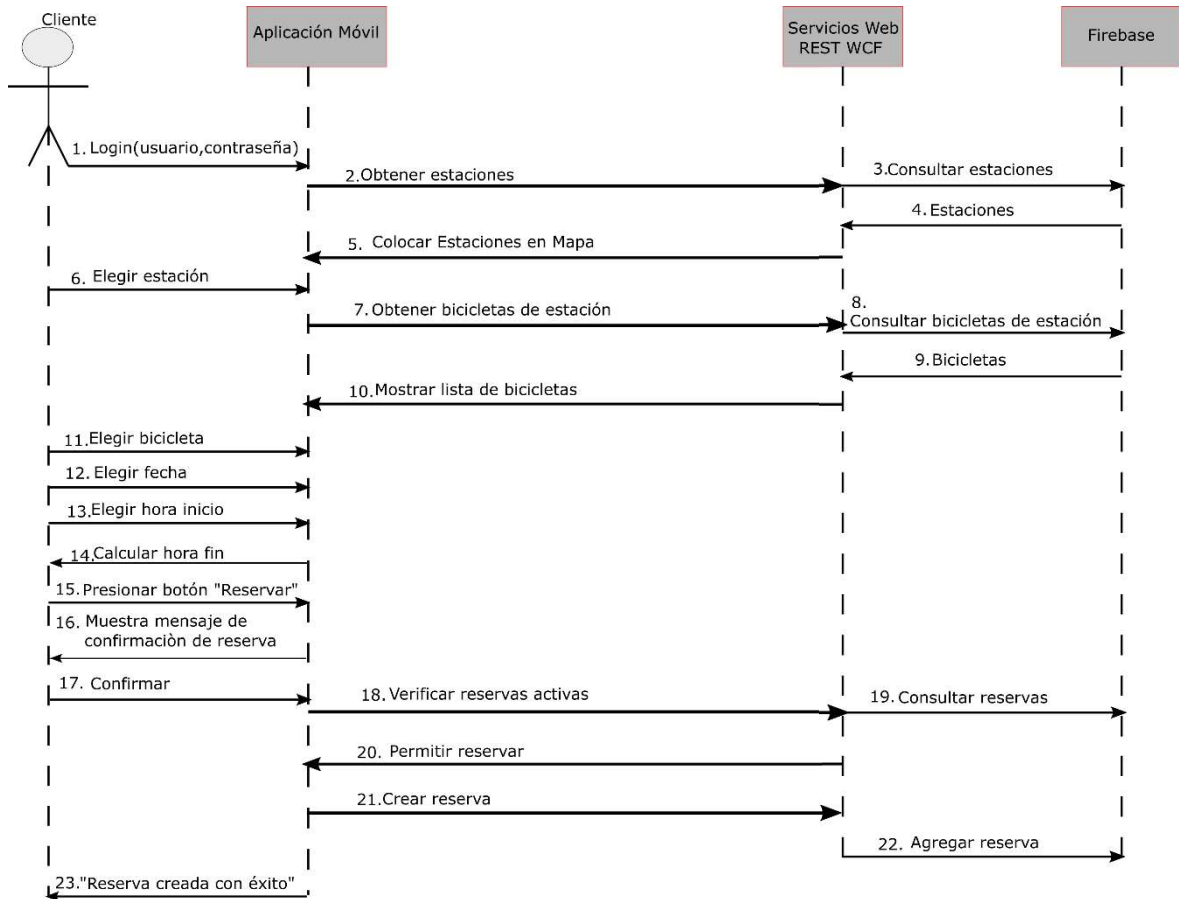
**Figura 2.7.** Actores y funciones del sistema

#### 2.1.4 Diagramas de secuencia

A continuación, se detallan los diagramas de secuencia.

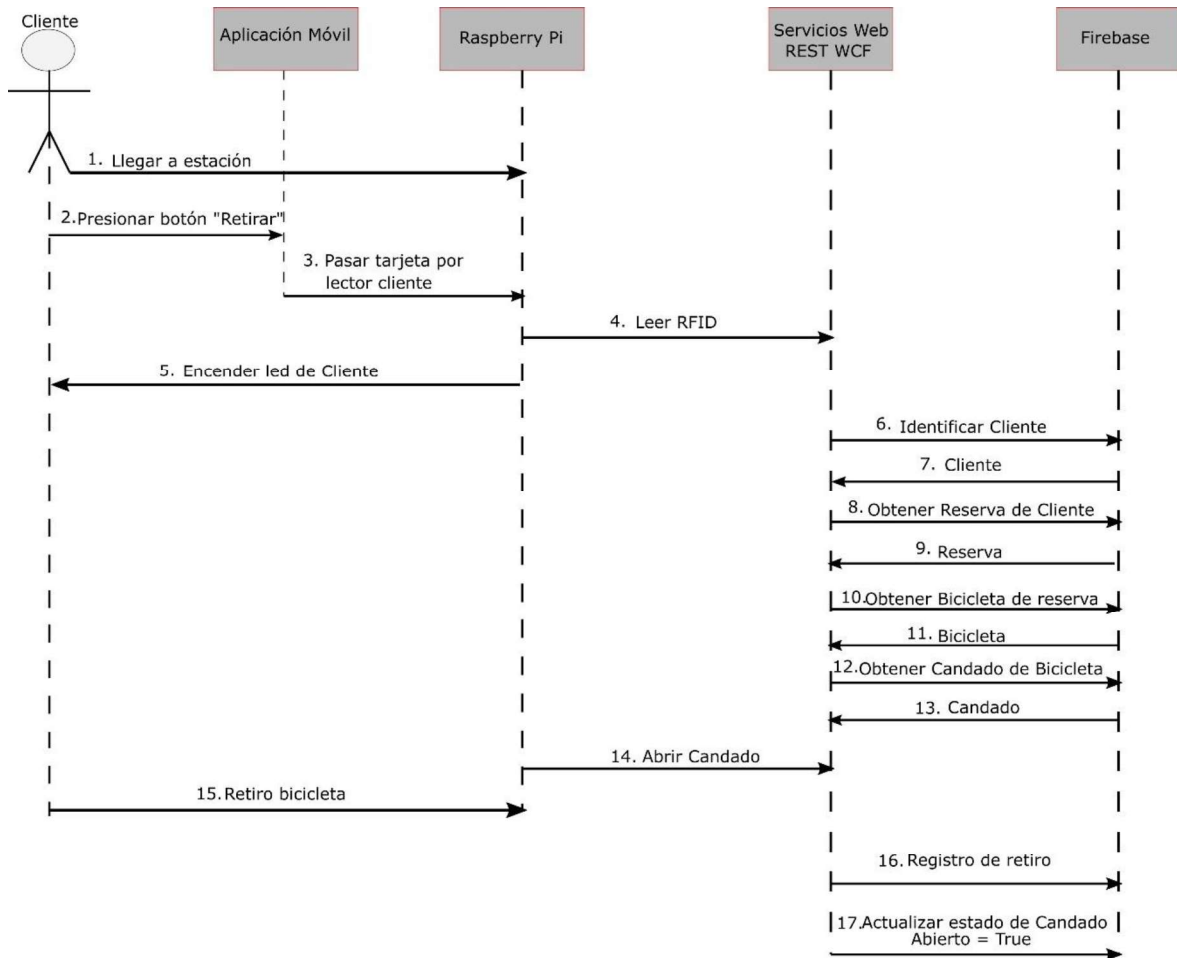
En Figura 2.8 se observa el proceso que se ejecuta cuando un cliente desea hacer una reserva. Para ello, debe realizar el login y en la pantalla principal visualiza un mapa con las

estaciones del sistema de préstamo de bicicletas. En ese momento el usuario puede elegir una estación y en la siguiente pantalla se despliegan las bicicletas disponibles. Acto seguido, la aplicación le permite escoger la fecha, hora inicio y hora fin de la reserva. Finalmente, la aplicación móvil solicita confirmación de los datos de la reserva, el cliente acepta y realiza con éxito la reserva.



**Figura 2.8.** Diagrama de secuencia de reserva de una bicicleta

En Figura 2.9 se observa la secuencia para retirar una bicicleta. El usuario llega a la estación y presiona el botón Retirar en la aplicación móvil, después pasa su tarjeta o tag RFID por el lector de cliente. Inmediatamente se enciende un led y se identifica el cliente para verificar la reserva asociada y otros parámetros como bicicleta y candado que debe liberarse con el fin de realizar el retiro. Para finalizar se crea el registro de retiro en la base de datos y se actualiza el estado no disponible de la bicicleta.



**Figura 2.9. Diagrama de secuencia retiro de una bicicleta**

En Figura 2.10 se detalla el proceso de entrega de bicicleta, en donde el usuario presiona el botón Entregar en la aplicación móvil. Acto seguido la aplicación muestra un mapa con las estaciones y la disponibilidad de candados disponibles y libres para enganchar bicicletas.

En este punto el cliente debe elegir la estación y el sistema le asigna el candado libre. En la aplicación móvil, el cliente presiona el botón finalizar, coloca la bicicleta y pasa su tarjeta RFID por el lector de Cliente para que pueda entregar con éxito, en ese momento se cierra el candado y se enciende led de lector de Bicicleta para crear el registro de entrega de bicicleta. Por último, se actualiza el registro donde se indica que el candado no está disponible.

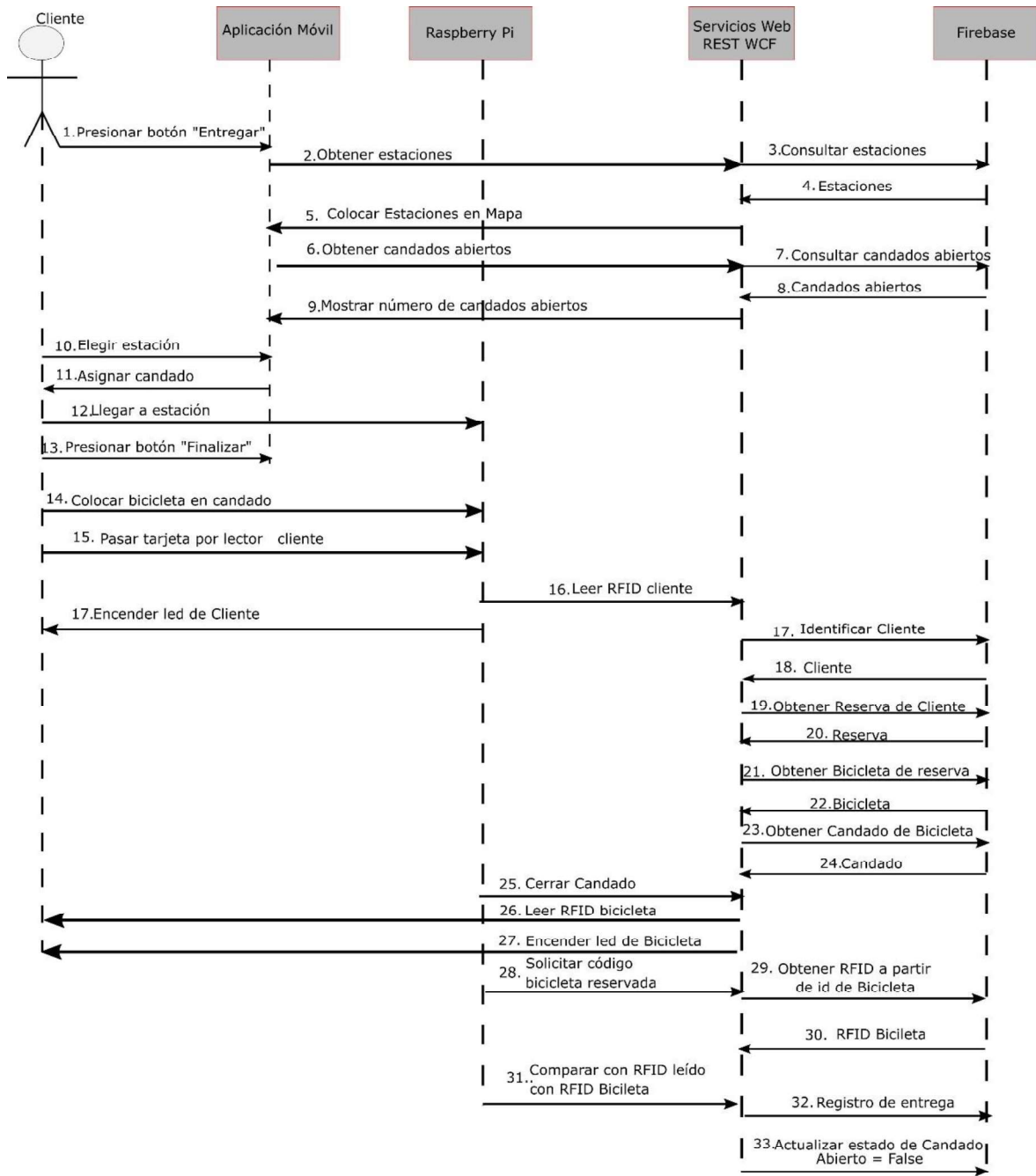
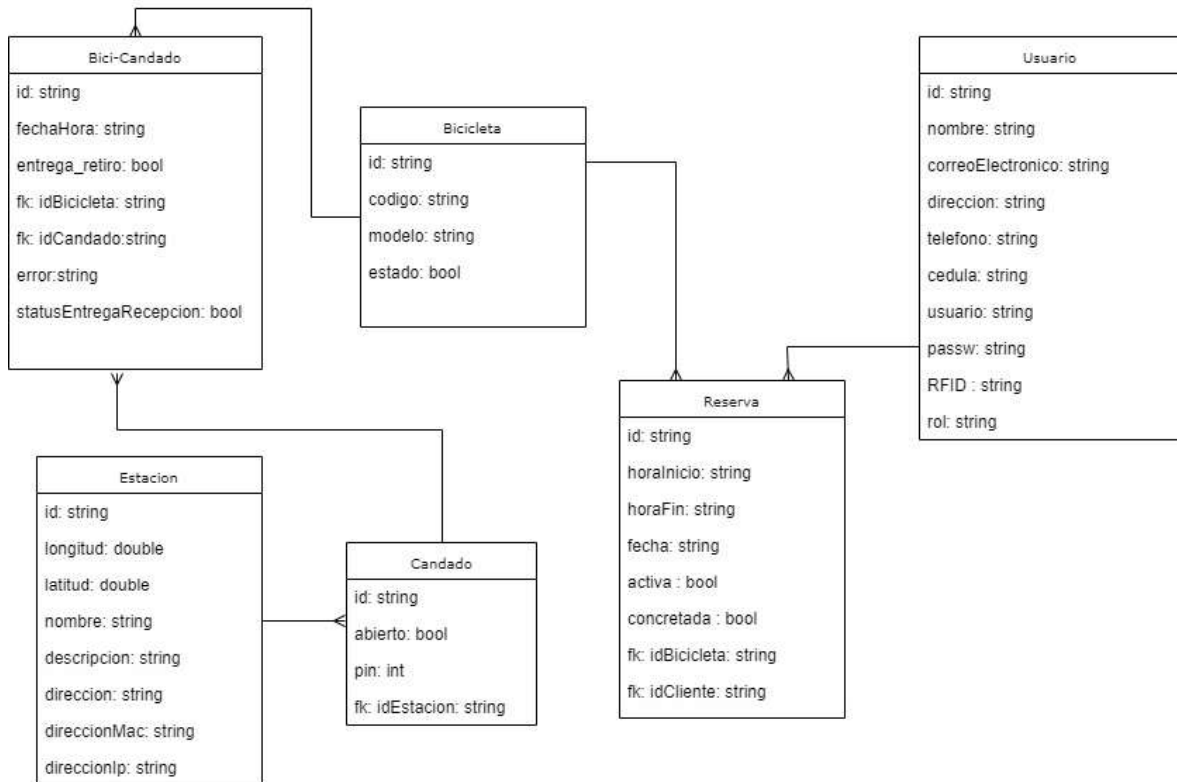


Figura 2.10. Diagrama de secuencia entrega de una bicicleta

### 2.1.5 Diagrama de clases servicio web WCF REST

En el diagrama de la Figura 2.11 se tiene un cliente que va a tener muchas reservas, cada reserva está relacionada al préstamo de una bicicleta, de este modo una bicicleta estará presente en muchas reservas. En este punto aparece la relación de muchos a muchos en donde los candados van a tener muchas bicicletas y las bicicletas van a estar en muchos candados por lo que entre bicicleta y candado surge una tabla BiciCandado que se aprovecha para utilizarla como la transacción del sistema, y la tabla que relaciona a las clases antes mencionadas. Finalmente se tiene que una estación va a tener muchos

candados. En resumen, toda bicicleta está asociada a un candado y a su vez el candado se asocia a una estación. De esta manera funcionan las clases del sistema. En la Figura 2.11 se visualizan las clases y métodos del sistema de préstamo de bicicletas.



**Figura 2.11.** Diagrama de clases servicio web

La clase BiciCandado es en donde se realizan las transacciones del sistema, es importante explicar los siguientes atributos:

- **id:** Es un string que identifica a la transacción BiciCandado.
- **fechaHora:** Es un string que almacena la fecha y hora del retiro o entrega
- **entrega\_retiro:** Es un atributo de tipo booleano que cuando toma el valor True significa que es una entrega de bicicleta y al contrario cuando es False es un retiro de bicicleta.
- **idBicicleta:** Es un string que determina la bicicleta que es retirada o entregada.
- **idCandado:** Es un string que indica el candado que se desbloquea en el retiro de una bicicleta o que se bloquea para enganchar la bicicleta entregada.
- **error:** Es un string que se muestra cuando la transacción no fue exitosa y muestra la falla ocurrida.
- **statusEntregaRecepcion:** Es un atributo booleano que al tomar el valor de True significa que la transacción se realizó con éxito y sin errores.

La clase Reserva se tienen los siguientes atributos:

- **id:** Es un string identificador único de una reserva.
- **horalnicio:** Es un string que indica la hora en la que inicia el préstamo de la bicicleta.
- **horaFin:** Es un un string que indica la hora límite en la que finaliza el préstamo de la bicicleta.
- **fecha:** Es un un string que indica la fecha en la que se realiza el préstamo de la bicicleta.
- **activa:** Indica que la reserva está en proceso. Se mantiene con valor True desde que se reserva hasta que se entrega la bicicleta. Una vez entregada la bicicleta, el valor que toma el atributo es False.
- **concretada:** Este atributo está en True cuando el cliente retiró la bicicleta en sitio. Éste es un indicador de las reservas ejecutadas por el sistema, debido a que hay usuarios que reservan bicicletas, pero no se presentan para retirar la bicicleta.
- **idBicicleta:** Indica la bicicleta reservada.
- **idCliente:** Indica el identificador del cliente que realiza la reserva.

Para la clase Bicicleta se debe destacar lo siguiente:

- **id:** String que identifica a una bicicleta.
- **código:** String que indica el código RFID propio de la bicicleta.
- **modelo:** String que define el modelo de la bicicleta.
- **estado:** Atributo booleano que cuando toma el valor True indica que la bicicleta está disponible y cuando toma valor False indica que la bicicleta está ocupada.

En la clase Candado se tiene:

- **id:** String identificador del candado.
- **abierto:** Atributo booleano que cuando toma el valor True indica que está libre y abierto, mientras que si es False indica que está cerrado y bloqueado.
- **pin:** Dato entero que indica el pin de la Raspberry que controla y envía señales en alto y bajo al candado.
- **idEstacion:** string que indica la estación en la cual el candado se encuentra.

En la clase Estación se tiene:

- **id:** String identificador de la estación.
- **longitud:** Dato double que indica coordenada de longitud donde se halla la estación.
- **latitud:** Dato double que indica coordenada de latitud donde se halla la estación.
- **nombre:** String que indica el nombre de la estación.

- **descripción:** String que almacena una descripción de la estación.
- **dirección:** String que muestra la dirección de la estación.
- **direccionMac:** String que muestra la dirección MAC de la Raspberry de la estación.
- **direccionIp:** String que indica la dirección IP que tiene la Raspberry Pi de la estación.

En la tabla Usuario se tiene:

- **id:** String identificador del usuario.
- **nombre:** String que define nombre del usuario.
- **correoElectronico:** String que indica el correo del cliente.
- **dirección:** String que indica la dirección del domicilio del cliente.
- **teléfono:** String que indica el teléfono del cliente.
- **cédula:** String que indica la cédula del cliente.
- **usuario:** String que indica el nickname o usuario del cliente.
- **passwd:** Es un string que almacena la contraseña del usuario.
- **RFID:** String que indica el código RFID que tiene el tag que identifica a cada cliente.
- **Rol:** String que muestra si el usuario es cliente o administrador.

### 2.1.5.1 Diseño de Servicio WCF REST

El diseño de la API se ha realizado siguiendo los lineamientos de la especificación OpenAPI<sup>8</sup>. De la tabla 2.16 a la tabla 2.21 se muestra el método y la URI de cada endpoint de la API. Además, se detalla el uso de cada endpoint.

---

<sup>8</sup> La especificación OpenAPI (OAS) define una interfaz estándar independiente del lenguaje para las API RESTful que permite y facilita la comprensión del mismo sin acceder al código fuente.[47]



**Tabla 2.16.** URI y métodos de servicio de bicicletas

<b>BicicletasServicio</b>	<b>Gestiona bicicletas</b>	
Uso	Método	URI
Obtiene todas las bicicletas de la base de datos.	GET	/bicicletas
Obtiene la bicicleta con el id especificado.	GET	/bicicletas/{idBici}
Crea una nueva bicicleta.	POST	/bicicletas/nueva
Modifica la bicicleta con el id especificado.	PUT	/bicicletas/{idBici}
Elimina la bicicleta con el id especificado.	DELETE	/bicicletas/{idBici}

**Tabla 2.17.** URI y métodos de servicio de clientes

<b>ClienteServicio</b>	<b>Gestiona clientes</b>	
Uso	Método	URI
Obtiene todos los clientes de la base de datos.	GET	/clientes
Obtiene el cliente con el id especificado.	GET	/clientes/{idCliente}
Crea un nuevo cliente.	POST	/clientes/nuevo
Modifica el cliente con el id especificado.	PUT	/clientes
Elimina el cliente con el id especificado.	DELETE	/clientes/{idCliente}
Realiza la autenticación de un cliente.	POST	/clientes/authentication
Obtiene el cliente con el rfid especificado.	GET	/clientes?rfid={rfid}

**Tabla 2.18.** URI y métodos de servicio de clientes

<b>ReservaServicio</b>	Gestiona reservas	
Uso	Método	URI
Obtiene todas las reservas de la base de datos.	GET	/reservas
Obtiene la reserva con el id especificado.	GET	/reservas/{idReserva}
Crea una nueva reserva.	POST	/reservas/nueva
Modifica la reserva con el id especificado.	PUT	/reservas/{idReserva}
Elimina la reserva con el id especificado.	DELETE	/reservas/{idReserva}

**Tabla 2.19.** URI y métodos de servicio de estaciones

<b>EstacionesServicio</b>	Gestiona estaciones	
Uso	Método	URI
Obtiene todas las estaciones de la base de datos.	GET	/estaciones
Obtiene la estación con el id especificado.	GET	/estaciones/{idEstacion}
Crea una nueva estación.	POST	/estaciones/nueva
Modifica la estación con el id especificado.	PUT	/estaciones/{idEstacion}
Elimina la estación con el id especificado.	DELETE	/estaciones/{idEstacion}

**Tabla 2.20.** URI y métodos de servicio de Bicicandados

<b>Bici-EstacionServicio</b>	Gestiona las bicis en las transacciones Bicicandados	
Uso	Método	URI
Obtiene todas los Bicicandados de la base de datos.	GET	/bicisCandados
Obtiene el Bicicandado con el id especificado.	GET	/ bicisCandados/{id}
Crea un nuevo Bicicandado.	POST	/ bicisCandados/nueva
Modifica el Bicicandados con el id especificado.	PUT	/ bicisCandados/{id}
Elimina el Bicicandado con el id especificado	DELETE	/ bicisCandados/{id}

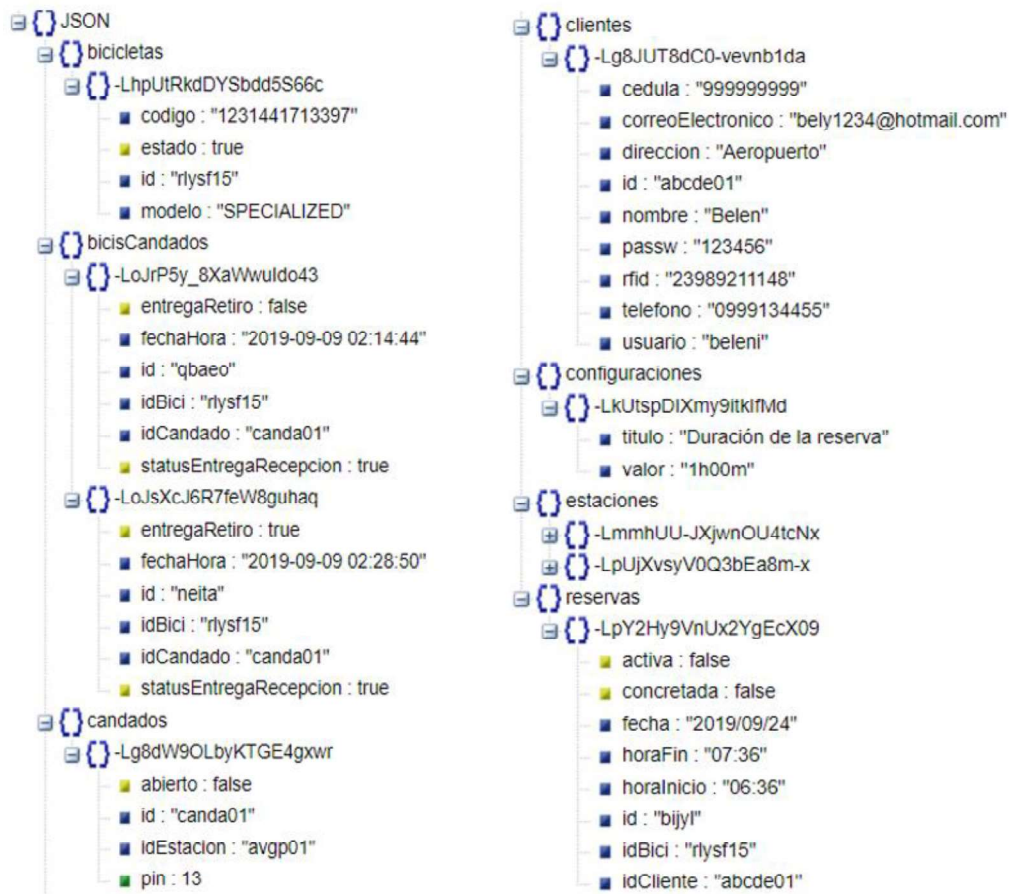
**Tabla 2.21.** URI y métodos de servicio de Candados

CandadosServicio	Método	URI
Obtiene todos los candados de la base de datos.	GET	/candados
Obtiene el candado con el id especificado.	GET	/candados/{id}
Crea un nuevo candado.	POST	/candados/nuevo
Modifica el candado con el id especificado.	PUT	/candados
Elimina el candado con el id especificado.	DELETE	/candados/{id}

### 2.1.6 Estructura del árbol de objetos JSON

En este proyecto se ha utilizado la base de datos Firebase que almacena los datos en forma de árbol JSON. Firebase maneja a cada nodo como padre y a cada subnodo u objeto lo considera como un hijo.

La actual base de datos está compuesta de 7 nodos y cada subnodo representa un objeto JSON. En la Figura 2.12 se aprecia la distribución de nodos.



**Figura 2.12.** Árbol JSON base de datos

Los nodos son:

- **bicicletas:** Almacena las bicicletas presentes en el sistema.
- **bicisCandados:** Almacena las transacciones de retiros y entregas de bicicletas del sistema.
- **candados:** Almacena los candado abiertos y cerrados del sistema.
- **clientes:** Almacena los clientes registrados del sistema.
- **configuraciones:** Almacena la configuración o parámetro de duración de las reservas.
- **estaciones:** Almacena las estaciones del sistema.
- **reservas:** Almacena las reservas del sistema.

El árbol JSON tiene su forma en base a la estructura de datos generada por el servicio Web WCF REST.

## 2.1.7 Sketchs de las pantallas para programar la interfaz de usuario

### 2.1.7.1 Aplicación Cliente

A continuación, se adjuntan los sketches de diseño de la aplicación móvil. En Android a cada pantalla se le conoce como activity:

En la Figura 2.13 se observa la activity de Registro para los usuarios que aún no tienen una cuenta en el sistema.

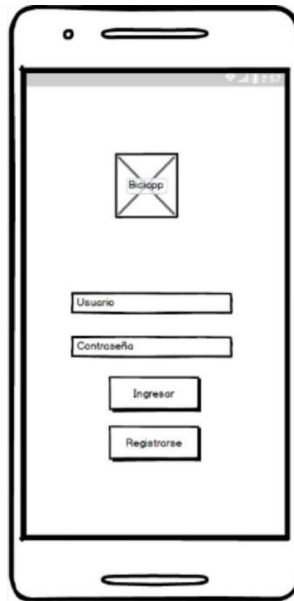


The sketch shows a mobile registration form titled "Registro de Usuario". It contains the following fields and buttons:

- Nombre
- Apellido
- Nombre de Usuario
- Cédula
- Contraseña
- Confirmar contraseña
- Email
- Dirección
- Teléfono
- Aceptar

**Figura 2.13.** Activity de registro

En la Figura 2.14 se aprecia el login o ingreso al sistema. Para ello el cliente hace uso de su usuario y contraseña que serán comparados con el nodo de Clientes de Firebase para verificar si está registrado o no.



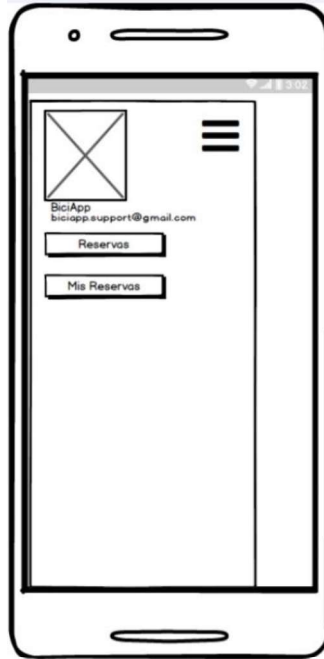
**Figura 2.14.** Activity de Login

En la Figura 2.15, se observa el mapa con las estaciones cercanas a la ubicación actual del cliente.



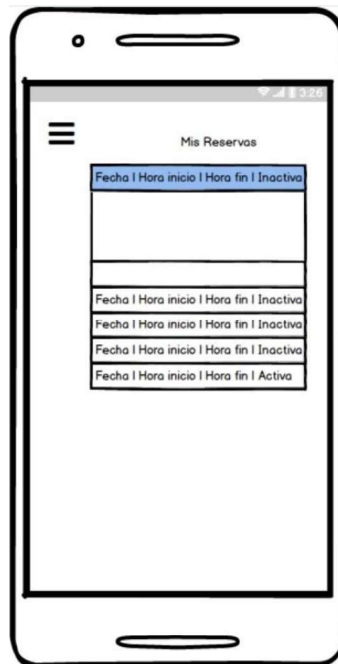
**Figura 2.15.** Activity pantalla principal

En Figura 2.16 se aprecia un menú que permite navegar entre Mis Reservas que son el reporte de reservas realizadas por el cliente y Reservas que es un atajo para poder acceder a una reserva.



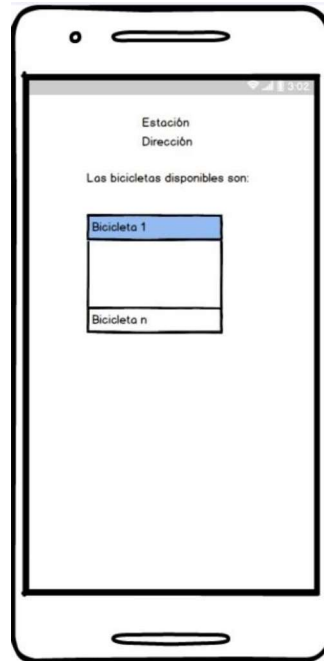
**Figura 2.16.** Menu de aplicación móvil

En Figura 2.17 se observa el boceto de cómo se visualiza el reporte de reservas de un cliente.



**Figura 2.17.** Activity reporte de reservas

En Figura 2.18 se observa la activity a la que se dirige la aplicación para continuar con la reserva. Una vez que en la Figura 2.15 se elige una estación, en esta pantalla se elige la bicicleta:



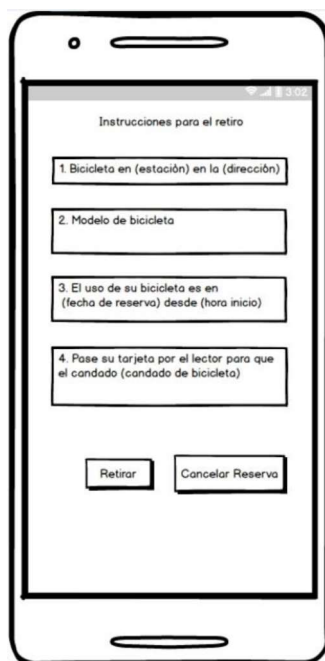
**Figura 2.18.** Activity de bicicletas disponibles de una estación

En la figura 2.19, se observa una pantalla con la fecha, hora inicio y hora fin de la reserva con un botón de confirmación para finalizar la reserva con éxito.



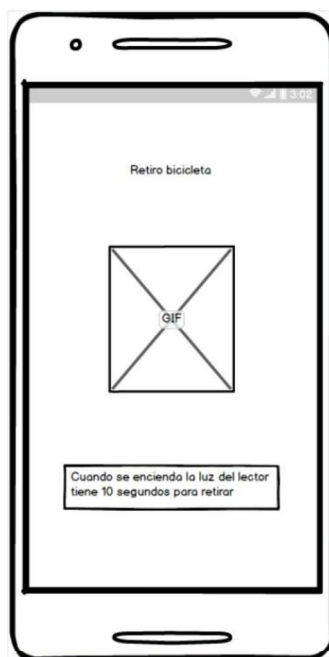
**Figura 2.19.** Activity de reserva

En la activity siguiente de la Figura 2.20 se observan las instrucciones para retirar la bicicleta y se muestra un botón para cancelar en caso de que el cliente tenga dificultades o ya no desee hacer uso de la bicicleta que reservó.



**Figura 2.20.** Activity instrucciones Retiro

En la pantalla de la Figura 2.21 se observa un mensaje que indica la espera mientras se valida la información del cliente y se desbloquea la cerradura.



**Figura 2.21.** Activity procesando retiro

En esta Figura 2.22 se observa una pantalla que indica que la bicicleta está bajo la responsabilidad del cliente y que en efecto fue retirada con éxito de la estación. Se incluye un botón que dice entregar, el mismo que debe ser usado una vez que esté por vencer la reserva y cuando el usuario desee entregarla:





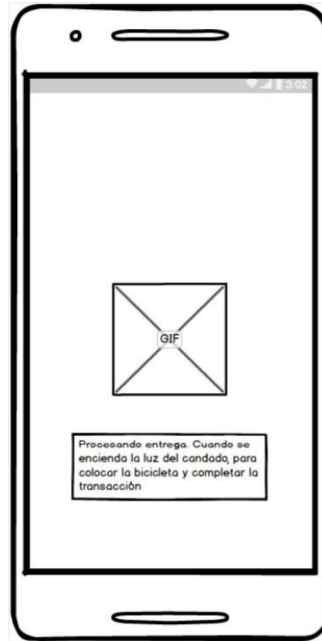
**Figura 2.22.** Activity que indica bicicleta rodando

Para concluir, se tiene una pantalla que aparece al presionar el botón Entrega como se observa en la Figura 2.22. La activity de la Figura 2.23 permite al cliente elegir una estación y muestra un campo en el que el sistema asigna un candado:



**Figura 2.23.** Activity de entrega de bicicleta

La Figura 2.24 cumple el formalismo de procesar la entrega y que cambiará a la activity inicial Figura 2.15 para continuar con otra reserva en el momento que el cliente lo requiera.

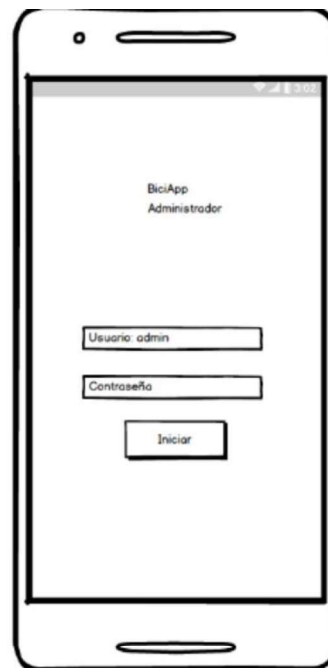


**Figura 2.24.** Activity de procesando Entrega

### 2.1.7.2 Aplicación Administrador

Esta aplicación es mucho más simple que la del cliente, debido a que tiene menos actividades.

En la Figura 2.25 se observa la interfaz de bienvenida:



**Figura 2.25.** Activity de login Administrador

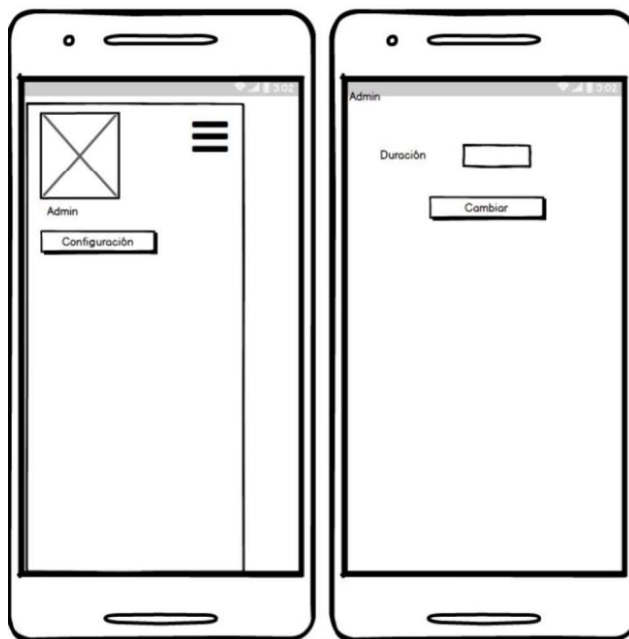
Luego, en la Figura 2.26, se pasa a la actividad principal que muestra unos íconos para gestionar parámetros del negocio tales como: usuarios, estaciones, bicicletas y reservas.

En esta aplicación se utiliza la misma pantalla para cambiar datos y gestionar los diferentes parámetros:



**Figura 2.26.** Activity principal Administrador

Finalmente, para cambiar configuraciones como modificar la duración de una reserva se utilizan 2 Activities como se aprecia en la Figura 2.27:



**Figura 2.27.** Activities para cambiar duración de reserva

### 2.1.8 Circuito de control de la cerradura electromagnética

En esta sección se diseña un circuito, ver Figura 2.258, que utiliza un modo de conexión que permite el uso de un relé para manejar voltajes de alimentación diferentes.

Para dimensionar el esquema del circuito se toma en cuenta que la alimentación de la cerradura es de 12v, la señal de control del Raspberry Pi es de 5v por lo que se requiere utilizar un relé de 5v, y un transistor que funcione en modo de switch para manejar las dos fuentes de alimentación. El relé de 5v tiene como objetivo aislar la alimentación de la cerradura electromagnética de la alimentación de la Raspberry Pi. También se incluye un diodo para proteger la bobina del relé de corrientes reversas que pueden dañarlo.

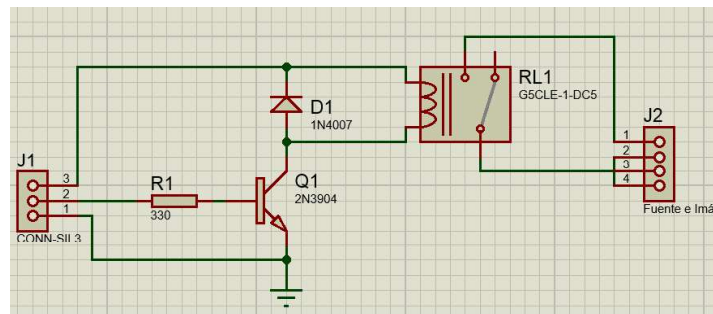


Figura 2.28. Esquema circuito

El voltaje que se tiene entre los pines del relé es de 5v, el voltaje necesario para que el transistor ingrese en saturación es de 0.6v. Y la corriente que admite el transistor es de 15mA. Por lo que la resistencia se obtiene según la Ecuación 2.1:

$$R_1 = \frac{V - V_{sat}}{I} \quad (2.1)$$

Por lo que el valor de la resistencia es 293.33Ω que se utiliza 330Ω

### 2.1.9 Diagramas de flujo

Los diagramas de flujo definen los algoritmos empleados para bloquear y desbloquear la cerradura electromagnética, y para la lectura de los tags RFID utilizados en el sistema de bicicletas. Aplican para ser programados en Python en la Raspberry Pi. En la Figura 2.29 se observa el diagrama global de los scripts (Anexo Q y Anexo R) que manejan esta lógica de la cerradura electromagnética conocida como candado en este proyecto. En el diagrama de flujo existe un punto denominado A que indica que el algoritmo se mantiene en un bucle infinito en el que apaga el led del cliente y la bicicleta para realizar la lectura cíclica del RFID cliente.

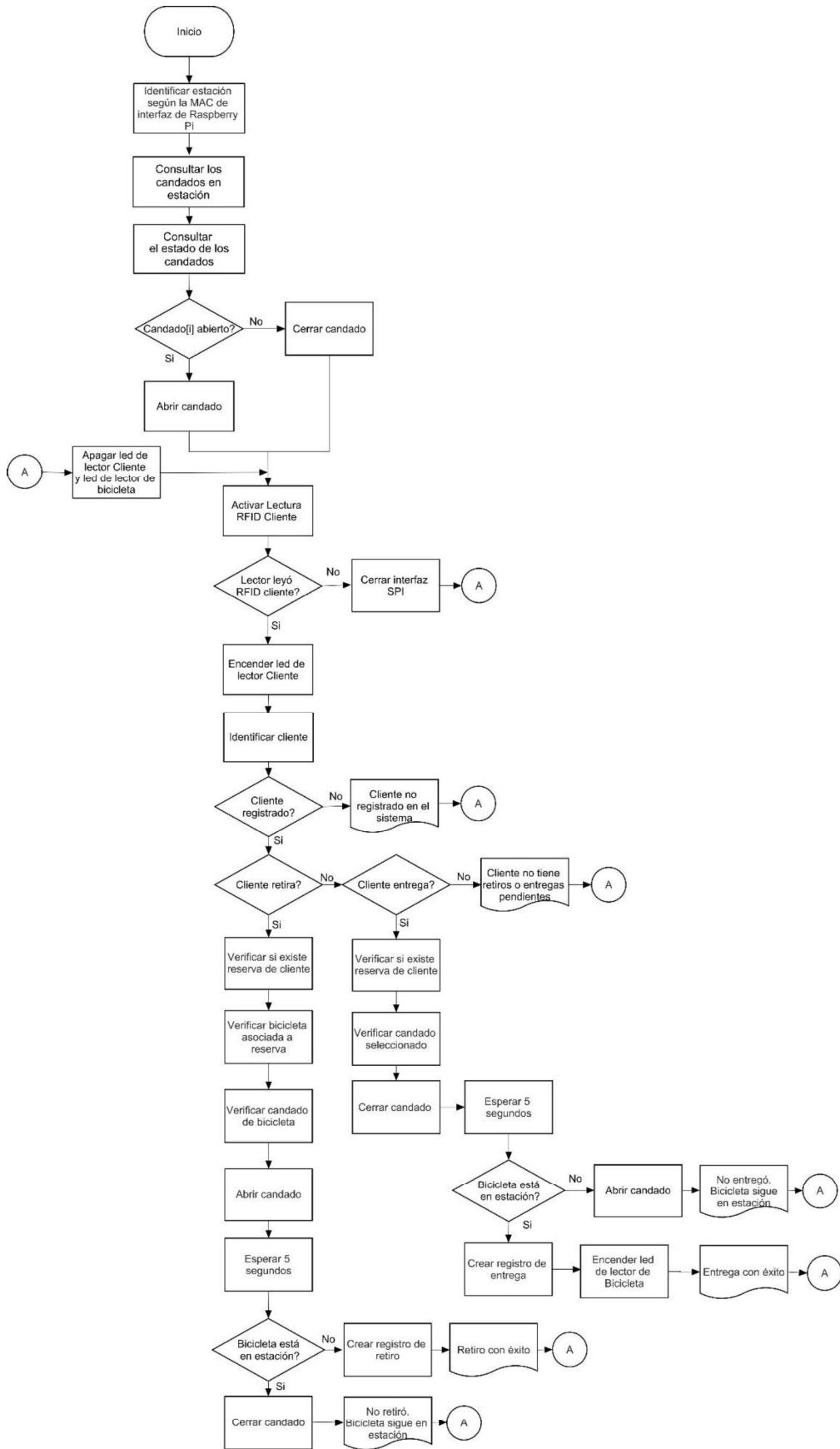
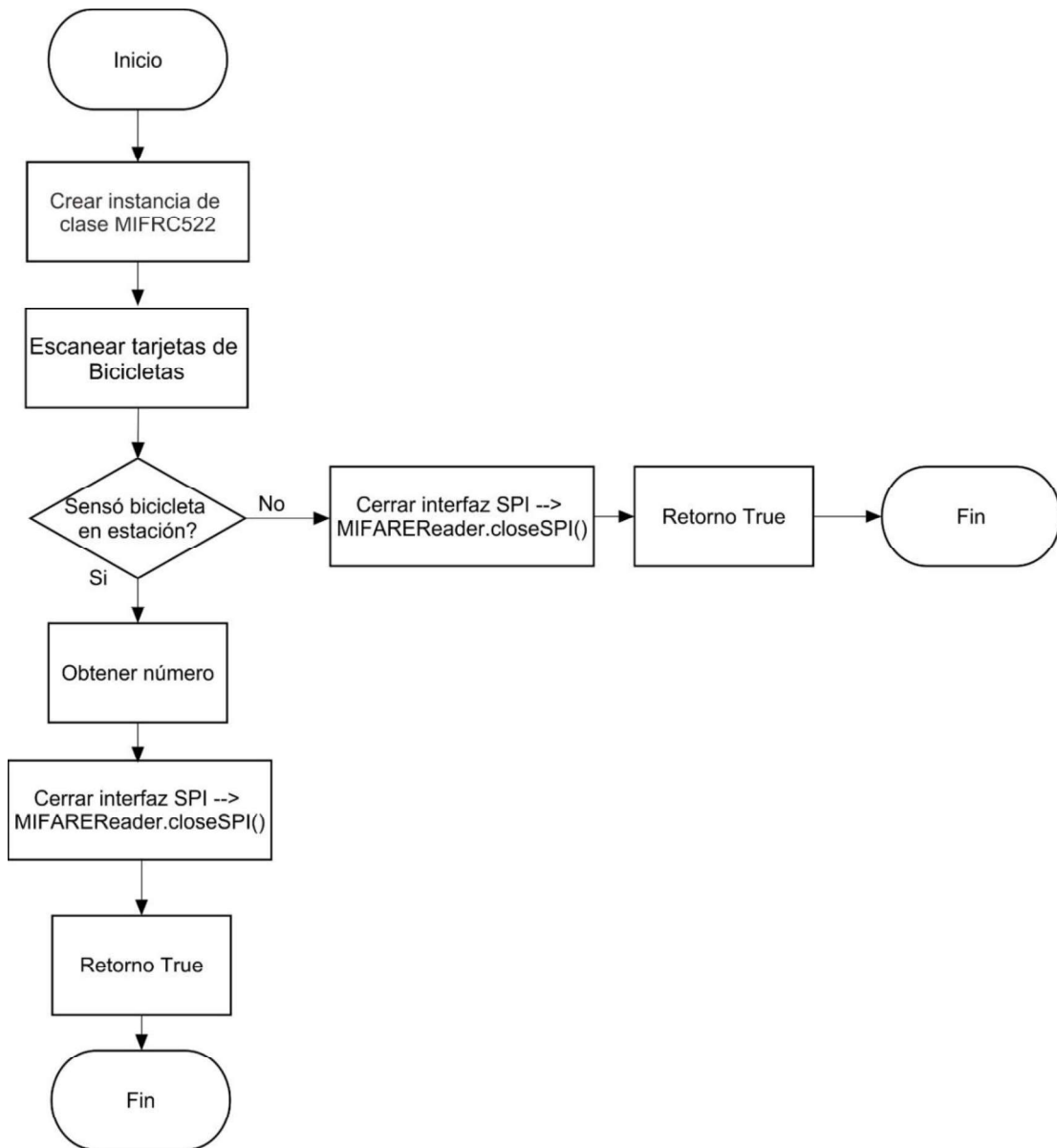


Figura 2.29. Algoritmo de bloqueo y desbloqueo de cerradura electromagnética

Ahora en la Figura 2.30 se muestra el algoritmo que verifica si la bicicleta está en la estación (Anexo R):



**Figura 2.30.** Algoritmo que verifica si bicicleta está en estación

## 2.2 IMPLEMENTACIÓN

La implementación del módulo principal se enfoca en el desarrollo de un servicio web WCF que tiene clases e interfaces, y alberga la lógica del negocio. Contempla la estructura de la base de datos Firebase que almacena los datos del sistema de bicicletas a manera de árbol de objetos JSON. En cuanto a la interfaz de usuario se toma en cuenta la definición de los requerimientos: funcionales y no funcionales. Adicionalmente, se realiza la conexión de la interfaz de usuario y el módulo principal con el fin de realizar el consumo del servicio web y se utilizan los sketches para implementar la app móvil.

A continuación, para el módulo de control se arma la circuitería necesaria para la construcción de la estación de bicicletas y se codifican los scripts con los algoritmos de bloqueo / desbloqueo de la cerradura, la lectura de los tags RFID y envío / recepción de datos entre la Raspberry Pi y Firebase.

## 2.2.1 Módulo principal e interfaz de usuario

### 2.2.1.1 Implementación de código

Para la implementación del código de las aplicaciones se ha utilizado dos lenguajes de programación. Java y XML han sido utilizados para las aplicaciones cliente Android y el lenguaje C# sobre el framework de WCF (Windows Communication Foundation) para el servidor. Las aplicaciones Android se basan en el patrón de arquitectura MVVM (Model View View-Model) mientras que el servidor WCF sigue los lineamientos de diseño de la especificación de OpenAPI. Por otra parte, aspectos como la comunicación entre servidor y base de datos o servidor y aplicaciones han sido optimizados a través del uso de librerías asíncronas correspondientemente para Java como para C#. Para el manejo de dependencias se ha utilizado el gestor de dependencias de Android Studio llamado Gradle que utiliza como lenguaje de programación Kotlin dsl y para el manejo de dependencias se ha utilizado la herramienta Nugget propia de Visual Studio.

### 2.2.1.2 Gradle de Proyecto Android

Un proyecto de Android Studio incluye dependencias por defecto, en el desarrollo del presente trabajo de titulación se añaden las dependencias que se destacan a continuación, en el Código 2.1: Las líneas 26 y 27 son necesarias para utilizar la arquitectura retrofit, la línea 32 es una librería para compilar mapas, la línea 36 es una librería necesaria para incluir GIFs en la aplicación a desarrollar.

```
21 dependencies {
22     implementation fileTree(dir: 'libs', include: ['*.jar'])
23     implementation 'com.android.support:appcompat-v7:28.0.0'
24     implementation 'com.android.support:design:28.0.0'
25     implementation 'com.android.support.constraint:constraint-layout:1.1.3'
26     implementation 'com.squareup.retrofit2:retrofit:2.5.0'
27     implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
28     implementation 'android.arch.lifecycle:extensions:1.1.1'
29     testImplementation 'junit:junit:4.12'
30     androidTestImplementation 'com.android.support.test:runner:1.0.2'
31     androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
32     implementation 'com.google.android.gms:play-services-maps:16.1.0'
33     implementation 'com.android.support:support-media-compat:28.0.0'
34     implementation 'com.android.support:support-v4:28.0.0'
35     implementation 'com.android.support:support-annotations:28.0.0'
36     implementation 'pl.droidsonroids.gif:android-gif-drawable:1.2.+'
37 }
```

**Código 2.1.** Dependencias de Gradle

### 2.2.1.3 Codificación de la Aplicación cliente (no administrador)

Para la codificación de la aplicación cliente, según los lineamientos de diseño, se ha dividido el código Java en tres subcarpetas principales, datos, ui (user interface) y servicios, mientras que el código XML ha sido dividido automáticamente por Android Studio en algunas subcarpetas que representan los recursos gráficos y el archivo de manifiesto de la aplicación.

### 2.2.1.4 Carpetas de Proyecto Android

#### Carpeta datos (Anexo C)

La carpeta datos contiene a las subcarpetas modelos y repos. La carpeta modelos contiene las clases que describen un objeto del dominio de la aplicación según el diagrama de clases. El Código 2.2 muestra un fragmento de código Java del archivo Bicicleta.java que representa un objeto Bicicleta del dominio de negocio como clase; allí se puede apreciar en la línea 5 la definición de la clase. Además, ésta implementa la interfaz serializable que permite que sus objetos puedan ser manejados a través de dos o más activities; las líneas 6-8 muestran los atributos de la clase y las líneas 10-35 muestran el encapsulamiento de los atributos a través de los getters y setters.

```
5 public class Bicicleta implements Serializable {
6     private boolean estado;
7     private String modelo;
8     private String id;
9
10    public boolean getEstado() { return estado; }
13
14    public String getModelo() { return modelo; }
17
18    @Override
19    public String toString() { return modelo; }
22
23    public String getId() { return id; }
26
27    public void setId(String id) { this.id = id; }
30
31    public void setEstado(boolean estado) { this.estado = estado; }
34
35    public void setModelo(String modelo) { this.modelo = modelo; }
38 }
```

**Código 2.2** Clase Bicicleta

#### Carpeta repos (Anexo D)

Por otra parte, la carpeta repos contiene los repositorios para cada objeto del dominio de la aplicación, es decir, allí se encuentra gran parte del código que permite realizar las solicitudes HTTP y almacenar las respuestas como objetos Java de forma que estos objetos puedan ser mutados en caso de necesitarlo. El Código 2.3 muestra un fragmento de código Java en donde se puede observar cómo se almacena un objeto proveniente de una solicitud HTTP. La línea 21 define una lista de bicicletas envuelta por la clase MutableLiveData, los objetos de esta clase permiten ser mutados por las solicitudes HTTP



mientras se mantienen ligados a un View-Model. En la línea 23 empieza la definición del método `getBicicletas`, este método recibe como parámetro el `String idEstación` que representa el identificador de la estación y retorna una lista de Bicicletas envuelta por la clase `LiveData`. Las líneas 25-28 inicializan un objeto `Retrofit` el cual es el principal encargado de formar la solicitud HTTP; la URL es especificada como parámetro del método `baseUrl` (línea 26) y se indica que las solicitudes manejarán lenguaje JSON (línea 27). Luego de ello se referencia al servicio que se utilizará (línea 30) en este caso se empleará el servicio `IServicioBiciCandados` donde como se apreciará posteriormente se encuentran los endpoints que apuntan hacia las API. Luego, de la línea 32 a la línea 47 se realiza una llamada asíncrona que maneja las posibles respuestas o fallos de la llamada HTTP. La línea 49 define el método que funcionará como `getter` del atributo `data`.

```

21 private MutableLiveData<List<Bicicleta>> data = new MutableLiveData<>();
22
23 public LiveData<List<Bicicleta>> getBicicletas(String idEstacion) {
24
25     Retrofit retrofit = new Retrofit.Builder()
26         .baseUrl(IServicioCliente.BASE_URL)
27         .addConverterFactory(GsonConverterFactory.create())
28         .build();
29
30     IServicioBicisCandados service = retrofit.create(IServicioBicisCandados.class);
31
32     Call<List<Bicicleta>> requestBicisEstacion = service.obtenerBicisByEstacion(idEstacion);
33     requestBicisEstacion.enqueue(new Callback<List<Bicicleta>>() {
34         @Override
35         public void onResponse(Call<List<Bicicleta>> call, Response<List<Bicicleta>> response) {
36             if (!response.isSuccessful()) {
37                 data.setValue(null);
38             } else {
39                 data.setValue(response.body());
40             }
41         }
42         @Override
43         public void onFailure(Call<List<Bicicleta>> call, Throwable t) { data.setValue(null); }
44     });
45     return data;
46 }
47
48
49 public MutableLiveData<List<Bicicleta>> getData() { return data; }

```

**Código 2.3.** Método que obtiene bicicletas según estación

### Carpeta ui (Anexo F)

La carpeta `ui` contiene carpetas que representan las actividades de la aplicación con su respectivo `View-Model`. Por lo general una de estas carpetas contiene tres archivos. El primero que representa la actividad, el segundo el `View-Model` correspondiente a la actividad y el tercero la clase creadora de tal `View-Model`.

### Activity

Una actividad por lo general obtiene el `intent` proveniente de una actividad anterior y genera un nuevo `intent` para la actividad posterior; inicializa los controles de la interfaz gráfica para que

estos puedan ser manipulados a través de código Java; genera peticiones al View-Model; y puede contener listeners para los controles y eventos o manejo de cuadros de diálogo. El Código 2.4 muestra un fragmento de una Activity; en la línea 38 se crea la actividad a través del método onCreate y en la línea 39, a través del método setContentView se indica qué archivo XML irá ligado a la actividad. Por otra parte, las líneas 42 hasta 44 muestran cómo a través de un objeto intent se obtiene información proveniente de la actividad anterior. En las líneas 46 hasta 48 se inicializa controles que pertenecen la interfaz gráfica, a través del método findViewById, para poder ser manipulados desde la actividad. En las líneas 50 y 51 se cambia el texto de dos controles tipo TextView.

### Viewmodel

En la línea 54 y 55 se inicializa el View-Model de la actividad a través del método estático ViewModelProviders.of, a este método es necesario especificarle el contexto al cual se desea que el View-Model esté ligado. La línea 57 acciona la llamada asíncrona solicitando al View-Model un recurso o petición, mientras que las líneas 59-65 describen el método manejador de la llamada de la línea 57. En otras palabras, este método manejador de llamadas asíncronas define qué se hará luego que la llamada tenga una respuesta.

```

37 protected void onCreate(Bundle savedInstanceState) {
38     super.onCreate(savedInstanceState);
39     setContentView(R.layout.activity_estacion_bicicletas);
40
41     //Obtener los objetos provenientes de la actividad anterior (MapsActivity)
42     Intent intent = getIntent();
43     clienteView = (LoginClienteView) intent.getSerializableExtra(CLIENT_VIEW);
44     estacionView = (Estacion) intent.getSerializableExtra(ESTACION_VIEW);
45
46     final TextView titleEstacionTextView = findViewById(R.id.titulo_estacion);
47     final ListView bicisEstacionListView = findViewById(R.id.bicis_estacion);
48     final TextView direccionEstacionTextView = findViewById(R.id.direccion_estacion);
49
50     titleEstacionTextView.setText(estacionView.getNombre());
51     direccionEstacionTextView.setText(estacionView.getDireccion());
52
53
54     viewModel = ViewModelProviders.of(this, new EstacionBicicletasViewModelFactory())
55         .get(EstacionBicicletasViewModel.class);
56
57     viewModel.obtenerBicicletas(estacionView.getId());
58
59     viewModel.observeBicicletas().observe(this, (bicicletas) -> {
60         adapter = new ArrayAdapter<Bicicleta>(getApplicationContext(),
61             android.R.layout.simple_list_item_1, new ArrayList<Bicicleta>(bicicletas));
62         bicisEstacionListView.setAdapter(adapter);
63     });
64
65 }

```

**Código 2.4.** Método onCreate

### Carpeta Servicios (Anexo E)

La carpeta servicios contiene los archivos Java que apuntan hacia los endpoints de la API del servidor WCF. Estos archivos no son clases sino interfaces, según los requerimientos de uso de Retrofit, y cada método está compuesto por ciertas anotaciones que ayudan a describir una solicitud HTTP. El Código 2.5 muestra la interface IServicioBicicletas. Esta

interface contiene un único método utilizado para obtener un objeto Bicicleta de un identificador específico. Este método contiene una anotación en la línea 12 que especifica el método HTTP (GET) y la URI a la cual irá la petición, esta URI utiliza la característica de reemplazo de parámetros (idBici) para colocar el parámetro del método en la URI. Además, se debe recalcar que el retorno del método obtenerBicicleta es un objeto del tipo Bicicleta envuelto en la clase Call, esto se realiza para que la llamada a este método cuente con los métodos asíncronos necesarios para manejar solicitudes en segundo plano.

```
10
11 public interface IServicioBicicletas {
12     @GET("/BicicletasServicio.svc/bicicletas/{idBici}")
13     Call<Bicicleta> obtenerBicicleta(@Path("idBici")String idBici);
14 }
```

**Código 2.5.** Interface IServicioBicicletas

### 2.2.1.5 Interfaz gráfica Android Studio

La interfaz gráfica está codificada en archivos XML (Anexo G), los cuales se encuentran en la carpeta res autogenerada por Android Studio. En esta carpeta se pueden apreciar diferentes subcarpetas como drawable, menu o values. Sin embargo, la que más se debe detallar es la carpeta layout debido a que ésta contiene los controles de la interfaz gráfica predicha en la etapa de diseño.

En la carpeta layout se encuentran archivos XML que describen la interfaz gráfica de cada activity o fragment. En el Código 2.6 se muestra un fragmento de código XML que describe la interfaz gráfica para la activity de login LoginActivity. Los archivos XML del proyecto de Android Studio comienzan con un tag descriptor de la versión XML y la codificación. De este modo, la línea 1 indica que se está empleando la versión 1.0 y la codificación utf-8. El segundo tag (líneas 2 hasta 7) define el layout sobre el cual estarán los componentes de la interfaz gráfica, en este caso el layout seleccionado ha sido el ConstraintLayout, además se especifican otros parámetros de inicialización del layout.

En la línea 9 hasta la línea 23, se describe el componente username del tipo EditText. Dentro de la etiqueta EditText se especifican atributos que permiten identificar el componente, su posición y otras propiedades adicionales. El atributo layout\_marginStart de la línea 13 especifica que el componente empezará a 25dp del margen izquierdo del layout. Los atributos de las líneas 20-23 son mandatorias para un componente que esté dentro de la layout ConstraintLayout, estas propiedades se utilizan para describir la restricción que tiene el componente respecto de otros o de la layout. La línea 20, especifica que la restricción superior será el límite de la layout.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".ui.login.LoginActivity">
8
9      <EditText
10         android:id="@+id/username"
11         android:layout_width="0dp"
12         android:layout_height="wrap_content"
13         android:layout_marginStart="24dp"
14         android:layout_marginTop="216dp"
15         android:layout_marginEnd="24dp"
16         android:hint="Email"
17         android:inputType="text"
18         android:selectAllOnFocus="true"
19         android:text="beleni"
20         app:layout_constraintEnd_toEndOf="parent"
21         app:layout_constraintHorizontal_bias="0.0"
22         app:layout_constraintStart_toStartOf="parent"
23         app:layout_constraintTop_toTopOf="parent" />

```

**Código 2.6.** Fragmento de código del archivo activity\_login.xml

De la misma manera como se describen layouts, es posible también describir componentes personalizados, tales como un menú o una fila para un ListView. En el Código 2.7 se muestra un componente del tipo menú que podrá ser utilizado dentro de una layout o dentro de otro componente. Los componentes del tipo menú se manejan a través de ítems por sí solos o a través de grupos de ítems.

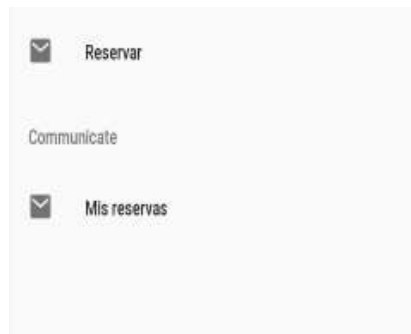
```

1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      tools:showIn="navigation_view">
5
6      <group android:checkableBehavior="single">
7          <item
8              android:id="@+id/nav_message"
9              android:icon="@drawable/ic_message"
10             android:title="Reservar" />
11      </group>
12
13      <item android:title="Communicate">
14          <menu>
15              <item
16                  android:id="@+id/nav_share"
17                  android:icon="@drawable/ic_message"
18                  android:title="Mis reservas" />
19          </menu>
20      </item>
21
22  </menu>

```

**Código 2.7.** Código XML del archivo drawer\_menu.xml

La interfaz gráfica se observa en la Figura 2.31:



**Figura 2.31.** Diseño de Menu

### Mostrar Mapa

Para mostrar un mapa en la aplicación móvil, se utiliza el servicio de mapas de Google. Primero se agrega un fragment en el layout asociado a la activity donde se muestra el mapa y las estaciones tanto en la reserva como en la devolución de la bicicleta. Como se observa en el Código 2.8.

```
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
  
<fragment  
    android:id="@+id/map"  
    android:name="com.google.android.gms.maps.SupportMapFragment"  
    android:layout_width="400dp"  
    android:layout_height="550dp"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintHorizontal_bias="0.5"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/imageViewEstacion"  
    tools:context=".ui.map_estaciones.MapsActivity" />
```

**Código 2.8.** Fragmento para mapa de layout

Y en la actividad asociada a esta pantalla, se llama a este recurso a través del método setContentView como se observa en la línea 60 del Código 2.9:

```
58  
59  
60  
  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_maps);  
}
```

**Código 2.9.** Actividad de mapa

En el archivo AndroidManifest.xml, en la línea 9 y 10 del Código 2.10 se incluye permiso para que la aplicación tenga acceso a una ubicación aproximada y la ubicación precisa respectivamente. En la línea 11 se habilita el permiso para que la aplicación pueda abrir sockets de red y en línea 41-43 del Código 2.11 se registra la referencia de la clave del API de Google maps.

```
9  
10  
11  
12  
  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />  
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

**Código 2.10.** Permisos de la aplicación

```

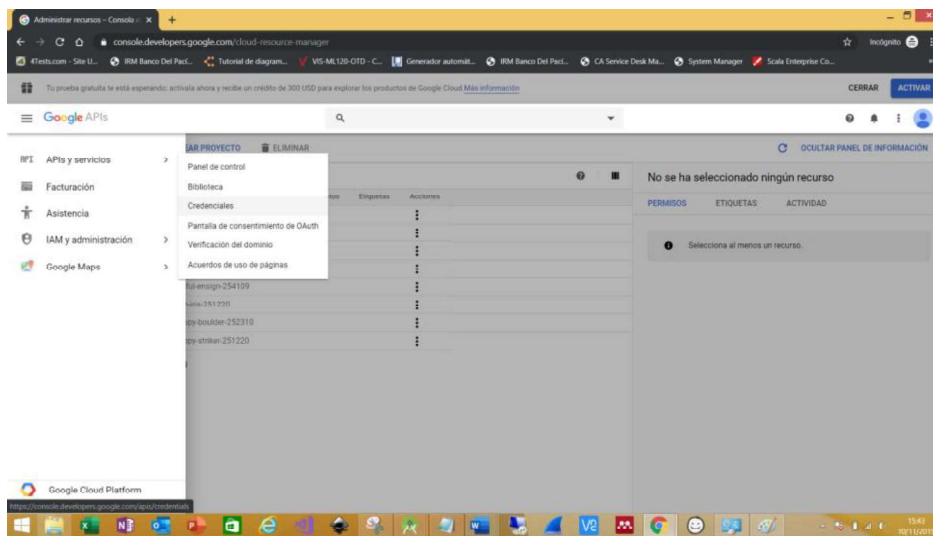
41 <meta-data
42     android:name="com.google.android.geo.API_KEY"
43     android:value="@string/google_maps_key" />

```

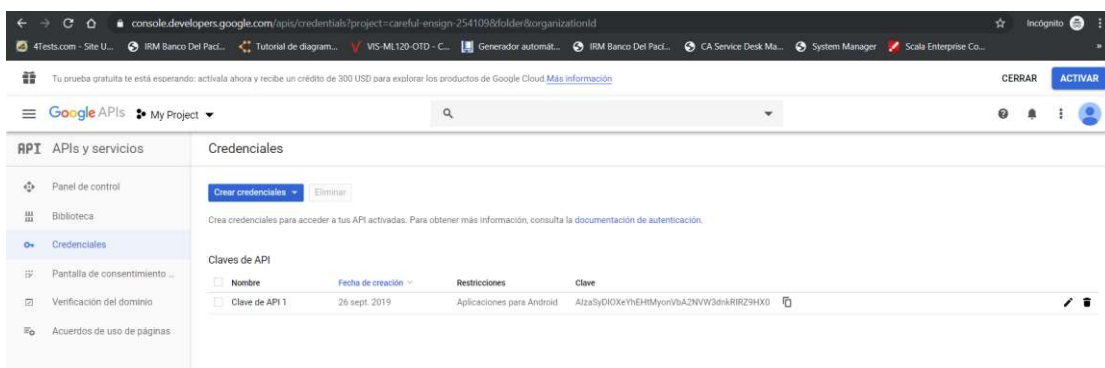
**Código 2.11.** Credenciales para acceder al mapa

Acto seguido, se debe generar la clave para almacenarla en el string `Google_maps_key`, siguiendo lo siguiente: Ingresar en <https://console.developers.google.com/project> con las credenciales de Google y asignar un nombre al proyecto.

Luego, se escoge el botón `Credenciales` de la sección `APIs y credenciales`. Y se las genera al seleccionar `Clave de API` en la sección `Crear credenciales`:

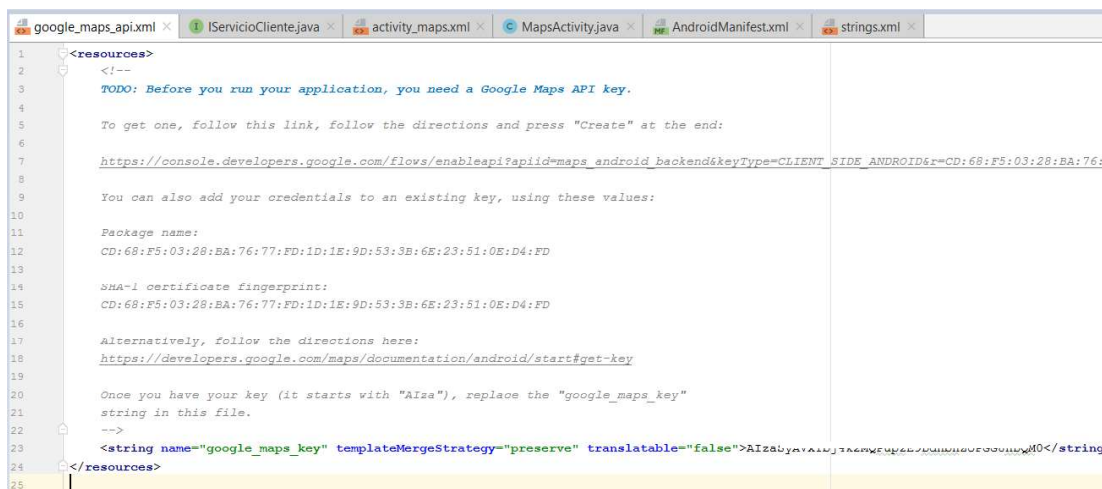


**Figura 2.32.** Sección APIs y servicios



**Figura 2.33.** Selección de credenciales

Para concluir este proceso, en línea 23 de Código 2.12 se inserta la clave generada en el archivo xml correspondiente a Google\_maps\_api que fue generado automáticamente:



```
1 <resources>
2 <!--
3  TODO: Before you run your application, you need a Google Maps API key.
4
5  To get one, follow this link, follow the directions and press "Create" at the end:
6
7  https://console.developers.google.com/flows/enableapi?apiid=maps\_android\_backend&keyType=CLIENT\_SIDE\_ANDROID&r=CD:68:F5:03:28:BA:76:77
8
9  You can also add your credentials to an existing key, using these values:
10
11  Package name:
12  CD:68:F5:03:28:BA:76:77:FD:1D:1E:9D:53:3B:6E:23:51:0E:D4:FD
13
14  SHA-1 certificate fingerprint:
15  CD:68:F5:03:28:BA:76:77:FD:1D:1E:9D:53:3B:6E:23:51:0E:D4:FD
16
17  Alternatively, follow the directions here:
18  https://developers.google.com/maps/documentation/android/start#get-key
19
20  Once you have your key (it starts with "AIza"), replace the "google_maps_key"
21  string in this file.
22  -->
23  <string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">AIzaSyA1A1J3A6KwE66000000000000000000000</string>
24 </resources>
25
```

**Código 2.12.** Clave de API de Google

### 2.2.1.6 Codificación de la Aplicación administrador

Para la codificación de la aplicación administrador (Anexo L), se ha dividido el código Java en dos subcarpetas principales, datos y ui (user interface). En esta aplicación se ha intentado mejorar el uso de los View-Models, antes se creaba un View-Model por cada activity o fragment, ahora se tiene un View-Model por cada objeto del dominio. En esta aplicación se ha trabajado mucho con el concepto de fragments por encima de las activities. También, se han utilizado controles que funcionan como componentes reutilizables (cuadros de diálogo, menús que van sobre las barras de acción o layouts personalizadas) que están ligados a una activity y aparecen en todos los fragments. Además, en esta aplicación se ha utilizado muchos más recursos gráficos personalizados y estandarizados que han permitido obtener una mejor interfaz de usuario.

Esta aplicación inicia con una activity para el login de usuario y luego pasa a una activity principal. Esta activity principal está poblada por una barra de navegación superior, una barra de navegación inferior y en la parte central se tiene un componente FrameLayout que permite que, a través de un gestor de fragments, se reemplace dinámicamente un fragment por otro, sin abandonar la activity.

El Código 2.13 muestra un fragmento del archivo activity\_main.xml el cual alberga al componente FrameLayout. Como se puede apreciar, las propiedades son las mismas que cualquier otro componente, la diferencia es que este componente sirve de contenedor para otros layouts del tipo fragment.

```

14 <FrameLayout
15     android:id="@+id/fragment_container"
16     android:layout_width="0dp"
17     android:layout_height="0dp"
18     android:layout_marginStart="8dp"
19     android:layout_marginEnd="8dp"
20     app:layout_constraintBottom_toTopOf="@+id/bnav_view"
21     app:layout_constraintEnd_toEndOf="parent"
22     app:layout_constraintHorizontal_bias="0.29"
23     app:layout_constraintStart_toStartOf="parent"
24     app:layout_constraintTop_toBottomOf="@+id/appBar" />

```

**Código 2.13.** Componente FrameLayout

Como se mencionó anteriormente, la actividad principal es la encargada de gestionar los fragments. Para ello, una actividad que herede de AppCompatActivity posee el método `getSupportFragmentManager`, el mismo que permite reemplazar el contenido del contenedor `FrameLayout` de acuerdo a las acciones del usuario. En el Código 2.14 se muestra cómo se sobrescribe el método `onNavigationItemSelectedListener` para que sea este método el que maneje los eventos generados por los botones de navegación. La línea 48 indica que en el caso de que se dé clic sobre el componente de id, `navigation_reservas`, se utiliza el gestor de fragmentos para reemplazar cualquier contenido que este ocupando el componente `FrameLayout` por el fragment `ReservasFragment` (líneas 49 y 50).

```

44 @Override
45 public boolean onNavigationItemSelectedListener(@NonNull MenuItem item) {
46     bnavView.getMenu().setGroupCheckable( group: 0, checkable: true, exclusive: true);
47     switch (item.getItemId()) {
48         case R.id.navigation_reservas:
49             getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container,
50                 new ReservasFragment()).commit();
51             setHintSearchView("Reservas");
52             return true;
53         case R.id.navigation_bicicletas:
54             getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container,
55                 new BicicletasFragment()).commit();
56             setHintSearchView("Bicicletas");
57             return true;
58         case R.id.navigation_clientes:
59             getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container,
60                 new ClientesFragment()).commit();
61             setHintSearchView("Clientes");
62             return true;

```

**Código 2.14.** Fragmento de código del archivo MainActivity.java

### 2.2.1.7 Codificación del servicio web WCF

Para la implementación del servicio web (Anexo P) se ha utilizado el lenguaje de programación C# sobre el framework WCF. La implementación se ha realizado siguiendo los lineamientos de diseño de OpenAPI y se los ha mezclado con los conceptos de servicio que ofrece WCF, es decir que para cada objeto del dominio de negocio exista un servicio



el cual esté atado a un conjunto de endpoints relacionados al mismo objeto de dominio de negocio. Por otra parte, la comunicación entre base de datos y servidor ha sido muy ligera gracias a la ayuda de las bibliotecas FirebaseDotNet y FireSharp.

### Archivo de configuración

Primero, se debe especificar que la aplicación será del tipo RESTful y que no se utilizarán conexiones seguras (HTTPS). Para ello se manipula el archivo web.config (Anexo Q) se añaden las líneas 27-29, tal como se muestra en el Código 2.15. Además, con el objetivo de depurar de manera más sencilla se activan las excepciones web en la línea 18 del Código 2.15.

```
11 <system.serviceModel>
12 <behaviors>
13 <serviceBehaviors>
14 <behavior>
15 <!-- Para evitar revelar información de los metadatos, establezca los valores siguientes en false antes
16 <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
17 <!-- Para recibir detalles de las excepciones en los fallos, con el fin de poder realizar la depuración
18 <serviceDebug includeExceptionDetailInFaults="true"/>
19 </behavior>
20 </serviceBehaviors>
21 <endpointBehaviors>
22 <behavior>
23 <webHttp/>
24 </behavior>
25 </endpointBehaviors>
26 </behaviors>
27 <protocolMapping>
28 <add binding="webHttpBinding" scheme="http"/>
29 </protocolMapping>
30 <serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true"/>
```

**Código 2.15.** Fragmento de código del Archivo web.config

### Servicio WCF: interfaces y métodos manejadores

Para cada objeto del dominio de negocio existirá un servicio WCF, así, para el objeto cliente existirá un servicio WCF (ClientesServicio) que sea el encargado de manipular tal objeto en la comunicación cliente-base de datos o viceversa. Un servicio WCF está conformado por una interfaz donde se detallan los parámetros de conexión HTTP, para exposición del endpoint, y una clase donde se implementan los métodos de la interfaz que son los manejadores de cada endpoint. Los métodos manejadores son los encargados de procesar las peticiones que llegan a cada endpoint y responder o no de forma adecuada.

El Código 2.16 muestra un fragmento del archivo IClientesServicio.cs en el cual se codifica una interfaz de nombre IClientesServicio (línea 10). Este archivo está compuesto por dos partes, la primera el ServiceContract y el DataContract. ServiceContract contiene las firmas de los métodos ancladas a dos decoradores Operation Contract y WebInvoke, este último permite especificar parámetros HTTP. En la línea 13 y 14 del Código 2.16 se especifica que se utilizará el método GET HTTP para acceder a la URI /clientes y que el formato de

las respuestas será de tipo JSON. El endpoint antes descrito será atendido por el método ObtenerClientes (línea 15) que no recibe ningún parámetro de entrada y retorna una Lista de clientes envuelta en la clase Task para que pueda ser tratada como una respuesta asíncrona.

Así, a través del cambio de parámetros en el decorador WebInvoke es posible especificar diferentes URI y métodos HTTP, como formatos de respuesta. Por ejemplo, la línea 19 del Código 2.16 especifica un parámetro de entrada de la petición que vendrá embebido en la URI de la petición y será tomado como parámetro de entrada del método ObtenerCliente (línea 20). Otro ejemplo se puede apreciar en la línea 28 del código en cuestión, que muestra cómo se especifica otro método HTTP (PUT) para una misma URI (línea 29 igual a la línea 14). De la misma forma es posible manejar parámetros query y asociarlos como parámetros de entrada de un método manejador.

```
10 public interface IClientesServicio
11 {
12     [OperationContract]
13     [WebInvoke(Method = "GET", ResponseFormat = WebMessageFormat.Json,
14             UriTemplate = "/clientes")]
15     Task<List<Cliente>> ObtenerClientes();
16
17     [OperationContract]
18     [WebInvoke(Method = "GET", ResponseFormat = WebMessageFormat.Json,
19             UriTemplate = "/clientes/{idCliente}")]
20     Task<Cliente> ObtenerCliente(string idCliente);
21
22     [OperationContract]
23     [WebInvoke(Method = "POST", ResponseFormat = WebMessageFormat.Json,
24             UriTemplate = "/clientes/nuevo")]
25     Task<Cliente> AgregarCliente(Cliente cliente);
26
27     [OperationContract]
28     [WebInvoke(Method = "PUT", ResponseFormat = WebMessageFormat.Json,
29             UriTemplate = "/clientes")]
30     Task<bool> ModificarCliente(Cliente cliente);
31
32     [OperationContract]
33     [WebInvoke(Method = "DELETE", ResponseFormat = WebMessageFormat.Json,
34             UriTemplate = "/clientes/{idCliente}")]

```

**Código 2.16.** Fragmento del archivo IClientesServicio.cs (ServiceContract)

### Interfaz

Las interfaces de servicio, además de definir los endpoints y sus métodos manejadores, también definen las clases que describen los objetos del dominio de negocio. DataContract es el decorador que indica que una clase será utilizada en el procesamiento de alguna solicitud HTTP. En el Código 2.17 se muestra cómo se define una clase, sus atributos, getters y setters. El decorador DataMember se utiliza para corresponder los nombres de los atributos con el nombre JSON que se utilizará en las solicitudes HTTP.

```

48 [DataContract]
49     31 referencias
    public class Cliente
50     {
51         string id;
52         string nombre;
53         string correoElectronico;
54         string direccion;
55         string telefono;
56         string cedula;
57         string usuario;
58         string passw;
59         string rfid;
60         string rol;
61
62         [DataMember(Name = "id")]
63         12 referencias
        public string Id { get => id; set => id = value; }
64         [DataMember(Name = "nombre")]
65         3 referencias
        public string Nombre { get => nombre; set => nombre = value; }
66         [DataMember(Name = "correoElectronico")]
67         3 referencias
        public string CorreoElectronico { get => correoElectronico; set => correoElectronico = value; }
68         [DataMember(Name = "telefono")]
69         3 referencias
        public string Telefono { get => telefono; set => telefono = value; }
70         [DataMember(Name = "cedula")]

```

**Código 2.17.** Fragmento del archivo ICientesServicio.cs (DataContract)

### Clase que implementa la interfaz

Otro archivo importante que compone el servicio para un objeto es la clase que implementa la interfaz del servicio. El Código 2.18 muestra un fragmento de código la clase ClientesServicio que implementa la interfaz ICientesServicio, en esta clase se implementa todo el procesamiento necesario de la información receptada de la solicitud HTTP, se genera una serie de acciones y de ser necesario una respuesta. Específicamente, el Código 2.18 muestra el método AgregarCliente que recibe como parámetro un objeto cliente y regresa otro objeto cliente. Este método funciona como manejador de una solicitud HTTP, en este caso de la solicitud apuntada hacia /clientes/nuevo (según se indica en las líneas 19 y 20 del Código 2.16). Una vez que se ha recibido una solicitud esta será procesada para validar la información e insertarla en la base de datos, tal y como se muestra en las líneas 35-46 del Código 2.18. La validación de la información se la realiza en el servidor (líneas 35-41) con el objetivo de verificar el contenido de cada solicitud antes de ser ingresado a la base de datos. Las líneas 44-46 del Código 2.18 hacen uso de la biblioteca FirebaseDotNet para crear una solicitud asíncrona hacia la base de datos y en este caso, insertar el registro. Si el registro se ha insertado correctamente, este será retornado como objeto, tal como se muestra en la línea 48 del Código 2.18.

```

27 public async Task<Cliente> AgregarCliente(Cliente cliente)
28 {
29     try
30     {
31         //Política de datos
32         if (cliente.Id != null) throw new Exception();
33         cliente.Id = Utils.Utils.RandomString(5, true) + (await ObtenerNumeroClientes() + 1);
34
35         if (cliente.Nombre == null) throw new Exception();
36         if (cliente.Cedula == null) throw new Exception();
37         if (cliente.CorreoElectronico == null) throw new Exception();
38         if (cliente.Direccion == null) throw new Exception();
39         if (cliente.Usuario == null) throw new Exception();
40         //if (cliente.Passw == null) throw new Exception();
41         if (cliente.Telefono == null) throw new Exception();
42
43         //Inserción sobre Firebase
44         var clienteDb = await clienteFirebase
45             .Child("clientes")
46             .PostAsync(cliente, false);
47
48         return clienteDb.Object;
49     }
50     catch (Exception e)
51     {
52         throw new FaultException("Error occurred while saving data...");
53     }
54 }

```

**Código 2.18.** Fragmento del archivo ClientesServicio.svc.cs

Por otra parte, los métodos manejadores de las solicitudes HTTP pueden ser utilizados por otros métodos manejadores sin necesidad de hacer una solicitud HTTP. Por ejemplo, en el Código 2.19 se muestra cómo dentro del método (manejador) retirarBici que se encarga de realizar el retiro de la bicicleta a través del RFID del cliente utiliza el método (manejador) ObtenerClienteByRfid para identificar al cliente que desea hacer el retiro como se observa en la 449.

```

444 public async Task<BiciCandado> retirarBici(string rfidCliente)
445 {
446     BiciCandado transaccion = new BiciCandado();
447
448     ClientesServicio clientesServicio = new ClientesServicio();
449     Cliente cliente = await clientesServicio.ObtenerClienteByRfid(rfidCliente);
450
451     if (cliente is null)
452     {
453         transaccion.Error = "No se ha encontrado un cliente con el RFID leído.";
454         transaccion.StatusEntregaRecepcion = false;
455         return transaccion;
456     }
457
458     ReservasServicio reservasServicio = new ReservasServicio();
459     Reserva reserva = await reservasServicio.ObtenerReservaActiva(cliente.Id);
460
461     if (reserva == null || !reserva.Activa)
462     {
463         transaccion.Error = "El cliente no registra Reservas activas.";
464         transaccion.StatusEntregaRecepcion = false;
465         return transaccion;
466     }

```

**Código 2.19.** Fragmento del archivo BicisCandadosServicio.svc.cs

### 2.2.1.8 Migración de servicio web a nube de AWS

Se alquila un servidor AWS que funcione como máquina virtual y que posea salida a la web. AWS ofrece una herramienta llamada Amazon Elastic Compute Cloud (Amazon EC2), que ofrece servidores virtuales en la nube de AWS con un sistema operativo

predeterminado. Cuando se elige un sistema operativo, se crea una instancia de ese sistema operativo. Amazon EC2 ofrece una serie de características que permiten realizar conexiones seguras y definir políticas de acceso.

A continuación, se detalla el proceso para lanzar una instancia de Windows Server. Cabe recalcar que se utilizó una cuenta estándar AWS con suscripción gratuita, esto limita las características de la instancia. También, se ha utilizado directamente la documentación AWS para lanzar una nueva instancia[42]:

- El primer paso es abrir la consola EC2, ver Figura 2.34:

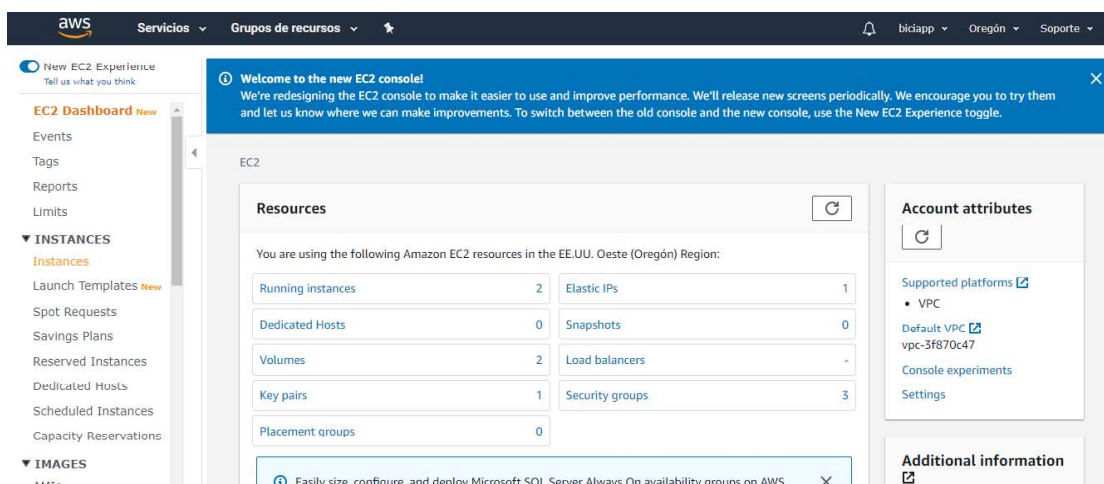


Figura 2.34. Consola de EC2

- En la barra de costado elegir Instancias y luego sobre el botón azul “Launch Instance”, clic para empezar a crear una nueva instancia, ver Figura 2.35:

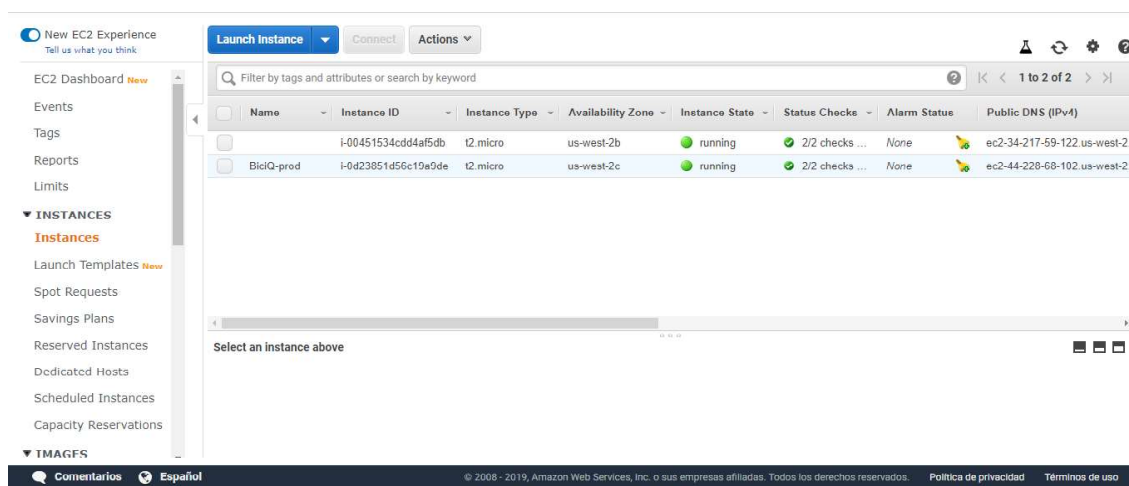
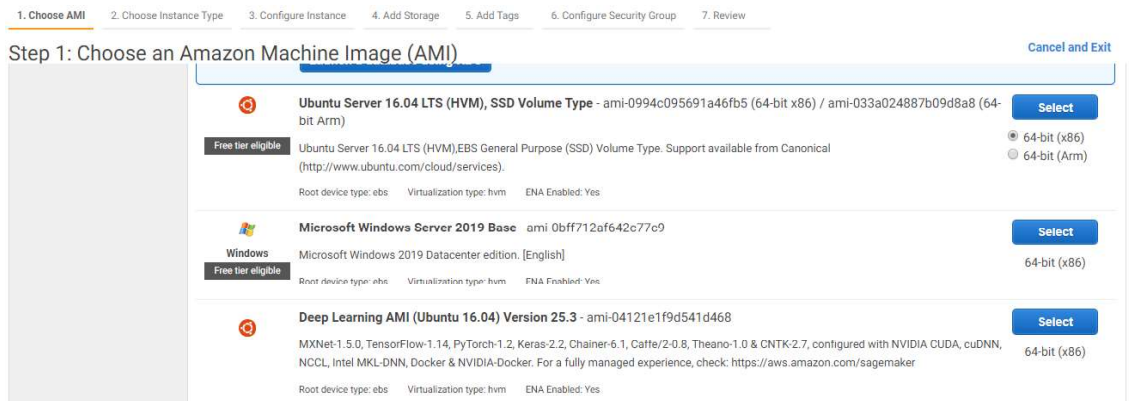


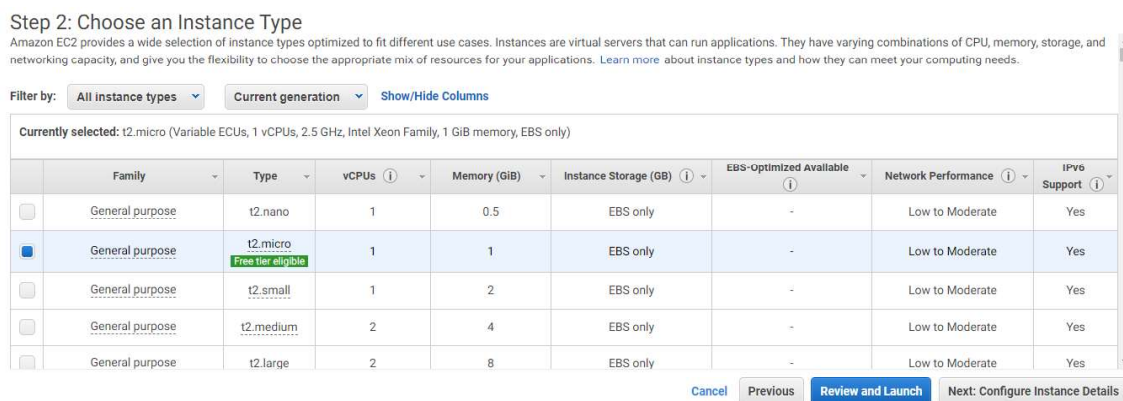
Figura 2.35. Instancias del usuario

- Elegir una plantilla AMI (Amazon Machine Image) que se acople a los requerimientos necesarios. En este caso necesitamos un servidor gratuito Windows Server 2019, ver Figura 2.36:



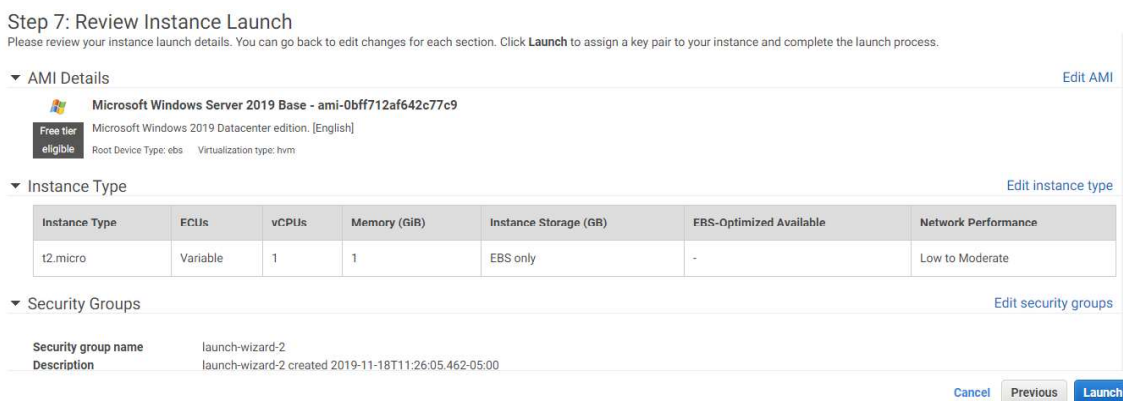
**Figura 2.36.** Plantilla AMI

- En Figura 2.37, se elige el tipo de instancia, es decir el número de CPUs virtuales, memoria RAM, entre otras cosas. Una opción sin costo es la instancia de 1GB de memoria. Luego de seleccionar tal instancia, se elige el botón “Review and Launch”.



**Figura 2.37.** Tipo de instancia

- Se muestra un resumen de lo que se ha elegido anteriormente. Políticas de seguridad, memoria de almacenamiento y más detalles de la instancia son mostrados para ser revisados en Figura 2.38. Por último, se lanza la instancia a través del botón “Launch”.



**Figura 2.38.** Enviar instancia – botón Launch

- En Figura 2.39 se selecciona un par de claves que se utilizarán para acceder a la máquina virtual. Sino se cuenta con un par de claves, entonces en la misma pantalla se puede generar un par a través de un archivo.pem.

**Select an existing key pair or create a new key pair** ✕

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Choose an existing key pair ▼

**Select a key pair**

claves\_aws\_biciapp ▼

I acknowledge that I have access to the selected private key file (claves\_aws\_biciapp.pem), and that without this file, I won't be able to log into my instance.

Cancel Launch Instances

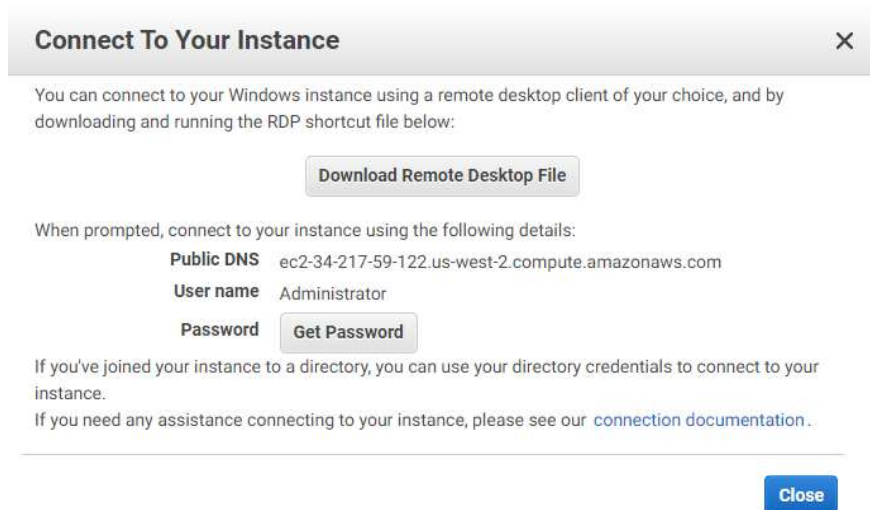
**Figura 2.39.** Clave de acceso a máquina virtual

- Una vez creada la instancia, esta aparecerá en la consola EC2, en la pestaña instancias. En Figura 2.40 se podrá apreciar entre otras cosas su IP pública.

Instance ID	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key Name	Monitoring	Launch Time
s...	None	ec2-34-217-59-122.us-west-2.compute.amazonaws.com	34.217.59.122	-	claves_aws_bi...	disabled	Novemb
s...	None	ec2-44-228-68-102.us-west-2.compute.amazonaws.com	44.228.68.102	-		disabled	Novemb

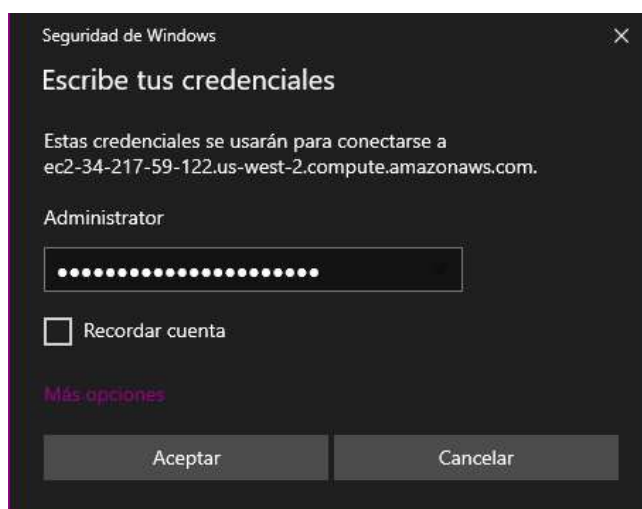
**Figura 2.40.** IP pública de servicio

- Para conectarse a la instancia se selecciona la misma y se elige conectar. A continuación, se mostrará un aviso para descargar un archivo .rdp (archivos asociados a escritorio remoto) y se deberá generar una contraseña mediante el archivo.pem de la Figura 2.41.



**Figura 2.41.** Conectar a instancia

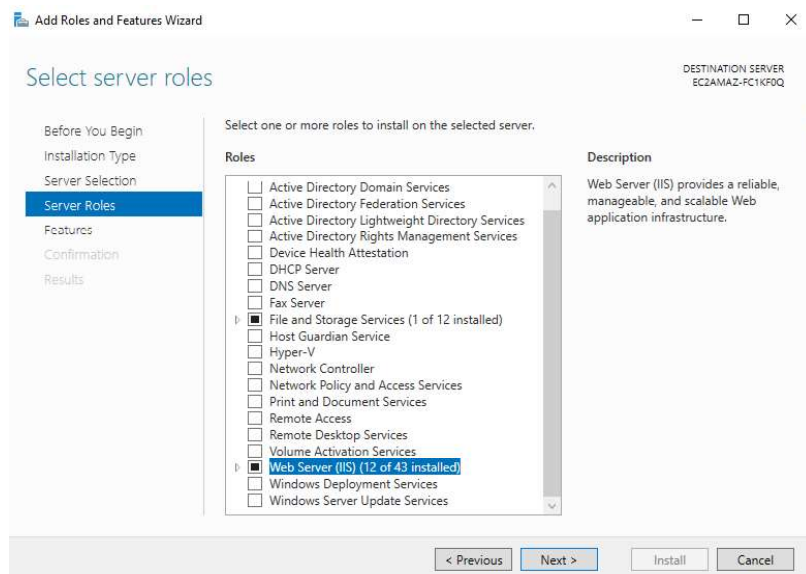
- Con la contraseña y el usuario “Administrator” se procede a abrir el archivo, ver Figura 2.42:



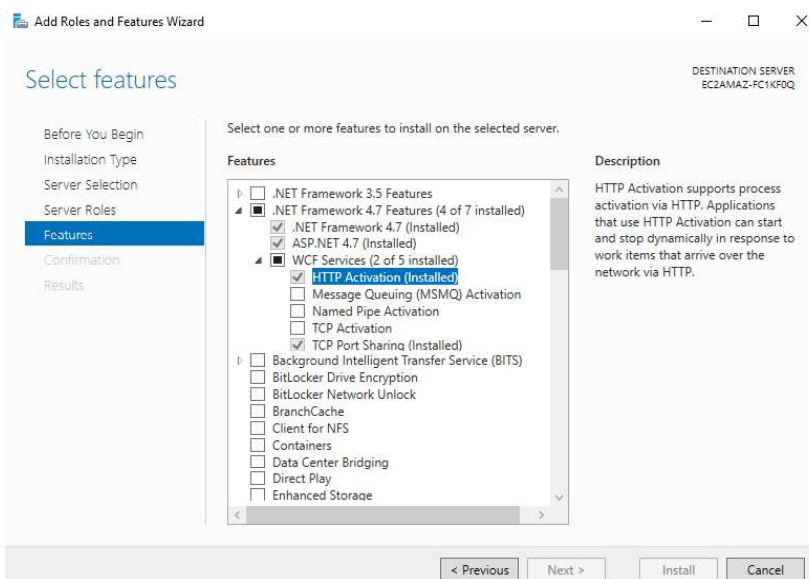
**Figura 2.42.** Credenciales

Una vez conectado a la máquina virtual se procede a instalar el servidor IIS. También es necesario que esté instalado el framework .NET 4.7 y activada la característica “HTTP Activation”, ver Figura 2.43. Todo esto se realiza desde el Panel de Control -> Programas -> Activar/Desactivar características de Windows.



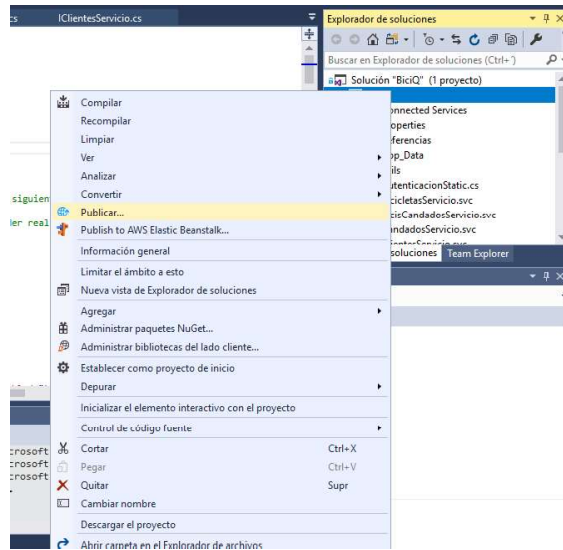


**Figura 2.43. Activar Web Server IIS**



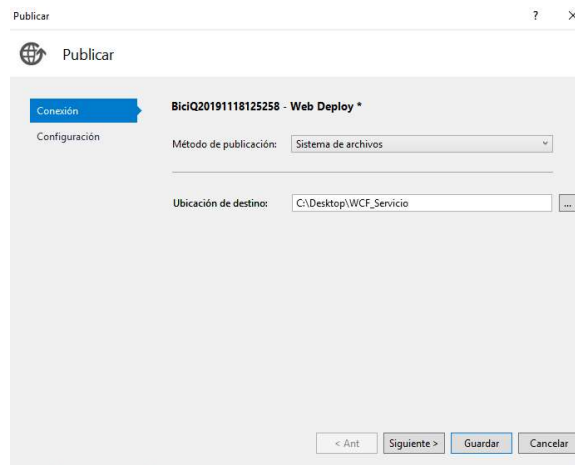
**Figura 2.44. Activar características HTTP**

- Una vez instalado el servidor IIS y activado las características necesarias. Entonces se procede a publicar el proyecto de Visual Studio como un sistema de archivos, ver Figura 2.45. Para ello, en la máquina local, en el Explorador de Soluciones de Visual se selecciona el proyecto que se desea desplegar, clic derecho y Publicar.



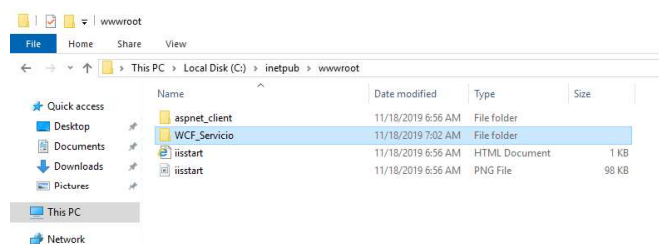
**Figura 2.45.** Publicar servicio

- Se elige como método de publicación la opción Sistema de Archivos se elige una ubicación en el disco, ver Figura 2.46 y el método de publicación como Release. Esto creará un conjunto de archivos que serán colocados en el servidor remoto para que funcionen a través de IIS.



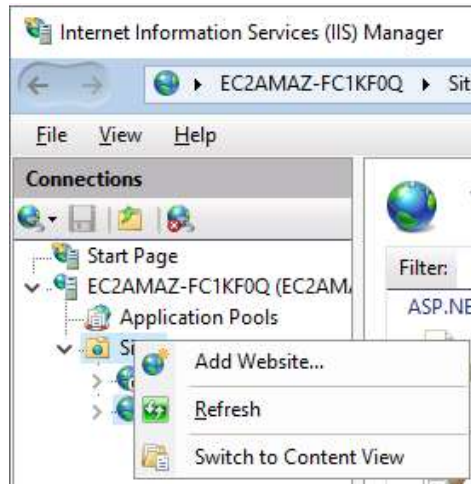
**Figura 2.46.** Ubicación de servicio

- Los archivos generados en el destino seleccionado deberán ser copiados en la carpeta wwwroot ubicada en la ubicación C:/inetpub de la máquina remota.



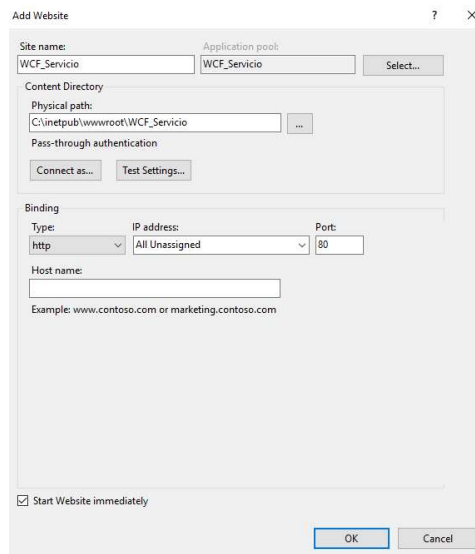
**Figura 2.47.** Carpeta wwwroot

- Luego en Figura 2.48, ya en la máquina remota, se abre el administrador IIS. Ahí se agrega un nuevo sitio y se selecciona la carpeta WCF\_Servicio.



**Figura 2.48.** Agregar sitio en IIS

- En Figura 2.49, se selecciona un nombre para el sitio web y se elige la carpeta WCF\_Servicio y las demás opciones por defecto.



**Figura 2.49.** Nombre para sitio web

- En Figura 2.50, se asegura que el sitio web este activo y se desactiva el sitio web por defecto. En la máquina remota prueba la conexión a través de localhost.

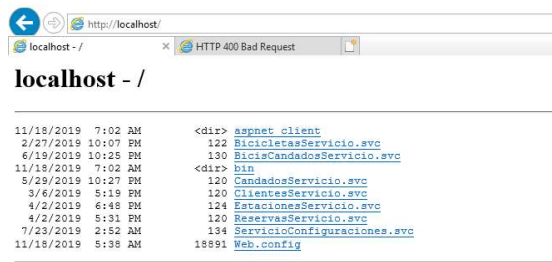


Figura 2.50. Prueba localhost

- Para tener salida a la web es necesario configurar la instancia WC2. Para ello se modifica el grupo de seguridad de la instancia, ver figura 2.51.

Filter by tags and attributes or search by keyword	IPv4 Public IP	IPv6 IPs	Key Name	Monitoring	Launch Time	Security Groups	Owner
nazonaws.com	34.217.59.122	-	claves_aws_bi...	disabled	November 18, 2019 at 1:40...	launch-wizard-1	372285223528
nazonaws.com	44.228.68.102	-	-	disabled	November 17, 2019 at 11:53...	awseb-e-bevdan3...	372285223528

Figura 2.51. Grupo de seguridad de instancia

- Se edita las reglas de entrada en Figura 2.52. Y se añade las direcciones IP que podrán conectarse a este servicio. Si se desea que no exista restricción entonces se añade una regla que permita todo el tráfico desde cualquier origen.

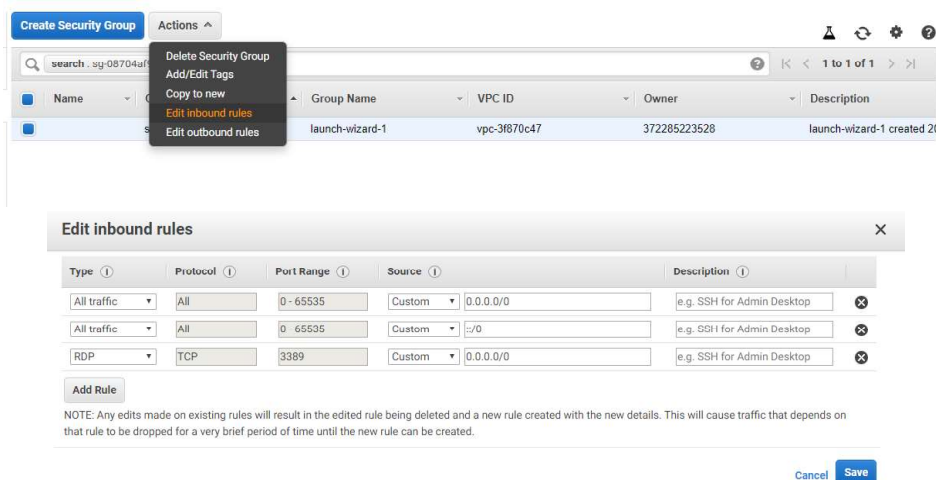


Figura 2.52. Lista de acceso -ACLs

- Se utiliza la dirección IP pública o el DNS público para conectarse al servicio, ver Figura 2.53.

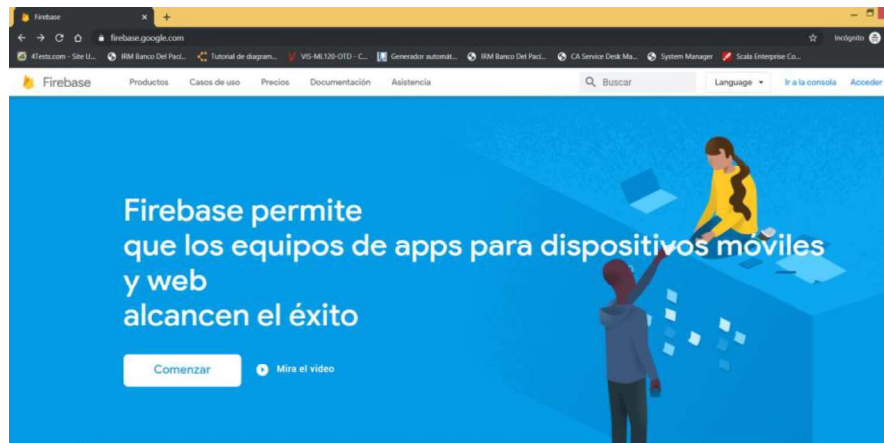
Public DNS (IPv4)	IPv4 Public IP
ec2-34-217-59-122.us-west-2.compute.amazonaws.com	34.217.59.122

Figura 2.53. IP pública del servicio

### 2.2.1.9 Base de Datos

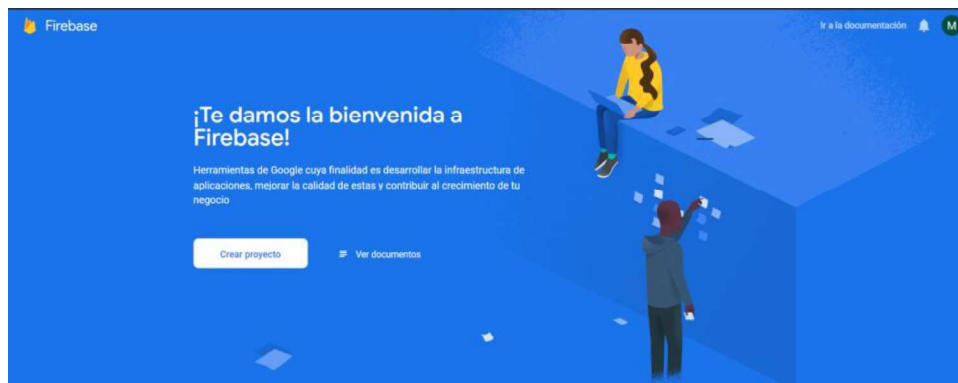
Ahora se van a revisar pasos importantes para construir la base de datos no relacional empleada en el presente proyecto.

Se ha utilizado Firebase y se accede a través de las credenciales de Google, se da clic en el botón Consola de la parte superior derecha para crear un nuevo proyecto como se observa en la Figura 2.54:

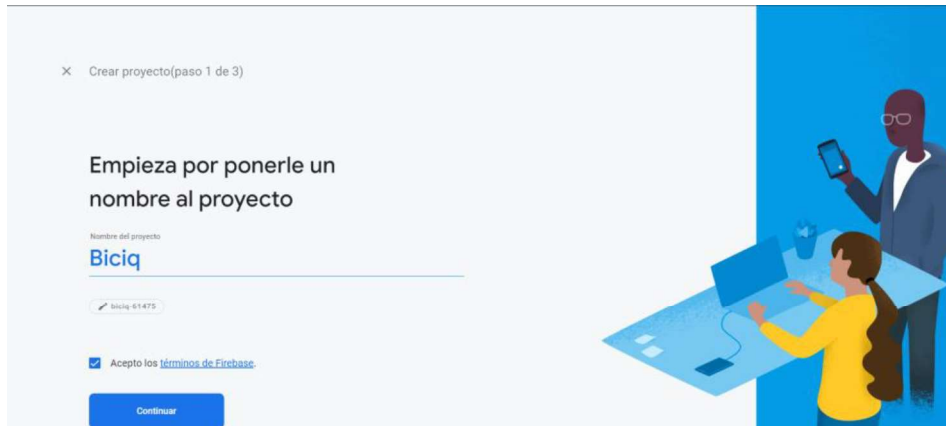


**Figura 2.54.** Pantalla de Inicio de Firebase

A continuación, en Figura 2.55 y Figura 2.56, se da clic en Crear Proyecto y se asigna un nombre a la base de datos

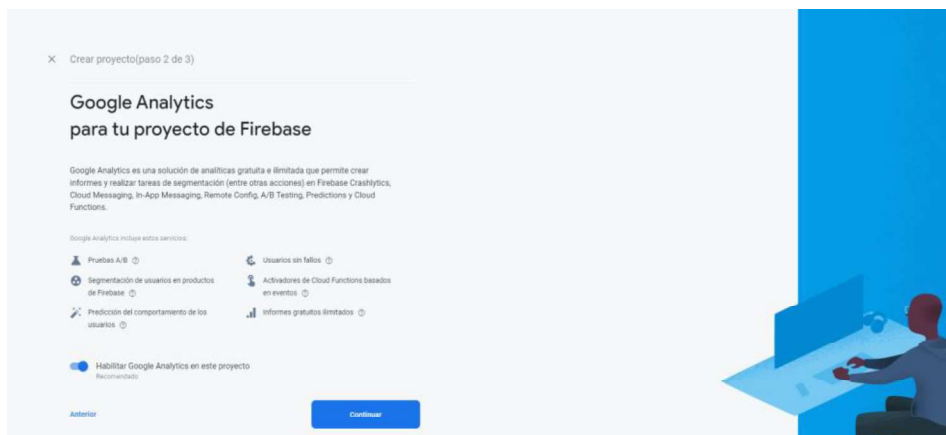


**Figura 2.55.** Interfaz para crear proyecto



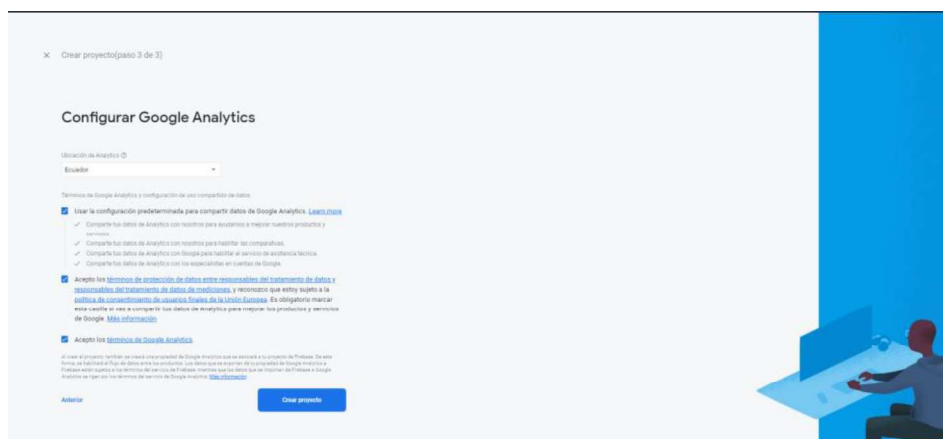
**Figura 2.56.** Ingreso de título de base de datos a crear

Luego, en Figura 2.57 se da clic en continuar:



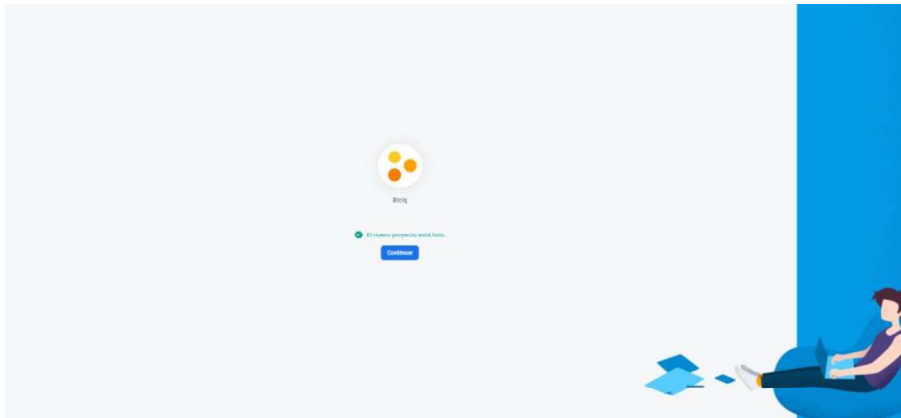
**Figura 2.57.** Activar análisis de Google

En Figura 2.58, se aceptan términos y condiciones:



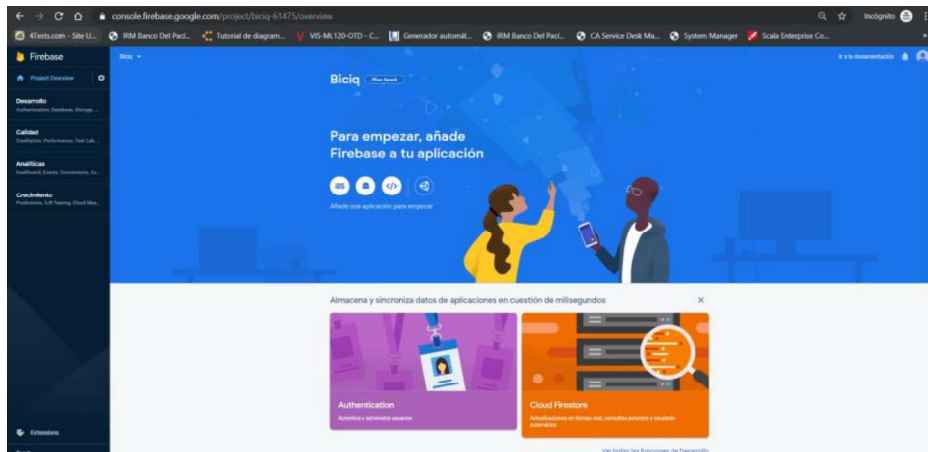
**Figura 2.58.** Paso final de creación de base de datos

Se espera un momento hasta que el proyecto esté listo, ver Figura 2.59:



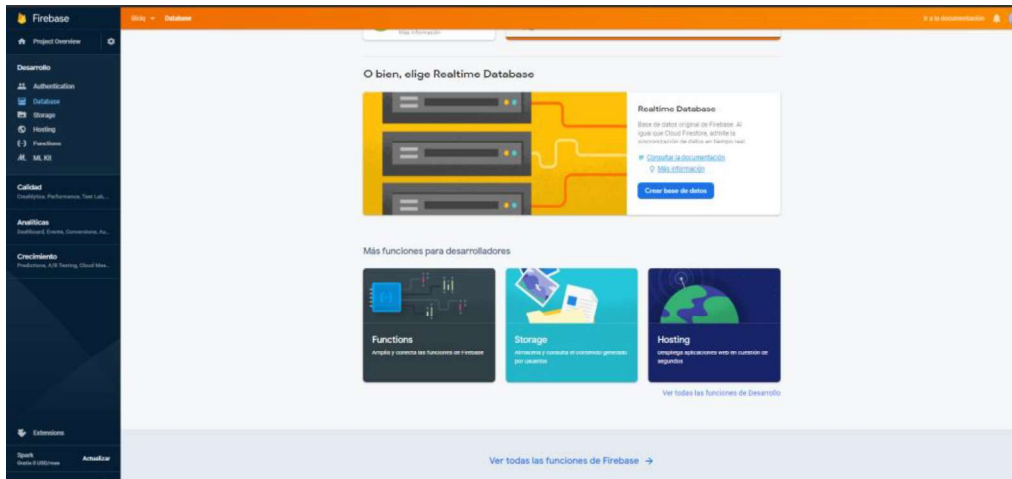
**Figura 2.59.** Proyecto listo

En Figura 2.60, se da clic en continuar y se visualizará en el extremo izquierdo superior la opción Desarrollo:



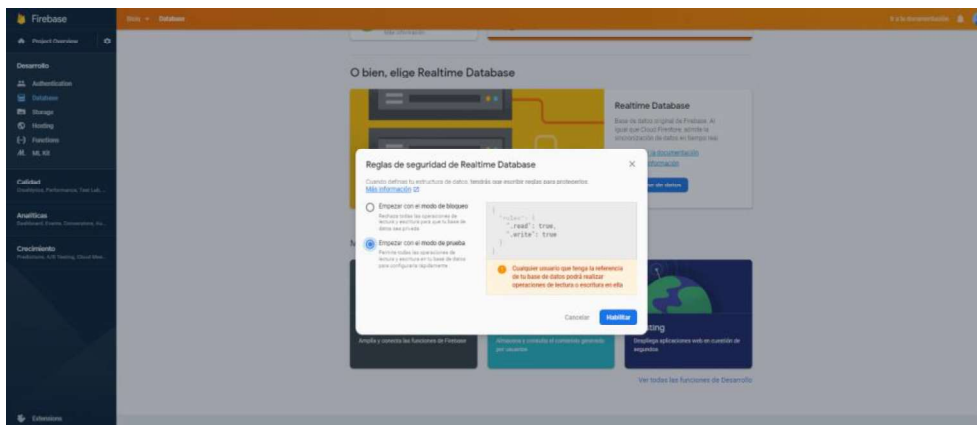
**Figura 2.60.** Interfaz de bienvenida

Y se despliegan las siguientes opciones, para crear la base de datos Firestore o Realtime DataBase, ver Figura 2.61:

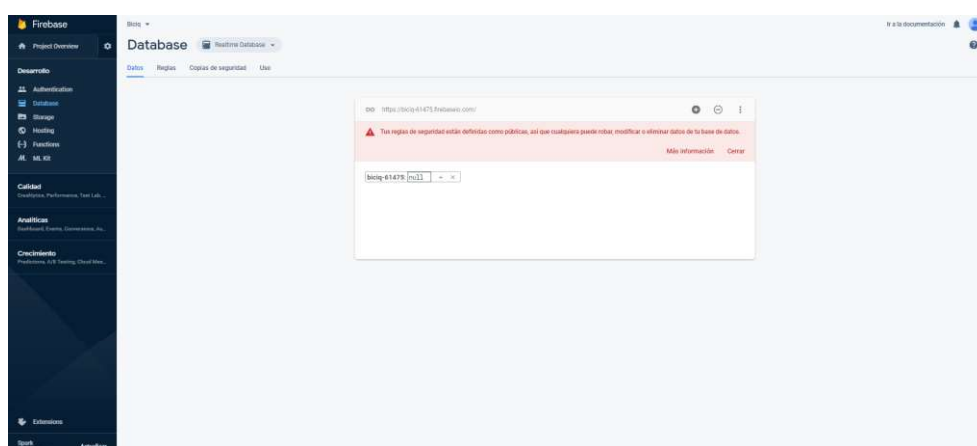


**Figura 2.61.** Realtime Database

En Figura 2.62 y Figura 2.63, se elige Crear base de datos - Realtime DataBase y aparece la siguiente ventana y se selecciona habilitar modo de Prueba:



**Figura 2.62.** Habilitar modo de prueba



**Figura 2.63.** Base de datos creada



En la sección superior Reglas se incluyen las reglas que normarán la base de datos por ejemplo en la línea 24 y 25 de la Figura 2.64 se observa que tiene permisos de lectura y escritura:

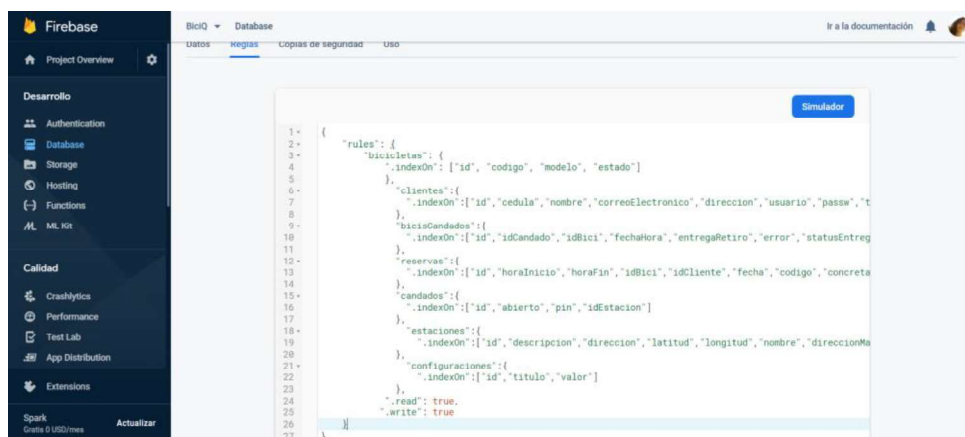


Figura 2.64. Reglas de la base de datos

## 2.2.2 Módulo de control

El módulo de control está compuesto por 1 Raspberry Pi, 1 cerradura electromagnética y 2 lectores RFID MRF522: para identificar al cliente y sensar la bicicleta.

### 2.2.2.1 Configurar Raspberry Pi

Para habilitar la Raspberry Pi se le debe instalar el sistema operativo.

Para ello se descargó el sistema operativo Raspbian y se lo hizo booteable para instalarlo en la tarjeta SD que se inserta en la Raspberry Pi.

Después, se ejecutan los siguientes comandos:

```
sudo apt-get update
```

```
sudo reboot
```

```
sudo apt-get upgrade
```

Luego se instala python-dev que es un paquete que permite construir extensiones de Python. Con el siguiente comando:

```
sudo apt-get install python-dev
```

Luego instalo gpio zero, con el comando siguiente:

```
sudo apt install python-gpiozero
```

### 2.2.2.2 Uso de Interfaz SPI

La Raspberry Pi se comporta como maestro y tiene 2 interfaces SPI para manejar dos esclavos. Para ello, los lectores comparten los pines de:

- Transmisión o salida de datos (Master In Slave Out)
- Entrada SPI al módulo RC522 (Master Out Slave In)
- Señal de reloj (Serial Clock)

Se diferencian los siguientes pines:

- Señal de recepción o entrada de datos que va conectada a los pines de interfaz SPI de la Raspberry Pi SPICE0 (pin 24) y SPICE1 (pin 26)
- Reset utilizados para restablecer y apagar el módulo: el lector de cliente usa el pin 22 y el otro lector de bicicletas usa el pin 11.

En la programación se definen dos scripts que tienen los mismos métodos y cuya única diferencia es el valor de la variable NRSTPD y la ruta de las interfaces SPI.

- **Script Cliente:** En la línea 31 del script MFRC522 de cliente se asigna el puerto 22 como pin de Reset y en la línea 131 se configura la dirección /dev/spidev0.0 como se observa en Código 2.20 y Código 2.21:

```
30 class MFRC522:
31     NRSTPD = 22
```

**Código 2.20.** Pin 22 - pin Reset de cliente

```
131 def __init__(self, dev='/dev/spidev0.0', spd=1000000):
132     spi.openSPI(device=dev, speed=spd)
133     GPIO.setmode(GPIO.BOARD)
134     GPIO.setup(self.NRSTPD, GPIO.OUT)
135     GPIO.output(self.NRSTPD, 1)
136     self.MFRC522_Init()
```

**Código 2.21.** Ruta de pin 24 - primera interfaz SPI de Raspberry Pi

- **Script Bicicleta:** En la línea 31 del script MFRC522bici de cliente se asigna el puerto 11 como pin de Reset y en la línea 131 se configura la dirección /dev/spidev0.1 como se observa en Código 2.22 y Código 2.23:

```
30 class MFRC522:
31     NRSTPD = 11
```

**Código 2.22.** Pin 11 - pin Reset de cliente

```
131 def __init__(self, dev='/dev/spidev0.1', spd=1000000):
132     spi.openSPI(device=dev, speed=spd)
133     GPIO.setmode(GPIO.BOARD)
134     GPIO.setup(self.NRSTPD, GPIO.OUT)
135     GPIO.output(self.NRSTPD, 1)
136     self.MFRC522_Init()
```

**Código 2.23.** Ruta de pin 26 – segunda interfaz SPI de Raspberry Pi

A nivel físico, la conexión es la que se muestra en la Figura 2.65:

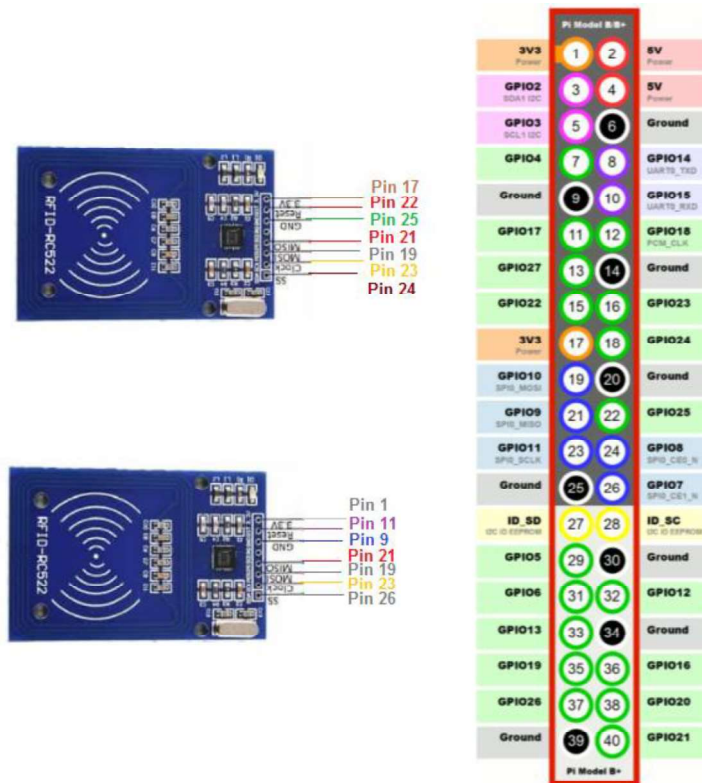


Figura 2.65. Conexión de lectores RFID a Raspberry Pi

### 2.2.2.3 Consumo de servicio web desde Python - Raspberry Pi

Para poder realizar el bloqueo y desbloqueo de la cerradura electromagnética, Python se comporta como un cliente que consume el servicio web WCF REST haciendo uso de los endpoints.

Para ello se utilizan las librerías: json, urllib y urllib2 como se observa en la línea 29, 30 y 31 del script principal que se observa en el Código 2.24.

```

29 import json
30 from urllib import urlencode
31 import urllib2

```

Código 2.24. Importe de librerías json, urllib y urllib2

Urllib2 es un módulo de Python que busca URLs utilizando el protocolo HTTP. HTTP se basa en solicitudes y respuestas: el cliente realiza solicitudes y el servidor envía respuestas. A continuación, en el Código 2.25 se observa un ejemplo: En la línea 55 de Código 2.25 se crea una función para realizar una solicitud get, en la línea 56 se define un objeto Request que especifica la URL a buscar. En línea 57 se llama al método urlopen

para abrir este objeto Request y en la línea 58 se devuelve un objeto de respuesta para la URL solicitada que puede ser leída a través del método read():

```
54 #GET
55 def http_get(url):
56     req = urllib2.Request(url)
57     response = urllib2.urlopen(req)
58     return response.read()
```

### **Código 2.25.** Función que implementa solicitud HTTP GET

Existe otro tipo de solicitud denominada POST. Como se ve en el Código 2.26, se observa el caso en el que se desea realizar una solicitud para enviar datos en la URL o realizar una solicitud POST que usualmente se usa cuando uno realiza un login de una página, por ejemplo. En este tipo de solicitud, los datos deben ser codificados en una forma estándar por lo que se usa el método urlencode() y se pasa al objeto Request como argumento. Esta codificación es realizada a través de la librería urllib.

```
42 #POST
43 def http_post(url):
44     post = urlencode("")
45     req = urllib2.Request(url, post)
46     response = urllib2.urlopen(req)
47     return response.read()
```

### **Código 2.26.** Función que implementa solicitud POST

En todas estas solicitudes se consulta la base de datos Firebase, se obtienen objetos JSON y se requiere pasar o deserializar a objetos Python. Este proceso se realiza a través de la librería json que hace uso del método loads() como se observa en línea 82 del Código 2.27:

```
80 #Consulta candados de la estacion
81 response = http_get(ipServidor+"/CandadosServicio.svc/candados?estacion="+estacion.id);
82 candados = json.loads(response, object_hook=lambda d: namedtuple("X", d.keys())(*d.values()))
```

### **Código 2.27.** Deserializar objetos JSON

Es importante mencionar que para desarrollar el script en Python y definir las señales de la Raspberry Pi se importa la librería RPi, línea 25 de Código 2.28.

```
25 import RPi.GPIO as GPIO
```

### **Código 2.28.** Importe de librería de pines de Raspberry Pi

Luego se configuran los puertos con la notación BOARD como se observa en línea 129. En línea 130 del Código 2.29 se configura el número del pin a controlar y finalmente en la

línea 131 se determina si la señal va a estar en alto o en bajo según el valor True o False respectivamente:

```
129 GPIO.setmode(GPIO.BOARD)
130 GPIO.setup(16, GPIO.OUT)
131 GPIO.output(16, True)
```

**Código 2.29.** Encendido pin 16 de Raspberry Pi.

Para conectar la cerradura electromagnética con un pin de la Raspberry Pi se utilizó el esquema de la Figura 1.13. En este circuito se utilizan 3 pines de la Raspberry Pi:

- Alimentación del circuito: pin 2 de 5v
- Señal de control: pin 13
- Tierra: pin 6

De esta forma a partir del diseño se implementa el sistema y se alcanzan los requerimientos del sistema. En el Anexo R, S, T, se adjunta el código correspondiente a la programación de la Raspberry Pi.

El Anexo B, C, D, E, F, G, H, I, J se encuentra la programación de la aplicación cliente y en el anexo K, L, M, N y O se encuentran la programación de la aplicación Administrador.

## 3 RESULTADOS Y DISCUSIÓN

En base a la metodología Rapid Application Development se desarrolla el sistema para que cumpla con los requerimientos definidos y analizados en el capítulo 2. En esta sección se analiza el funcionamiento de cada uno de los módulos del prototipo planteado en el presente proyecto de titulación: en el módulo principal, se verifica la correcta gestión del servicio web referente a las solicitudes y respuestas realizadas entre la interfaz de usuario y la base de datos. En el módulo de control se prueba el envío y recepción de datos entre la Raspberry Pi y Firebase. Finalmente, se comprueba el funcionamiento general de los módulos del sistema.

### 3.1 Prueba de Interfaz de Usuario, módulo principal y módulo de control

Se verifica la correcta gestión del servicio web referente a las solicitudes y respuestas realizadas entre Android, servicio web WCF y Firebase en un dispositivo real.

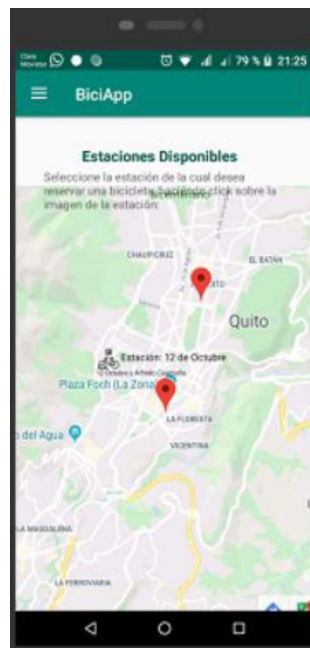
**Prueba 1:** Autenticación en el sistema, ver Figura 3.1.

Para ello se ingresa el usuario y contraseña:



**Figura 3.1.** Autenticación de cliente

Lo que demuestra el éxito de la prueba es que se observe la pantalla principal de la aplicación donde muestra un mapa y sus estaciones, ver Figura 3.2:



**Figura 3.2.** Pantalla principal de sistema de reservas

Se verifica la autenticación correcta de un usuario cliente que consta en la base de datos:

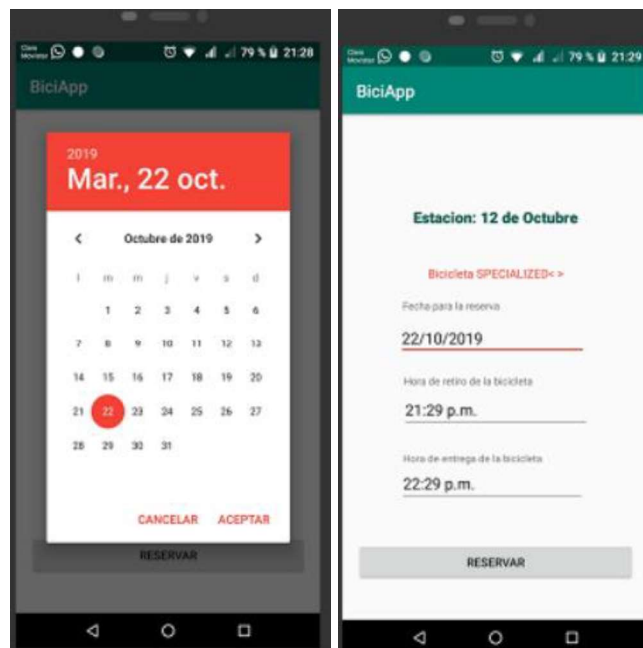
**Prueba 2:** Reserva de bicicleta.

El usuario elige una estación, se muestra la pantalla de la Figura 3.3 :



**Figura 3.3.** Bicicletas disponibles de estación 12 de octubre

Acto seguido, se despliega la actividad de la Figura 3.4 para elegir la fecha de reserva. El usuario elige la hora de inicio, se autocalcula la hora fin y presiona el botón Reservar.



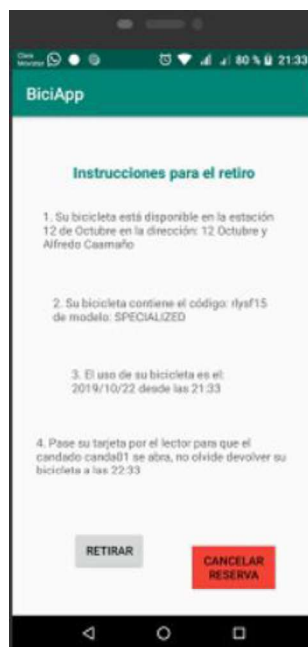
**Figura 3.4.** Pantalla de reserva – Elegir fecha de reserva

En la Figura 3.5 se visualiza la confirmación de la reserva.



**Figura 3.5.** Mensaje de confirmación de reserva

Se confirma la creación de la reserva con éxito debido a que aparece la pantalla de la Figura 3.6 con los detalles de la estación, bicicleta, candado y horario de reserva.



**Figura 3.6.** Activity de instrucciones de retiro



### Prueba 3: Retiro de Bicicleta

El usuario se acerca a la estación físicamente y se evidencia el estado de la estación en la Figura 3.7.



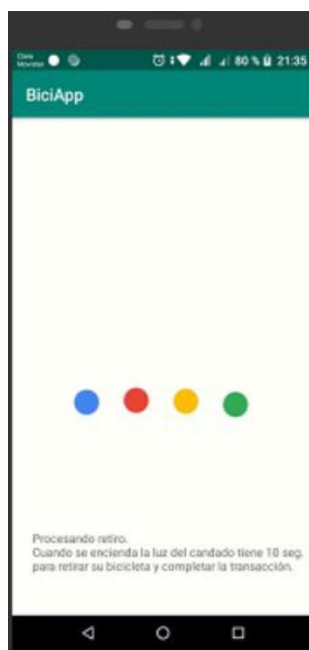
**Figura 3.7.** Estación con bicicleta lista para retiro

En este momento, el usuario pasa su tag RFID para iniciar retiro y se enciende el led del lector de cliente, Figura 3.8:



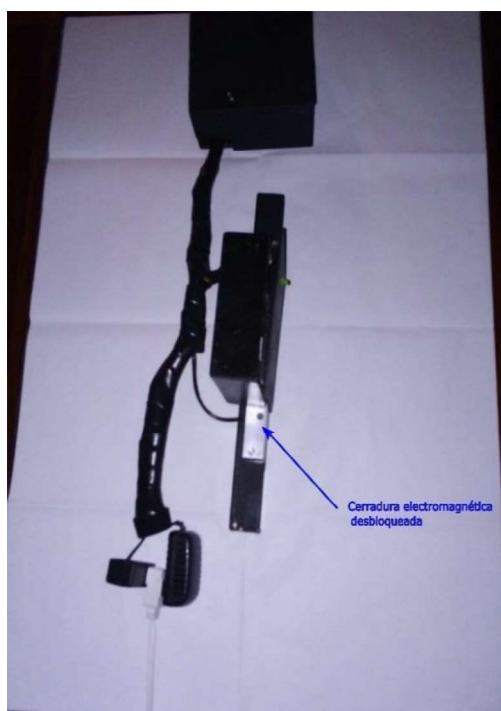
**Figura 3.8.** Lectura de tag cliente enciende led

Se valida en el sistema y se observa la pantalla de la Figura 3.9:



**Figura 3.9.** Procesando retiro

Finalmente, una vez que se valida que existe reserva, se desbloquea la cerradura, ver Figura 3.10, y se observa la pantalla de Figura 3.11.



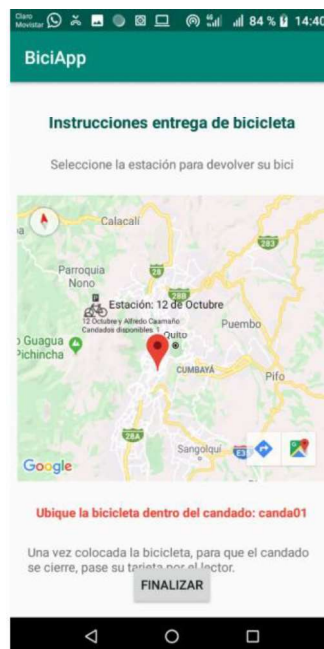
**Figura 3.10.** Candado libre



**Figura 3.11.** Retiro con éxito

#### **Prueba 4:** Devolución de bicicleta

Después de haber realizado el recorrido, el usuario decide entregar. Para esto debe elegir una estación y el sistema automáticamente le asigna el primer candado disponible, ver Figura 3.12.



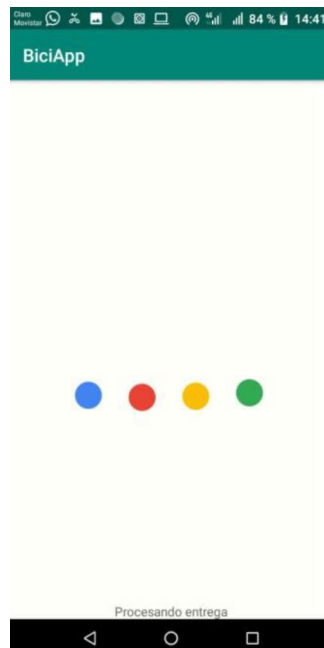
**Figura 3.12.** Candado libre en estación elegida

Usuario se acerca a la estación y pasa su identificador RFID



**Figura 3.13.** Usuario en estación con bicicleta

Finalmente se observa la pantalla de la Figura 3.14, y se ubica la bicicleta en el candado asignado, el lector sensa la bicicleta, en Figura 3.15 enciende led y bloquea la cerradura. En este punto finaliza el proceso de entrega.



**Figura 3.14.** Activity procesando entrega

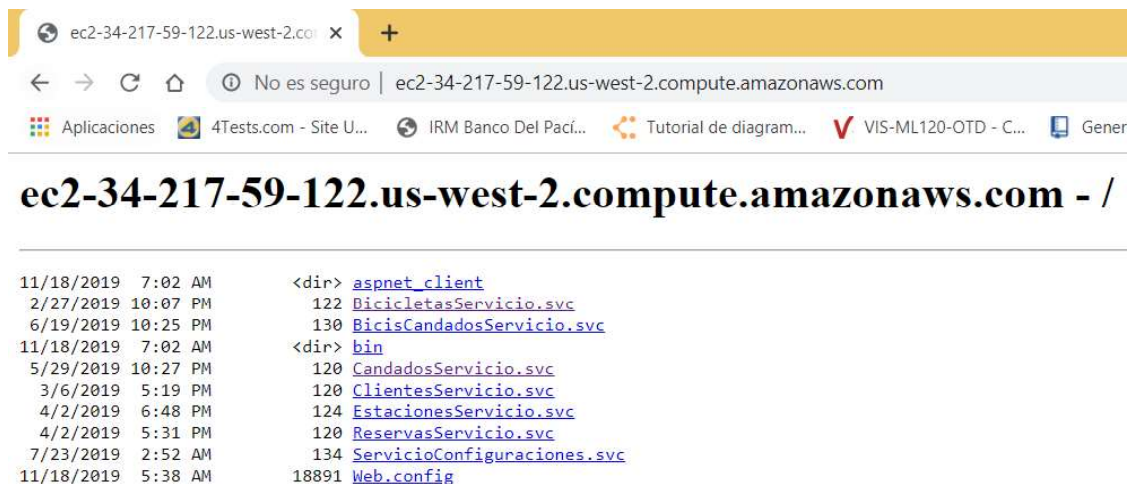


**Figura 3.15.** Lector sensa bicicleta en estación

Todas las gestiones y la lógica del sistema han sido administradas por el servicio web que almacena los datos inmersos en Firebase. A continuación, en Figura 3.17, 3.18 y 3.19, se observan los registros de reserva, retiro y entrega de bicicleta.

**Prueba 5:** Prueba en la nube del Servicio WCF

En Figura 3.16 se muestran los servicios de bicicletas, candados, clientes, estaciones y reservas.



**Figura 3.16.** Servicio web en nube de AWS

### Prueba 6: Prueba de reserva y transacción de retiro y entrega

En Figura 3.17, se observa que se reservó el día 18 de noviembre de 2019 una bicicleta desde las 14:32 hasta las 15:32.

```
-LtzziqSlidD1c-ZTVIC
activa: false
concretada: true
fecha: "2019/11/18"
horaFin: "15:32"
horaInicio: "14:32"
id: "k11ku"
idBici: "rlysf15"
idCliente: "abcde01"
```

Figura 3.17. Registro de reserva

En Figura 3.18 se muestra que el cliente retiró la bicicleta a las 14:33 del 18 de noviembre de 2019.

```
-LsZ8rTbRxE2ENe0TUAu
entregaRetiro: false
fechaHora: "2019-11-18 14:33:1"
id: "pifau"
idBici: "rlysf15"
idCandado: "canda01"
statusEntregaRecepcion: true
```

Figura 3.18. Registro de retiro

En Figura 3.19 se muestra que el cliente entregó la bicicleta a las 15:09 del 18 de noviembre de 2019.

```
-LsZ8ayRhs1lgHDk97mF
entregaRetiro: true
fechaHora: "2019-11-18 15:09:1"
id: "uzvoc"
idBici: "rlysf15"
idCandado: "canda01"
statusEntregaRecepcion: true
```

Figura 3.19. Registro de entrega

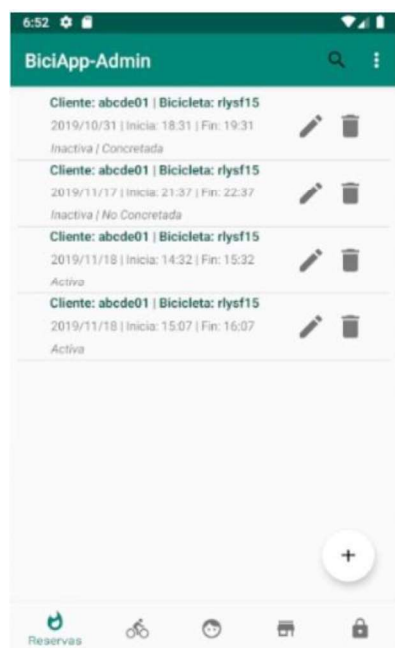
### Prueba 7: Funcionalidad app administrador

Se realiza la autenticación del administrador como se observa en la Figura 3.20.



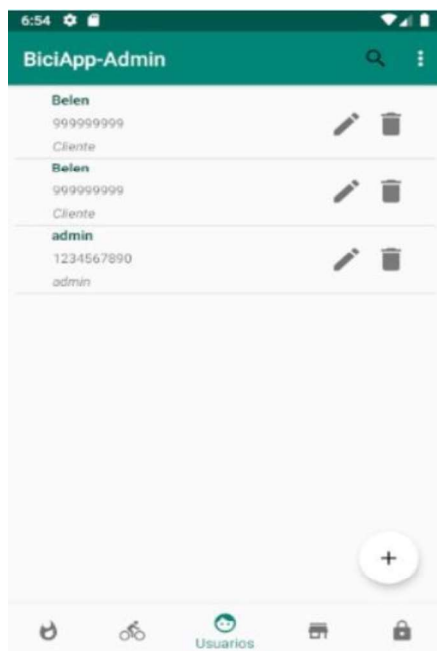
**Figura 3.20.** Autenticación app administrador

En Figura 3.21, se accede al dashboard del administrador:



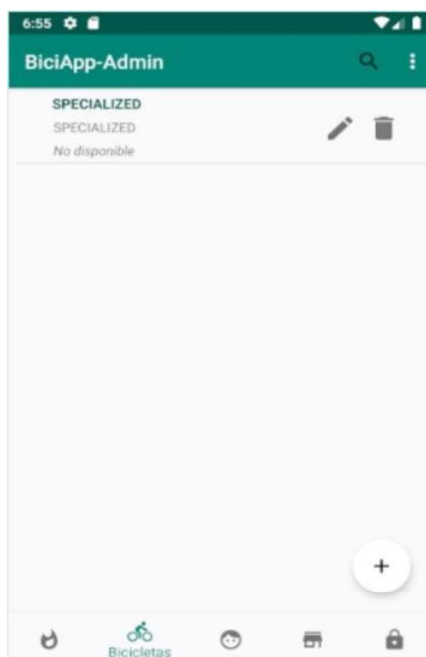
**Figura 3.21.** Dashboard

En Figura 3.22, se observa la administración de clientes:



**Figura 3.22.** Administración clientes

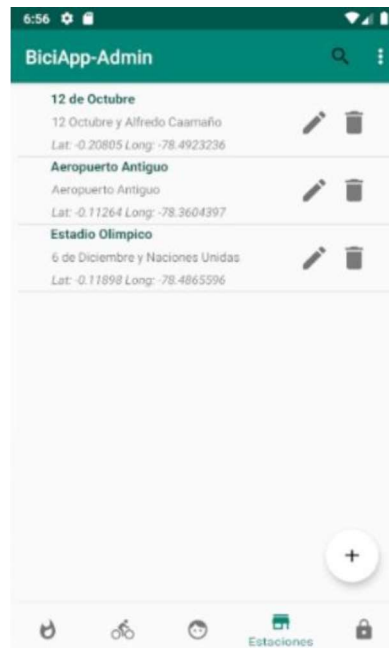
En Figura 3.23, se observa la administración de bicicletas:



**Figura 3.23.** Administración bicicletas

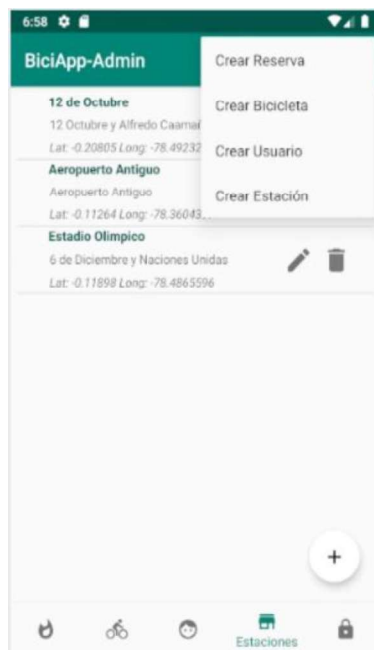


En Figura 3.24, se observa la administración de estaciones:



**Figura 3.24.** Administración de estaciones

Finalmente, en Figura 3.25, se verifica el menú para crear estaciones, reservas, bicicletas y usuarios.



**Figura 3.25.** Menú - Crear elementos del sistema

En este capítulo se verifican las pruebas que comprueban la correcta funcionalidad del sistema, módulo principal, módulo de control e interfaz de usuario tanto en la aplicación móvil del cliente como del administrador.

## 4 CONCLUSIONES Y RECOMENDACIONES

### 4.1 CONCLUSIONES

- Las peticiones HTTP que se utilizan en este sistema de bicicletas requieren de una conexión a Internet para poder ejecutar las transacciones de retiro y entrega. Esto representa una debilidad del prototipo ya que es necesario realizar el bloqueo y desbloqueo del candado de bicicletas de forma local en caso de que falle el enlace de Internet.
- El modelo empezó a fallar en la arquitectura cliente servidor debido a que muchas veces el cliente Android se vuelve servidor ya que está a la escucha de cambios en la base de datos para realizar retiros y entregas de bicicletas.
- Una falla identificada en el prototipo es la gran sobrecarga que representan las numerosas solicitudes HTTP que realizan Android y la Raspberry Pi para realizar las validaciones previas a las transacciones. Esta comunicación centralizada hace que los sistemas no sean óptimos.
- Se tuvo dificultades con el servidor por las numerosas peticiones HTTP ya que el cliente Android maneja diferentes hilos en ejecución y el servidor tiene una RAM limitada de 1GB. Es decir, Android está a la escucha de cambios en la base de datos y para ello se crean varias solicitudes asíncronas que deben ser controladas (eliminadas) una vez que se detecte una respuesta satisfactoria que implica una gran demanda hacia el servidor. Para solucionar estas fallas en el servidor fue fundamental utilizar un método que remueve esta escucha de cambios desde Android.

### 4.2 RECOMENDACIONES

- El prototipo desarrollado requiere que el usuario porte un tag RFID que identifica la presencia física del cliente para realizar un retiro o devolución de bicicletas. Para mayor seguridad se recomienda la integración de un display y teclado numérico en la estación para que el usuario además de utilizar el tag RFID, ingrese un código único que se genere en cada transacción de retiro y entrega.
- Para un futuro desarrollo se podría hacer uso de utilidades que ofrece Firebase, por ejemplo, la autenticación de Google y Facebook y el envío de mensajes push para generar códigos.
- Este proyecto utilizó como frontend una aplicación móvil para la gestión y administración del sistema, para el futuro se podría implementar esta administración en una aplicación web.

- El prototipo actual identifica la presencia de la bicicleta a través de la lectura del código RFID, en un desarrollo futuro se recomienda implementar un gps para poder hacer el track de las distancias recorridas y rastrear la bicicleta.
- El módulo de control realiza las transacciones a partir del frontend de Android del dispositivo móvil, se recomienda implementar un botón en la estación para retirar y otro para entregar la bicicleta, de este modo el cliente tiene un método directo para desbloquear y bloquear la bicicleta en la estación una vez que el sistema lo identifique localmente.
- Una gran mejora que se puede implementar es desarrollar el sistema de bicicletas utilizando el protocolo MQTT que es ideal para comunicaciones donde el cliente se vuelve servidor.

## 5 REFERENCIAS BIBLIOGRÁFICAS

- [1] Transport for London, «Santander Cycles - Transport for London,» 2010. [En línea]. Available: <https://tfl.gov.uk/modes/cycling/santander-cycles>. [Accessed: 20-Jan-2019].
- [2] Spokes Bicycle Rentals, «Spokes Bicycle Rentals,» 2017. [En línea]. Available: <https://spokesbicyclerentals.com/>. [Accessed: 20-Jan-2019].
- [3] Donkey Republic, «Alquiler de bicicletas 24/7 - Donkey Republic,» 2017 [En línea]. Available: <https://www.donkey.bike/es/>. [Accessed: 20-Jan-2019].
- [4] BiciQuito, «BiciQuito - Bicicleta Pública,» [En línea]. Available: <http://www.biciquito.gob.ec/index.php/info/que-es.html>. [Accessed: 19-Jan-2019].
- [5] CiclóPolis, «CiclóPolis Ec,» [En línea]. Available: <https://ciclopolis.wordpress.com/quienes-somos/>. [Accessed: 10-Mar-2019].
- [6] Google Inc., «Firebase Realtime Database,» [En línea]. Available: <https://firebase.google.com/docs/database/>. [Accessed: 23-Apr-2019].
- [7] Lucid Software Inc., «4 Phases of Rapid Application Development Methodology | Lucidchart Blog,» [En línea]. Available: <https://www.lucidchart.com/blog/rapid-application-development-methodology>. [Accessed: 10-Mar-2019].
- [8] I. Automated Architecture, «blueink.biz - Rapid Application Development - An Application Development Technique Using Prototypes, Iterative Customization, and CASE Tools,» 2019. [En línea]. Available: <http://www.blueink.biz/RapidApplicationDevelopment.aspx>. [Accessed: 21-Apr-2019].
- [9] K. Kikama, Securing the Rapid Application Development (RAD) Methodology , 2010.
- [10] High Performance Computing System Laboratory, Introduction to Distributed Systems, 2007.
- [11] D. Serain, International Seminar on Client/Server Computing, 1995, vol. 1995, pp. v1-1-v1-1.
- [12] J. Löwy, Programming WCF services. O'Reilly, 2007.
- [13] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, Web Services, Berlin, Springer, 2004, pp. 123–149.
- [14] H. M. Abdullah and A. M. Zeki, Frontend and Backend Web Technologies in Social Networking Sites: Facebook as an Example, 2014, pp. 85–89.
- [15] D. McComb, «Web Services: Definition and Implications,» [En línea]. Available: <https://flylib.com/books/en/2.820.1.107/1/>. [Accessed: 01-May-2019].
- [16] Coursera, «4.1.5 – HTML / XML / JSON - Web Technologies,» [En línea]. Available:

- <https://www.coursera.org/lecture/service-oriented-architecture/4-1-5-html-xml-json-rK9X5>. [Accessed: 06-Oct-2019].
- [17] Coursera, «4.1.6 – HTTP - Web Technologies,» [En línea]. Available: <https://www.coursera.org/lecture/service-oriented-architecture/4-1-6-http-MsRrv>. [Accessed: 06-Oct-2019].
- [18] Dotnetpattern.com, «WCF Architecture,» 2019 [En línea]. Available: <http://dotnetpattern.com/wcf-architecture>. [Accessed: 15-Oct-2019].
- [19] L. Oliva, Design and development of a REST-based Web service platform for applications integration.
- [20] S. Mumbaikar and P. Padiya, Web Services Based On SOAP and REST Principles, 2013.
- [21] Android Developers, «Introducción a Android Studio,» [En línea]. Available: <https://developer.android.com/studio/intro>. [Accessed: 10-Oct-2019].
- [22] M. Zechner, J. F. DiMarzio, R. Green, M. Zechner, J. F. DiMarzio, and R. Green, First Steps with Android Studio, Apress, 2016, pp. 15–32.
- [23] B. Hohensee, Android For Beginners. Developing Apps Using Android Studio, Babelcube Inc. , 2014, pp.13-35
- [24] H. Esmaeel, «(PDF) Apply Android Studio (SDK) Tools,» [En línea]. Available: [https://www.researchgate.net/publication/331673953\\_Apply\\_Android\\_Studio\\_SDK\\_Tools](https://www.researchgate.net/publication/331673953_Apply_Android_Studio_SDK_Tools). [Accessed: 31-Oct-2019].
- [25] Pitechnologies.com, «Advantages of Using Firebase For App Development,» [En línea]. Available: <https://pитеchnologies.org/pitechblog/Advantages-of-Using-Firebase-For-App-Development/185>. [Accessed: 12-Oct-2019].
- [26] Google Inc., «Estructura tu base de datos | Firebase Realtime Database,» [En línea]. Available: [https://firebase.google.com/docs/database/web/structure-data?hl=es\\_419](https://firebase.google.com/docs/database/web/structure-data?hl=es_419). [Accessed: 12-Oct-2019].
- [27] Google Inc., «Firebase Pricing | Firebase,» [En línea]. Available: <https://firebase.google.com/pricing/?hl=es-419>. [Accessed: 14-Oct-2019].
- [28] W.-J. Li, C. Yen, Y.-S. Lin, S.-C. Tung, and S. Huang, JustIoT Internet of Things based on the Firebase real-time database, 2018, pp. 43–47.
- [29] «Non-relational data and NoSQL | Microsoft Docs,» [En línea]. Available: <https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data>. [Accessed: 14-Oct-2019].
- [30] «NuGet Gallery | FirebaseDatabase.net 4.0.2,» [En línea]. Available: <https://www.nuget.org/packages/FirebaseDatabase.net/>. [Accessed: 14-Oct-2019].
- [31] F. Pérez, B. Granger, and J. Hunter, Python: An Ecosystem for Scientific Computing,

- 2011.
- [32] B. Horan, *Practical Raspberry Pi*. Apress, 2013.
  - [33] M. Saluch *et al.*, Raspberry Pi 3B + microcomputer as a central control unit in intelligent building automation management systems, 2018, vol. 196.
  - [34] «Simple Guide to the Raspberry Pi GPIO Header - Raspberry Pi Spy, » [En línea]. Available: <https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/>. [Accessed: 21-Oct-2019].
  - [35] V. Vujovic and N. Davidović, Raspberry Pi as Internet of Things hardware: Performances and Constraints, 2015.
  - [36] N. Semiconductors, MFRC522 Standard performance MIFARE and NTAG frontend.
  - [37] Instructables, «All You Need to Know About a Relays: 6 Steps (with Pictures),» [En línea]. Available: <https://www.instructables.com/id/All-You-Need-to-Know-About-Relays/>. [Accessed: 08-Oct-2019].
  - [38] 12 Volt Planet, «Automotive Relay Guide,» [En línea]. Available: <https://www.12voltplanet.co.uk/relay-guide.html>. [Accessed: 21-Oct-2019].
  - [39] Electric lock Systems, «How Does a Magnetic Lock Work?,» [En línea]. Available: <https://www.electricklock.net/how-does-a-magnetic-lock-work/>. [Accessed: 07-Oct-2019].
  - [40] Seco-Larm, Cerraduras Electromagnéticas de 600-lb Datasheet.
  - [41] Pace University, Introduction to Software Engineering - Requirements Engineering, New York, 2014.
  - [42] J. Bezos, «Amazon Web Services Documentation,» [En línea]. Available: [https://docs.aws.amazon.com/es\\_es/AWSEC2/latest/WindowsGuide/EC2\\_GetStarted.html](https://docs.aws.amazon.com/es_es/AWSEC2/latest/WindowsGuide/EC2_GetStarted.html) y [https://docs.aws.amazon.com/es\\_es/AWSEC2/latest/UserGuide/using-instance-addressing.html](https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/using-instance-addressing.html). [Accessed: 18-Nov-2019].
  - [43] Guidgenerator.com, «Online GUID Generator,» [En línea]. Available: <https://www.guidgenerator.com/>. [Accessed: 04-May-2019].
  - [44] W3C Working Group, «Web Services Glossary,» 2004. [En línea]. Available: <https://www.w3.org/TR/ws-gloss/>. [Accessed: 01-May-2019].
  - [45] K. V. Dyshlevoi, V. E. Kamensky, and L. B. Solovskaya, MARSHALLING IN DISTRIBUTED SYSTEMS: TWO APPROACHES, 1997.
  - [46] I. Nandrajog, «Simplified Object Access Protocol.» [En línea]. Available: <https://web.njit.edu/~turoff/coursenotes/IS679/sample/soap.htm>. [Accessed: 30-Sep-2019].
  - [47] J. Ponelat, «Home - OpenAPI Initiative,» [En línea]. Available: <https://www.openapis.org/>. [Accessed: 31-Oct-2019].

# ORDEN DE EMPASTADO

# ANEXOS

## Anexo A. Diagrama de Clases utilizado en el servicio WCF REST

