Pontificia Universidad
JAVERIANA
Cali

Facultad de Ingeniería
y Ciencias
Ingeniería Electrónica

Monografía de Trabajo de Grado

Sistema de parqueo de bicicletas Bistander

Nicolás Castaño Calderón
Gabriel Jordan Lopera

Director
Dr. Alexánder Martínez

8 de noviembre de 2023

Pontificia Universidad JAVERIANA Cali | Facultad de Ingeniería y Ciencias

Acta de Correcciones al Proyecto de Grado
Ingeniería Electrónica

Fecha: 7 de noviembre de 2023

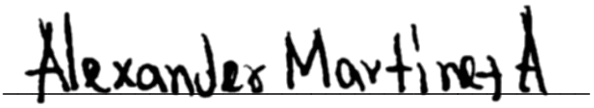
Autores: Gabriel Jordán Lopera
Nicolás Castaño Calderón

Nombre del Proyecto de Grado:

Sistema de Parqueo de Bicicletas Bistander

Director: Dr. Alexánder Martínez Álvarez

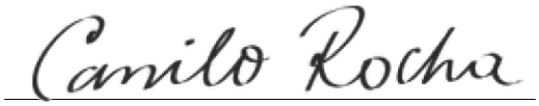
Como indica el artículo 2.27 de las Directrices de Trabajo de Grado, he verificado que los estudiantes indicados arriba han implementado todas las correcciones que los Jurados del Proyecto de Grado definieron que se efectuaran, como consta en el Acta de Calificación correspondiente.



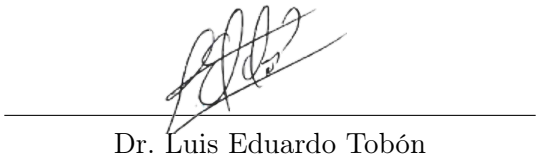
Alexánder Martínez Álvarez
Director del Proyecto de Grado

Nota de Aceptación

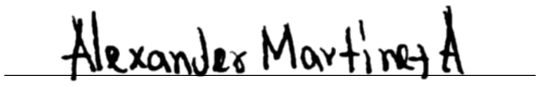
Aprobado por el Comité de Trabajo de Grado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana para optar al título de Ingeniero Electrónico.



Dr. Hernán Camilo Rocha Niño
Decano Facultad de Ingeniería y Ciencias



Dr. Luis Eduardo Tobón
Director Carrera Ingeniería Electrónica



Dr. Alexánder Martínez
Director Trabajo de Grado

Dr. Ana Victoria Prados Arboleda
Jurado 1

Dr. Eugenio Tamura Morimitsu
Jurado 2

Santiago de Cali, 8 de noviembre de 2023

Señores
Pontificia Universidad Javeriana – Cali
Dr. Hernán Camilo Rocha Niño
Decano
Facultad de Ingeniería y Ciencias
Ciudad

Cordial saludo.

Por medio de la presente nos permitimos presentarle el Trabajo de Grado titulado “Sistema de parqueo de bicicletas Bistander ”.

Esperamos que este trabajo reúna todos los requisitos académicos, cumpla el propósito para el cual fue creado y sirva de apoyo para futuros proyectos relacionados con la materia.

Atentamente,

Nicolás Castaño Calderón

Gabriel Jordan Lopera

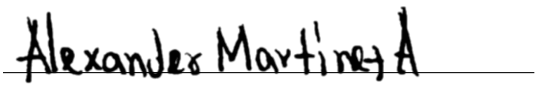
Santiago de Cali, 8 de noviembre de 2023

Señores
Pontificia Universidad Javeriana – Cali
Dr. Hernán Camilo Rocha Niño
Decano
Facultad de Ingeniería y Ciencias
Ciudad

Cordial saludo.

Certifico que el presente Trabajo de Grado titulado "Sistema de parqueo de bicicletas Bistander ", realizado por Nicolás Castaño Calderón y Gabriel Jordan Lopera, estudiantes de Ingeniería Electrónica, se encuentra terminado y puede ser presentado para su sustentación.

Atentamente,



Dr. Alexánder Martínez
Director Trabajo de Grado

Agradecimientos

Este trabajo se lo dedicamos a todas las personas que influyeron en nosotros en este camino a ser ingenieros, motivándonos a seguir y a no rendirnos aún en condiciones adversas. Principalmente, queremos agradecer a nuestros padres: Lizeth Lopera, por ser una gran madre, siempre dispuesta a ayudar cuando se necesitaba y que nunca dejó de dar alientos y amor, una mujer fuerte y única que siempre compartió su gran inteligencia y sabiduría; Luis Alfredo Jordan, que siempre ha podido centrar a las personas en lo que tienen que hacer, dando a su vez ideas clave de cómo realizar las cosas más prácticas, un hombre con gran capacidad de dar, siempre preocupado por los demás y con principios sólidos; José Wilson Castaño, por estar presente durante nuestro proceso como estudiantes y dándonos el apoyo necesario en todo momento; Edith Calderón, por darnos motivación en los momentos mas difíciles durante el proceso, por sus muestras de cariño y comprensión y estar siempre al cuidado de nosotros durante las extensas reuniones. También, agradecer a nuestros hermanos: Juan Manuel Jordan y Natalia Castaño. Aprovechamos para incluir igualmente a Alexánder Martínez, por su paciencia y acompañamiento a través de la creación de este documento y su interés en ver a sus dos estudiantes más allá de la meta final. Finalmente, nos gustaría extender este agradecimiento a todo el personal de la Javeriana que nos brindó un apoyo de alguna manera u otra, como los laboratoristas del laboratorio de electrónica, el personal de aseo que nos permitía estudiar en instalaciones limpias y los compañeros que hicieron de nuestra experiencia en la universidad algo especial e inolvidable. Gracias, gracias y gracias.

Resumen

Los parqueaderos de bicicletas públicos, al igual que los parqueaderos que se encuentran en el campus de la Universdiad Javeriana, suelen contar con una seguridad deficiente, al asegurar solo una parte de la bicicleta o necesitar de implementos propios del dueño de dicha bicicleta para asegurarla (cadenas, candados, entre otros); esto motiva la realización de un parqueadero que integre estas funciones y logre un objetivo, el cual es ser más seguro que alternativas tradicionales.

En el presente trabajo de grado se expone el desarrollo del parqueadero *Bistander*, explicando paso a paso el diseño, componentes y código que se requirieron para llegar a un prototipo, en el cual encontramos un sistema que integra un factor de seguridad física, así como seguridad informática, a partir de la incorporación de diferentes componentes que conforman una estructura que asegura dos partes vitales de una bicicleta, el marco y la rueda, e incluye alertas en forma de notificaciones que surgen de la aplicación y van dirigidas a los usuarios.

Al integrar dos tipos diferentes de seguridad, se logra un sistema con posibilidades de mira al mercado, y mucho más eficiente; la unión de la seguridad cibernética junto con la seguridad física, permiten proteger tanto la bicicleta (propiedad tangible) como la información de quien crea la cuenta (propiedad intangible). Con estos desarrollos, se desafía el paradigma actual colombiano, donde la existencia de parqueaderos públicos es muy reducida, inspirado en gran parte por la alta inseguridad.

El resultado es un prototipo con una aplicación funcional, en el cual se logran identificar desarrollos de electrónica sobre un parqueadero que tiene características mecánicas diferenciales, como el aseguramiento de diferentes partes de la bicicleta; sin embargo, se requiere de un estudio de materiales posterior y perfeccionamiento del sistema mecánico para resistir intentos de robo.

Palabras clave:

- Parqueadero inteligente, bicicleta, seguridad, internet, internet of Things (IoT), movilidad, transporte, sensores, sistemas embebidos, aplicación móvil, infraestructura, ciclistas, sistemas de transporte inteligentes, trasnporte sostenible, movilidad urbana, monitoreo en tiempo real, geolocalización, soluciones basadas en la nube, autenticación de usuarios

Abstract

Public bicycle parking lots, just like those found in the Javeriana University, usually have insufficient security as they only keep a part of the bicycle secured, or the owner needs his own tools (chains, locks and others); this motivates the building of a parking lot that integrates these functions and achieves an objective, which is to be safer than traditional alternatives. In this current project we present the development of a parking lot, explaining the design step by step, components and the code needed to get to a prototype, in which we find a system that integrates a physical security factor, as well as an informatic security one, through the inclusion of different components that make up the structure which secures two vital parts of the bicycle, the frame and the wheel, and includes alerts in the form of notifications that show up in the app and are directed towards the users. As the two different types of security are combines, a much more efficient system is created with the possibility to enter the market, defying the current colombian paradigm in which the existence of bicycle public parking systems is reduces, inspired highly because of the high insecurity. The result is a prototype with a functional application, in which electronic developments can be identified over a parking lot which has differential mechanical characteristics, like the securing of different parts of the bicycle; however, further material study is required, as well as improvement of the mechanical system to better resist theft attempts.

Keywords:

- Smart parking, bicycle, security internet, internet of Things (IoT), mobility, transportation, sensors, embedded systems, mobile application, infrastructure, cyclists, smart transportation systems, sustainable transportation, urban mobility, real time monitoring, geolocation, cloud based solutions, user authentication

Índice general

1. Introducción **1**

2. Contextualización del proyecto **3**

2.1. Planteamiento . . . 3
2.2. Justificación . . . 4
2.3. Objetivos . . . 4
2.3.1. Objetivo General . . . 4
2.3.2. Objetivos Específicos . . . 4
2.4. Marco de Referencia . . . 5
2.4.1. Definición de términos . . . 5
2.4.2. Marco Teórico . . . 6
2.4.3. Trabajos relacionados . . . 8

3. Desarrollo del sistema de parqueo **10**

3.1. Requisitos del usuario . . . 10
3.2. Planteamiento de solución . . . 11
3.3. Software . . . 12
3.3.1. Base de datos . . . 13
3.3.2. Front-End . . . 13
3.3.3. Back-End . . . 13
3.3.4. Firebase authentication . . . 14
3.3.5. Tecnologías de comunicación . . . 14
3.3.6. Interfaces de visualización . . . 17
3.4. Hardware . . . 17
3.5. Selección de hardware . . . 17
3.5.1. Procesador . . . 17
3.5.2. Selección del procesador . . . 18
3.5.3. Acelerómetro . . . 19
3.5.4. Selección del acelerómetro . . . 19
3.5.5. Sensor de proximidad . . . 20
3.5.6. Selección del sensor de proximidad . . . 21
3.5.7. Pad Numérico . . . 22
3.5.8. Cerradura . . . 22

4. Resultados y Discusión **23**
4.1. Visión global del diseño 23
4.1.1. Forma de aseguramiento de la bicicleta 23
4.1.2. Diseño exterior 24
4.1.3. Anclado 26
4.1.4. Materiales 27
4.2. Consideraciones de diseño 29
4.3. Resultados generales 30

5. Conclusiones **38**

6. Recomendaciones **39**
6.1. Materiales 39
6.2. Respaldo 39
6.3. Aplicación móvil 39

7. Anexos **40**
Anexo 1 – Encuesta sobre parqueaderos de bicicletas 40
Anexo 2 – Código de Arduino para funcionamiento de ESP32 y sensores 45

Bibliografía **70**

Índice de figuras

3.1. Diagrama del sistema 11
3.2. Marco del sistema de aseguramiento 12
3.3. Diagrama del software del sistema 12
3.4. Módulo ESP32 18
3.5. Módulo ADXL345 20
3.6. Módulo VL6180 21
3.7. Cerradura Solenoide 12V 22

4.1. Diseño original del parqueadero con dos barras. 24
4.2. Bicicleta asegurada en diseño original del parqueadero. 24
4.3. Nuevo modelo del parqueadero. 25
4.4. Diagrama de las partes de una bicicleta. 25
4.5. Área útil para medición de distancia entre radios. 26
4.6. Medidas externas del parqueadero. 27
4.7. Inicio de sesión 31
4.8. Registro de usuarios 32
4.9. Pantalla de inicio del usuario 33
4.10. Cuenta del usuario 34
4.11. Ubicación de los parqueaderos 35
4.12. Parqueadero utilizado 36
4.13. Notificaciones 37

7.1. Frecuencia de uso de parqueaderos de bicicletas públicos. 40
7.2. Seguridad de los parqueaderos públicos de bicicletas. 41
7.3. Potencial uso de la aplicación para monitoreo de la bicicleta parqueada. 41
7.4. Incomodidad de uso de parqueaderos no inteligentes. 42
7.5. Percepción de seguridad y falta de parqueaderos de bicicletas. 42
7.6. Preferencias de tecnología para desbloqueo. 43
7.7. Posibles costos del parqueadero. 43
7.8. Preferencias de tecnología para desbloqueo. 44
7.9. Interés en el uso de un parqueadero de bicicleta inteligente. 44
7.10. Librerías utilizadas. 45
7.11. Declaraciones globales. 46
7.12. Código del Pad numérico. 47
7.13. Variables de parqueadero y conexión a internet. 47
7.14. Tareas HTTP y de Pad numérico. 48
7.15. Inicialización de los módulos y sensores. 48

7.16. Bucle de funciones del programa. 49
7.17. Función WiFi. 49
7.18. Condicional de la función WiFi. 50
7.19. Acceso de hipervínculo y solicitud de texto. 50
7.20. Impresión HTTP. 50
7.21. Función Rango. 51
7.22. Función Acelerar. 51
7.23. Función KeyPad. 52
7.24. Archivo principal. 53
7.25. Controladores de estado. 54
7.26. Objeto usuario, interfaz typeScript y contexto. 55
7.27. Administradores de estado. 55
7.28. Función auth.onAuthStateChanged. 56
7.29. Proveedor de contexto. 57
7.30. Configuración Firebase 57
7.31. Hook useAuth 1. 58
7.32. Hook useAuth 2. 59
7.33. Hook useAuth 3. 59
7.34. Hook useParking 1. 60
7.35. Hook useParking 2. 60
7.36. Hook useParking 3. 61
7.37. Hook useParking 4. 62
7.38. Hook useParking 5. 63
7.39. Componente AuthLayout 64
7.40. Loading. 64
7.41. AuthPage 1. 65
7.42. AuthPage 2. 66
7.43. AuthPage 3. 66
7.44. AuthPage 4. 67
7.45. AuthPage 5. 68
7.46. Stack de navegación. 69

Índice de cuadros

3.1. Comparación de diferentes alternativas 15
3.2. Criterios de selección del procesador 18
3.3. Criterios de selección del acelerómetro 19
3.4. Criterios de selección del sensor de proximidad 21

4.5. Criterios de selección del material 28

Capítulo 1

Introducción

Existen tecnologías, métodos y objetos que no cambian a través del tiempo, debido a que cumplen con su función de manera excepcional, así como hay otros que están en constante cambio y cada vez tienen una mayor efectividad; los parqueaderos de bicicleta, por un largo tiempo, han tenido una funcionalidad similar, a pesar de tener diferentes formas, pero recientemente se han desarrollado sistemas más avanzados, capaces y seguros para dar una mayor confianza a su usuario, así como cierto nivel de monitoreo a este, por medio del uso de tecnologías tales como el RFID (Radio Frequency Identification o identificación por radiofrecuencia), las conexiones inalámbricas a internet, las matrices programables y sensores de cada vez menor tamaño. [1]

Al observar estas tecnologías, es posible inferir que no son precisamente nuevas, pero están cada vez más presentes en diferentes proyectos, impulsado principalmente por el auge de los proyectos individuales, los startups, y la facilidad de acceso al conocimiento que otorgan las redes, así como la constante disminución de dos factores vitales: el costo y su eficiencia. Esto ha dado como resultado proyectos más inteligentes y más automatizados.

Otro aspecto cuya revisión es indispensable para verificar la validez de estos parqueaderos radica en la necesidad de la seguridad adicional que brinda un parqueadero inteligente sobre uno tradicional. Examinando un país como Colombia, podemos notar que no hay una gran cantidad de parqueaderos, y también se cuenta con altas tasas de hurtos de bicicletas.

Con este proyecto se busca solucionar o, en su defecto, mejorar los problemas encontrados con el uso de un parqueadero tradicional, brindando una forma efectiva de asegurar el marco y ruedas de la bicicleta, así como un método fácil de identificación para poder retirar el vehículo sin necesidad de elementos adicionales (tarjetas, llaves, etc.), todo con el uso de tecnologías clave, sensores, y una aplicación que permite el manejo remoto del sistema, así como alertas oportunas, disminuyendo así la incidencia del hurto y perfeccionando los parqueaderos de este estilo. También se busca tener más documentación de parqueaderos de bicicleta con el desarrollo de este, ya que es bastante escasa y no se le presta la atención debida aún cuando la gran mayoría del tiempo, las bicicletas se encuentran parqueadas. [2]

A continuación se procede a mostrar los diferentes elementos que componen este documento; empezando con una contextualización del proyecto, que incluye el razonamiento para iniciar el proyecto, y un marco de referencia para establecer bases teóricas de las tecnologías a usar. Posteriormente, se habla de los materiales y métodos utilizados, iniciando con unos requisitos de usuario a cumplir y que difieren de los requisitos del proyecto, luego se hace explicación del software implementado y el hardware necesario para el proyecto. Para finalizar, contamos con los resultados del proyecto, que son el aprendizaje y producto de las secciones previas.

Capítulo 2

Contextualización del proyecto

2.1. Planteamiento

Hay una tendencia en la ciudad de Cali en la cual se puede ver una cantidad creciente de usuarios de bicicleta, como se puede ver con el aumento entre 5.4 % y 6.9 % de usuarios de bicicleta que se vio del 2018 al 2019[3], gracias a factores como la instauración de ciclovías en trayectos de alta importancia, como las avenidas Quinta y Cañasgordas, así como la mayor conciencia en cuanto a la contaminación y la cultura del ejercicio. Este crecimiento en el uso de bicicletas debería, idealmente, venir acompañado de una mayor seguridad a la hora de parquearlas, lo cual no es el caso. Generalmente, las bicicletas deben guardarse al interior de las casas o en sistemas de parqueo básicos, cuya seguridad depende de un candado y una cadena que debe portar el usuario. Debido a que muchos ciclistas no aseguran sus bicicletas, sea por desinterés, exceso de confianza o falta de equipos, o en su defecto las aseguran muy superficialmente con candados de dudosa calidad o asegurando solo una rueda, se facilita el hurto de partes o la totalidad de las bicicletas, [4] lo cual tiene posibilidad de generar una pérdida de interés en el uso del vehículo. Uno de los factores determinantes para el uso de la bicicleta es la disponibilidad de buenos parqueaderos; en un estudio del 2015 en Montreal, se notó que la mitad de las bicicletas robadas eran por causa de un parqueo en sitios no designados paras esta actividad. [5] Sumado a estos problemas, en la ciudad hay puntos muy limitados para dejar bicicletas aun siendo un vehículo que ocupa muy poco espacio, posiblemente porque requieren una mayor inversión (los vehículos y motos solo necesitan un espacio pavimentado, mientras que las bicicletas requieren estructuras para poder poner la cadena sin que esta pueda ser retirada fácilmente), y un espacio dedicado, ya que un parqueadero de bicicletas no puede ser utilizado por automóviles ni motocicletas. Encontramos entonces un dilema con el mayor uso de la bicicleta, que a su vez genera menor capacidad de lugares para el parqueo de estas, obligando a parquear ilegalmente y dañando la imagen del espacio donde se encuentran; este caso también se observa en países donde se ha promovido el uso de este vehículo, como Dinamarca y Países Bajos. [6] Recientemente en los puntos de mayor flujo de personas, como las universidades, centros comerciales, entre otros, se ha aumentado el número de parqueaderos para bicicletas, pero debido a un interés que aún se encuentra en etapas iniciales, la seguridad y comodidad que se le presta a los usuarios no es la mejor.[7] Con este proyecto se crea un prototipo cuyo enfoque es mejorar los problemas encontrados en los parqueaderos de bicicletas tradicionales, brindando así una forma de asegurar el marco, un método fácil de identificación para poder retirar la bicicleta propia sin necesidad de gastar mucho tiempo, como también brindando la posibilidad de hacer reservas por determinado tiempo para asegurar un espacio fijo y no gastar tiempo buscando donde parquear.

2.2. Justificación

Partiendo de las distintas problemáticas tanto existentes como posibles en la seguridad de las bicicletas en la ciudad, se genera el proyecto de seguridad electrónica para bicicletas. Es importante tener en cuenta que la mayoría de Smart Locks o candados inteligentes del mercado actual no son sólo costosos, adicionalmente cuentan con una baja satisfacción por parte de sus usuarios, mostrando fallas evidentes en su funcionamiento y seguridad.

El proyecto plantea un sistema de parqueadero individual para las bicicletas más efectivo que los parqueaderos actuales, además brindando un método de identificación para que la seguridad no sea únicamente física, sino que incluya tecnologías electrónicas de la actualidad, reforzando la seguridad. Este sistema disminuye la necesidad de personal, facilita la entrada y salida de bicicletas, evita la necesidad de cargar con candados y cadenas, e incentiva el uso de bicicleta al disminuir inconvenientes debidos a la inseguridad.

Se espera que el proyecto cuente con éxito, logrando así incentivar el uso de parqueaderos de bicicleta novedosos y de alta seguridad, y a su vez impulsando el uso de bicicleta a nivel regional y nacional al reducir las dudas en sus usuarios.

2.3. Objetivos

2.3.1. Objetivo General

Diseñar e implementar un sistema de parqueadero individual de bicicletas modular para zonas públicas que permita tener una mejor seguridad con ayuda de tecnología electrónica.

2.3.2. Objetivos Específicos

- Identificar las necesidades para parqueaderos de bicicletas de los usuarios de este medio de transporte en la ciudad para establecer los requisitos base del proyecto.
- Diseñar e implementar un prototipo de la estructura mecánica para soporte y seguridad de una bicicleta.
- Diseñar e implementar un método de desbloqueo personal inalámbrico del parqueadero de bicicletas.
- Diseñar e implementar un sistema de alarma ante eventos de movimiento no autorizado de la bicicleta.
- Desarrollar una aplicación para reservar parqueaderos de forma remota.
- Implementar y analizar el funcionamiento del sistema completo, comparándolo con sistemas existentes.

2.4. Marco de Referencia

2.4.1. Definición de términos

1. Parqueadero inteligente

   Un parqueadero inteligente es un término utilizado para referirse a soluciones que incluyen sensores de parqueo inteligentes, cámaras o sensores de conteo. Estos dispositivos suelen estar integrados en el punto de parqueo o posicionados al lado de ellos para detectar si la bahía está libre o desocupada. Esto ocurre a través de recolección de datos en tiempo real, la cual es luego transmitida a una aplicación o sitio web que se encarga de comunicar la disponibilidad a los usuarios. También puede haber otras funciones, como visualización de costos o de la ubicación de parqueaderos disponibles.[8]

2. Bicicleta

   Según el diccionario de la Real Academia Española (RAE), una bicicleta es un "vehículo de dos ruedas, normalmente de igual tamaño, cuyos pedales transmiten el movimiento a la rueda trasera por medio de un plato, un piñón y una cadena." [9]

3. Sensor

   Un sensor es un dispositivo que produce una señal de salida con el propósito de detectar y medir un fenómeno físico. Esta salida suele estar conectada a otros dispositivos electrónicos (generalmente un procesador). [10]

   Los sensores permiten, en un espacio de parqueo, recolectar y procesar información en tiempo real, así determinando si un espacio de parqueo está disponible, si está ocupado para contabilizar tiempo en caso de un parqueadero de pago. [11] En el caso de un parqueadero de bicicletas, también puede tener funciones adicionales, como alertas en caso de actividad sospechosa.

4. Front-end

   El front-end está compuesto por múltiples tecnologías que, en términos amplios, hacen que una interfaz de usuario se vea mejor estéticamente y funcione de forma más agradable; entre estas tecnologías encontramos el marcado, el estilo, la compatibilidad a través de dispositivos diferentes, la accesibilidad, las plantillas y marcos web, la usabilidad y el desempeño.

   El desarrollo en front-end tiene como objetivo la creación de interfaces fáciles de manejar, de entender y que cuenten con buena agilidad para que el usuario se interese y pueda comprender con facilidad la información contenida. [12]

5. Back-End

   El desarrollo back-end trata del lado de servidor de una página web. Guarda y ordena datos, y se asegura que todo del lado de cliente del servidor funcione correctamente. Es la parte de una página web con la que no se puede interactuar y que no se puede ver. Es la porción de software que no entra en contacto directo con los usuarios.

Las actividades tales como escribir APIs, crear librerías y trabajar con componentes de sistema sin interfaces de usuario también se incluyen en el back-end.

Los lenguajes más comunes con los que funciona el back-end, son: PHP, C++, Java, Python, JavaScript y Node.js. [13]

2.4.2. Marco Teórico

2.4.2.1. Tecnologías de comunicación

Se cuenta con una variedad de tecnologías inalámbricas, todas con diferentes usos y características que las distinguen, y todas con un uso creciente en un mundo cada vez más interconectado. Algunas de estas tecnologías son la RFID, la NFC y el Bluetooth. Asimismo, se cuenta con alternativas a las tecnologías inalámbricas que permiten un nivel de interacción entre una aplicación y un sistema, con beneficios en costo, como los códigos QR.

El **RFID** (Radio Frequency Identification) o identificación por radiofrecuencia, es una tecnología de identificación inalámbrica que consta de dos partes, un lector y una tarjeta o etiqueta. Tanto el lector como la etiqueta deben tener una antena RF que puede funcionar a diferentes frecuencias que determinan la distancia y seguridad con la que funcionan, y debe haber un código para determinar las funciones, así como acceder a una base de datos para los diferentes permisos según la identificación utilizada.[14] La frecuencia utilizada principalmente por las tarjetas y etiquetas actuales es de 13,56 MHz, pero existen múltiples tecnologías para frecuencias determinadas por sus ventajas y desventajas específicas.[15][16]

La tecnología **NFC** (Near Field Communication) es una tecnología por radiofrecuencia de corto alcance entre dispositivos que permite, por medio de etiquetas compatibles, enviar información sin necesidad de contacto ni pasos adicionales, además de ser compatible con otras tecnologías inalámbricas y ser compatible con una variedad de transmisiones de la misma tecnología. Funciona permitiendo a un dispositivo (lector) crear una corriente de radio frecuencia que se comunica con otro dispositivo que cuente con la información que el lector necesita; también es posible la comunicación de igual a igual de forma que ambos dispositivos envíen y reciban información.[17]

El **Bluetooth** es una tecnología con dos opciones de radio, Bluetooth Classic y Bluetooth Low Energy (LE). Ambas encargadas de la transferencia de información inalámbrica entre dispositivos, utilizan ondas de radiofrecuencia en la banda de 2.4 GHz (banda ISM). Generalmente es utilizada para transferencia de audio o conexión de dos dispositivos electrónicos.[18]

El **QR** es un código que contiene información de manera tanto horizontal como verticalmente, lo que permite incluir una mucho mayor cantidad de esta en un espacio reducido frente un código de barras, que solo contiene información de manera horizontal. Adicionalmente, cuenta con marcas a las esquinas, pequeños cuadros que incrementan la velocidad de lectura al ayudar a identificar la existencia del código QR mismo.

Las **Aplicaciones móviles** son, como su nombre lo expresa, aplicaciones creadas para funcionar en dispositivos móviles; por lo tanto, el proceso de desarrollo de aplicaciones móviles se relaciona con crear paquetes de software instalables como códigos, binarios, assets, etc., implementando servicios

backend como acceso de datos con una API. Se puede identificar con precisión con un número y un usuario únicos sin necesidad de la implementación de otras tecnologías.[19]

2.4.2.2. Bases de datos

Hay gran variedad de tecnologías de base de datos, más nos centramos en aquellas que contaban con características básicas como una versión gratuita, y gran disponibilidad en el mercado para tener acceso a más información sobre ellas. A continuación se encuentran algunas:

AWS o **Amazon Web Services** es una plataforma que ofrece hasta 14 diferentes motores para bases de datos. Es una plataforma de servicios en la nube con un acercamiento "pay-as-you-go". Cuenta con encriptación de parte del servidor y debido a su gran cantidad de servidores, otorga un servicio de confianza. Está limitado en ciertas regiones y tiene una gran abundancia de servicios.[20]

Firebase Realtime Database es una base de datos NoSQL, la cual cuenta con una sincronización en tiempo real que permite a los usuarios tener acceso a los datos desde el dispositivo que deseen, así facilitando el trabajo en conjunto, esta sincronización permite que los datos se almacenen y sincronicen en tiempo real entre los usuarios.[21]

MongoDB es una base de datos orientada a documentos, clasificada como NoSQL (no-relacional). MongoDB utiliza documentos tipo JSON con esquemas opcionales. Es una base de datos open-source escrita en C++, Go, JavaScript y Python. Es una aplicación mejor orientada a aplicaciones de gran escala.[22]

2.4.2.3. Software de programación

JavaScript es un lenguaje abierto, multiplataforma orientado a objetos para la creación y personalización de aplicaciones en Internet. Es el lenguaje por excelencia de cara al cliente, debido a la posibilidad de crear aplicativos que conectan objetos y recursos tanto en cliente como en servidor. Su principal objetivo es la mejora de la interacción con el usuario, significando un uso predominante en interfaces de usuario y páginas web dinámicas. Es utilizado por plataformas muy conocidas, como Google, YouTube, Facebook, Amazon, entre muchas otras.[23]

HTML es un lenguaje de marcado para documentos, diseñado para ser representado en un navegador web. Generalmente, trabaja de la mano con CSS y JavaScript, y funciona con etiquetas. Los documentos HTML consisten en tres elementos y texto; cada elemento es denotado por una etiqueta de inicio, y una etiqueta de finalización. Cada elemento puede tener atributos, lo cual controla la forma en que los elementos funcionan.[24]

CSS es un lenguaje para describir la presentación de páginas web, incluyendo colores, la disposición y las fuentes. Permite adaptar la presentación a diferentes tipos de dispositivos y pantallas. CSS es independiente de HTML, lo cual facilita el mantenimiento de páginas y el compartir hojas de estilo entre estas, y puede ser utilizado con lenguaje de marcado XML.[25]

Node.js es un runtime (software en tiempo de ejecución) asíncrono y dirigido por eventos diseñado para construir aplicaciones de redes escalables diseñado en el ecosistema JavaScript. Node.js permite ejecutar scripts del lado servidor para producir contenido dinámico de páginas web antes

de que el usuario reciba la página. Node.js permite la unificación de JavaScript para usos de lado de cliente y lado de servidor, evitando tener que utilizar lenguajes diferentes.

NodeJS provee un bucle de eventos como una construcción de runtime y no como una librería. En Node.js no hay necesidad de llamar una línea de bloqueo para iniciar el evento; simplemente accede al bucle después de ejecutar el script de entrada y finaliza una vez no hay llamadas de regreso para realizar (similar a la versión de navegador de JavaScript).[26]

npm es un repositorio en línea utilizado principalmente para la publicación de proyectos código abierto de Node.js, y es, además, una utilidad de línea de comando para interactuar con dicho repositorio que asiste en instalación de paquetes, manejo de versiones y manejo de dependencias. Múltiples librerías y aplicaciones de Node.js son publicadas en npm constantemente, y su integración permite un uso rápido con unas cuantas líneas de comando.[27][28]

React Native es un marco de referencia de software en código abierto. Es utilizado para desarrollar aplicaciones para diferentes sistemas operativos (principalmente móviles) y desarrollos web con el marco React (librería JavaScript de Facebook para construir interfaces de usuario) y capacidades de plataforma nativas.

Las aplicaciones de React Native se escriben utilizando una mezcla de JavaScript y simbología similar a XML conocida como JSX. React Native invoca APIs nativas de renderización en Objective-C (para iOS) y en Java (para Android), así utilizando componentes del UI del sistema respectivo para verse como una aplicación "nativa". [29][30]

TypeScript es un lenguaje de programación desarrollado por Microsoft. TypeScript es un superconjunto de JavaScript que expande en las funciones de este, como con tipado estático opcional para el lenguaje, y puede ejecutar programas de JavaScript. Este lenguaje puede ser utilizado para desarrollar aplicaciones tanto para ejecución de lado de cliente como lado de servidor.[31][32]

Expo es un marco de referencia y plataforma para aplicaciones universales de React. Es una serie de herramientas y servicios construidos alrededor de React Native y plataformas nativas para auxiliar con el desarrollo, construcción, liberación e iteración rápida en iOS, Android, y aplicaciones web del mismo código base de TypeScript.[33]

2.4.3. Trabajos relacionados

Guidelines for the Design and Management of Bicycle Parking Facilities
En este trababajo podemos observarlos lineamientos en la ciudad de Toronto para los parqueaderos de bicicleta, y aunque es un borrador, muestra un trabajo necesario para una ciudad principal de Canadá. El trabajo habla desde definiciones hasta diferentes tipos de parqueaderos y de múltiples elementos como el espacio ocupado por bicicletas dependiendo de la arquitectura y disposición del parqueadero. Es un trabajo muy completo para hacerse una idea de las generalidades de los diferentes tipos de parqueo de bicicletas y sus necesidades, tanto en espacio como en requerimientos técnicos.[34]

SMART BICYCLE PARKING: Bicycle Parking Solution for the Riihimäki Railway Station Area

En este proyecto se revisa la creación del concepto de parqueadero de bicicletas para Riihimäki, una ciudad en Finlandia. A través del proyecto se investigan diferentes frentes que abarcan el concepto en relación a las necesidades de un cliente, como lo son los materiales, características técnicas del parqueadero y algoritmos para localizarlo en la vida real, llegando finalmente a un modelo 3D de un parqueadero completamente dimensionado.[35]

Capítulo 3

Desarrollo del sistema de parqueo

3.1. Requisitos del usuario

Al desarrollar un parqueadero de bicicletas, se deben tener en cuenta ciertas necesidades básicas, y adicionalmente, considerar otros retos que propone la construcción de un parqueadero inteligente sin perder de vista el potencial usuario final. Considerando la afluencia de usuarios de bicicleta en la Pontificia Universidad Javeriana Cali, se hizo énfasis en estos y sus experiencias.

Para este enfoque, se realizó una encuesta a 111 usuarios potenciales, entre los que se encuentran estudiantes y profesores pertenecientes a diversas facultades, considerando aspectos como la frecuencia de uso de parqueaderos de bicicletas públicos, así como las necesidades para determinar un buen parqueadero.

A partir del análisis de la encuesta, se encuentran algunos puntos clave para la integración del parqueadero inteligente, como el interés por conocer el estado de la bicicleta en tiempo real, el posible pago de este y las condiciones actuales de parqueaderos en la ciudad. En la encuesta pudimos ver comportamientos y deseos de los usuarios:

- **Los usuarios estarían dispuestos a instalar una aplicación en su celular para saber el estado de su bicicleta. (Pregunta 1, Sí: 92.8 %)**
- **Los usuarios usarían más la bicicleta si tuviesen lugares seguros para parquearla (Pregunta 3, Sí: 84.7 %)**
- **Muchos usuarios no aseguran sus bicicletas cuando no llevan elementos de seguridad (Pregunta 4, Nunca: 39.6 %)**
- **Muchos usuarios utilizan otros medios de transporte por falta de parqueaderos aptos para bicicleta (Pregunta 5, Sí: 84.7 %)**
- **La mayoría de usuarios encuestados preferirían una aplicación como medio de desbloqueo (Pregunta 6, Aplicación móvil: 54.1 %)**
- **Las personas estarían más dispuestas a usar bibicleta si los diferentes lugares contaran con parqueaderos con mayor seguridad (Pregunta 9, Sí: 94.6 %)**

A partir de esta encuesta y estudios donde se muestra la corelación entre la inseguridad, experiencia e importancia de parqueaderos de bicicleta[4], se definen los requerimientos funcionales para el diseño del parqueadero inteligente que hemos denominado Bistander:

- Poseer un parqueadero con mayor seguridad que los parqueaderos de bicicletas comunes.
- Contar con una aplicación para monitorear la bicicleta parqueada en tiempo real.
- Recibir notificaciones sobre el estado de la bicicleta a partir de la aplicación.

3.2. Planteamiento de solución

El parqueadero de bicicletas inteligente que se ha desarrollado se ha nombrado Bistander. Este parqueadero se desarrolla por la situación social de la ciudad de Cali, donde se ha presentado un aumento crítico de los hurtos en los últimos años, presentándose 5877 casos en el 2021, pero cabe mencionar que el sistema también es utilizable en cualquier ciudad del mundo con cambios mínimos en la aplicación, y tiene la capacidad de asegurar un gran rango de tamaños de bicicleta con una seguridad incrementada al utilizar una estructura resistente, puede avisar al usuario para acción oportuna en caso de detectarse una actividad anormal, como un movimiento fuerte de la bicicleta o la ausencia de esta, además de ser de uso individual de forma que cada parqueadero solo está asociado a un usuario por el tiempo que su bicicleta esté parqueada en él.

El parqueadero está compuesto por distintas partes, entre las que encontramos el marco del parqueadero, el pasador, el sistema electrónico, y la aplicación móvil. En la figura 3.1 se puede observar un diagrama general del sistema.

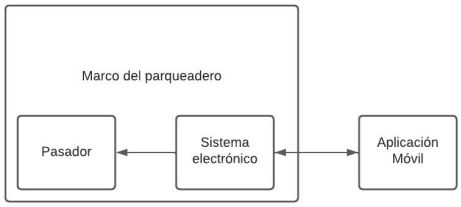


Figura 3.1: Diagrama del sistema

El marco del parqueadero se encarga de asegurar la mayor parte de la bicicleta, siendo una estructura que permite asegurar el marco y la rueda trasera de la bicicleta, y que, adicionalmente, se adapte a bicicletas de diversos tamaños dentro un rango limitado. Esta estructura se construye en un material resistente y apenas dúctil, preferiblemente metálico, evitando su desmontaje y minimizando daños.

El pasador es una parte vital del sistema, y consta de la barra que asegura el marco y la rueda de la bicicleta, atravesando ambos y finalizando en otra de las barras metálicas del marco. Es delgado, de un material rígido, y finaliza con una apertura para bloquearla.

Figura 3.2: Marco del sistema de aseguramiento

El sistema electrónico se encuentra dividido, a su vez, en dos partes: el sistema de sensores y el módulo de control. El sistema de sensores es el encargado de enviar información al módulo de control en caso de irregularidades en el parqueadero, como cualquier movimiento intenso, o, en un caso más extremo, la no detección de la rueda de la bicicleta, de esta forma siendo el encargado de la parte "inteligente" del parqueadero y la seguridad electrónica. El módulo de control se encarga de recibir información tanto de la aplicación (entradas de usuario) para el desbloqueo del parqueadero, como de los sensores para enviar información a la aplicación.

Finalmente, se cuenta con la aplicación móvil, que es la interacción con el usuario y donde este podrá desbloquear el parqueadero, así como recibir la información relevante, como la ubicación y estado de los sensores.

3.3. Software

El software es una parte importante del proyecto y comprende toda la parte de la comunicación entre usuario y parqueadero. Esta está compuesta por distintas secciones, ya que la comunicación con el usuario se hace a partir de una aplicación, más se debe acceder a una base de datos para guardar la información única del usuario y debe haber un método de comunicación para poder enviarle notificaciones en caso de detectarse anomalías en el parqueadero. En la figura 3.3 se logra ver el funcionamiento general del software del proyecto.

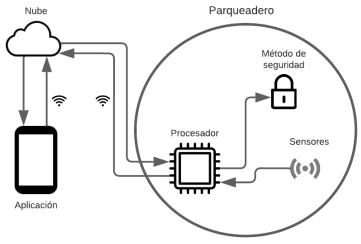


Figura 3.3: Diagrama del software del sistema

Debido a la gran variedad de tecnologías en esta sección del desarrollo del proyecto, se hizo un análisis sobre las diferentes alternativas y se seleccionó la mejor dependiendo de las necesidades que se daban en el desarrollo del proyecto. Por la gran variedad de funciones y variaciones en características, se determinó hacerlo de forma empírica; es decir, probando una muestra de esta gran variedad de programas y eligiendo el mejor entre estas.

3.3.1. Base de datos

Entre las posibles bases de datos, se consideraron distintas plataformas como lo son AWS, Firebase y MongoDB Atlas. La base de datos era necesaria para poder comunicarse con el parqueadero, ya que el usuario podía utilizar un parqueadero y debería ser desbloqueado únicamente por él (autenticación). Debido a la facilidad de prototipado y las actualizaciones en tiempo real necesarias para el tipo de aplicación, así como la inclusión de APIs OAuth que facilitan la implementación de funcionalidades de inicio de sesión, se decide trabajar con Firebase. Además, a pesar de contar con un modelo "pay as you go", así como lo hace Amazon Web Services, Firebase cuenta con un plan gratuito con una capacidad considerable. Este plan tiene por nombre "Plan Spark", y permite autenticación telefónica limitada (10 SMS al día), 50.000 usuarios activos por mes, 1 Gb de almacenamiento en la nube y miles de operaciones de escritura, lectura, eliminación de documentos, carga y descarga por día. Firebase se utiliza tanto para la base de datos como sus funcionalidades adicionales, para poder enviar notificaciones en el aplicativo al usuario y la autenticación.

3.3.2. Front-End

Una de las herramientas principales utilizadas para el desarrollo de la aplicación fue Node.js, que facilita la configuración del entorno de desarrollo y es una herramienta ideal para servidores conducidos por eventos, con capacidades push en tiempo real. Una vez se cuenta con Node.js, se hace uso de la herramienta de gestión de paquetes npm (Node Package Manager) que permite la instalación de múltiples componentes a través de un repositorio, tanto librerías como diferentes dependencias. Para la parte interactiva (front-end) de la aplicación, se utiliza React Native junto con Typescript y Expo, un marco de referencia para aplicaciones de React que agiliza su creación que se instala a través de comandos de forma global en el sistema. Para la autenticación del usuario y almacenamiento de información generada por el parqueadero se utilizan las capacidades adicionales de Firebase, instalando las librerías de compatibilidad directamente desde npm (de manera local). Para la funcionalidad del mapa se utiliza React Native maps, siendo compatible con el aplicativo y pudiéndose mostrar de forma rápida.

3.3.3. Back-End

Para el lado del servidor, se utilizó como herramienta principal FireBase. Se debe asignar un nombre al proyecto y una plataforma de trabajo, que en este caso será Expo. Expo trabaja como aplicación web, así que se selecciona la opción Web y se selecciona adicionalmente la opción de

Android para poder manejar la parte de las notificaciones en dicho sistema operativo. Una vez seleccionado, se dan las credenciales del nuevo proyecto, las cuales deben ponerse en la aplicación, en un archivo de configuración.

Con el fin de construir el back-end se utiliza next.js, que cuenta con la opción de crear APIs (Application Programming interface; Sistema que permite intercambiar información entre dispositivos o aplicaciones) a pesar de enfocarse en el front-end. El microcontrolador hace uso de esta API para enviar la información que se almacena al crear una función cloud con node.js que permite recibir los datos y almacenarlos en Firebase posteriormente. Se utiliza npm dentro de node.js para la creación del proyecto, para instalar la herramienta global -create next app, además de la carpeta llamada API, donde se guardan datos que quedan disponibles como vínculos para envío de información, y el servidor se encarga de recibirla.

Adicionalmente a esto, se creó un archivo llamado send data, de forma que, al correr el servidor, queda una URL que es el dominio que se está utilizando en el momento. Esta URL es la utilizada para enviar los datos.

Se debe crear, finalmente, un archivo de configuración donde se almacenarán las credenciales del proyecto para finalmente unir el front-end con la autenticación y la información guardada; se guarda el archivo y se usan las funciones de la librería Firebase para hacer la conexión y así se genera conexión para envío de datos a la aplicación.

3.3.4. Firebase authentication

Firebase Authentication provee servicios de back-end, SDKs fáciles de utilizar, y librerías de UI finalizadas. Soporta la autenticación utilizando claves, números de celular e identificadores de proveedores federados como Google, Facebook y Twitter, entre otros. Esta función se integra con facilidad con otros servicios de Firebase, y aprovecha estándares tales como OAuth 2.0 y OpenID Connect, así que puede ser fácilmente integrado con un back-end personalizado.

Entre sus capacidades clave se encuentra la solución de autenticación directa, un sistema completo de autenticación directa. También encontramos el SDK basado en correo y contraseña que permite la recuperación por medio de correo de restablecimiento, integración con proveedores de identidad federada, autenticación con número de teléfono, integración de sistemas de autenticación personalizados y autenticación anónima, todo como funciones separadas para desarrollarlo de la mano con un back-end propio realizado en lenguajes tales como JavaScript, entre otros.

3.3.5. Tecnologías de comunicación

Se cuenta con diferentes alternativas para comunicar los dispositivos del parqueadero con el usuario, previamente mencionados en el marco teórico; RFID, códigos QR, NFC, Bluetooth y Wifi. A partir de ellos, se hace una matriz de selección para elegir la mejor opción.

Criterios	RFID	QR	Wifi	NFC	Bluetooth
Necesidades para implentación	•Tags RFID •Conexión Wifi	•Código QR •Lector QR •Conexión Wifi	•Conexión Wifi	•Lector NFC (Disponible en algunos móviles) •Tag NFC •Conexión Wifi	•Conexión Bluetooth
Alcance	LF: <20cm HF: <50cm UHF 1 – 5 m Tags activos: 100 m – 150 m	Pocos centímetros (lectura por cámara)	Cualquier lugar con Wifi)	Pocos centímetros	Variable. Limitado por conexión directa al dispositivo.
Consumo	Pasivo: muy bajo Activo: relativamente alto.	Ninguno.	Bajo.	Ninguno.	Bajo con Bluetooth LE.
Dificultad de implentación	Media	Baja.	Baja	Media-alta	Baja
Dificultad de uso (usuario)	Media-alta: Requiere mayor interacción del usuario, y una aplicación.	Media: Requiere lectura del código QR.	Media-baja: Solo requiere el uso de la aplicación.	Media: Requiere el uso de NFC y un móvil o dispositivo adicional con NFC.	Media-baja: Solo requiere el uso de la aplicación.
Pros	•Mayor seguridad	•Simple de instaurar. • Método conocido y fácil de usar.	•Bajo costo. •Integrado en procesadores como el ESP32. •Método conocido y fácil de usar.	•Alta seguridad	•Método conocido y fácil de usar.
Contras	•Necesidad de tags •Mayor consumo de energía •Necesidad del usuario de llevar un dispositivo adicional •Costo de tags activos es alto.	•Necesita conexión Wifi •Necesita lector QR.	•Requiere de una aplicación. •Requiere de una forma de identificar al usuario por medio de la aplicación	•No es económica. •Complicada de utilizar (problemas para ubicar el tag)	•Rango limitado.

Cuadro 3.1: Comparación de diferentes alternativas

Finalmente, se decide usar únicamente una aplicación móvil con conexión Wifi para manejar todo el proceso; desde el desbloqueo del parqueadero, hasta el envío de las notificaciones de alarma. La conexión Wifi otorga una mayor versatilidad de uso y una conexión a larga distancia, permitiendo notificar al usuario aún si se encuentra lejos del parqueadero. No es imprescindible agregar otra tecnología si el desbloqueo se administra desde la aplicación; también es importante notar que los resultados de la encuesta realizada a usuarios de la javeriana en el año 2019, en la cual se votaba por el método de seguridad preferido para retirar la bicicleta del parqueadero, guiaban al uso de una aplicación móvil por conveniencia de cara al usuario, con un 54.1 % de votos a favor de esta opción frente al 25.2 % de la segunda.

En caso de que no haya conexión, se decide usar un pad numérico para comunicación directa con la aplicación en caso de que no haya conexión que permita el acceso a la base de datos.

Igualmente, cabe considerar que para conseguir una funcionalidad completa y factible comercialmente, es posible que se requieran desarrollos posteriores a la elaboración de este trabajo, para hacer del parqueadero más seguro, más práctico, o menos propenso a problemas en su operación, ya que esto influiría en el alcance del proyecto actual; esto incluiría pruebas en campo de un prototipo en términos de resistencia, pruebas de elementos necesarios o sobrantes de la aplicación, y pruebas de resistencia del material a los elementos naturales, los cuales requieren de recursos adicionales.

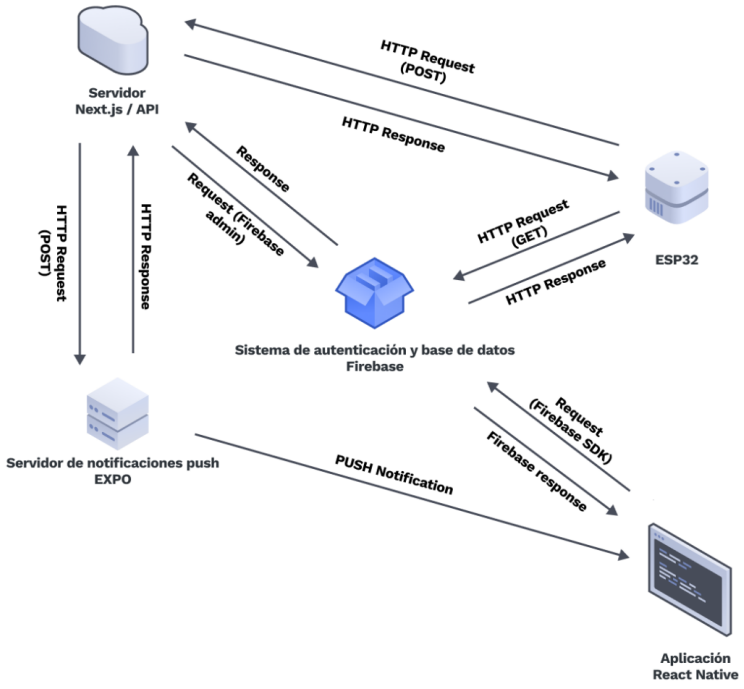


Figura 3.4. Arquitectura del software del sistema.

3.3.6. Interfaces de visualización

Para las interfaces de visualización se consideró la integración de una página web, al igual que una aplicación de celular, para la interacción con el usuario. Se consideran, entonces, una variedad de factores; facilidad de programación, costo en términos de tiempo, y practicidad. Comenzando por la página web, se logra identificar una variedad de lenguajes para su programación. Entre los más comunes para el Front-end está JavaScript, HTML y CSS. Alternativamente, tenemos la aplicación para celular, que se puede programar en una variedad de lenguajes, como Python, Swift, JavaScript, entre otros. Finalmente, se elige desarrollar una aplicación, mas no una página web, utilizando TypeScript, que es una extensión de JavaScript en cuanto a que permite utilizar sintaxis adicional y una mayor instrumentación. Se decide utilizar, a su vez, React Native, que se enfoca en la creación de interfaces de usuario para sistemas operativos móviles, como iOS y Android. La razón para no crear la página radica en la funcionalidad del parqueadero, de forma que se requiere algo móvil y la página sería un desarrollo adicional poco necesario.

3.4. Hardware

En esta sección se explica el uso, función y proceso de selección que tienen los componentes que conforman el parqueadero y se encargan de la seguridad de la bicicleta. Estos determinan la parte "inteligente" del parqueadero y son un requerimiento para cumplir con los objetivos base.

3.5. Selección de hardware

A continuación, se muestran las opciones posibles tanto para los sensores como el procesador requerido por el proyecto. Debido a tener unas características más fijas y compartir múltiples funciones, se decidió por una matriz de selección considerando especificaciones tales como el precio, memoria, información disponible, entre otros.

3.5.1. Procesador

Se utiliza un SoC (System on a Chip, o Sistema en Chip) para controlar y procesar la información que llega de los sensores. Igualmente, se utiliza para la comunicación con la aplicación por medio de Wi-Fi o algún otro método inalámbrico.

En el parqueadero, el procesador es el eje central, de forma que tiene que recibir la información de los sensores con los que este cuenta, procesarla, y enviarla por medio de una conexión inalámbrica (en este caso Wifi) a la aplicación, para lograr notificar al usuario sobre el estado de la bicicleta en tiempo real. Adicionalmente, el procesador tiene la conexión al pad numérico. El procesador maneja su propio código, el cual se programa en Arduino IDE, diferente al de la aplicación que se realiza para el parqueadero. Este código se considera back-end de la operación.

3.5.2. Selección del procesador

Criterios	ESP32	ESP8266	MKR1000	NRF52
Costo (COP)	$25 - 30,000	$10-20,000	$140 - 170,000	$130 – 160,000
No. de puertos	36 GPIO	17 GPIO	8 (I/O)	20
Memoria FLASH	4 Mb +	4 Mb	256 kb	512 kb
Memoria SRAM	520 kb	<50 kb	32 kb	64 kb
Alimentación	2.5V - 3.6V 80mA	2.5V - 3.6V 80mA	3.3V	1.7V - 3.6V
Conectividad	Bluetooth, WiFi	WiFi	WiFi	Bluetooth, NFC
Documentación	En demasía	En demasía	Suficiente	Escasa

Cuadro 3.2: Criterios de selección del procesador

Por disponibilidad, precio, conectividad y documentación disponible, se decidió utilizar el ESP32 en alguna de sus múltiples variaciones. Es una herramienta útil que además da un rango de operabilidad en caso de querer extender las funciones. El ESP32 es, además, fácil de conseguir en el país por un costo reducido y cuenta con más pines que el 8266, que también cuenta con el beneficio previamente mencionado.

3.5.2.1. ESP32-WROOM-32

El ESP32-WROOM-32 es un módulo cuyo núcleo es el chip ESP32-D0WDQ6, diseñado para ser escalable y adaptable. La frecuencia de reloj de la CPU es ajustable desde 80 MHz hasta 240 MHz. El chip cuenta con un co-procesador que puede ser utilizado en lugar de la CPU para ahorrar potencia cuando realiza tareas como monitorear periféricos. Este dispositivo cuenta con conexión Bluetooth y Wi-Fi. La corriente en standby del ESP32 es de menos de 5 μA, permitiendo ser utilizado en aplicaciones con baterías. Soporta un rango de datos hasta los 150 Mbps, y una antena de potencia de salida de 20 dBm para asegurar un largo alcance físico. Cuenta con el sistema operativo freeRTOS.[36]

Figura 3.4: Módulo ESP32

3.5.3. Acelerómetro

Un acelerómetro permite cumplir con múltiples funcionalidades, como la medición inercial de velocidad y posición, medida de vibración e impacto, medida de la gravedad para determinar orientación, entre otras. El acelerómetro permite medir el campo de gravedad estático, de forma que, al cambiar su inclinación, cambia el vector de aceleración.

Uno de los métodos de seguridad del parqueadero consta en la detección de movimientos significativos detectados en él, sea por interacciones indebidas directamente con el parqueadero, o con la bicicleta. Para detectar este movimiento, nos encontramos con dos tecnologías que realizan una función similar; el sensor de vibración y el acelerómetro. Se decidió utilizar el último debido a que su ubicación estratégica, casi en contacto con la barra aseguradora del marco y la rueda de la bicicleta, nos permite asegurarnos de enfocarnos en ciertos ejes del movimiento, así necesitando umbrales específicos para notificar al usuario. Al centrar nuestra atención en el movimiento vertical y horizontal, descartamos movimientos por causa del tráfico, construcciones, u otras eventualidades cercanas. De manera adicional, se ve que la disponibilidad de acelerómetros es mayor en Colombia.

3.5.4. Selección del acelerómetro

Criterios	**ADXL345**	**MMA8452Q**	**ADXL335**	**MMA7361**
Costo (COP)	$13,000	$14,800	$19,000	$14,500
Ejes	3 ejes	3 ejes	3 ejes	3 ejes
Interfaz	I2C +	I2C	Analógico	Analógico
Rango de detección	±2g, 4g, 8g, 16g	±2g, 4g, 8g	±3g	±1.5g, 6g
Filtro	No	Sí, paso alto (opcional)	No	No
Resolución	13 bits	12 bits	N/A	N/A
Voltaje de operación	2.0V - 3.6V	1.95V - 3.6V	3.3V - 5V	2.2V - 3.6V
Consumo	0.1 µA – 23 µA	6 µA – 165 µA	350 µA – 3 mA	3 µA – 0.5 mA
Programabilidad	Programable	Programable	No	No
Documentación	Suficiente	Suficiente	Suficiente	Suficiente

Cuadro 3.3: Criterios de selección del acelerómetro

Ya que la documentación de los distintos acelerómetros es suficiente, se decidió determinar por otros factores, como la resolución y la capacidad de programación. El ADXL345, con su alta resolución (necesaria para el sistema), bajo costo, muy bajo consumo y diferentes rangos de detección, se seleccionó como el mejor candidato. Para poder limitar las notificaciones al usuario exclusivamente a intentos de robo, se filtra la información recibida desde el acelerómetro de forma que esta solo se

dispare cuando hay demasiado movimiento; así se logra evitar confusiones con movimientos naturales (como alguien usando un parqueadero adyacente o un vehículo haciendo vibrar la acera) de manera digital, ya que no cuenta con un filtro como lo hace el MMA8452Q, dispositivo que, además, es más difícil de encontrar.[37]

3.5.4.1. Acelerómetro ADLX345

El ADXL345 es un acelerómetro de tres ejes pequeño, delgado, de consumo muy bajo (23 μA en modo de medición y 0.1 μA en modo de standby) y con alta resolución (resolución fija de 10-bit y resolución completa de hasta 13 bits, midiendo hasta ± 16g). Es adecuado para uso en aplicaciones móviles y su resolución permite medir inclinaciones de hasta 1°. Cuenta con funciones especiales como la detección de actividad e inactividad al verificar la presencia o falta de movimiento, comparando la aceleración en cualquier eje con umbrales establecidos por el usuario. Detecta toques en cualquier dirección, así como permite conocer si el dispositivo se encuentra en caída libre. Cuenta con un sistema de manejo de memoria que puede ser utilizado para guardar datos para minimizar la actividad del procesador host y disminuir el consumo total de potencia. Trabaja en temperaturas desde los -40°C hasta los +85°C.[38]

Figura 3.5: Módulo ADXL345

3.5.5. Sensor de proximidad

Los sensores de proximidad pueden funcionar con distintas tecnologías. Una de las más conocidas es la tecnología óptica, que a su vez se divide en otras ramas, como los sensores fotoeléctricos por barrera de luz, los sensores de tipo retro reflectivo, y los sensores de reflexión difusa. También se encuentran los sensores inductivos, que solo detectan objetos metálicos, los sensores capacitivos, que detectan todo tipo de objetos al observar la diferencia de capacitancia, y los sensores ultrasónicos, que detectan la presencia de los objetos emitiendo frecuencias en el rango ultrasónico y midiendo el tiempo de su retorno.

El sensor de proximidad, en el caso del parqueadero inteligente Bistander, es el encargado de verificar que la rueda de la bicicleta se encuentre en su lugar, implicando por la forma del parqueadero

que la bicicleta está en su lugar. Es una forma de otorgar redundancia a la seguridad del parqueadero y se encontrará en la parte inferior del parqueadero, con vista directa a la rueda trasera.[39]

3.5.6. Selección del sensor de proximidad

Criterios	VL6180	RCWL0516	APDS-9930	TMD27713
Costo (COP)	$42,000	$8,000	$14,000	$15,000
Forma de sensado	Óptico (Con tecnología FlightSense) *	Microondas	LED Infrarrojo	Radar
Rango de sensado	0cm – 62cm	5m – 7m	0mm – 100mm	360°, 6m – 9m
Salida	Digital I2C	Voltaje análogo	Digital I2C	Digital I2C
Voltaje de operación	2.6V – 3.0V	3.3V	2..2V – 3.6V	3.3V – 20V
Consumo	<1µA – 1.7mA	3mA	2.2 µA – 90µA	<3mA
Documentación	Suficiente	Suficiente	Escasa	Escasa

Cuadro 3.4: Criterios de selección del sensor de proximidad

Para la aplicación en cuestión, se requiere de un sensor preciso para distancias cortas, unidireccional, que detectara la cercanía de la rueda de la bicicleta. Por disponibilidad y bajo consumo en estado de espera, así como contar con una salida digital necesaria para comunicarse con el procesador, se eligió el VL6180, que cuenta además con una tecnología que evitaría problemas al encontrarse con llantas de materiales de reflectancias diferentes.

3.5.6.1. Sensor de proximidad VL6180

El sensor VL6180 es un sensor de proximidad que funciona a pesar de la reflectancia del objeto a detectar. El VL6180 mide el tiempo que toma a la luz viajar desde el objeto más cercano hasta el sensor, combinando un emisor infrarrojo y un sensor de rango en un paquete único. Está diseñado para funcionar con baja potencia (menor a 1 μA en standby y 1.7 mA en uso). Funciona con un voltaje de 2.6 a 3V y mide hasta una distancia de 62 centímetros (dependiendo de las condiciones).[40]

Figura 3.6: Módulo VL6180

3.5.7. Pad Numérico

Se utiliza un pad numérico genérico para tener una forma de interactuar con el parqueadero que no requiera de conexión inalámbrica o internet. La aplicación genera una serie de cuatro números aleatorios una vez separado el parqueadero, entonces se permitiría desbloquear en caso de que se pierda la conexión WiFi del ESP32 o simplemente se desee desbloquear sin tener que sacar el celular.

3.5.8. Cerradura

Una cerradura es un mecanismo metálico que se fija en diferentes lugares con la función de cerrar, por medio de uno o más pestillos. Para la seguridad del parqueadero, se hace indispensable tener una forma de asegurar la barra que atraviesa el marco y rueda de la bicicleta; por la naturaleza del sistema, se nombra como "pasador" al sistema barra-cerradura.[41]

3.5.8.1. Cerradura Solenoide 12V (genérica)

Una cerradura solenoide está diseñada para trabar y destrabar. Esta cerradura se desbloquea al darle energía, de forma que se encuentra cerrada hasta recibir la señal de apertura. Este modelo cuenta con una demora de disparo de 1 segundo, está compuesto de acero niquelado y tiene un recorrido de pestillo de 10 mm. Para casos de falla energética, la cerradura quedará cerrada, así asegurando la seguridad de la bicicleta; para poder tener acceso a la bicicleta en estos casos, se podrían instalar baterías en los parqueaderos.

Figura 3.7: Cerradura Solenoide 12V

Capítulo 4

Resultados y Discusión

4.1. Visión global del diseño

En esta sección se explican los distintos componentes que conforman el diseño del parqueadero. El diseño debía cumplir con ciertos requerimientos establecidos tanto por usuarios como creadores; ser visualmente atractivo, ser funcional, ser resistente y no ocupar un espacio excesivo. Para la estructura exterior se pensaron diferentes alternativas, evaluando diferentes formas y tamaños. La mayor dificultad se encontró al tomar en cuenta que un parqueadero de bicicleta comercialmente viable tendría un tamaño reducido, así como una posibilidad de juntar dichos parqueaderos. Se decidió, entonces, tratar con una estructura que se pudiese conectar de alguna forma, y de un aspecto que no chocara con las estructuras urbanas conocidas. La estructura también tenía que cumplir, claramente, su función principal: asegurar la bicicleta de forma firme, haciendo lo posible para evitar el hurto de no solo el marco, sino también, al menos, una de las ruedas. En las próximas subsecciones, se detallan el proceso y los materiales para crear un parqueadero efectivo para las bicicletas, considerando que se ubicaría en un lugar público y de fácil acceso.

4.1.1. Forma de aseguramiento de la bicicleta

La bicicleta debe ser asegurada, y esto se puede hacer de diferentes maneras, aunque las más útiles siempre consideran el marco de la bicicleta, ya que es la pieza central y más costosa del vehículo. Con esta información, sabemos que se debe asegurar el marco, y para dar un valor adicional y diferencial, también una o dos de las ruedas.

Investigando alternativas de parqueaderos inteligentes de bicicletas, se encontraron diseños ya existentes como lo son Bikeep[42], vadeBike[43], y BikeLink[44], entre otros, contando con características diferentes, así como sensores, funcionalidades y características únicas. Bikeep asegura una de las ruedas y el marco con una estructura metálica única, ocupando un espacio reducido frente a otros parqueaderos de bicicletas inteligentes convencionales.

Se determina, inicialmente, asegurar la bicicleta con barras metálicas que atraviesen tanto el marco como la rueda trasera, logrando el objetivo de proteger partes adicionales al marco. Posteriormente, se nota la redundancia de tener dos barras, y se disminuye a 1 para ahorrar en materiales, dificultad y tamaño total del parqueadero.

Esta barra pasa a través de los radios de la rueda de la bicicleta, pasando a su vez por el marco de la bicicleta; la lógica de esta solución consta en que en el caso que intenten robar la rueda, dañarían a su vez los radios, e igual no podrían retirar el marco en caso de atacar este.

4.1.2. Diseño exterior

Se inició el proceso de diseño con una lluvia de ideas sobre cómo podría ser el diseño exterior, así como basándose en modelos ya existentes. Se decide usar como base figuras rectangulares, formando una estructura monolítica de metal. Se decidió utilizar la forma de pilar en vez de la más redondeada manejada en otros parqueaderos inteligentes para bicicletas, o en vez de la cantidad de material utilizada en lockers para bicicletas. El primer diseño realizado se puede apreciar en la figura 4.1 y cuenta con las dos barras metálicas que se iban a utilizar para asegurar la bicicleta. En la figura 4.2 se aprecia cómo se vería la bicicleta asegurada.

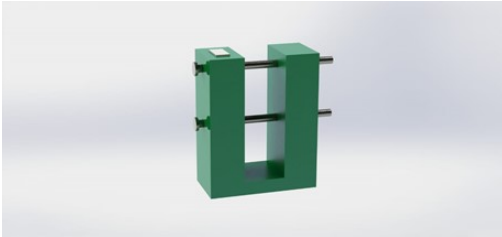


Figura 4.1: Diseño original del parqueadero con dos barras.

Figura 4.2: Bicicleta asegurada en diseño original del parqueadero.

Para evitar hacer un parqueadero muy grande e incómodo para los usuarios, se disminuyó el tamaño eliminando una de las barras aseguradoras, también debido al problema que podría generar al tener que atravesar radios de dos partes diferentes de la rueda trasera. El parqueadero resultante se muestra en la figura 4.3.

Las medidas de la bicicleta que importan para el diseño del parqueadero son la altura del marco o cuadro de la bicicleta, particularmente el espacio entre el tubo del asiento y las vainas superiores e inferiores, el ancho de la manzana donde se ubican los piñones en la parte trasera de la bicicleta,

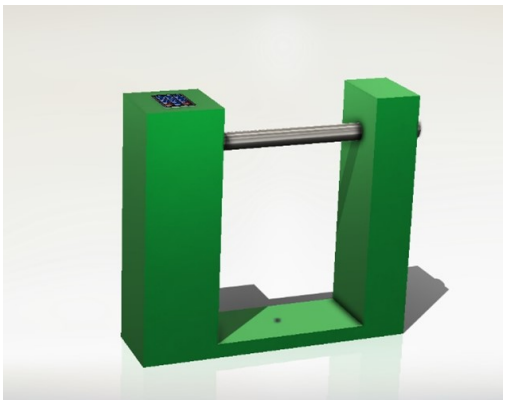


Figura 4.3: Nuevo modelo del parqueadero.

así como la distancia entre radios en la llanta trasera. Se puede ver un diagrama de las partes de la bicicleta en la figura 4.4.

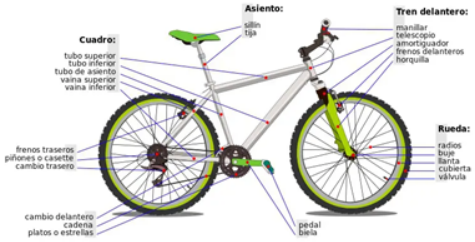


Figura 4.4: Diagrama de las partes de una bicicleta.

La mayoría de manzanas miden entre 130 y 135 mm entre sus tuercas de bloqueo, pero adicionando la tuerca que ajusta los piñones y el cambio trasero, la distancia puede incrementar alrededor de 100 mm.[45] Para dejar un margen de manejo de la bicicleta y en caso de tuercas de bloqueo alargadas (como se pudo observar en algunas bicicletas) o detalles adicionales, se tiene una distancia libre entre pilares de 27 cm.[46] Para la altura del espacio del marco y distancia entre radios se toma un promedio de mediciones hechas manualmente de un total de 20 bicicletas, así como la información disponible en línea[47]. Las mediciones entre los radios se hacían de forma que se pudiese atravesar la barra del pasador del parqueadero tanto por estos como por el marco, de forma que se hacía en un lugar muy específico donde se evitaba la curvatura de la rueda y la cadena, como se puede ver en la figura 4.5.

Figura 4.5: Área útil para medición de distancia entre radios.

- Promedio distancia entre radios (desde menor a mayor circunferencia): 3.5 a 5.5 cm
- Promedio de altura del radio útil para bloqueo: 36 cm[48]

Con esta información, se comienza con las dimensiones finales del diseño del parqueadero, considerando igualmente el recorrido de la barra del pasador dentro de los dos pilares del parqueadero. Estas medidas pueden apreciarse en la figura 4.6, en centímetros. También podemos determinar que el área de suelo que ocuparía un parqueadero es de 48 centímetros por 13 centímetros, para un total de 624 cm^2.

4.1.3. Anclado

El ancla para el sistema de parqueo se decide a partir de los posibles lugares donde se podría instalar. Debido a que el parqueadero requiere de conexión a internet, se instalaría en lugares dentro de la ciudad y dentro de edificaciones que lo requieran, por lo cual el método de aseguramiento se haría de forma universal con anclas de piso. Un ancla de piso con tornillo de 30 milímetros de ancho resiste cargas desde 160 hasta 340 kilogramos dependiendo de la longitud del ancla, así que se trabaja con dichas consideraciones. A partir de la estructura del parqueadero, se concluye que se puede utilizar más de un ancla para sostener cada esquina en su lugar, contando con un total de cuatro. Si se quiere anclar el parqueadero aún mejor, la forma del parqueadero permitiría utilizar

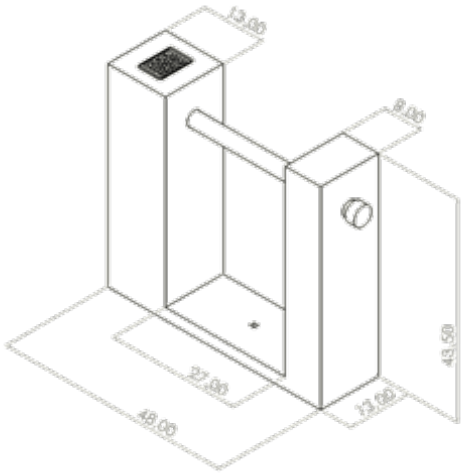


Figura 4.6: Medidas externas del parqueadero.

seis u ocho, que podría ser redundante debido el tamaño de este. Es importante considerar que las tuercas utilizadas para las anclas deben ser a prueba de manipulaciones, para evitar que se pueda atacar el parqueadero como unidad antes que la bicicleta. Previo a realizar el anclado, es necesario revisar las tuberías presentes en el sitio para poder evitar el daño de estas estructuras. Igualmente, se deben revisar los lineamientos legales de la ciudad y pedir permisos en caso de instalación en propiedad pública, ya que se requeriría taladrar en andenes.[48]

4.1.4. Materiales

El material con el que se decide hacer el parqueadero requirió de una búsqueda de información sobre los distintos materiales posibles, a sabiendas de que el metal sería la mejor opción para un modelo final por su maleabilidad y resistencia.

Debido a que hay distintos metales y aleaciones, así como variedad de costos, se decide ir con la solución comercial más económica y recomendada para este uso, el acero[49].

El acero cuenta con resistencia a cortes y dobleces, sin llegar a precios tan altos como los del titanio, ni tener la relativa fragilidad del aluminio.

La dificultad recae en los acabados, para los cuales se creó una matriz de selección:

Material	Costos	Puntos positivos	Puntos negativos	Garantia(tipica)
Acero inoxidable	Alto	•Resistente •Inoxidable •Durabilidad •Antimicrobiano	•Dificil de soldar •Costo alto •Falta de protección contra rayones	5 años
Acero galvanizado	Bajo	•Costo • Durabilidad	• Apariencia • Falta de protección conta rayones	5 años
Recubrimiento en polvo	Bajo	• Añade colores • Añade resistencia • Protección a rayones • Protección a corrosión	• Imposibilidad de reciclaje del recubrimiento	1 año
Vinilo termoplástico	Medio	• Añade colores • Protección a corrosión • Protección a rayones • Protección UV • Antimicrobiano	• Costo más alto que recubrimiento en polvo • Mayor resistencia a impactos	1 año

Cuadro 4.5: Criterios de selección del material

Con estas consideraciones, se estiman los costos finales aproximados en pesos colombianos (COP), a partir de los precios encontrados en línea en sitios como Mercadolibre.com, Amazon.com, Alibaba.com, Bricometal.com, entre otros:

- Módulo ESP32: $30.000
- Acelerómetro ADXL345: $13.000
- Sensor de alcance VL6180: $50.000
- Cerradura solenoide 12V: $40.000
- Estructura (acero galvanizado 3mm espesor): $306.112 (10 láminas de 48 x 13 cm)
- Anclado *8 (costo de compra al por mayor): $2,000 c/u ->$16,000
- Varilla de acero: $140.000

Esto nos da un total estimado de $581.112,00 por parqueadero únicamente en partes y materiales.

4.2. Consideraciones de diseño

Para el sistema en cuestión, se trabaja inicialmente con un prototipo de madera debido al alto costo de los materiales para un modelo funcional, lo cual influye en el grosor de algunas partes del parqueadero, así como el movimiento y flexibilidad que este experimenta.

La resistencia de los materiales, así como las pruebas estructurales del parqueadero salen del espectro del proyecto actual y son aspectos para desarrollar en trabajos futuros, así como una forma de evitar el alto rango de movimiento con el que cuenta una bicicleta parqueada en el parqueadero, ya que no se asegura actualmente la rueda delantera.

Originalmente se planeaba utilizar un módulo GSM para el envío de los mensajes de movimientos extraños al celular del usuario, para lo cual se seleccionó un módulo GSM GPRS SIM800l. Se hicieron pruebas iniciales en las cuales el módulo funcionaba perfectamente, pero después de un tiempo los mensajes dejaron de llegar; se probaron distintas soluciones sin llegar a una conclusión concreta de por qué no llegaban, probando tanto distintos códigos como con antenas de mayor efectividad, hasta darnos cuenta de un factor externo que influyó en esta funcionalidad: El Ministerio de Tecnologías de la Información y las Comunicaciones (MinTIC) anunció la desconexión de redes 2G de forma definitiva para el mes de Junio de 2021.

La señal 2G era necesaria para la funcionalidad SMS que se le quería dar, por lo cual se retiró completamente ya que utilizar un módulo que funcionara con redes de nueva generación impactaría el costo de los parqueaderos, así como la implementación.

Asimismo, se hizo un cambio en las herramientas para la creación de la aplicación; originalmente se empezó a crear con el MIT App Inventor, pero sus utilidades eran limitadas y a pesar de no necesitar casi experiencia ni código, limita la escalabilidad y la capacidad de cambios una vez ubicado un elemento en la interfaz. Por añadidura, dificultaba la entrada de usuarios y había cierta incompatibilidad visual con las imágenes con la función de cambio automático de tamaño. Por estas dificultades y otras adicionales en la integración de Firebase, se prefirió cambiar totalmente (en especial para conservar las funcionalidades requeridas de la base de datos).

Si se decide implementar más de un parqueadero, se debe tener en cuenta que, por el diseño de cada parqueadero, deben ser ubicarlos de manera en que las barras de bloqueo puedan moverse sin problema alguno; esto implica que se pueden hacer grupos sin dejar espacio entre ellos de 4 parqueaderos juntos. En cuanto a la cantidad de ESP32, es necesario utilizar uno para cada parqueadero, ya que los sensores y el teclado numérico ocupan cada uno de los pines del ESP32 y los dos núcleos de este.

Para una operación optima del sistema se debe asegurar que tenga cerca un router el cual les de una cobertura a todos los módulos ESP32 que se utilizaran en los parqueaderos, de esta forma asegurando una buena comunicación del parqueadero con el servidor y brindando una buena experiencia al usuario. También se debe ubicar el parqueadero en una zona donde la cobertura celular sea buena para que así los usuarios puedan hacer uso de la aplicación.

4.3. Resultados generales

Finalmente, se consiguen resultados en las pruebas del proyecto. El parqueadero, a través del correcto funcionamiento de la aplicación, se puede controlar remotamente y notifica al usuario cuando ocurren las eventualidades de movimiento excesivo del parqueadero y cuando no hay presencia de la bicicleta.

El relé, como se ha explicado con anterioridad, es el que se encarga del bloqueo o desbloqueo del parqueadero, activando el mecanismo de seguro de este. Una vez la barra metálica que asegura el eje y el cuerpo de la bicicleta se encuentra en el lugar y se manda la señal desde la aplicación, el seguro se activa, bloqueando la barra en la pestaña que fue creada para ese propósito.

Las notificaciones de alerta y la capacidad de desbloqueo se aplican únicamente al usuario que solicitó el parqueadero en primer lugar, para poder hacer de la seguridad de la aplicación algo verdaderamente útil.

El correcto funcionamiento del parqueadero requiere de una fuente de energía constante, así como acceso a redes inalámbricas WiFi, ya que la comunicación con la aplicación desde el ESP 32 es vital para cumplir con todas las funciones básicas de *Bistander*.

Se logra concluir el proyecto con un prototipo funcional, con una aplicación completa y la posibilidad de volverse un producto completo una vez se hagan desarrollos en la parte mecánica y se establezca el funcionamiento en lugares públicos.

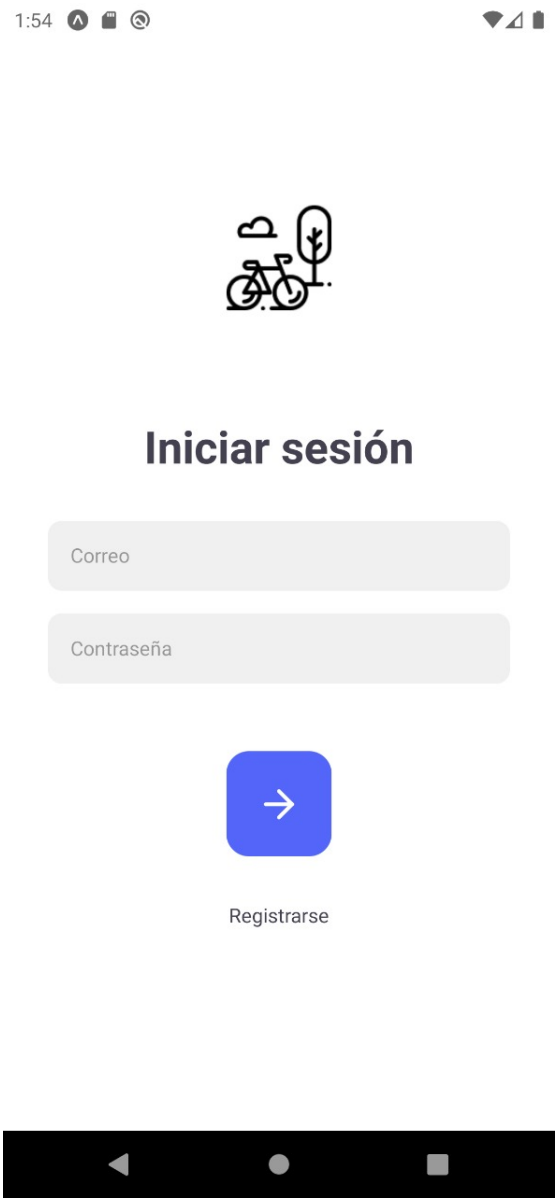


Figura 4.7: Inicio de sesión

La Figura 4.7 muestra el inicio de sesión. Este es necesario para atribuir el valor único que identificará al usuario al parquear su bicicleta, de forma que sea sólo esta persona quien pueda desbloquearlo una vez lo utiliza. Como podemos observar, también es posible acceder al registro desde esta pantalla.

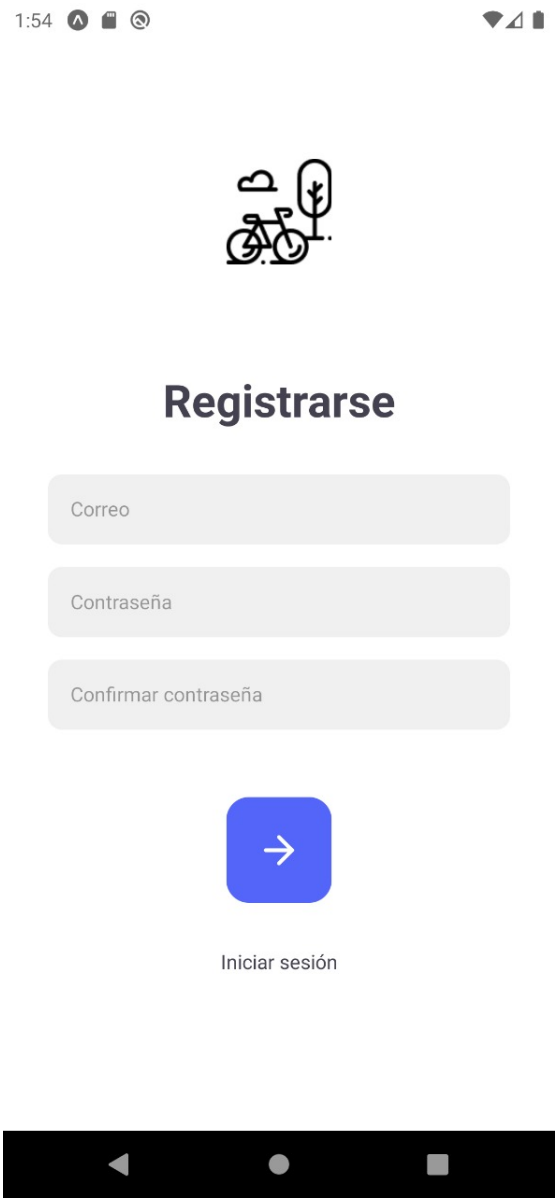


Figura 4.8: Registro de usuarios

En la figura 4.8 se puede observar la pantalla de registro. Aquí se podrá registrar el correo y crear una clave para acceder en ocasiones futuras a la aplicación.

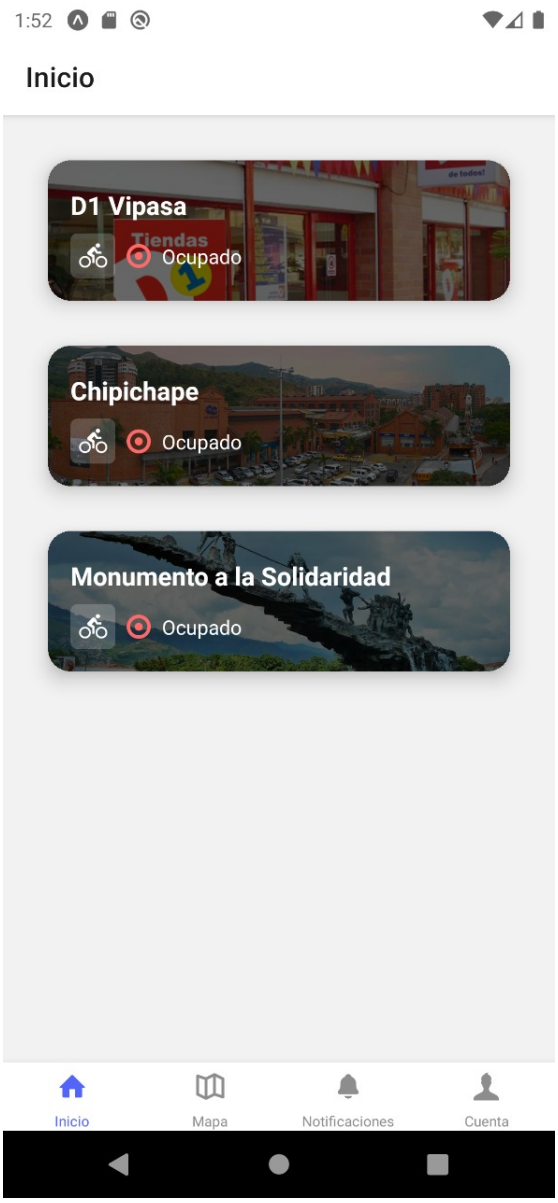


Figura 4.9: Pantalla de inicio del usuario

En la pantalla de Inicio (Figura 4.9) se muestra lo primero que verá el usuario al acceder a su cuenta. Aquí se encuentra la información de los parqueaderos, que se debe modificar por parte de los programadores al instalar nuevos parqueaderos, ya que registra si hay disponibilidad en las diferentes ubicaciones de la ciudad. Se puede observar una imagen de fondo representativa para las ubicaciones, y desde esta pantalla se puede acceder a las pantallas de mapa, notificaciones y cuenta.

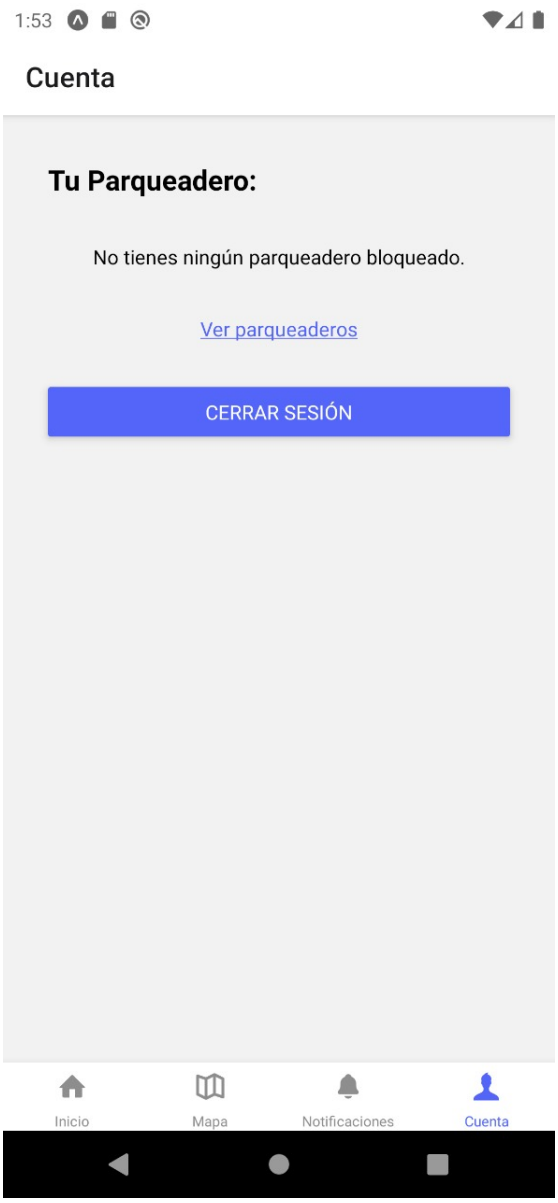


Figura 4.10: Cuenta del usuario

En la figura 4.10 se puede ver la pantalla donde el usuario tendrá la información de si el usuario tiene algun parqueadero en uso y también el boton para cerrar la sesión del usuario.

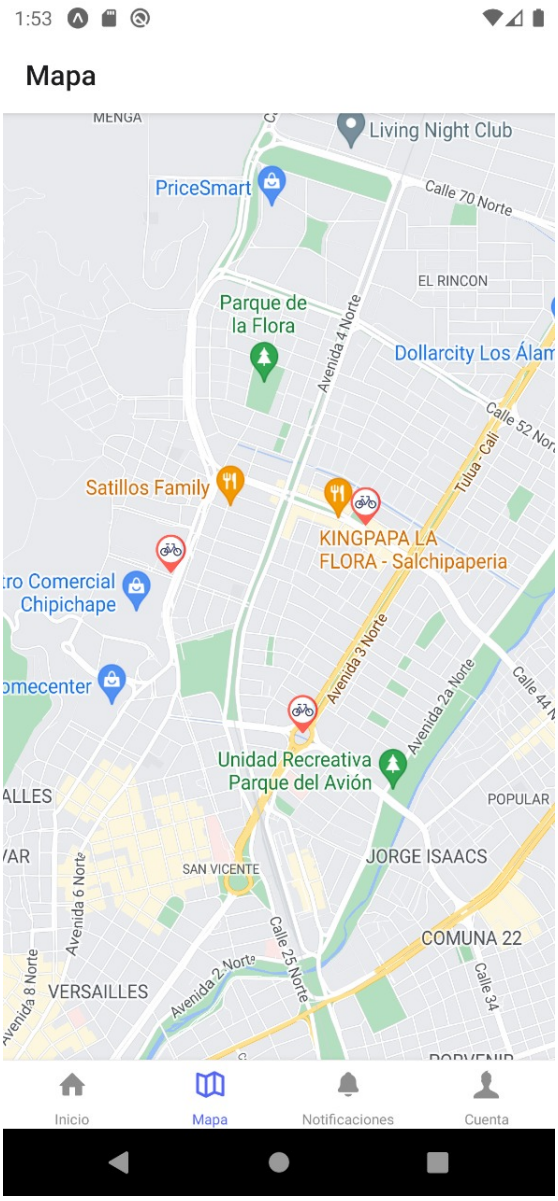


Figura 4.11: Ubicación de los parqueaderos

En la figura 4.11 se observa la la ubicacion de los parqueaderos disponibles en las diferentes zonas donde esten ubicados. Este mapa se puede actualizar fácilmente con la ubicación de nuevos parqueaderos.

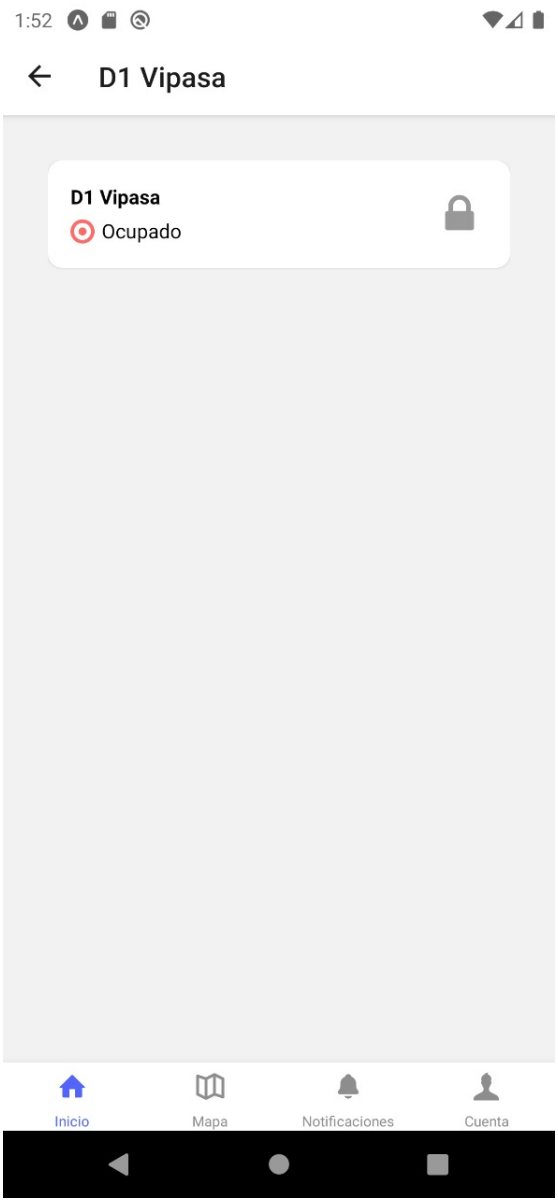


Figura 4.12: Parqueadero utilizado

La figura 4.12 muestra el parqueadero que esta utilizando el usuario, el estado de este y el acceso al bloqueo o desbloqueo del sistema. En el estado puede verse una imagen que lo acompaña, indicando así si está ocupado por algún otro usuario o nadie lo tiene en uso. El título del parqueadero es la ubicación de este o algo representativo que lo identifica.

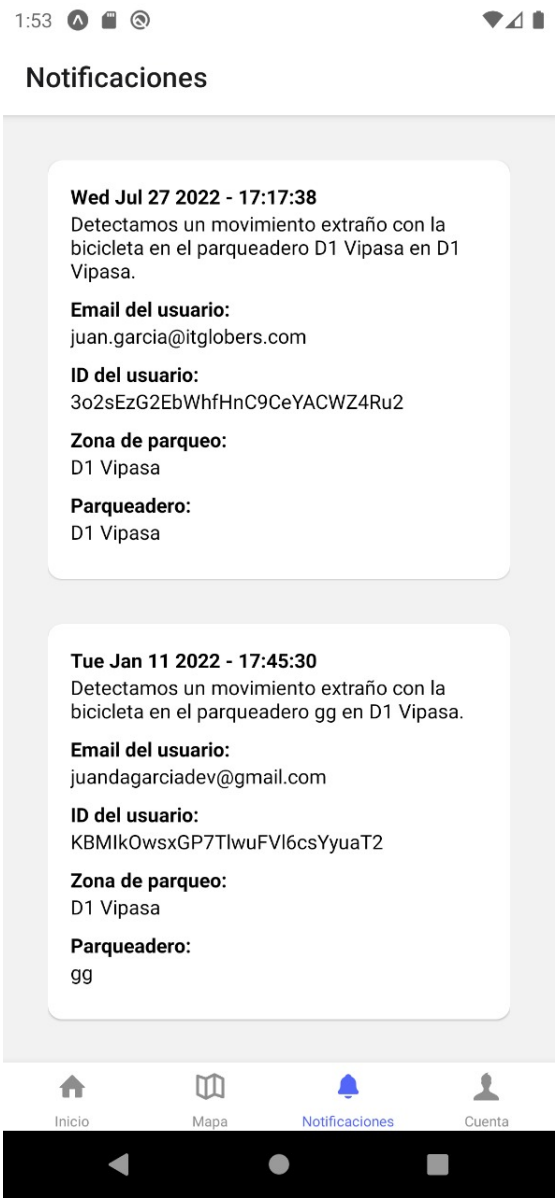


Figura 4.13: Notificaciones

En la pantalla de notificaciones se muestra las diferentes alertas que el sistema envía al usuario. En la figura 4.13 vemos dos notificaciones, ambas indicando movimiento extraño de la bicicleta, lo cual es una señal recibida por el acelerómetro ADLX345, al recibir fuertes vibraciones que pasan el umbral. Esto podría indicar que alguien está forzando el parqueadero.

Capítulo 5

Conclusiones

A través del presente proyecto se pudo llegar a una variedad de resultados; el primero y más importante es el sistema completo del parqueadero de bicicletas, comprendiendo así las dificultades en el diseño de un sistema complejo comprendido por una parte mecánica así como un software que debe funcionar a la par.

Adicionalmente nos dimos cuenta de que al consultar a los usuarios, en este caso por medio de una encuesta, podemos evitar la subjetividad al momento de crear el proyecto. Así, en proyectos con aplicación comercial, se pueden eludir desarrollos innecesarios o irrelevantes para el paradigma del momento.

Después del uso de los componentes, se ha logrado establecer la necesidad de tener en cuenta elementos varios que en ambientes controlados no se suelen considerar, como la necesidad de una alimentación de energía constante, el requerimiento de materiales que permitan el paso de señales WiFi, la importancia de los cambios en las tecnologías de comunicación del país (como con el caso del 2G volviéndose obsoleto), entre otros.

Al crear la aplicación, también se pudieron identificar la gran cantidad de software que son necesarios para crear una aplicación; no solo es necesario programar una comunicación con los componentes electrónicos, sino que se requieren elementos que se conecten con la nube, otros para la interfaz de usuario e incluso algunos adicionales para la creación de contenido dinámico e interactivo. Estos programas requieren de código especial y librerías para poder tener interacción entre ellos.

En el desarrollo del proyecto se logra, a su vez, realizar un diseño, que requiere de estudio, modelado y prototipado para ser realmente efectivo. Como ingenieros electrónicos, pudimos abordar una pequeña parte de lo que es el área de la mecánica, conociendo además de materiales y su importancia, principalmente considerando que el parqueadero de bicicletas *Bistander* ha de contar con una seguridad al ser destinado para uso en lugares públicos y necesitar, como uno de sus objetivos, ser más seguro que una simple cadena como las que suelen ser utilizadas para asegurar bicicletas a falta de mejores opciones.

Capítulo 6

Recomendaciones

6.1. Materiales

Con el fin de que el sistema tenga una mayor durabilidad, resistencia y seguridad se sugiere hacer una investigación de materiales para la realización de la base del sistema, de forma que este tenga una mayor durabilidad al estar expuesto a la intemperie. También se sugiere la búsqueda de un material que cuente con una mayor dureza, para así tener una mayor resistencia a golpes, cortes, desgaste por tiempo y uso.

6.2. Respaldo

Se podría considerar el desarrollo de un respaldo adicional para lugares rurales o espacios donde no haya flujo constante de energía eléctrica. Debido a que en gran medida el parqueadero es manual, no tiene un consumo muy alto y puede ser alimentado con una batería pequeña para alimentar el módulo ESP32 y la cerradura, pero se puede considerar automatizar el pasador y esto requeriría de mayor alimentación.

6.3. Aplicación móvil

El prototipo de la aplicación que se desarrolló en esta tesis, puede mejorarse para que así cada cliente pueda personalizar la aplicación a su conveniencia. Algunas mejoras futuras podrían ser:

- Un diseño de la aplicación más amigable con el usuario.
- Mayor funcionalidad para el control del parqueadero, notificaciones, etc. por parte del usuario.

Capítulo 7

Anexos

Anexo 1 – Encuesta sobre parqueaderos de bicicletas

Se realizó una encuesta a 111 usuarios potenciales, entre los que se encuentran estudiantes y profesores pertenecientes a diversas facultades de la Universidad Javeriana, considerando aspectos como la frecuencia de uso de parqueaderos de bicicletas públicos, así como las necesidades para determinar un buen parqueadero, obteniendo los siguientes resultados.

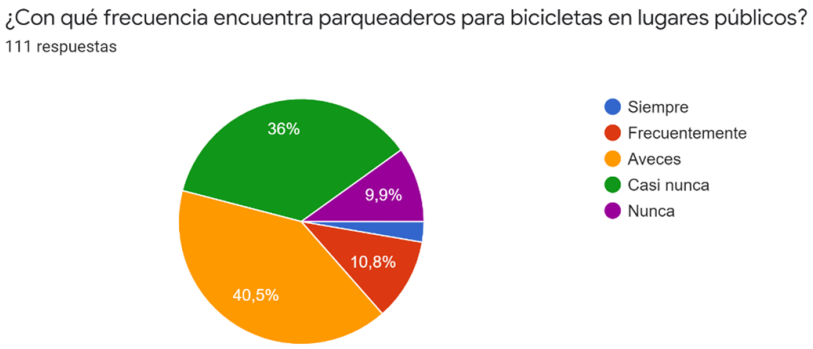


Figura 7.1: Frecuencia de uso de parqueaderos de bicicletas públicos.

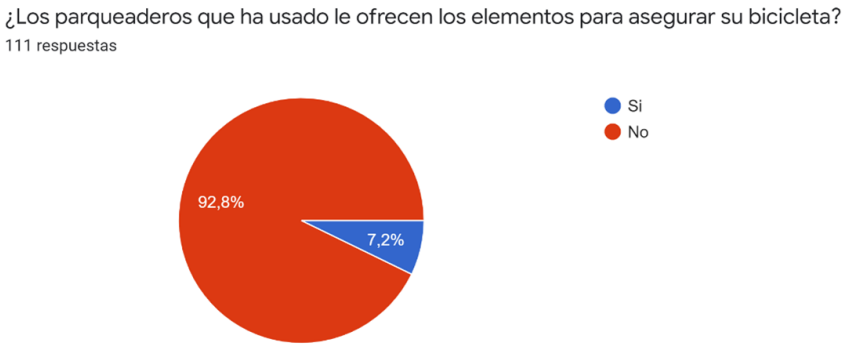


Figura 7.2: Seguridad de los parqueaderos públicos de bicicletas.

Figura 7.3: Potencial uso de la aplicación para monitoreo de la bicicleta parqueada.

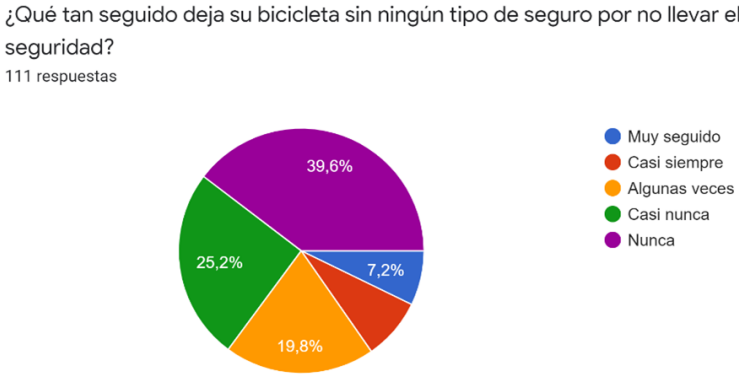


Figura 7.4: Incomodidad de uso de parqueaderos no inteligentes.

Figura 7.5: Percepción de seguridad y falta de parqueaderos de bicicletas.

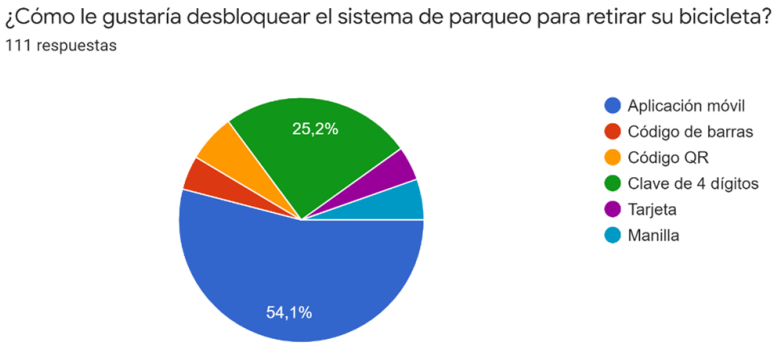


Figura 7.6: Preferencias de tecnología para desbloqueo.

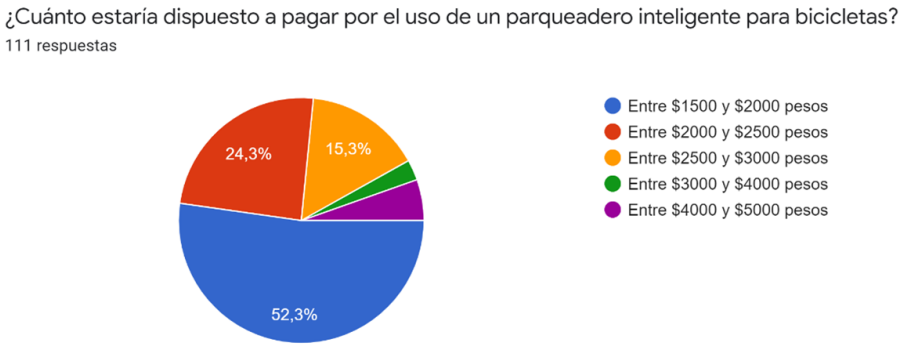


Figura 7.7: Posibles costos del parqueadero.

Esta pregunta fue descartada al ser considerada inexacta por falta de información suficiente de costos de préstamo del servicio, y al posiblemente influir en el pensamiento del encuestado al dar opciones fijas, en lugar de un espacio para que el encuestado diera una opinión más libre.

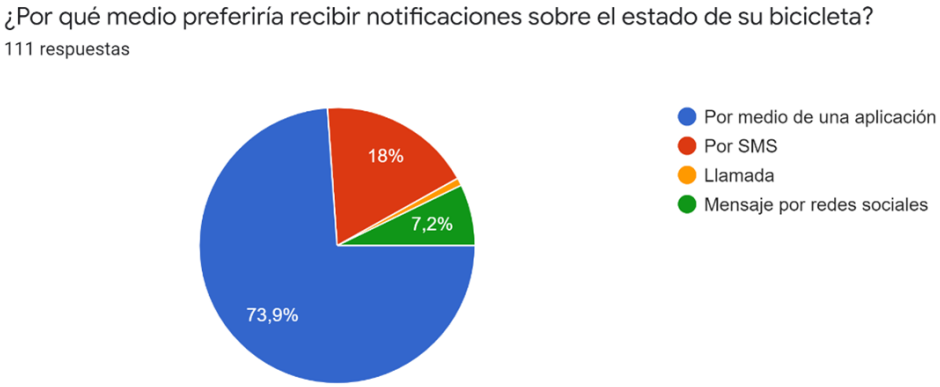


Figura 7.8: Preferencias de tecnología para desbloqueo.

Figura 7.9: Interés en el uso de un parqueadero de bicicleta inteligente.

Anexo 2 – Código de Arduino para funcionamiento de ESP32 y sensores

A continuación, se explica el programa que se debe descargar al módulo ESP32, el cual permite el intercambio de los datos de los sensores ADXL345 y VL6180X, el pad numérico y la base de datos Firebase, encargada del almacenamiento de datos, el sistema de autenticación y las notificaciones.

```
1 //Librerias
2 #include <SPI.h>
3 #include <Wire.h>
4 #include <SparkFun_ADXL345.h>
5 #include "Adafruit_VL6180X.h"
6 #include <Arduino_JSON.h>
7 #include <WiFi.h>
8 #include <HTTPClient.h>
9 #include <Keypad.h>
```

Figura 7.10: Librerías utilizadas.

Para tener la funcionalidad total del sistema se deben descargar las librerías que se explican seguidamente, permitiendo de esta manera el acoplamiento de los módulos, la aplicación y el microcontrolador.

- SPI.h: Es un protocolo de datos serial síncrono que es utilizado en microcontroladores para comunicar uno o varios dispositivos de manera rápida en distancias cortas.
- Wire.h: Esta librería permite la comunicación con el bus I2C.
- SparkFun_ADXL345.h : Con esta librería se tiene la opción de usar SPI y I2C para así poder comunicarse con el sensor ADXL345.
- Adafruit_VL618X.h Es una librería que permite la comunicación del sensor VL6180 con el microcontrolador..
- Arduino_JSON.h : La librería permite leer y procesar datos en formato JSON al microcontrolador.
- WiFi.h: Esta librería permite la conexión del microcontrolador a internet.
- HTTPClient.h: Permite que la interacción entre el microcontrolador y los servidores web sea más sencilla.

- Keypad,h: Es una librería usada para el uso de matrices de estilo teclado numérico puedan ser utilizadas con un microcontrolador.

Para guardar los datos obtenidos por los módulos, obtener y enviar información a la base de datos se crean variables.

La variable adxl se crea para guardar los datos del acelerómetro, la variable vl es la variable donde se obtienen los datos del sensor de proximidad, y, x y z son variables en las cuales se guardará el valor detectado en cada eje del acelerómetro y ParkingID es la variable en la cual se guarda el ID único de Firebase para cada parqueadero.

Adicionalmente se asigna el pin 19 del microcontrolador, para tener comunicación entre este y el módulo relé.

```
11 //Global
12 String ParkingID = "F7VhSYEn4yiGnnc0c9fw";
13 #define relay  19
14 ADXL345 adxl = ADXL345();
15 Adafruit_VL6180X vl = Adafruit_VL6180X();
16 int x, y, z;
```

Figura 7.11: Declaraciones globales.

Para el teclado numérico se establecen dos constantes, las cuales van a dar el número de filas y columnas, a estas constantes se les da el nombre ROW_NUM y COLUMN_NUM , teniendo estas un valor asignado de 4 y 3 respectivamente. También es necesario crear un arreglo que concede cada carácter a cada uno de los botones. Después se procede a asignar los pines del microcontrolador, para lo cual se crean dos vectores que se asociarán a las filas y columnas del teclado numérico.

Finalmente se crea el objeto keypad para así obtener los datos de la instancia Keypad (nótese la diferencia en la “k”), a la cual se le asignaron los pines 25, 26, 27 y 13 para las filas, y para las columnas se asignaron los pines 33, 32 y 14.

```
//Pad Numerico
const int ROW_NUM = 4;
const int COLUMN_NUM = 3;

char keys[ROW_NUM][COLUMN_NUM] = {
  {'3','2','1'},
  {'6','5','4'},
  {'9','8','7'},
  {'#','0','*'}
};
byte pin_rows[ROW_NUM] = {25, 26, 27,13};
byte pin_column[COLUMN_NUM] = {33, 32, 14};

Keypad keypad = Keypad( makeKeymap(keys), pin_rows, pin_column, ROW_NUM, COLUMN_NUM );
```

Figura 7.12: Código del Pad numérico.

Para los diferentes módulos se crearon variables y constantes, las cuales permiten almacenar datos para enviar al servidor, desbloquear el sistema, conectar el sistema a internet y enviar notificaciones al usuario.

La variable passwordPad se usa para guardar un numero aleatorio entre 1000 y 9999, el cual servirá para asignar una contraseña para el desbloqueo del sistema. La variable desbloqueo es con la cual se identifica si el parqueadero está libre o en uso, prox se declara para controlar el envío de notificaciones al usuario, en la variable input_password se guardan los dígitos ingresados por el usuario (por medio de la matriz numérica), ssid es una constante creada para guardar el nombre de la red wifi a la que el sistema debe estar conectado y la constante password es la contante a la que se le asigna la contraseña de la red wifi.

```
//Variables modulos
String passwordPad;
bool desbloqueo = true;
bool prox = true;
String input_password;
const char* ssid = "Ojoaereo";
const char* password =  "438foto11";
```

Figura 7.13: Variables de parqueadero y conexión a internet.

Ya que al ingresar los dígitos por la matriz numérica se entraba en conflicto con los sensores, se decide crear una tarea llamada Pad, para así asignar esta más adelante al otro procesador y no crear conflicto con ninguno de los módulos.

Se debe crear el objeto http, para así poder acceder a la información de la base de datos y hacer requerimientos de los datos que se encuentran en esta.

```
41 //ESP32
42 TaskHandle_t Pad;
43 HTTPClient http;
44 int httpCode = http.GET();
```

Figura 7.14: Tareas HTTP y de Pad numérico.

Para el correcto funcionamiento del sistema es necesario iniciar la comunicación serial entre el computador y el microcontrolador, también deben iniciarse los sensores VL618X y ADXL345 y el módulo WiFi; a este último es necesario darle los parámetros necesarios para que pueda conectarse a la red. Todo esto debe hacerse en la función void setup.

También en la función void setup se debe iniciar la tarea Pad, la cual debe apuntar a la función KeyPad, y donde también se le asigna el procesador 0, permitiendo así que esta tarea se ejecute en simultáneo con todo lo demás que se está realizando en el procesador 1.

```
46 void setup() {
47
48   if (! vl.begin()) {
49     Serial.println("NO se encontro el sensor");
50     while (1);
51   }
52   Serial.println("Sensor encontrado");
53
54   xTaskCreatePinnedToCore(
55     KeyPad,
56     "Tarea2",
57     10000,
58     NULL,
59     1,
60     &Pad,
61     0
62     );
63
64   Serial.begin(115200);
65   adxl.powerOn();
66   adxl.setRangeSetting(4);
67
68   input_password.reserve(32);
69   WiFi.begin(ssid, password);
70   pinMode(relay, OUTPUT);
71
72 }
```

Figura 7.15: Inicialización de los módulos y sensores.

En el void loop hacemos el llamado a las funciones creadas posteriormente, teniendo así una obtención de datos constante para el monitoreo y funcionamiento del sistema.

```
74 void loop() {
75   Serial.print(passwordPad);
76   Rango();
77   Wifi();
78   Acelerar();
79 }
```

Figura 7.16: Bucle de funciones del programa.

En las figuras 7.17, 7.18, 7.19 y 7.20 se ve el código con el se conecta el módulo al servidor y se da el intercambio de datos. Inicialmente se verifica si el módulo ESP32 se conecta a la red WiFi, a la variable http se asigna de tipo HTTPClient, a esta variable http se le asigna la URL de donde se va a obtener la información de la base de datos. Después se usa la instrucción http.GET para traer la información de cada parqueadero, esta información se guarda en la variable httpCode. Si la variable la cual recibe los datos pedidos anteriormente de la nube, se convierten los datos en tipo string para así convertirlo después en un objeto tipo JSON.

```
81 void Wifi(){
82
83   if ((WiFi.status() == WL_CONNECTED)) {
84     HTTPClient http;
85     http.begin("https://firestore.googleapis.com/v1/projects/bike-parking-app/databases/(default)/documents/parking_lots/" + ParkingID);
86     int httpCode = http.GET();
```

Figura 7.17: Función WiFi.

En la variable httpPost se crea para poder guardar cada uno de los datos de las variables, de los sensores, para así enviarlos a la base de datos y poder utilizarlas en el sistema.

```
89    if (httpCode > 0) { //Check for the returning code
90
91        String payload = http.getString();
92        JSONVar myObject = JSON.parse(payload);
93        JSONVar keys = myObject.keys();
94        if (myObject["fields"]["available"]["booleanValue"]){
95        //Serial.print("Apagado!");
96        digitalWrite(relay, HIGH);
97        desbloqueo=true;
98        }
99        else {
100         digitalWrite(relay, LOW);
101         //Serial.print("Prendido!!");
102           if (desbloqueo== true){
103           passwordPad = random(1000,9999);
104         }
105       desbloqueo= false;
106         }
107     }
```

4er

Figura 7.18: Condicional de la función WiFi.

La variable httpRequestData recibe todos los datos necesarios del sistema convertidos en tipo string, para así poder guardarlos y tenerlos de una manera mas clara en la base de datos.

```
115   http.addHeader("Content-Type", "application/x-www-form-urlencoded");
116
117   String httpRequestData = "acelerometroX=" + String(x) +
118   "&password=" + String(passwordPad) +
119   "&ParkingID=" + String(ParkingID) +
120   "&desbloqueo=" + String(desbloqueo) +
121   "&proximidad=" + String(prox);
122
123   int httpResponseCode = httpPost.POST(httpRequestData);
```

Figura 7.19: Acceso de hipervínculo y solicitud de texto.

Se envían los datos a la base de datos.

```
123   int httpResponseCode = httpPost.POST(httpRequestData);
124
125      Serial.print("HTTP Response code: ");
126      Serial.println(httpResponseCode);
127
128      httpPost.end();
129      }
130      }
```

Figura 7.20: Impresión HTTP.

En la función rango se crea la variable range para guardar los valores del sensor y ver el valor correspondiente a la distancia donde se detecta un objeto. Teniendo este valor se hace un condicional el cual va a alertar al usuario cuando el sensor no detecte un objeto en el rango establecido.

```
132 void Rango(){
133    uint8_t range = vl.readRange();
134    uint8_t status = vl.readRangeStatus();
135       if (range > 50){
136         prox = false;
137     Serial.print("No está la bicicleta");
138     }
139     else{
140       prox = true;
141     Serial.print("Bicicleta parqueada");
142       }
143   }
```

Figura 7.21: Función Rango.

En la función acelerar se leen los valores detectados por el sensor en los ejes x, y y z. En este caso, por motivos de la ubicación del sensor, solo se necesita el valor del eje x; para este valor se crea un condicional el cual envía una alerta cuando el valor de x no se encuentre en los rangos establecidos.

```
145 void Acelerar(){
146   adxl.readAccel(&x, &y, &z);
147   Serial.print(x);
148   if(x>5){
149   Serial.print("\n ALERTA DE ROBO!!! \n");
150   }
151   else {
152     Serial.println("\n Bicicleta segura \n");
153     }
154     delay(10000);
155    }
```

Figura 7.22: Función Acelerar.

La función KeyPad se encarga de guardar los valores ingresados por la matriz numérica y así compararlo con el valor aleatorio guardado anteriormente en la variable passwordPad. Si la contraseña coincide, se habilita el relé para así poder desbloquear el sistema.

La variable key es la encargada de guardar el numero presionado en la matriz, teniendo los números del 1 al 9, * siendo la manera de borrar la totalidad de los dígitos ingresados y # sirviendo como botón de aceptar al ingresar la contraseña. Esta función KeyPad cuenta con algunos condicionales que permiten asignar el * y el # como teclas de acciones y también condicionales para habilitar el desbloqueo.

```
157 void KeyPad(void *pvParameter){
158   for(;;){
159     //Serial.print(passwordPad);
160     char key = keypad.getKey();
161   if (key){
162     Serial.println(key);
163     if(key == '*') {
164
165       input_password = "";
166     }
167     else if(key == '#') {
168       if(passwordPad == input_password) {
169         Serial.println("Contraseña correcta");
170         digitalWrite(relay, HIGH);
171         desbloqueo = true;
172       } else {
173         Serial.println("Contraseña incorrecta, intente de nuevo");
174       }
175
176     }
177     else {
178       input_password += key;
179     }
180   }
181   vTaskDelay(10);
182   }
183
184   }
```

Figura 7.23: Función KeyPad.

Anexo 3 – Código App

A continuación, se explica y se muestra el código de la aplicación, en el que se puede ver las funciones con las cuales se da la lógica a la aplicación y también la parte visual de esta. Para el desarrollo de la aplicación se utilizaron componentes de React, los cuales facilitan la creación de la interfaz de usuario, en este documento no se explicarán cada uno de los componentes utilizados en el desarrollo del proyecto.

En la figura 7.25 se puede observar el archivo principal de la aplicación, contiene una función llamada app la cual se encargará de ejecutar la aplicación, inicialmente se definen algunos estados o variables de React para manejar las notificaciones de la aplicación.

Después de definir estas variables se ejecuta un método de React llamado useEffect para que una vez se cargue la aplicación, se habilite el uso de notificaciones de esta.

Al final se retorna mediante JSX (JavaScript XML) los componentes principales para almacenar la información de la aplicación y renderizar la parte visual de esta. El archivo appContext se encargará de almacenar el estado de la aplicación

```
21  const App = () => {
22      const [expoPushToken, setExpoPushToken] = useState('')
23      const [notification, setNotification] = useState<Notification | null>(null)
24      const notificationListener = useRef<any>()
25      const responseListener = useRef<any>()
26
27      useEffect(() => {
28          registerForPushNotificationsAsync().then(
29              token => token && setExpoPushToken(token)
30          )
31
32          notificationListener.current =
33              Notifications.addNotificationReceivedListener(notification => {
34                  setNotification(notification)
35              })
36
37          responseListener.current =
38              Notifications.addNotificationResponseReceivedListener(response => {
39                  /* console.log(response) */
40              })
41
42          return () => {
43              notificationListener.current &&
44                  Notifications.removeNotificationSubscription(
45                      notificationListener.current
46                  )
47              responseListener.current &&
48                  Notifications.removeNotificationSubscription(responseListener.current)
49          }
50      }, [])
51
52      return (
53          <AppContextProvider notificationToken={expoPushToken}>
54              <StatusBar style="auto" />
55              <AuthLayout>
56                  <NavigationContainer>
57                      <Tabs />
58                  </NavigationContainer>
59              </AuthLayout>
60          </AppContextProvider>
61      )
62  }
```

Figura 7.24: Archivo principal.

En la figura 7.25 se importan los controladores de estado y el estado inicial de cada uno de ellos, para luego utilizarlos en la aplicación mediante el hook useReducer de React.

```
import React, {
    createContext,
    useEffect,
    FC,
    useReducer,
    Dispatch,
} from 'react'
import { IUserState, userInitialState, userReducer } from './states/userState'
import {
    parkingLotsReducer,
    parkingLotsInitialState,
    IParkingLotsState,
    parkingLotsActionType,
} from './states/parkingLotsState'
import firebase, { auth, db } from '../config/firebase'
import {
    IParkingPlacesState,
    parkingPlacesActionType,
    parkingPlacesInitialState,
    parkingPlacesReducer,
} from './states/parkingPlacesState'
import { adminDocumentId } from '../utils/constants'
import {
    INotificationsState,
    notificationsActionType,
    notificationsInitialState,
    notificationsReducer,
} from './states/notificationsState'
```

Figura 7.25: Controladores de estado.

En el codigo de la figura 7.26 se crea un tipo que representa el objeto usuario en la aplicación, se crea una interfaz de typeScript la cual modela el estado global de la aplicación y tambien se crea un contexto con el método createContext de React el cual será usado mas adelante para crear el proveedor de contexto y consumir el mismo en otros componentes de la aplicación.

```
30  type FirebaseUser = firebase.User | null
31
32  interface IContext {
33      userState: IUserState
34      parkingLotsState: IParkingLotsState
35      parkingLotsDispatch: Dispatch<parkingLotsActionType>
36      parkingPlacesState: IParkingPlacesState
37      parkingPlacesDispatch: Dispatch<parkingPlacesActionType>
38      notificationsState: INotificationsState
39      notificationsDispatch: Dispatch<notificationsActionType>
40      notificationToken: string
41  }
42
43  export const Context = createContext({} as IContext)
```

Figura 7.26: Objeto usuario, interfaz typeScript y contexto.

En la figura 7.27 se puede ver que se crean los reducer (administradores de estado) con el método userReducer de react y los controladores y estados iniciales importados anteriormente.

```
context > TS AppContext.tsx > AppContextProvider
45  export const AppContextProvider: FC<{ notificationToken: string }> = ({
46      children,
47      notificationToken,
48  }) => {
49      const [userState, userDispatch] = useReducer(userReducer, userInitialState)
50      const [parkingLotsState, parkingLotsDispatch] = useReducer(
51          parkingLotsReducer,
52          parkingLotsInitialState
53      )
54      const [parkingPlacesState, parkingPlacesDispatch] = useReducer(
55          parkingPlacesReducer,
56          parkingPlacesInitialState
57      )
58      const [notificationsState, notificationsDispatch] = useReducer(
59          notificationsReducer,
60          notificationsInitialState
61      )
```

Figura 7.27: Administradores de estado.

A continuación, en la figura 7.28 se crea un efecto de React el cual se encarga de validar si hay un usuario con una sesión activa en la aplicación mediante la función auth.onAuthStateChanged, a esta función se le pasa otra función llamada handleUser la cual si recibe un usuario lo almacenará en el estado global y si no enviará nada, esto se ejecuta una vez al momento de que la aplicación carga.

Se crea otro efecto el cual se ejecuta cuando hay un usuario con una sesión activa en la aplicación y en caso de que este sea el usuario administrador se enviara su token de notificaciones a la base de datos ya que el administrador recibe notificaciones especiales (de todos los parqueaderos).

```
63      const handleUser = async (user: FirebaseUser) => {
64          if (user) {
65              const admin = await db.collection('admin').doc(adminDocumentId).get()
66              const isUserAdmin = admin.data()?.user_admin_id === user.uid
67
68              userDispatch({ type: 'logged-in', payload: { user, isUserAdmin } })
69          } else {
70              userDispatch({ type: 'not-logged-in' })
71          }
72      }
73
74      useEffect(() => {
75          const unsubscribe = auth.onAuthStateChanged(handleUser)
76
77          return () => unsubscribe()
78      }, [])
79
80      useEffect(() => {
81          if (userState.isUserAdmin && notificationToken) {
82              db.collection('admin')
83                  .doc(adminDocumentId)
84                  .update({ admin_notification_token: notificationToken })
85          }
86      }, [userState.isUserAdmin, notificationToken])
```

Figura 7.28: Función auth.onAuthStateChanged.

Se crea el proveedor de contexto y se le dan los valores que serán accesibles globalmente por todos los componentes. Este proveedor de contexto se utilizará en el archivo principal de la aplicación.

```
88      return (
89          <Context.Provider
90              value={{
91                  userState,
92                  parkingLotsState,
93                  parkingLotsDispatch,
94                  parkingPlacesState,
95                  parkingPlacesDispatch,
96                  notificationsState,
97                  notificationsDispatch,
98                  notificationToken,
99              }}
100         >
101             {children}
102         </Context.Provider>
103     )
104 }
```

Figura 7.29: Proveedor de contexto.

Para utilizar firebase en la aplicación se crea un archivo de configuración, como se ve en la figura 7.30, en el cual se importan las librerías de firebase que se usarán en la aplicación. Para inicializar la conexión con firebase se crea un objeto el cual contiene las credenciales para conectarse a los servicios de firebase. Luego se crean una variable db la cual es una instancia del servicio firestore de firebase. También se crea una variable auth la cual es un servicio de autenticación de firebase. Estas dos variables se exportan para poder ser utilizadas en cualquier componente de la aplicación.

config > TS firebase.ts > ...

```
1   import firebase from 'firebase/app'
2   import 'firebase/firestore'
3   import 'firebase/auth'
4
5   const firebaseConfig = {
6       apiKey: 'AIzaSyB2pgMkvOGalV-J53xRgL4bja4I554BzaY',
7       authDomain: 'bike-parking-app.firebaseapp.com',
8       projectId: 'bike-parking-app',
9       storageBucket: 'bike-parking-app.appspot.com',
10      messagingSenderId: '168739402423',
11      appId: '1:168739402423:web:f395b91f0ac26232d85332',
12      measurementId: 'G-NSRK70XG63',
13  }
14
15  !firebase.apps.length && firebase.initializeApp(firebaseConfig)
16
17  export const db = firebase.firestore()
18  export const auth = firebase.auth()
19
20  export default firebase
```

Figura 7.30: Configuración Firebase

En las figuras 7.31, 7.32 y 7.33 se ve el código para la creación de la lógica de programación de la aplicación se crearon hooks de React como por ejemplo el hook useAuth el cual se conecta al estado global de la aplicación y contiene las funciones encargadas de controlar la sesión de usuario en la aplicación.

hooks > TS useAuth.ts > useAuth

```
1  import { adminDocumentId } from './../utils/constants'
2  import { Context } from '../context/AppContext'
3  import { useContext } from 'react'
4  import { auth, db } from '../config/firebase'
5
6  const useAuth = () => {
7      const {
8          userState,
9          parkingLotsDispatch,
10         notificationToken,
11         notificationsDispatch,
12     } = useContext(Context)
13     const { user, loadingUser, isUserAdmin } = userState
14
15     const signInWithEmailAndPassword = async (email: string, password: string) =>
16         await auth.signInWithEmailAndPassword(email, password)
17
18     const clearGlobalStateOnSignOut = () => {
19         parkingLotsDispatch({ type: 'clear' })
20     }
```

Figura 7.31: Hook useAuth 1.

```
22    const createUserWithEmailAndPassword = async (
23        email: string,
24        password: string
25    ) =>
26        await auth
27            .createUserWithEmailAndPassword(email, password)
28            .then(data => data.user?.sendEmailVerification())
29
30    const signOut = async () => {
31        if (isUserAdmin) {
32            db.collection('admin')
33                .doc(adminDocumentId)
34                .update({ admin_notification_token: '' })
35        }
36        await auth
37            .signOut()
38            .then(clearGlobalStateOnSignOut)
39            .catch(error => {})
40
41        notificationsDispatch({ type: 'clear' })
42    }
```

Figura 7.32: Hook useAuth 2.

```
44        return {
45            user,
46            loadingUser,
47            isUserAdmin,
48            signOut,
49            signInWithEmailAndPassword,
50            createUserWithEmailAndPassword,
51            notificationToken,
52        }
53    }
```

Figura 7.33: Hook useAuth 3.

También se creó el hook useParking el cual contiene las funciones que se encargan de traer la información de los parqueaderos desde la base de datos y almacenarla en el estado global de la aplicación, el codigo de esto se puede ver en las figuras 7.34, 7.35, 7.36, 7.37, 7.38. Estos dos hooks utilizan métodos de firebase para conectarse con el servicio firestore y con firebase authentication

```
1  import { Context } from '../context/AppContext'
2  import { db } from '../config/firebase'
3  import { useContext } from 'react'
4  import ParkingLot from '../utils/types/ParkingLot'
5  import ParkingPlace from '../utils/types/ParkingPlace'
6  import useAuth from './useAuth'
7  import Notification from '../utils/types/Notification'
8
9  const useParking = () => {
10     const {
11         parkingLotsState,
12         parkingLotsDispatch,
13         parkingPlacesState,
14         parkingPlacesDispatch,
15         notificationsState,
16         notificationsDispatch,
17     } = useContext(Context)
```

Figura 7.34: Hook useParking 1.

```
19     const { user, isUserAdmin } = useAuth()
20
21     const { parkingLots, loadingParkingLots, errorParkingLots, userParkingLot } =
22         parkingLotsState
23     const {
24         parkingPlaces,
25         loadingParkingPlaces,
26         errorParkingPlaces,
27         currentPlace,
28     } = parkingPlacesState
29     const { notifications, loadingNotifications, errorNotifications } =
30         notificationsState
31
32     const getData = async () => {
33         await getParkingPlaces()
34         await getParkingLots()
35         await getNotifications()
36     }
```

Figura 7.35: Hook useParking 2.

```
62  const getParkingLots = async ({
63      update = false,
64  }: { update?: boolean } = {}) => {
65      if (!parkingLots || update) {
66          if (!update) {
67              parkingLotsDispatch({ type: 'init-list-request' })
68          }
69          try {
70              db.collection('parking_lots').onSnapshot(querySnapshot => {
71                  const docs = querySnapshot.docs.map(doc => ({
72                      id: doc.id,
73                      ...doc.data(),
74                  }))
75                  user &&
76                      parkingLotsDispatch({
77                          type: 'set-list',
78                          payload: { list: docs as ParkingLot[], uid: user.uid },
79                      })
80              })
81          } catch (error) {
82              parkingLotsDispatch({ type: 'error-list-request', payload: error })
83          }
84      }
85  }
86
87  const getNotifications = async ({
88      update = false,
89  }: { update?: boolean } = {}) => {
90      if (!notifications || update) {
91          if (!update) {
92              notificationsDispatch({ type: 'init-list-request' })
93          }
```

Figura 7.36: Hook useParking 3.

```
94              try {
95                  if (isUserAdmin) {
96                      db.collection('notifications').onSnapshot(querySnapshot => {
97                          const docs = querySnapshot.docs.map(doc => ({
98                              id: doc.id,
99                              ...doc.data(),
100                         }))
101
102                         notificationsDispatch({
103                             type: 'set-list',
104                             payload: { list: docs as Notification[] },
105                         })
106                     })
107                 } else {
108                     db.collection('notifications')
109                         .where('user_id', '==', user?.uid)
110                         .onSnapshot(querySnapshot => {
111                             const docs = querySnapshot.docs.map(doc => ({
112                                 id: doc.id,
113                                 ...doc.data(),
114                             }))
115
116                             notificationsDispatch({
117                                 type: 'set-list',
118                                 payload: { list: docs as Notification[] },
119                             })
120                         })
121                 }
122             } catch (error) {
123                 notificationsDispatch({ type: 'error-list-request', payload: error })
124             }
125         }
126     }
```

Figura 7.37: Hook useParking 4.

```
128     const isPlaceAvailable = (id: string) => {
129         const isAvailable = parkingLots?.some(
130             lot => lot.parking_place === id && lot.available
131         )
132         return isAvailable
133     }
134
135     const setCurrentPlace = (place: ParkingPlace) =>
136         parkingPlacesDispatch({ type: 'set-current-place', payload: place })
137
138     return {
139         getParkingLots,
140         getParkingPlaces,
141         parkingLots,
142         loadingParkingLots,
143         errorParkingLots,
144         parkingPlaces,
145         loadingParkingPlaces,
146         errorParkingPlaces,
147         getData,
148         isPlaceAvailable,
149         setCurrentPlace,
150         currentPlace,
151         userParkingLot,
152         notifications,
153         loadingNotifications,
154         errorNotifications,
155     }
156 }
157
158 export default useParking
```

Figura 7.38: Hook useParking 5.

En la figura 7.39 se observa el componente AuthLayout, que se encarga de validar si hay un usuario activo e la aplicación y en caso de que lo haya muestra la pantalla de inicio de la aplicación y en caso de que no mostrará la pantalla para registrarse o crear un usuario de la aplicación. En este comoponente se utiliza el archivo principal de la aplicación.

components > TS AuthLayout.tsx > ...

```tsx
import React, { FC } from 'react'
import useAuth from '../hooks/useAuth'
import Loading from '../screens/Loading'
import AuthPage from '../screens/AuthPage'

const AuthLayout: FC = ({ children }) => {
    const { user, loadingUser } = useAuth()

    return <>{loadingUser ? <Loading /> : user ? children : <AuthPage />}</>
}

export default AuthLayout
```

Figura 7.39: Componente AuthLayout

El componente loading se renderiza en pantalla un loader junto con una imagen y se utiliza cuando la aplicación este cargando. Para darle estilo a esta pantalla se utiliza el método StyleSheet.create de React native para definir las dimensiones de los elementos en esta pantalla.

screens > TS Loading.tsx > ...

```tsx
import React from 'react'
import {
    View,
    StyleSheet,
    Dimensions,
    Image,
    ActivityIndicator,
} from 'react-native'
import { primary_color } from '../styles/globals'

const Loading = () => {
    return (
        <View style={styles.loading}>
            <Image
                style={styles.loading__img}
                source={require('../assets/img/bike.png')}
            />
            <ActivityIndicator size="large" color={primary_color} />
        </View>
    )
}

const styles = StyleSheet.create({
    loading: {
        alignItems: 'center',
        justifyContent: 'center',
        width: Dimensions.get('window').width,
        height: Dimensions.get('window').height,
    },
    loading__img: {
        width: 75,
        height: 75,
        marginBottom: 75,
    },
})

export default Loading
```

Figura 7.40: Loading.

AuthPage es toda la parte visual de la pantalla de inicio de sesión y de registro de la app. Esta pantalla se conecta al hook useAuth para traer las funciones de inicio de sesión o registro. Explicación de componentes react native: Para la maquetación de pantallas con React native se utilizan componentes que la librería trae, como, por ejemplo:

- El componente view que representa un contenedor de elementos en pantalla.
- El componente text se utiliza para colocar texto en la aplicación.
- El componente image se utiliza para colocar imágenes en los componentes de react native

screens > TS AuthPage.tsx > [@] AuthPage

```tsx
import React, { useState } from 'react'
import {
    View,
    TextInput,
    Pressable,
    Image,
    Text,
    ActivityIndicator,
    Alert,
    StyleSheet,
    Dimensions,
} from 'react-native'
import useAuth from '../hooks/useAuth'
import Icons from 'react-native-vector-icons/Feather'
import useLanguage from '../hooks/useLanguage'
import { primary_color } from '../styles/globals'

const AuthPage = () => {
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')
    const [confirmPassword, setConfirmPassword] = useState('')
    const [loading, setLoading] = useState(false)

    const [isLogin, setIsLogin] = useState(true)

    const { firebaseErrors } = useLanguage()
    const { signInWithEmailAndPassword, createUserWithEmailAndPassword } =
        useAuth()
```

Figura 7.41: AuthPage 1.

```
30  const onError = (error: string) => {
31      Alert.alert('Ocurrió un error 🌵', error, [{ text: 'Intertar de nuevo' }])
32  }
33
34  const onSubmit = async () => {
35      setLoading(true)
36      if (isLogin) {
37          await signInWithEmailAndPassword(email, password).catch(error => {
38              setLoading(false)
39              onError(firebaseErrors(error))
40          })
41      } else {
42          if (password === confirmPassword) {
43              await createUserWithEmailAndPassword(email, password).catch(error => {
44                  setLoading(false)
45                  onError(firebaseErrors(error))
46              })
47          } else {
48              Alert.alert('Ocurrió un error 🌵', 'Las contraseñas no coinciden', [
49                  { text: 'Intertar de nuevo' },
50              ])
51          }
52      }
53  }
```

Figura 7.42: AuthPage 2.

```
55      return (
56          <View style={styles.auth}>
57              <Image
58                  style={styles.auth__img}
59                  source={require('../assets/img/bike.png')}
60              />
61              <Text style={styles.auth__title}>
62                  {isLogin ? 'Iniciar sesión' : 'Registrarse'}
63              </Text>
64              <TextInput
65                  style={styles.auth__input}
66                  onChangeText={setEmail}
67                  placeholder="Correo"
68                  keyboardType="email-address"
69              />
70              <TextInput
71                  style={styles.auth__input}
72                  secureTextEntry={true}
73                  onChangeText={setPassword}
74                  placeholder="Contraseña"
75              />
76              {!isLogin && (
77                  <TextInput
78                      style={styles.auth__input}
79                      secureTextEntry={true}
80                      onChangeText={setConfirmPassword}
81                      placeholder="Confirmar contraseña"
82                  />
83              )}
```

Figura 7.43: AuthPage 3.

```
84              <Pressable
85                  disabled={loading}
86                  style={styles.auth__buttom}
87                  onPress={onSubmit}
88              >
89                  {loading ? (
90                      <ActivityIndicator size="large" color="#fff" />
91                  ) : (
92                      <Icons name="arrow-right" color="#fff" size={32} />
93                  )}
94              </Pressable>
95              <Text style={styles.auth__link} onPress={() => setIsLogin(!isLogin)}>
96                  {isLogin ? 'Registrarse' : 'Iniciar sesión'}
97              </Text>
98          </View>
99      )
100 }
101
102 const styles = StyleSheet.create({
103     auth: {
104         alignItems: 'center',
105         justifyContent: 'center',
106         width: Dimensions.get('window').width,
107         height: Dimensions.get('window').height,
108         padding: 32,
109     },
110     auth__img: {
111         width: 75,
112         height: 75,
113         marginBottom: 75,
114     },
115     auth__title: {
116         fontWeight: 'bold',
117         fontSize: 32,
118         marginBottom: 32,
119         color: '#444251',
120     },
```

Figura 7.44: AuthPage 4.

```
121     auth__input: {
122         width: '100%',
123         height: 50,
124         borderRadius: 10,
125         backgroundColor: '#f0f0f0',
126         marginBottom: 16,
127         padding: 16,
128     },
129     auth__buttom: {
130         alignItems: 'center',
131         justifyContent: 'center',
132         height: 75,
133         width: 75,
134         backgroundColor: primary_color,
135         borderRadius: 16,
136         marginTop: 32,
137     },
138     auth__link: {
139         color: '#444251',
140         marginTop: 32,
141     },
142 })
143
144 export default AuthPage
```

Figura 7.45: AuthPage 5.

Para la creación de la barra de navegación y sistema de navegación entre pantallas de la aplicación se utiliza una librería llamada React navigation y se crea un stack de navegación por cada botón. Cada uno de estos stack contiene las otras pantallas de la aplicación. Los diferentes stack se explicarán a continuación.

- **HomeStack:** Contiene la pantalla donde se muestran los parking places y la pantalla donde se muestra la información especifica de cada una de ellos.
- **MapStack:** Este contiene la pantalla que representa el mapa mediante la librería react-native-maps y también contiene la pantalla que contiene la información específica de cada una de los parking places.
- **NotificationStack:** contiene la pantalla que muestra la información de alertas de robo en la aplicación

- **Account:** Contiene la pantalla que muestra la información del parqueadero actual y un botón para cerrar sesión.

Cada uno de estos stacks y pantalla utiliza los hooks useAuth y useParking para interactuar con la base de datos y con la información del usuario.

```
const Tabs = () => {
    return (
        <Tab.Navigator>
            <Tab.Screen
                name="Inicio"
                component={HomeStack}
                options={{
                    tabBarIcon: ({ color }) => (
                        <Icons name="home" color={color} size={20} />
                    ),
                    tabBarActiveTintColor: primary_color,
                    headerShown: false,
                }}
            />
            <Tab.Screen
                name="Mapa"
                component={MapStack}
                options={{
                    tabBarIcon: ({ color, size }) => (
                        <Icons name="map" color={color} size={20} />
                    ),
                    tabBarActiveTintColor: primary_color,
                    headerShown: false,
                }}
            />
            <Tab.Screen
                name="Notificaciones"
                component={NotificationsStack}
                options={{
                    tabBarIcon: ({ color, size }) => (
                        <Ionicons name="notifications" color={color} size={20} />
                    ),
                    tabBarActiveTintColor: primary_color,
                    headerShown: false,
                }}
            />
            <Tab.Screen
                name="Cuenta"
                component={Account}
                options={{
                    tabBarIcon: ({ color, size }) => (
                        <Icons name="user" color={color} size={20} />
                    ),
                    tabBarActiveTintColor: primary_color,
                }}
            />
        </Tab.Navigator>
    )
}

export default Tabs
```

Figura 7.46: Stack de navegación.

Bibliografía

[1] J. Karsten. (2018) What is smart parking? [Online]. Available: https://www.parkeagle.com/2018/05/12/what-is-smart-parking/

[2] Eva Heinen Ralph Buehler, "Bicycle parking," 2019, a systematic review of scientific literature on parking behaviour, parking preferences, and their influence on cycling and travel behaviour. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/01441647.2019.1590477?scroll=top&needAccess=true&role=tab

[3] *Boom de Bicicletas*, 2021. [Online]. Available: https://www.larepublica.co/analisis/simon-gaviria-munoz-401830/boom-de-bicicletas-3249806

[4] *¿Cómo asegurar y parquear bien la bicicleta para evitar robos?*, 2020. [Online]. Available: https://www.eltiempo.com/bogota/como-asegurar-y-parquear-bien-una-bicicleta-509252

[5] *Bike theft across Europe and securing better bike parking*, 2023. [Online]. Available: https://ecf.com/news-and-events/news/bike-theft-across-europe-and-securing-better-bike-parking

[6] VANDERSPEK201539k, "The importance of bicycle parking management," *Research in Transportation Business Management*, vol. 15, pp. 39–49, 2015, managing the Business of Cycling. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210539515000127

[7] P. V. Tran, "Will implementation of high-quality bicycle parking facilities make cycling more attractive in city central areas," *Institutt for arkitektur og planlegging*, 2021, master thesis. [Online]. Available: https://hdl.handle.net/11250/2824639

[8] Real Academia Española. (1995) Bicicleta. [Online]. Available: https://dle.rae.es/bicicleta

[9] ——. (1995) Bicicleta. [Online]. Available: https://dle.rae.es/bicicleta

[10] S. Bennett, *A History of Control Engineering 1930-1955*, ser. Control, Robotics and Sensors. The Institution of Engineering and Technology, 1993.

[11] MaxBotix. (2018) Smart parking, sensors for vehicle detection. [Online]. Available: https://www.digikey.com/en/pdf/m/maxbotix/smart-parking-sensors-vehicle

[12] I. Codesido. (2009) What is front-end development? [Online]. Available: hhttps://www.theguardian.com/help/insideguardian/2009/sep/28/blogpost

[13] A. Goins. (2020) Front end vs. back end: What's the difference? [Online]. Available: https://kenzie.snhu.edu/blog/front-end-vs-back-end-whats-the-difference/

[14] Food and Drug Administration. (2018) Radio frequency identification (rfid). [Online]. Available: https://www.fda.gov/radiation-emitting-products/electromagnetic-compatibility-emc/radio-frequency-identification-rfid

[15] RFID POINT. ¿a qué distancia puede leerse un tag rfid? [Online]. Available: http://www.rfidpoint.com/consultas-al-experto/a-que-distancia-puede-leerse-un-tag-rfid/

[16] European Telecommunications Standards Institute. (2012) Electromagnetic compatibility and radio spectrum matters (erm); short-range devices (srd) for operation in the 13,56 mhz band; system reference document for radio frequency identification (rfid) equipment. [Online]. Available: https://www.etsi.org/deliver/etsi_tr/103000_103099/103059/01.01.01_60/tr_103059v010101p.pdf

[17] (2017) About near field communication. [Online]. Available: http://nearfieldcommunication.org/about-nfc.html

[18] ABI Research. Bluetooth technology overview. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

[19] What is mobile application development? [Online]. Available: https://aws.amazon.com/es/mobile/mobile-application-development/

[20] Amazon Web Services, Inc. Soluciones de aws. [Online]. Available: https://aws.amazon.com/es/solutions/

[21] Firebase documentation. [Online]. Available: https://firebase.google.com/docs/web/setup?hl=es-419#available-libraries

[22] MongoDB, Inc. Database. [Online]. Available: https://www.mongodb.com/es/atlas/database

[23] (1995) Netscape and sun announce javascript, the open, cross-platform object scripting language for enterprise networks and the internet. [Online]. Available: https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html

[24] HTML Living Standard. [Online]. Available: https://html.spec.whatwg.org/multipage/introduction.html#a-quick-introduction-to-html

[25] HTML CSS. [Online]. Available: https://www.w3.org/standards/webdesign/htmlcss#whatcss

[26] OpenJS Foundation. (1995) Musica. [Online]. Available: https://nodejs.org/en/about/

[27] O. Foundation. About npm. [Online]. Available: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

[28] npm, Inc. About npm | npm docs. [Online]. Available: https://docs.npmjs.com/about-npm/

[29] O'Reilly Online Learning. Learning react native. [Online]. Available: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

[30] (1995) Meta platforms. introduction (to react native). [Online]. Available: https://reactnative.dev/docs/getting-started

[31] M. J. Foley, "Microsoft takes the wraps off typescript, a superset of javascript. zdnet," 2012. [Online]. Available: https://www.zdnet.com/article/microsoft-takes-the-wraps-off-typescript-a-superset-of-javascript/

[32] Meta Platforms. Using typescript. [Online]. Available: https://reactnative.dev/docs/typescript

[33] Introduction to expo. [Online]. Available: https://docs.expo.dev/

[34] "Guidelines for the design and management of bicycle parking facilities (draft)." 2008. [Online]. Available: https://americantrails.s3.us-west-2.amazonaws.com/files/pdf/Torontobikeparkguide.pdf

[35] A. Polishchuk, *SMART BICYCLE PARKING: Bicycle Parking Solution for the Riihimäki Railway Station Area.*, 2016. [Online]. Available: https://www.hamk.fi/wp-content/uploads/2018/07/Final-Report-ASTI-Parkers.pdf

[36] Espressif systems. (2021) Esp32-wroom-32 datasheet. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[37] (2008) Accelerometer design and applications. [Online]. Available: https://web.archive.org/web/20081213001559/http://www.analog.com/en/technical-library/faqs/design-center/faqs/CU_faq_MEMs/resources/fca.html#SECTION_A

[38] Analog Devices. (2009) Adxl345 datasheet. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

[39] OMRON Industrial Automation. (1995) Overview of proximity sensors. [Online]. Available: https://www.ia.omron.com/support/guide/41/introduction.html

[40] ST Microelectronics. (1995) Product overview. [Online]. Available: https://www.st.com/en/imaging-and-photonics-solutions/vl6180.html

[41] Real Academia Española. Cerradura. [Online]. Available: https://dle.rae.es/cerradura

[42] Bikeep. (2022) Secure and smart bike parking station. [Online]. Available: https://bikeep.com/smart-bike-parking-station-book-a-demo/

[43] Vadebike. (2020) La nueva estación híbrida de vadebike ya es una realidad. [Online]. Available: https://vadebike.es/blog/la-nueva-estacion-hibrida-de-vadebike-ya-es-una-realidad

[44] BikeLink. On-demand bike parking. [Online]. Available: https://www.bikelink.org/

[45] S. Brown, *Shimano Cassettes Freehubs*, 2007. [Online]. Available: https://www.sheldonbrown.com/k7.html

[46] Highpath Engineering. Cycle information : Freehubs and cassettes. [Online]. Available: https://highpath.co.uk/cycle-information-freehubs-and-cassettes/

[47] Belson Outdoors. Commercial bike rack how to buying guide. [Online]. Available: http://www.belson.com/Bike-Rack-Buying-Guide

[48] Pedrero, J. C. , *GEOMETRÍA DE UN CUADRO DE BICICLETA. Escuela La Bicicleta.*, 2019. [Online]. Available: https://labicicleta.net/escuela/geometria-de-un-cuadro-de-bicicleta/

[49] Madrax. (1995) Bike rack materials and finishes. [Online]. Available: https://www.madrax.com/materials-finishes