People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering
Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of the Requirements for the Degree of

'MASTER'

In **Electronics**

Option: **Computer Engineering**

Title:

Design and Implementation of a Web-Based Application for Bicycle Rentals with GPS Tracking

Presented By:

- **CHAOUADI Sami**
- **HANNOUN Najib**

Supervised by:

Dr. NAMANE R.

Registration Number:……/2023

Acknowledgement

This work was achieved thanks to hard work and perseverance. Much thanks are direct towards our supervisor, Dr. Namane, who accepted to guide us through this project on such short notice.

We wish to express special appreciation to the software engineer for initiating us to the basic concepts of web development.

We cannot forget that we were able to progress through this project because of the rich curriculum we partook in our institute.

Dedication

I dedicate this work to my parents, family and friends, to everyone who helped me in the realization of this project and my success.

HANNOUN Najib

I dedicate this work to my parents, family and friends, to everyone who helped me in the realization of this project and my success.

CHAOUADI Sami

Abstract

Bicycle rentals is a growing business in developed countries. It is a great investment that offers people a means of transportation, and allows for innovation in the fields of Internet of Things and Embedded Systems.

Throughout this work, our main goal is to design and implement a web application for a bicycle rental shop by providing a user-friendly software for both customers and administrators navigating the system. The application allows the users to reserve bicycles, start and finish transactions, browse the various bicycle types and subscription plans that the shop has to offer, redeem and use coupons for interesting reductions in transaction prices and it finally offers customers the possibility to rent their bicycles through the shop. All of this data can be easily stored and monitored in a database with the help of C#'s ASP.NET Core framework.

In order to condition our software system to the real-world application, we propose an implementation of a GPS Tracking device that should be able to update the location of a bicycle in real-time. This trackability offers additional security and further information display. This implementation is achieved using a microcontroller-based system which offers simplicity and reduces expenses. The wide range of available peripherals provides versatility and flexibility in the design process.

It is important to note that, even though this project was designed with only a single bicycle rental shop in mind, the overall system can be easily upgraded to handle multiple shops of different vehicle types, such as motorcycles, cars, trucks… etc.

Table of Contents

Acknowledgement I

Dedication II

Abstract III

Table of Contents IV

List of Figures VIII

List of Tables XII

List of Abbreviations XIII

General Introduction 1

CHAPTER I Bicycle Rentals and Web Applications 3

1. Bicycle Rentals 3

1.1. Existing Application Examples 3

1.2. System Architecture and Entities 6

2. Web Applications 7

2.1. Definition 7

2.2. Why a web application 8

2.3. Web Application Types 8

2.4. Frameworks 10

2.4.1. Difference between Frameworks and Libraries 10

2.5. Client-Server Model 10

2.5.1. Web Clients 11

2.5.2. Web Server 11

2.5.3. HTTP and HTTPS 11

2.5.3.1. HTTP Request 12

2.5.3.2. HTTP Response 12

2.5.3.4. Communication Sequence 13

3. Embedded Systems 13

3.1. Microcontrollers 14

4. GPS Tracking Systems 14

Conclusion 15

CHAPTER II Development Tools and Technologies 16

1. Software tools 16

1.1. MVC 16

1.1.1. Model 16
1.1.2. View 16
1.1.3. Controller 17
1.1.4. Business 17
1.2. Back-End Technologies 17
1.2.1. ASP.NET Core 17
1.2.1.1. Dependency Injection 18
1.2.1.2. Routes 18
1.2.2. Databases 18
1.2.2.1. Relational Database 19
a. Keys and Indices 19
b. Relations in Databases 19
c. Relation Types 20
d. Microsoft SQL Server 20
e. EntityFramework Core 21
1.2.2.2. NoSQL Database 21
a. Firebase Realtime Database 22
b. Firesharp Library 22
1.3. Front-End Technologies 22
1.3.1. HTML 22
1.3.2. CSS 22
1.3.3. JavaScript 23
1.3.4. jQuery 23
1.3.5. Bootstrap 23
1.3.6. Leaflet 23
1.3.7. Razor Pages 23
2. Hardware tools 24
2.1. Arduino Uno Microcontroller 24
2.2. NEO 6MV2 GPS 24
2.3. Sim800L 25
2.3.1. AT commands 26
Conclusion 26
CHAPTER III System Design 27
1. Programming principles 27
2. Modeling Language 27
2.1. Unified Modeling Language 28
3. Web Application Design 30
3.1. Modeling 30
3.1.1. System actors 30

3.1.2. Use Cases 31
3.1.2.1. Actors' Use Cases Description 31
3.1.2.3. Use Case Diagrams 34
3.1.2.4. Textual Description of Use Cases 39
3.1.3. Sequence Diagrams 59
3.1.3.1 Elements of Sequence Diagrams 59
3.1.3.2. Sequence Fragments 61
3.1.3.3 Actions used in Sequence Diagrams 61
3.2. Relational Database Design 73
3.2.1. Approach Types 73
3.2.2. Migrations 74
3.2.3. Entity Relationship Diagram 74
4. GPS Tracker Design 77
4.1. Circuit Diagram 77
4.2. Software Design 78
4.2.1 NEO 6MV2 78
4.2.2. SIM800L 78
4.2.3. Setup Function 78
4.2.4. Main Loop 79
4.3. Firebase real-time database 81
Conclusion 83
CHAPTER IV Results and Analysis 84
1. Web Application Results 84
1.1. Layout 84
1.2. Home Page 84
1.3. Register 86
1.4. Login 86
1.5. Catalogue 87
1.6. Customer 88
1.6.1. Making a Reservation 89
1.6.2. Reservations 90
1.6.3. Transactions 91
1.6.4. Profile 92
1.6.4.1. Redeem Coupons 94
1.6.4.2. Coupons 95
1.6.4.3. Subscriptions 96
1.6.5. Contracts 98
1.6.6. Plans 100
1.7. Admin 101

1.7.1. Bicycles 102
1.7.2. Bicycle Types 104
1.7.3. Customers 106
1.7.4. Admins 109
1.7.5. Reservations 111
1.7.6. Transactions 114
1.7.7. Coupons 117
1.7.8. Plans 119
1.7.9. Contracts 122
1.8. Terminal Admin 124
1.9. Mobile Responsiveness 125
2. GPS Tracker Results 127
2.1. GPS Output 127
2.2. GPRS Output 128
2.2.1. Read Data 128
2.2.2. Write Data 129
2.3. System Results 130
Conclusion 131
General Conclusion and Future Works 132
References XV

List of Figures

Figure 1 rentabike.nl home page 4
Figure 2 rentabike.nl bicycle type details 4
Figure 3 retnabike.nl reservation form 4
Figure 4 fontainebleaubikerental.com home page 5
Figure 5 fontainebleaubikerental.com reservation dates 5
Figure 6 Static vs. Dynamic websites 9
Figure 7 Multi-Page vs SPA Lifecycles 9
Figure 8 HTTP Request example 12
Figure 9 HTTP Response example 12
Figure 10 Client-Server Communication Sequence 13
Figure 11 Realtime GPS Tracking Operation 14
Figure 12 ASP.NET Core Logo 17
Figure 13 Route Configuration 18
Figure 14 Microsoft SQL Server Logo 20
Figure 15 Entity Framework Logo 21
Figure 16 Firebase Realtime Database Logo 22
Figure 17 Arduino Microcontroller Board 24
Figure 18 NEO 6MV2 Pin Diagram 25
Figure 19 SIM800L Pin Diagram 26
Figure 20 Entity Relationship Diagram Example 29
Figure 21 Use Case Diagram Example 29
Figure 22 Sequence Diagram Example 30
Figure 23 Admin with Roles Use Case Diagram 36
Figure 24 Customer Use Case Diagram 37
Figure 25 SuperAdmin Use Case Diagram 38
Figure 26 Sequence Fragment 61
Figure 27 Login Sequence Diagram 63
Figure 28 Logout Sequence Diagram 64
Figure 29 Mock Login Sequence Diagram 64
Figure 30 List All Entities Sequence Diagram 65
Figure 31 Entity Details Sequence Diagram 65
Figure 32 Edit Entity Sequence Diagram 66
Figure 33 Delete/Remove Entity Sequence Diagram 67
Figure 34 Confirm/Cancel Reservation Sequence Diagram 68
Figure 35 Return Bicycle Sequence Diagram 69
Figure 36 Cash In Sequence Diagram 70
Figure 37 Redeem Coupon Sequence Diagram 71
Figure 38 Enable/Disable Plan Sequence Diagram 72
Figure 39 Confirm/Deny BicycleContract Sequence Diagram 73

Figure 40 Entity Relationship Diagram 76
Figure 41 Circuit Diagram 77
Figure 42 Atduino Setup Function Flowchart 79
Figure 43 Arduino Loop Function Flowchart 80
Figure 44 Firebase Realtime Database Data Organization Diagram 82
Figure 45 Firebase Realtime Database Data Organization 82
Figure 46 Render Body Layout 84
Figure 47 The Presentation Section 85
Figure 48 The Sign-Up Section 85
Figure 49 The Affiliate Section 86
Figure 50 Register View 86
Figure 51 Login View 87
Figure 52 Catalogue View 87
Figure 53 Catalogue Card 88
Figure 54 Catalogue Details 88
Figure 55 Customer's Navigation Bar 88
Figure 56 Customer Drop Down Menu 89
Figure 57 Make A Reservation View 89
Figure 58 Customer’s List of Reservations View 90
Figure 59 Customer’s Reservation’s Details Popup 90
Figure 60 Customer’s Cancel Reservation Popup 91
Figure 61 Customer’s List of Transactions View 91
Figure 62 Transaction’s Details View 92
Figure 63 Customer’s Profile View 93
Figure 64 Customer’s Edit Profile View 93
Figure 65 Customer’s Change Password View 93
Figure 66 Customer’s Redeem Coupons View 94
Figure 67 Customer’s Confirm Redeem Coupons Popup 94
Figure 68 Customer’s List of Coupons View 95
Figure 69 Customer’s Coupons Sliding Menu 95
Figure 70 Customer’s List of Used Coupons View 96
Figure 71 Customer’s List of Deleted Coupons View 96
Figure 72 Customer’s List of Expired Coupons View 96
Figure 73 Customer’s List of Subscriptions View 97
Figure 74 Customer’s Subscription Details Popup 97
Figure 75 Customer’s Remove Subscription Popup 97
Figure 76 Customer’s List of Contracts View 98
Figure 77 Customer’s Contract Details Popup 98
Figure 78 Submit a Bicycle Contract Popup 99
Figure 79 Add an Image Bicycle Contract 99

Figure 80 Contracts Sliding Menu 100
Figure 81 Plans View 100
Figure 82 Customer's Plans Details 101
Figure 83 Admin's Navigation Bar 101
Figure 84 Admin's Drop-Down Menu 102
Figure 85 Bicycle's State 102
Figure 86 List of Bicycles View 103
Figure 87 Bicycle's Edit Popup 103
Figure 88 Bicycle's Details Popup 103
Figure 89 Bicycle's Delete Popup 104
Figure 90 Adding a Bicycle Popup 104
Figure 91 Admin's Catalogue View 105
Figure 92 Catalogue's Bicycle Edit Popup 105
Figure 93 Adding a Bicycle Type Popup 105
Figure 94 List of Customers View 106
Figure 95 Customer's Details Popup 106
Figure 96 Customer's Delete Popup 107
Figure 97 Customer's More Button 107
Figure 98 Admin's Customer Edit View 107
Figure 99 Admin's Customer List of Subscriptions View 108
Figure 100 Renew Subscription Popup 108
Figure 101 Add a Subscription to Customer Popup 109
Figure 102 Admin's Customer Coupons View 109
Figure 103 List of Admins View 110
Figure 104 Admin's Edit Popup 110
Figure 105 New Admin Popup 111
Figure 106 List of Reservations View 112
Figure 107 Reservation's Edit Popup 112
Figure 108 Reservation's Details Popup 113
Figure 109 Reservation's Confirm Popup 113
Figure 110 Reservation's Cancel Popup 113
Figure 111 List of Transactions View 114
Figure 112 Transaction's Edit Popup 114
Figure 113 Transaction's Delete Popup 115
Figure 114 New Rental Popup 115
Figure 115 CashIn Popup 116
Figure 116 CashIn View 116
Figure 117 Return Bicycle Popup 117
Figure 118 Return Bicycle View 117
Figure 119 List of Deleted Transactions View 117

Figure 120 List of Coupons View 118
Figure 121 Add a Coupon Popup 118
Figure 122 List of Deleted Coupons View 119
Figure 123 List of Plans View 119
Figure 124 Admin's Plan Details Popup 120
Figure 125 Plan's Buttons 120
Figure 126 Plan's State 120
Figure 127 New Subscription Plan View 121
Figure 128 New Subscription Plan Time Fields 121
Figure 129 Plan's Sliding Menu 122
Figure 130 List of Contracts View 122
Figure 131 Contract's Details 122
Figure 132 Contracts Sliding Menu 123
Figure 133 List of Inactive Contracts View 123
Figure 134 Contract's Confirm Popup 123
Figure 135 Contract's Delete Popup 124
Figure 136 List of Denied Contracts 124
Figure 137 Store Terminal View 125
Figure 138 Catalogue Mobile View 126
Figure 139 Login Mobile View 126
Figure 140 Bicycle Type Details Mobile View 126
Figure 141 Bicycles List Mobile View 126
Figure 142 GPS Output Logs 127
Figure 143 GPS Output Logs with higher precision 127
Figure 144 GPRS Reading Data Output Logs 129
Figure 145 GPRS Writing Data Output Logs 130
Figure 146 Firebase Realtime Database Stored GPS Data 131

List of Tables

Table 1 EF Core mapping between a database and .NET software 21
Table 2 Use Case Descriptions 32
Table 3 Relationship Types in a Use Case Diagram 35
Table 4 Textual Description for the use case "Login" 39
Table 5 Textual Description for the use case "Register" 40
Table 6 Textual Description for the use case "Manage Admins" 40
Table 7 Textual Description for the use case "Manage Customers" 41
Table 8 Textual Description for the use case "Manage Bicycle Types" 43
Table 9 Textual Description for the use case "Manage Bicycles" 44
Table 10 Textual Description for the use case "Manage Bicycle Contracts" 45
Table 11 Textual Description for the use case "Manage Subscription Plans" 46
Table 12 Textual Description for the use case "Manage Subscriptions" 47
Table 13 Textual Description for the use case "Manage Coupon Types" 48
Table 14 Textual Description for the use case "Manage Coupons" 49
Table 15 Textual Description for the use case "Manage Reservations" 49
Table 16 Textual Description for the use case "Manage Transactions" 50
Table 17 Textual Description for the use case "Manage Personal User Information" 53
Table 18 Textual Description for the use case "Browse Bicycle Types" 54
Table 19 Textual Description for the use case "Manage Personal Bicycle Contracts" 54
Table 20 Textual Description for the use case "Manage Personal Reservations" 55
Table 21 Textual Description for the use case "Manage Personal Transactions" 56
Table 22 Textual Description for the use case "Browse Subscription Plans" 57
Table 23 Textual Description for the use case "Browse Coupon Types" 57
Table 24 Textual Description for the use case "Manage Personal Subscriptions" 58
Table 25 Textual Description for the use case "Manage Personal Coupons" 59
Table 26 Elements of Sequence Diagrams 60
Table 27 Fragment Type Operators 61
Table 28 Relations Symbols 75
Table 29 NoSQL Database Element Symbols and Descriptions 81

List of Abbreviations

- ADC: Analog to Digital Converter
- API: Application Programming Interface
- AT: Attention
- CPU: Central Processing Unit
- CRUD: Create, Read, Update, Delete
- DB: Database
- DBMS: Database Management System
- DI: Dependency Injection
- DRY: Don't Repeat Yourself
- EF: EntityFramework
- ERD: Entity Relationship Diagram
- GPRS: General Packet Radio Service
- GPS: Global Positioning System
- GSM: Global Systems for Mobile
- HTML: HyperText Markup Language
- HTTP: HyperText Transfer Protocol
- HTTPS: HyperText Transfer Protocol Secure
- IBM: International Business Machines Corporation
- IoC: Inversion of Control
- JS: JavaScript
- JSON: JavaScript Object Notation
- JSP: Java (Jakarta) Server Pages
- KISS: Keep It Simple
- LED: Light Emitting Diode
- LINQ: Language Integrated Query
- MPA: Multi Page Application
- MVC: Model, View, Controller
- MVP: Model, View, Presenter
- MVVM: Model, View, View Model
- MVW: Model, View, Whatever
- ORM: Object Relational Mapper

- OS: Operating System
- PWM: Pulse Wave Modulator
- RAM: Random Access Memory
- RDBMS: Relational Database Management System
- REST: Representational State Transfer Architecture
- SD: Sequence Diagram
- SDK: Software Development Kit
- SIM: Subscriber Identity Module
- SMS: Short Message Service
- SoC: Separation of Concerns
- SPA: Single Page Application
- SQL: Structured Query Language
- SSE: Server-Side Events
- SSL: Secure Sockets Layer
- TLS: Transport Layer Security
- UART: Universal Asynchronous Receiver/Transmitter
- UI: User Interface
- UML: Unified Modeling Language
- URL: Uniform Resource Locator
- USART: Universal Synchronous/Asynchronous Receiver/Transmitter
- USB: Universal Serial Bus
- WWW: World Wide Web

General Introduction

The speed at which the web is developing is exponentially increasing from day to day. A few years ago, websites were simple static pieces of text that displayed information. Nowadays, heavy machinery and equipment can be monitored and controlled through a web application from any browser on any device. With the new web, opportunities for innovation are endless.

As of today, building a web application is simplified with the help of dedicated frameworks that offer a convenient template that allow both beginners to learn the basics of web development, and advanced developers to quickly set up an environment for their large-scale applications.

Investors who would dive in the industry of bicycle rentals find themselves facing a serious issue, which is the lack of automated systems that handle bicycle rentals. This lack pushes startups to create traditional pen and paper businesses. This model repels potential customers who would prefer a more automated and faster way of interacting with the business, which in our age of automation is expected. We propose this system in hopes of inciting future investors to create local businesses in our country.

Two main solutions to this problem exist. The first one is to design and implement a desktop software that would be installed and hosted on a single computer machine.
The second solution suggests building a web application that would be hosted on a server device and accessible from anywhere through an internet connection.

From a development point of view, each solution requires its own set of skills and knowledge to properly implement, but from a practical point of view, a web application would be more beneficial for a number of reasons:

- Multiple devices would share the same program stored in the server, reducing the cost and complexity of the software.
- Access would not be limited to staff only, but customers would also be able to access the platform.
- An online application is more suitable for the use of various IoT devices.

The objective of our project is to create a stable Web Application for our bicycle rental platform that should be able to handle every operation that is required from a small or medium business. It would also ensure the safety of user information, without forgetting the user experience by making the entirety of the interface user-friendly and easy to use. We also aim to create a GPS tracking device for the bicycles that would be linked to the application and be able to read the geolocation data on the web.

This report is divided into four main chapters: We will first talk about how we perceive the renting operation and the other functionalities that the application will perform, we will also give an introduction to web applications with the hidden protocols that they rely on. In the second chapter, we will be giving an introduction to the various building blocks of our project alongside the development tools, techniques and technologies used to bring this system to life. After laying the building blocks and displaying the techniques to use, we can discuss, in the third chapter, the design of both the processes that our application can handle and the GPS tracker. Finally, in the fourth chapter, we showcase the results of our work before suggesting future works and improvements that could be implemented when concluding the work.

CHAPTER I
Bicycle Rentals and Web Applications

CHAPTER I Bicycle Rentals and Web Applications

In this chapter, we will take a first look into rental systems and how existing companies manage their bicycle rentals. After that we discuss how we could improve those structures for a better user experience and a more reliable system. Finally, we will provide an introduction to web applications and key concepts that rule over the development process of a web-based application alongside a short introduction to GPS trackers.

1. Bicycle Rentals

Renting is an agreement where a payment is made for the temporary use of a property owned by another.

If we apply this definition to a bicycle rental company, we can already have a basic understanding of the fundamental transaction which is the rental. A customer should be able to choose from a selection of kinds of bicycles and select the time and date for when they expect to return their bicycle. They should be able to view the price of their transaction and which bicycle they have been assigned. In a similar manner, we allow the customers to reserve a bicycle for the future.

In order to simplify the upcoming design, we decided that one customer cannot rent more than one bicycle at the same time, making verifications much simpler.

We have to pay attention to different scenarios that are logically unfeasible. A good example is that one bicycle cannot be rented twice at the same time or reserved at the same time, since it is physically impossible to give the same bicycle to two different customers.

1.1. Existing Application Examples

We took inspiration from the work of existing companies to see how these functionalities could be implemented.

rentabike.nl

This is a bicycle rental shop based in the Netherlands. Entering the website, we are greeted with a front page displaying the different bicycle types they offer with a short description and their prices. We can check on more details concerning each vehicle.

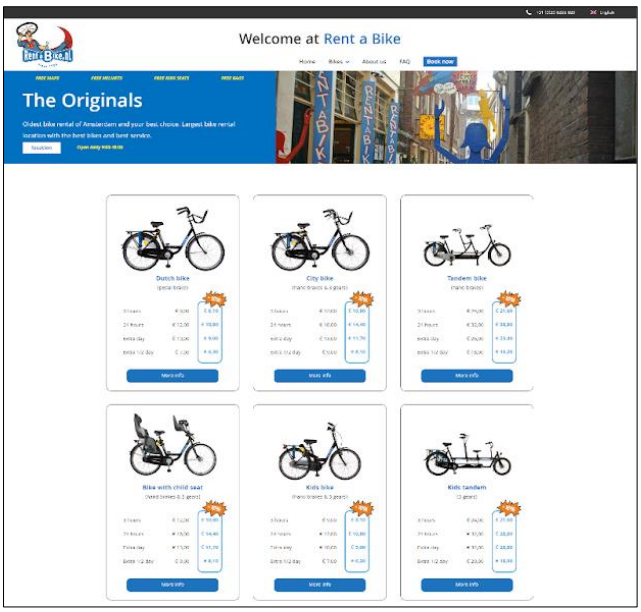


Figure 1 rentabike.nl home page

The "book" or reserve button displays a form requiring personal information alongside the number of bicycles we wish to reserve. We notice that the numbers are dropdown menus with the number of available bikes. Since their system relies on human interaction and manual operations, no further functionalities are implemented.

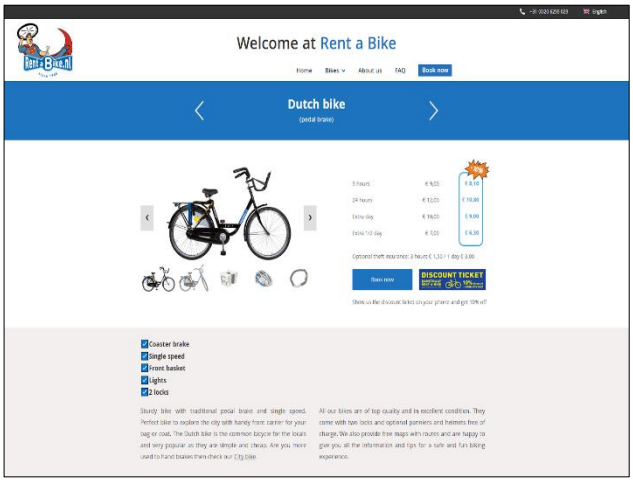


Figure 2 rentabike.nl bicycle type details

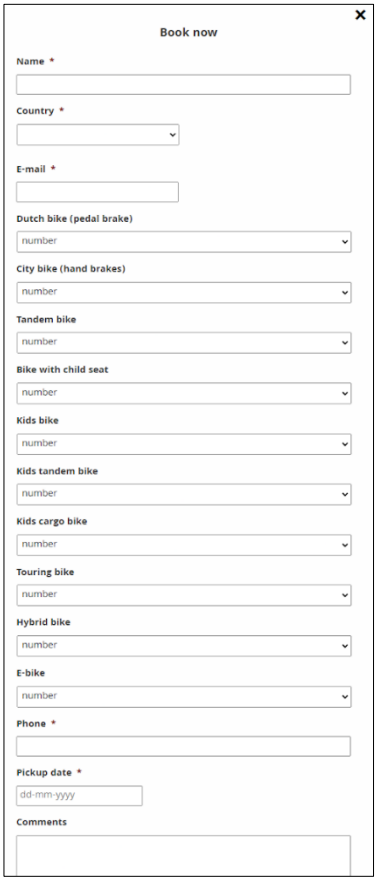


Figure 3 retnabike.nl reservation form

fontainebleaubikerental.com

This website is a French bicycle rental service. Similar to the previous example, we are greeted with a home page displaying various details and updates about their services, such as the prices of the different bicycle types.

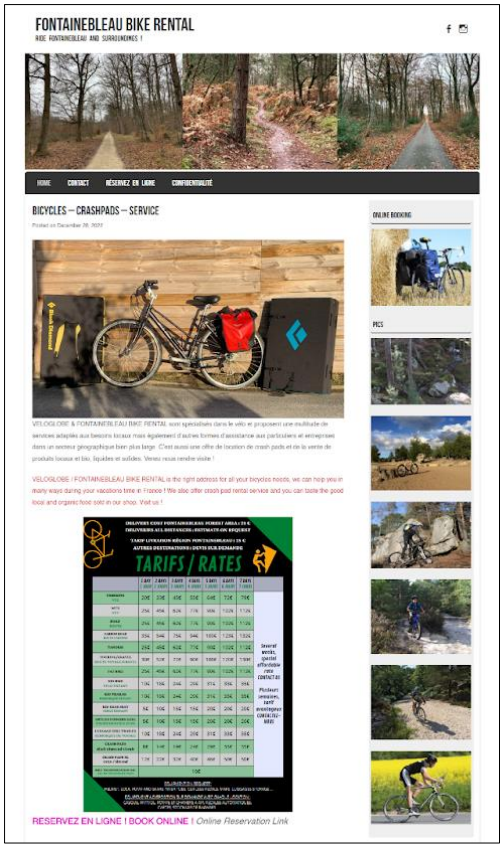


Figure 4 fontainebleaubikerental.com home page

The reservation process gives you the choice of the dates you wish to keep your bicycle and choosing the bicycle you want to take.

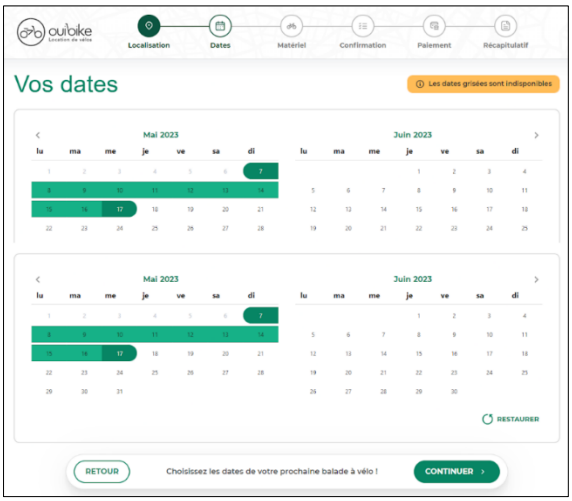


Figure 5 fontainebleaubikerental.com reservation dates

After defining the core functionalities of our web application, we should have a more detailed scheme in order to properly design the actual application in the upcoming chapters.

1.2. System Architecture and Entities

Users: For our application to be usable, we are required to properly separate the roles of our users, which will at the same time, give us the authorization levels for controlling access to the application.

- **Customers**: they are the users that interact with our application and benefit from our service. They should have restricted overall access with enough control over their own information.
- **Admins**: they are the users that monitor the flow of data through the application. They have a higher authorization level than customers which allows them to control more confidential information. There are multiple admin roles, this helps with the system organization and better control admin behaviors. They can be thought of as employees or the bicycle maintenance team.
- **Super Admins**: they are the users that have full access to the entirety of the application with no restriction whatsoever. They can be thought of as the higher administration within a company. There should be a single super admin but the possibility of more is still possible.

Bicycles and Bicycle Types: We are focusing our application for bicycle rental. In order to organize them, they are categorized into Bicycle Types.

Pricing Schemes: Regarding the pricing of our rentals, we associated various pricing schemes to the bicycle types. A pricing scheme dictates the unit price for renting a bicycle belonging to a certain bicycle type. Those units are:

- Per Hour (less than 24h)
- Per single day (1 to 2 days)
- Per 2 days (2 to 3 days)
- Per 3 days (3 to 4 days)
- Per 4 days (4 to 5 days)
- Per 5 days (5 to 6 days)
- Per extra day (for everyday over 5 days)

Reservations and Transactions: In order to make a rental, the user must have a customer account, once that's done there are 2 options:

- Make a reservation directly on the web application, then confirm it once at the pickup location. This method ensures the availability of the bicycle of choice, as well as flexibility in the pick-up and return dates.
- Physically coming to the store (or pickup location) and requesting a rental. We call this operation a transaction. This method does not ensure availability of the wanted bicycle and forces the customer to instantly pick it up once the transaction is confirmed.

The payment happens physically at the store location, before picking up the bicycle. This is mandatory for the customer in order to retrieve the key that unlocks the bicycle's lock.

GPS tracking: Once a transaction is confirmed, an embedded microcontroller within the selected bicycle will start sending the GPS locations to a real-time database in a repeated manner, tracking the concerned bicycle.

Returning the bicycle: At the end of the transaction, the customer should return their bicycle marking their transaction as "ended". If they exceed a certain amount of time to return their bicycle, this one will be marked as stolen and further measures should be taken from the Admins and Super Admins.

Customer score points and coupons: If the customer returns their bicycle within the time limits of the transaction (before the expected return time), they will receive a certain amount of score points which they can redeem into coupons to use for future transactions. These coupons act as reductions for the price of the transaction.

Bicycle contracts: A suggested feature to be implemented by the web application, is allowing customers to rent their own bicycles through the company. They can submit a contract request for the admins to review.

S*ubscriptions:* Customers can buy a monthly subscription plan. This operation would save our cyclers both time and money.

2. Web Applications

2.1. Definition

A web application is a computer program that utilizes web browsers and web technology to perform tasks over the Internet.

Web applications use a combination of server-side scripts (PHP and ASP) to handle the storage and retrieval of the information, and client-side scripts (JavaScript and HTML) to present information to users. This allows users to interact with the company using online forms, content management systems, shopping carts and more. In addition, the applications allow employees to create documents, share information, collaborate on projects, and work on common documents regardless of location or device[1].

2.2. Why a web application

Before exploring further aspects of web applications, an important question should be answered. Why implement this system with a web application?

- **Efficient storage:** Web applications store data on a server, so you don't have to install them on a hard drive. The ability to store data online allows companies to function without storage limitations, which can be particularly useful in remote or hybrid companies[2].
- **Few Compatibility issues:** Since web applications function using web browsers, they're usually accessible from a variety of devices. While native applications require certain operating systems and software, web applications work for anyone who can access the browsers that support them. Often, companies ask their employees to use the same browser when accessing certain applications so the information looks the same to everyone using the app, but most web browsers are free and adaptable to a wide range of computer and mobile devices. If an employee can't access their regular computer, they can still complete their work by using an alternate device[2].
- **Automatic updates:** A web application's connection to the internet allows the app developers to launch updates frequently, often without asking the users to do anything. These updates can ensure that users have the most current information, fix glitches and improve the user experience. While native applications often have a lengthy update process, web applications typically update fairly quickly if the user has a fast internet connection. This means that users can update their programs often without missing out on work time[2].

2.3. Web Application Types

We can differentiate two types of web applications:

- **Static Applications:** with no processing required from the server, these applications are mostly used for information display and portfolio websites. They often use simple HTML and CSS to display their data with no required logic.

- **Dynamic Applications:** these applications require server-side processing since a dynamic data flow exists between the client and the server. This type of application requires more processing power which HTML and CSS do not provide, hence the use of server dedicated languages (such as PHP, Python, C# …etc.).

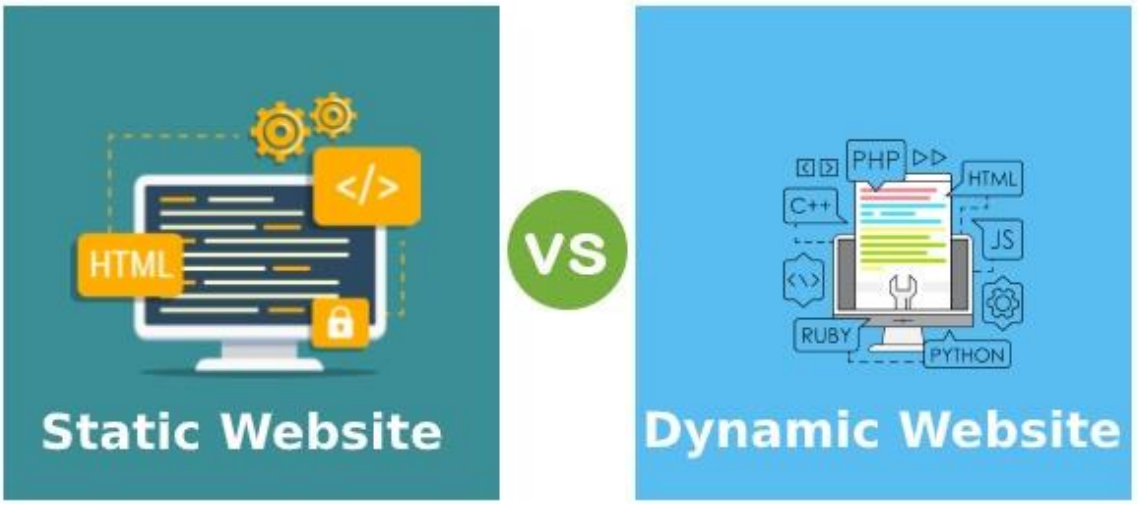


Figure 6 Static vs. Dynamic websites

When it comes to dynamic web applications, we can find two different kinds:

- **Multi-Page Applications:** MPAs are used for slow-growing data applications, where the instantaneous update of the data is not necessary. MPA's are usually large applications that require different pages. Any data change or data transfer to the server leads to a new page displayed in the browser[3].
- **Single-Page Applications:** An SPA is a type of web app loaded from one page, and all user interaction with this service is carried out, using one page[3]. The relevant data is then updated in a real time fashion from using JavaScript and advanced web protocols such as websockets, long polling, Server-Side Events (SSE) and others.

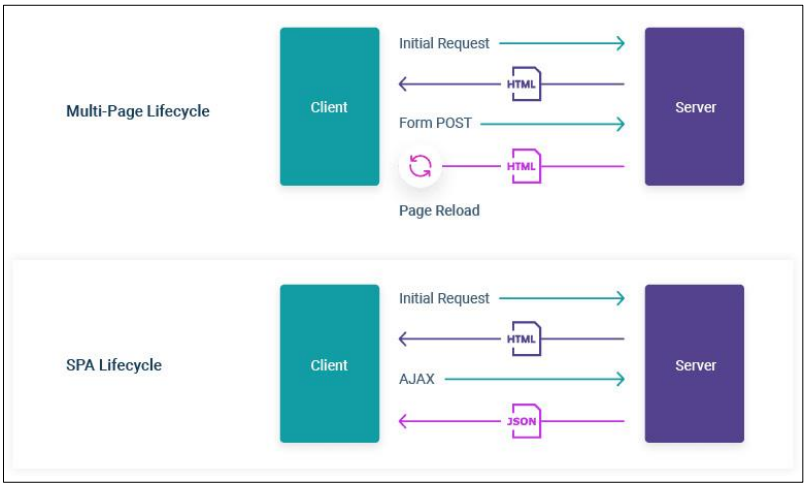


Figure 7 Multi-Page vs SPA Lifecycles

Nowadays, many dynamic web applications implement both techniques to offer an optimal user experience. Numerous technologies exist for server-side development, from ASP.NET to PHP and JSP and many others.

2.4. Frameworks

Frameworks are software that are developed and used by developers to avoid building applications from scratch. They are a group of libraries, API's and packages that are tied together, with a subtle difference from a regular library package.

Since they are built by several experienced software engineers, frameworks are versatile, robust and very efficient.

Using a framework as the base of a project lets the developers focus on the high-level functionalities of the application because any low-level operation is taken care of by the framework[4].

Frameworks usually manage both server-side and client-side operations. Some examples of the most popular web application frameworks are ASP.NET Core with C#, Laravel under PhP, NodeJs that utilizes JavaScript… etc.

2.4.1. Difference between Frameworks and Libraries

The difference between a library and a framework is that the latter calls the code. Opposite to this, the code calls the software library [4]. This principle is known as Inversion of Control (**IoC**) where the framework calls the code written in the application, instead of the code that calls functions from the libraries.

2.5. Client-Server Model

We have previously stated that a web application relies on client-server communication using several web protocols. The client-server model, or client-server architecture, is a distributed application framework dividing tasks between servers and clients, which either reside in the same system or communicate through a computer network or the Internet. The client relies on sending a request to another program in order to access a service made available by a server. The server runs one or more programs that share resources with and distribute work among clients[5].

2.5.1. Web Clients

Also known as service requesters, are pieces of computer hardware or server software that request resources and services made available by a server. Client computing is classified as Thick, Thin, or Hybrid.

- **Thick Client:** a client that provides rich functionality, performs the majority of data processing itself, and relies very lightly upon the server.
- **Thin Client:** a thin-client server is a lightweight computer that relies heavily on the resources of the host computer -- an application server performs the majority of any required data processing.
- **Hybrid Client:** possessing a combination of thin client and thick client characteristics, a hybrid client relies on the server to store persistent data, but is capable of local processing[5].

2.5.2. Web Server

A server is a device or computer program that provides functionality for other devices or programs. Any computerized process that can be used or called upon by a client to share resources and distribute work is a server[5]. Some common examples of servers include:

- **Application Server:** hosts web applications that users in the network can use without needing their own copy.
- **Computing Server:** shares an enormous amount of computer resources with networked computers that require more CPU power and RAM than is typically available for a personal computer.
- **Database Server:** maintains and shares databases for any computer program that ingests well-organized data, such as accounting software and spreadsheets.
- **Web Server:** hosts web pages and facilitates the existence of the World Wide Web.

2.5.3. HTTP and HTTPS

One of the famous communication protocols over the internet is HTTP/HTTPS.

HTTP stands for Hypertext Transfer Protocol, and it is a protocol used for transferring data over a network. Most information that is sent over the Internet, including website content and API calls, uses the HTTP protocol. There are two main kinds of HTTP messages: requests and responses.

HTTPS is HTTP with encryption and verification. The only difference between the two protocols is that HTTPS uses TLS (SSL) to encrypt normal HTTP requests and responses, and

to digitally sign those requests and responses. As a result, HTTPS is far more secure than HTTP. A website that uses HTTP has *http://* in its URL, while a website that uses HTTPS has *https://*.

2.5.3.1. HTTP Request

HTTP requests are generated by a user's browser as the user interacts with web properties. For example, if a user clicks on a hyperlink, the browser will send a series of "HTTP GET" requests for the content that appears on that page.

An HTTP request is just a series of lines of text that follow the HTTP protocol. A GET request might look like this:

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.63.0 libcurl/7.63.0 OpenSSL/1.1.1 zlib/1.2.11
Host: www.example.com
Accept-Language: en
```

Figure 8 HTTP Request example

This section of text, generated by the user's browser, gets sent across the Internet. The problem is, it's sent just like this, in plaintext that anyone monitoring the connection can read. (Those who are unfamiliar with the HTTP protocol may find this text hard to understand, but anyone with a baseline knowledge of the protocol's commands and syntax can read it easily)[6].

The primary or most commonly-used HTTP methods are POST, GET, PUT, PATCH, and DELETE. These methods correspond to create, read, update, and delete (or CRUD) operations, respectively[7].

2.5.3.2. HTTP Response

These HTTP requests go to a server, and that server will generate an HTTP response. HTTP responses are answers to HTTP requests which are similar:

```
HTTP/1.1 200 OK
Date: Wed, 30 Jan 2019 12:14:39 GMT
Server: Apache
Last-Modified: Mon, 28 Jan 2019 11:17:01 GMT
Accept-Ranges: bytes
Content-Length: 12
Vary: Accept-Encoding
Content-Type: text/plain
Hello World!
```

Figure 9 HTTP Response example

2.5.3.4. Communication Sequence

The following figure summarizes the client-server communication sequence.

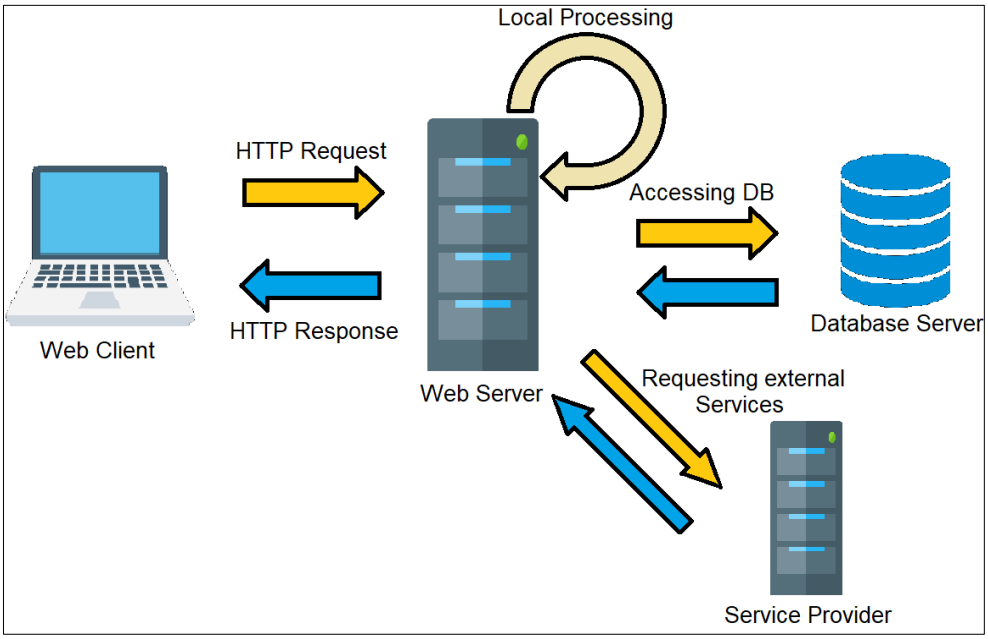


Figure 10 Client-Server Communication Sequence

The first action is initiated by the web client where an HTTP(S) Request is issued to the web server, this one receives the request and makes the necessary checks (in case of a secure connection) then proceeds to process the request, this could include requesting data from a local database, a different database server or even requesting services from an external service provider such as Google or Facebook. Finally, the web server sends an HTTP(S) Response back to the client containing the results of the initial request.

3. Embedded Systems

An embedded system is a microprocessor-based computer hardware system with software that is designed to perform a dedicated function, either as an independent system or as a part of a large system. At the core is an integrated circuit designed to carry out computation for real-time operations.

Complexities range from a single microcontroller to a suite of processors with connected peripherals and networks; from no user interface to complex graphical user interfaces. The complexity of an embedded system varies significantly depending on the task for which it is designed.

Embedded system applications range from digital watches and microwaves to hybrid

vehicles and avionics. As much as 98 percent of all microprocessors manufactured are used in embedded systems[8].

3.1. Microcontrollers

A microcontroller is a small and low-cost microcomputer, which is designed to perform the specific tasks of embedded systems like displaying information, receiving remote signals, and sending data to more powerful computers… etc. [9].

Microcontrollers include a microprocessor, memory and different peripherals depending on the manufacturer and the purpose of the microcontroller. Different brands manufacture microcontrollers such as Texas Instruments, Zilog, Toshiba and many others.

4. GPS Tracking Systems

GPS tracking is the surveillance of location through use of the Global Positioning System (GPS) to track the location of an entity or object remotely. The technology can pinpoint longitude, latitude, ground speed, and course direction of the target.

The GPS is a "constellation" of 24 well-spaced satellites that orbit the Earth and make it possible for people with ground receivers to pinpoint their geographic location. The location accuracy is anywhere from 100 to 10 meters for most equipment. Accuracy can be pinpointed to within one meter with special military-approved equipment. GPS equipment is widely used in science and has now become sufficiently low-cost so that almost anyone can own a GPS and many do in a smartphone, tablet or GPS navigation device[10].

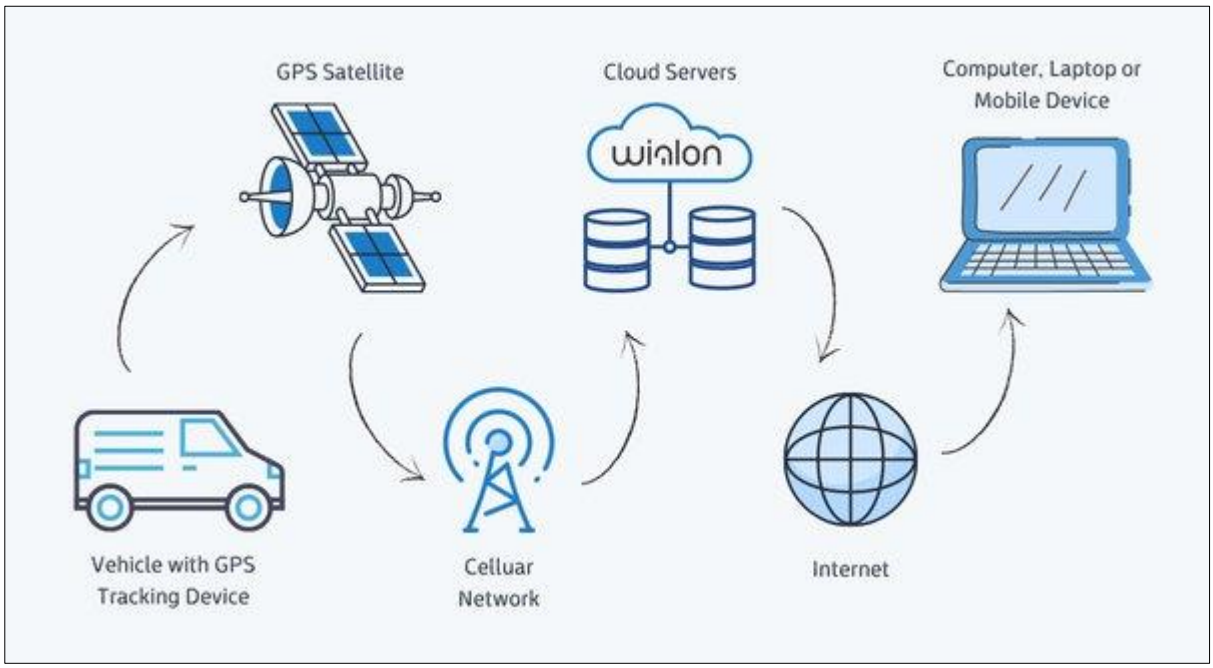


Figure 11 Realtime GPS Tracking Operation

In our project, we use GPS tracking on our bicycles for the following reasons:

- Display the course that the customer took in their previous rentals.
- In case a bicycle is stolen or lost, we can know its last known location.

Conclusion

In this first chapter, we introduced the basic idea behind our project (bicycle rentals), and the different functionalities that should be implemented from authorization levels to the rental process and GPS tracking. We also took a glimpse into the world of web applications and client-server communication protocols. In the next chapter, we will present the tools and technologies that allow us to properly use the previously presented building blocks.

CHAPTER II
Development Tools and Technologies

CHAPTER II Development Tools and Technologies

A variety of frameworks exist for both back-end and front-end development. Some require a certain expertise in software engineering, others are completely beginner friendly.

In this chapter we will cover the entire toolset for our development process, from the frameworks and techniques utilized to build our web application, to the different hardware components involved in the embedded system for our GPS tracker.

1. Software tools

We have already introduced web applications and their different types. Since our project involves data manipulation and verification, we concluded that implementing a **dynamic** application is the best approach. Building a **multi-page** application would be beneficial since our data is not rapidly changing and controlling access to different sections of the application can be easily managed that way.

1.1. MVC

Short for Model-View-Controller, is a pattern in software design commonly used to implement user interfaces, data, and controlling logic. It emphasizes a separation between the software's business logic and display. This **SoC** (separation of concerns) provides for a better division of labor and improved maintenance. Some other design patterns are based on MVC, such as MVVM (Model-View-Viewmodel), MVP (Model-View-Presenter), and MVW (Model-View-Whatever)[11].

1.1.1. Model

The first part of this design pattern is the Model, which defines the data structures and data types the app should be working with. If the data changes, the Model usually notifies both the View and the Controller for actions to be taken accordingly. As an example, in our application models would be BicycleType, Bicycle, PricingScheme… etc.

1.1.2. View

Second to the Model, we find the View which is the layer in the MVC architecture that is responsible for all the front-end displays that are sent to the client.

1.1.3. Controller

We finally have the Controller layer which is responsible for the logic that updates the model or the views in response to input from the clients. The controller is called so because it controls the flow of our web application. If a client initiates a server request (HTTP request) the controller receives it and decides on which action to perform.

1.1.4. Business

A common practice which helps achieve a better SoC (Separation of Concerns) is to separate the Business logic layer from the controller and model layers, thus creating a new layer called the Business Logic Layer or the Service Layer (not to be confused with external services)

1.2. Back-End Technologies

1.2.1. ASP.NET Core

ASP.NET Core is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled, Internet-connected apps. Millions of developers use or have used ASP.NET 4 to create web apps. ASP.NET Core is a redesign of ASP.NET 4, including architectural changes that result in a leaner, more modular framework[12].

Figure 12 ASP.NET Core Logo

ASP.NET Core offers a variety of templates so that anybody can start a new project with all the pre-configurations set up and ready to be used which allows developers to fully focus on their craft.

ASP.NET Core provides the following benefits: [12]

- A unified story for building web UI and web APIs.
- Architected for testability.
- Razor Pages makes coding page-focused scenarios easier and more productive.
- Blazor lets you use C# in the browser alongside JavaScript. Share server-side and client-side app logic all written with .NET.
- Ability to develop and run on Windows, macOS, and Linux.
- Open-source and community-focused.
- Integration of modern, client-side frameworks and development workflows.
- A cloud-ready, environment-based configuration system.
- Built-in dependency injection.
- A lightweight, high-performance, and modular HTTP request pipeline.

1.2.1.1. Dependency Injection

While talking about framework operation, we have stated that a certain Inversion of Control exists within the framework. ASP.NET Core is no stranger to this phenomenon and implements it using Dependency Injection.

ASP.NET Core MVC controllers request dependencies explicitly via constructors. ASP.NET Core has built-in support for dependency injection (DI). DI makes applicationss easier to test and maintain[13].

1.2.1.2. Routes

Navigating the web application, the client requests a certain functionality. The request reaches the server side of the program which will identify the request and execute the appropriate action of the corresponding controller. This path is the route and the handling of the route is an endpoint's responsibility.

Routing is responsible for matching incoming HTTP requests and dispatching those requests to the app's executable endpoints. Endpoints are the app's units of executable request-handling code. Endpoints are defined in the app and configured when the app starts. The endpoint matching process can extract values from the request's URL and provide those values for request processing. Using endpoint information from the app, routing is also able to generate URLs that map to endpoints[14].

In our application, we define the routes as follows

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

Figure 13 Route Configuration

The default route has the form of "ControllerName/ActionName/id?". The interrogation mark shows the possible existence of "id" within the route.

1.2.2. Databases

Different methods exist for storing data, from plain text files to highly secure databases. The common way of storing our model data in web development is using Databases.

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management

system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to database. There are multiple types of databases. The best database for a specific organization depends on how the organization intends to use the data[15].

In our case, we use two types of databases: Relational Databases and NoSQL Databases.

1.2.2.1. Relational Database

Relational databases use Structured Query Language (SQL) in their user and application program interfaces. A new data category can easily be added to a relational database without having to change the existing applications. A Relational Database Management System (RDBMS) is used to store, manage, query and retrieve data in a relational database[16].

In Relational Databases, the data is stored in the form of tables. Those tables are made of columns which represent the attributes of the data, and rows where each row represents one instance of the data. We can notice the convenience when it comes to structured data, since we can impose constraints on the attributes that must be respected to be inserted in the table.

a. Keys and Indices

In order to optimize the search across large data tables, we can define indices on frequently referred to columns or attributes. A special type of index is known as “Key”. We find two types of keys for two distinct uses.

The Primary Key: It is a constraint defined on one or more attributes in order to maintain the uniqueness of the rows within a table. Two rows of the same table will never have the same Primary Key. From the definition, database designers use a unique identifier as the primary key of tables, usually named “ID”.

The Foreign Key: It is a constraint defined on one attribute that refers to the primary key of another table. Foreign Keys are extremely useful since they allow us to divide the tables into smaller tables containing more specific and relevant data.

b. Relations in Databases

Relational Databases take advantage of the relations that exist between tables to optimize retrieving data from a large number of tables using the JOIN operation which constructs a singular table that includes the selected information.

c. Relation Types

In Relational Database Theory, entities are linked together using three different types of relations. They are differentiated by the conditions (uniqueness and multiplicity) on the two ends of the relation.

One to One: One entity of a table is uniquely related to a single other entity of another table. An example of a One-to-One relationship is Coupon-Transaction where one coupon can be applied to one transaction at a time.

One to Many: One entity of a table is to be related to any number of other entities of another table. An example of a One-to-Many relationship is BicycleType-Bicycle where one type can have multiple bicycles related to it.

Many to Many: Multiple entities of one table are related to multiple other entities of another table simultaneously. An example of a Many to Many relations is Admin-AdminRole where multiple admins can have multiple roles.

In the case of Many to Many relationships, a single foreign key between the two tables is not enough. Hence, we use a combination of two One to Many relationships using a third table called the Junction Table, where the first One to Many links the first table and the Junction Table, and the second One to Many links the second table with the Junction Table.

For the sake of storing our application's data such as user information, bicycle types, pricing schemes and other well-structured data, we use a relational database that uses Microsoft SQLServer as its RDBMS.

d. Microsoft SQL Server

Microsoft SQL Server is a relational database management system (RDBMS). Its primary function is to store and retrieve data requested by other applications. It supports various transaction processing, business intelligence and analytics applications, usually in corporate IT environments. Alongside Oracle Database and IBM's DB2, Microsoft SQL Server is one of the market-leading database technologies[17].

Figure 14 Microsoft SQL Server Logo

e. EntityFramework Core

Figure 15 Entity Framework Logo

Entity Framework Core, or EF Core, is a library that allows software developers to access databases. There are many ways to build such a library, but EF Core is designed as an object-relational mapper (O/RM). O/RMs work by mapping between the two worlds: the relational database with its own API, and the object-oriented software world of classes and software code. EF Core's main strength is allowing software developers to write database access code quickly[18].

We opted for EntityFramework Core since it is the recommended ORM to be used in an ASP.NET Core Application.

Entity Framework (EF) Core is a lightweight, extensible, open source and cross-platform version of the popular Entity Framework data access technology.

Table 1 EF Core mapping between a database and .NET software [19]

Relational Database	.NET software
Table	.NET class
Table columns	Class properties/fields
Rows	Elements in .NET collections
Primary keys: unique row	A unique class instance
Foreign keys: define a relationship	Reference to another class
SQL-for instance, WHERE	.NET LINQ-for instance, Where(p => …

1.2.2.2. NoSQL Database

NoSQL databases are good when dealing with large collections of distributed data. They can address big data performance issues better than relational databases. They also do well analyzing large unstructured data sets and data on virtual servers in the cloud. These databases can also be called non-relational databases[15].

Instead of tables, data is stored in documents and collections of documents. Inside of a same collection, documents can have different data types, with different constraints or no constraint at all. This lack of organization gives better performance while traversing and recovering the data.

We find the convenience in using a NoSQL Database for storing the incoming GPS data where we are sure that it will be fast growing, unstructured data.

a. Firebase Realtime Database

Figure 16 Firebase Realtime Database Logo

Conveniently, Google offers a NoSQL Database service. The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in real-time to every connected client. When you build cross-platform apps with our Apple platforms, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data[20].

b. Firesharp Library

Firesharp is a library that links our ASP.NET Core Web Application and the Firebase Realtime Database in an optimal and secure manner.

This library provides the following benefits: [21]

- Use Microsoft HTTP Client Libraries instead of RestSharp.
- Supports Streaming from the REST API (Firebase REST endpoints support the EventSource / Server-Sent Events protocol).
- It is fully asynchronous and designed to be non-blocking.

1.3. Front-End Technologies

1.3.1. HTML

HTML, or HyperText Markup Language, is the fundamental language used for creating web pages. It provides structure and defines the meaning of the content displayed on a page. HTML is typically used in conjunction with CSS (Cascading Style Sheets) for styling and JavaScript for interactivity.

The term “HyperText” refers to the ability of navigating through links that connect different web pages, making the web interconnected and accessible. “MarkupLanguage”, on the other hand, refers to a system that uses specific symbols (or tags) inserted in a text document to control its structure, formatting, and the relationship between its parts[22].

1.3.2. CSS

CSS, or Cascading Style Sheets, is a style sheet language used to define the visual appearance and layout of HTML documents. While HTML focuses on the structure and content of a web page, CSS is responsible for the presentation and styling aspects.

1.3.3. JavaScript

JavaScript, often abbreviated as JS, is a widely-used scripting language that allows developers to add interactivity, functionality, and dynamic behavior to web pages.

JavaScript can be used to perform a wide range of tasks on both the front-end side and the back-end side. It allows developers to create and implement interactive elements, handle events, create animations, validate forms, and much more.

1.3.4. jQuery

jQuery is a popular JavaScript library that simplifies the use of JavaScript on web pages. Developers can write shorter and more readable code compared to plain JavaScript code. jQuery achieves this by providing a set of methods that hide away the complexities of JavaScript.

1.3.5. Bootstrap

Bootstrap is a free, open-source front-end development framework for the creation of websites and web apps and the most popular CSS framework [23].

Bootstrap offers a wide range of pre-defined templates and components, such as forms, modals, navigation bars, tables and more. These ready-to-use templates can be easily implemented in projects, saving developers time and effort.

1.3.6. Leaflet

Leaflet is the leading open-source JavaScript library for interactive maps. It is simple to use, efficient and lightweight [24]. One of the key advantages of Leaflet is its simplicity. Leaflet provides a clear API that shields away the complexities of working with maps, making it accessible to developers. Furthermore, Leaflet makes customization easily achievable with its wide range of features and options for creating maps, including markers, paths, and tile layers.

Even though the MVC model allows us to utilize any kind of front-end technology, ASP.NET core offers a very convenient way to connect the front-end to the back-end of the application with its “Razor pages” and “Blazor” technologies.

1.3.7. Razor Pages

Razor is a templating engine that combines C# with HTML to build dynamic web content[25].

The file extension ".cshtml" refers to a Razor Page file which implies that the contents of the file are C# and HTML. This technology is very convenient since we can make verifications on the data and control the display directly on the page.

2. Hardware tools

In order to build our GPS tracker, we opted for a microcontroller-based system. These systems allow for a certain degree of freedom in their design, since we can use different peripherals that would otherwise be incompatible with each other.

2.1. Arduino Uno Microcontroller

Arduino Uno is a microcontroller board based on the ATmega328P, an 8bit microprocessor.

All Arduino boards have at least one serial port (also known as a UART or USART), and some have several[26].

This UART port allows our Arduino board to communicate with the peripherals that will be used to build the GPS tracker.

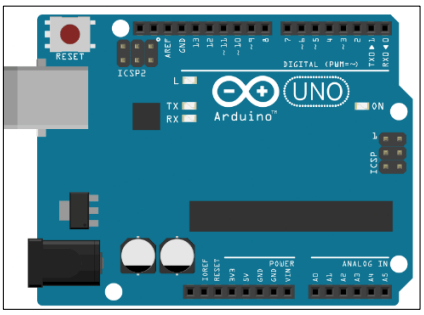


Figure 17 Arduino Microcontroller Board

The microcontroller has access to a flash memory of 32KB. This limitation can affect various aspects of the design process. Space optimization should be a top priority when writing any code for the microcontroller.

2.2. NEO 6MV2 GPS

The **NEO-6MV2** is a **GPS** (Global Positioning System) module and is used for navigation. The module simply checks its location on earth and provides output data which is the longitude and latitude of its position. It is from a family of stand-alone GPS receivers featuring the high-performance u-blox6 positioning engine. These flexible and cost-effective receivers offer numerous connectivity options in a miniature package[27].

This module supports the UART communication protocol making it directly compatible with the microcontroller.

The following figure shows the pin diagram of the NEO 6MV2

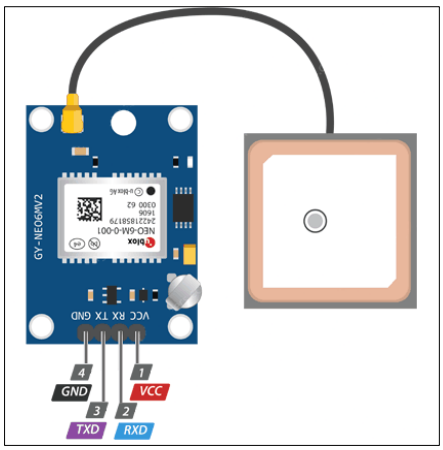


Figure 18 NEO 6MV2 Pin Diagram

When powered, the NEO 6MV2 starts to locate the GPS satellites in order to recover the current position on the surface of earth. This operation requires a clear view of the sky and any obstruction might affect the accuracy of the reading.

After various tests we came to the following results:

- If used indoors, the module cannot find any satellite and has no reading of the position.
- If used outdoors and the sky is clear (no clouds or light clouds), the module returns the position with an accuracy of a 5-10 meters radius.
- Is used outdoors and the sky is covered (heavy clouds), the module returns the position with an accuracy of a 30 meters radius.

2.3. Sim800L

Sim800L is a quad-band GSM/GPRS (Global Systems for Mobile/General Packet Radio Service) module, that works on frequencies GSM850MHz, EGSM900MHz, DCS1800MHz and PCS1900MHz [28].

This module is a modem that connects our system to the wireless network of the mobile provider that is accessed by the used SIM card.

The Sim800L and similar modules are used to send/receive SMS, initiate/intercept phone calls. If the chip supports GPRS, it can access the Mobile Data Network and carry out HTTP requests. The type and complexity of requests depend on the brand and quality of the used module. Sim800L only supports HTTP POST and HTTP GET requests.

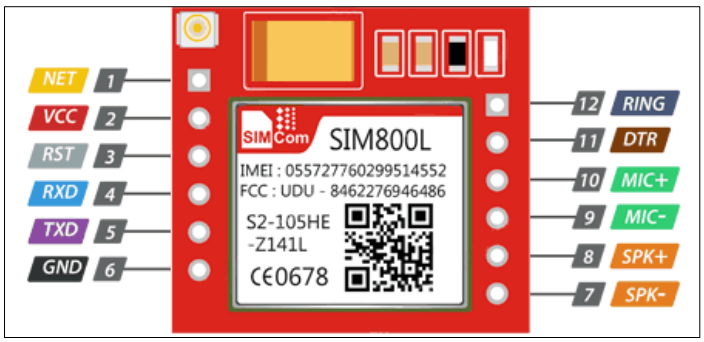


Figure 19 SIM800L Pin Diagram

Using a sophisticated and expensive module would unlock a wider range of functionalities and the ability to use complex protocols and have a more reliable data transfer.

2.3.1. AT commands

AT stands for Attention and these commands are used to control modems[29].

In application, controlling device controls the GSM engine by sending AT Command via its serial interface[30].

In order to communicate with the GSM/GPRS networks, we transmit a set of command words that are known as the AT Commands to the Sim800L. The complete set of AT Commands recognized by SIM800L are found in the dedicated manual.

Conclusion

In this chapter, we introduced in detail the tools and technologies that we used in the development process of our Web Application, alongside the modules and techniques that will be used in building the GPS tracker. In the next chapter, we will discuss the design process of each aspect of our system.

CHAPTER III
System Design

CHAPTER III System Design

Following the guidelines set by the MVC architecture, we utilize Microsoft's ASP.NET Core framework and the environment around it, from libraries to other frameworks, in order to build our Web based Application.

Our application follows the market standards and the various programming conventions for the absolute best user experience. We also kept in mind the simplicity of the program behind the application so that it can be easily maintained by other developers.

In this third chapter, we will finely detail the functional design of the application using the Unified Modeling Language (UML), from the different actors that interact with the system and their use case diagrams, to the sequence diagrams of every operation. Finally, we will take a deep look into the design of the GPS Tracker at both the hardware and the software level.

1. Programming principles

In order to write a readable and maintainable code, following a set of programming principles and rules is mandatory. Some of the unwritten conventions we worked with at all times are the following:

DRY: short for Don't Repeat Yourself. Specifically used for writing efficient functions where no two snippets of code are executing the same logic. If code is repeated, it must be encapsulated within a function.

KISS: Keep It Simple. Whenever designing a system, simplicity is advised over complication. A set of simple functionalities grow into a yet complex, but simple system.

SoC: Separation of Concerns. Previously seen at the MVC section. This rule states that a function is supposed to solve a single problem. Applying this rule produces short and readable functions that have different atomic operations. It drastically helps while debugging and maintaining the program.

2. Modeling Language

A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure. A modeling language can be graphical or textual.

Graphical modeling languages use diagram techniques with named symbols that represent concepts and lines that connect the symbols and that represent relationships and various other

graphical annotations to represent constraints. Textual modeling languages typically use standardized keywords accompanied by parameters to make computer-interpretable expressions[31].

2.1. Unified Modeling Language

UML, short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems.

UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects[32].

In order to model systems, UML imposes two main types of diagrams:

- *Structure diagrams:* they show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. In this category, we can find seven subtypes of Structure diagrams:
  - Class Diagram
  - Component Diagram
  - Deployment Diagram
  - Object Diagram
  - Package Diagram
  - Composite Structure Diagram
  - Profile Diagram
- *Behavior diagrams:* they show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time. We find another seven subtypes for this category:
  - Use Case Diagram
  - Activity Diagram
  - State Machine Diagram
  - Sequence Diagram
  - Communication Diagram
  - Interaction Overview Diagram
  - Timing Diagram

During our design process, we are using two of the previously mentioned diagrams, alongside a special kind of Structure Diagram:

Entity Relationship Diagram: An ERD is a representation of data within a domain. It consists of entities as well as relationships between entities[33]. An entity is generally a reference to a table in the database. Every entity comes with its own set of attributes (or columns).

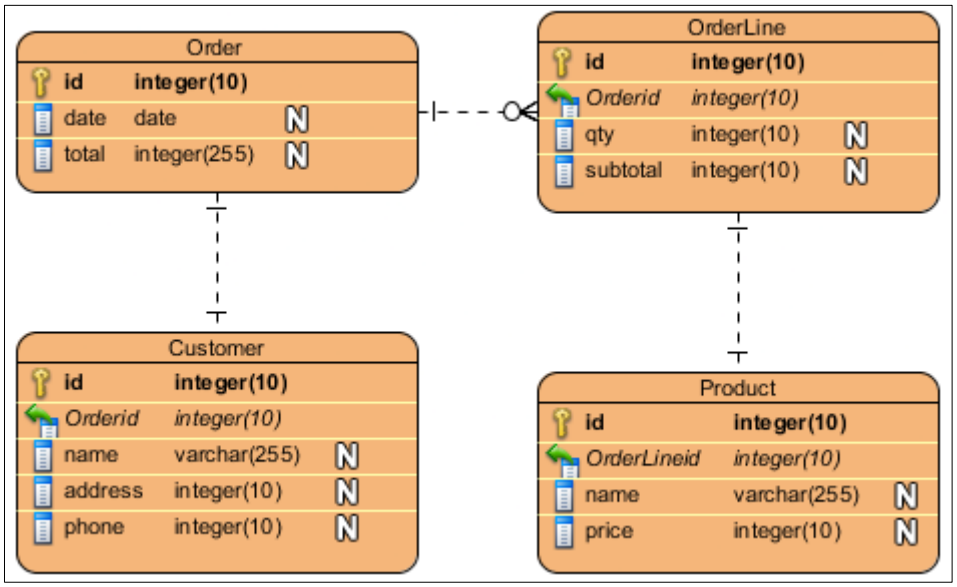


Figure 20 Entity Relationship Diagram Example

Use Case Diagram: A use-case model describes a system's functional requirements in terms of use cases. It is a model of the system's intended functionality (use cases) and its environment (actors). Use cases enable you to relate what you need from a system to how the system delivers on those needs. Because it is a very powerful planning instrument, the use-case model is generally used in all phases of the development cycle by all team members[32].

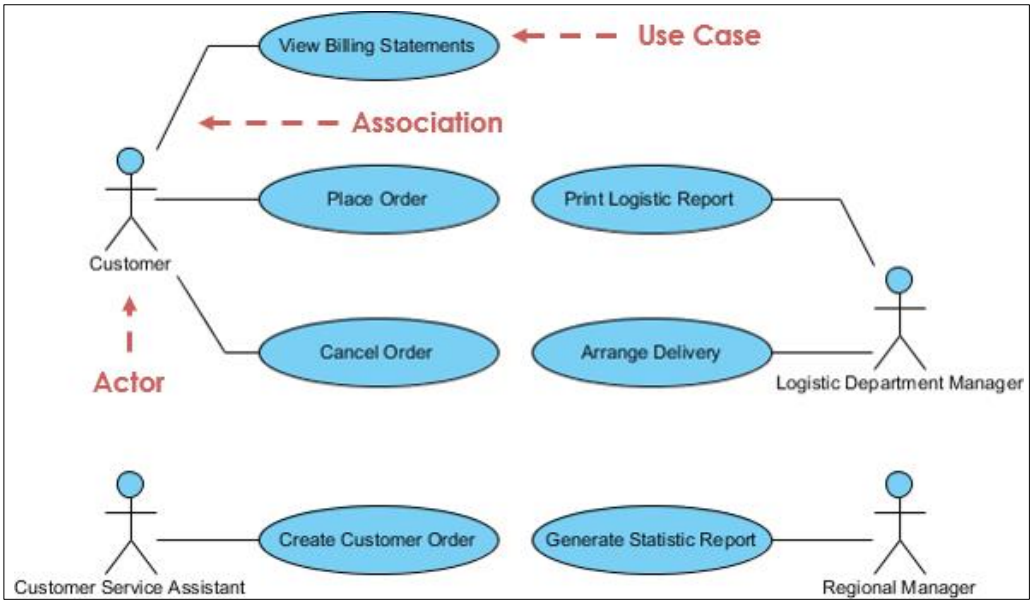


Figure 21 Use Case Diagram Example

Sequence Diagram: The Sequence Diagram models the collaboration of objects based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case[32].

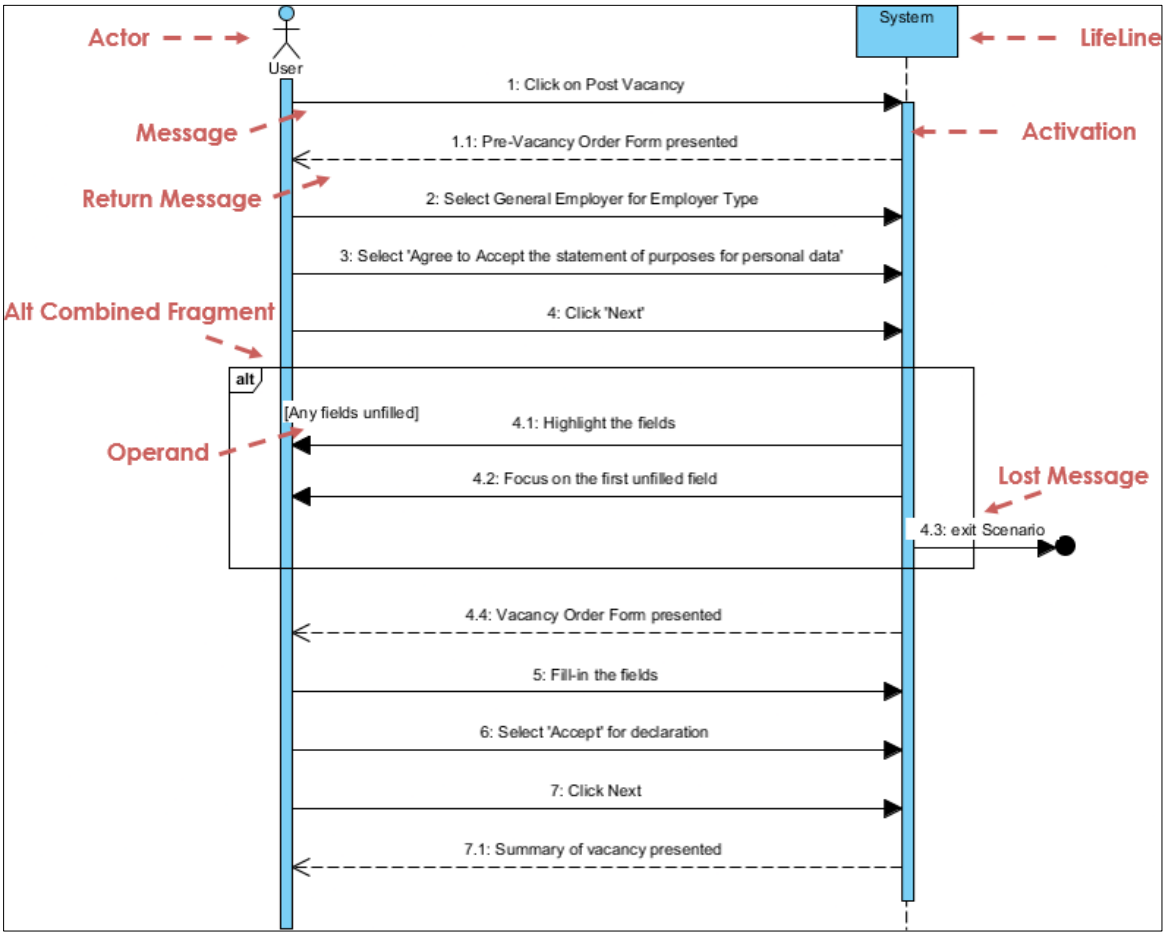


Figure 22 Sequence Diagram Example

3. Web Application Design

3.1. Modeling

3.1.1. System actors

A System Actor represents any entity that interacts with the application, it could be a human, a machine, an organization or any other program.

Since we are using the MVC architecture, we can easily isolate the data from the rest of the application (business logic, configurations, authentication services… etc.) which is represented by the Models.

Our application expects three different actors:

- ***SuperAdmins:*** A SuperAdmin has access to the entire system without restriction, granting him/her total control over the application with all possible privileges.

- ***Admins:*** An Admin has partial access to the system that allows him/her to control a certain area of the application. This makes the repartition of tasks easier within the management team and helps supervisors track all the actions of admins. The restriction is controlled using a Role. An Admin can have multiple of the six different roles
  - *Transactions:* Admins with this role are able to create, edit and delete transactions.
  - *Reservations:* Admins with this role are able to create, edit, delete, confirm or cancel reservations.
  - *Bicycles:* Admins with this role have access to all bicycles, bicycle types and bicycle contracts data types and all their related actions.
  - *Customers:* Admins with this role have access to customers, subscription plans, subscriptions, coupon types and coupons data types with all their related actions.
  - *Admins:* Admins with this role have access to the admin data type and all related actions.
  - *StoreTerminal:* this is a special role since admins connected with this role don't have access to any data. Instead, this role's purpose is to allow customers to make instantaneous transactions at the store's location or confirm reservations.
- ***Customers:*** They are the clients to our business. Customers can create an account, modify their user details, redeem coupons, browse the bicycle types, browse subscriptions, view their past transactions and make reservations.

3.1.2. Use Cases

Use cases are a set of actions and operations that actors can execute at different levels of the system.

3.1.2.1. Actors' Use Cases Description

The following table describes all the use cases for each actor depending on their roles.

Table 2 Use Case Descriptions

Actors	Roles	Use Cases
SuperAdmin	-	**-Login:** A Super Admin receives their privileges at the authentication after logging into the system through the login action. **-Logout:** Leaving a client with a SuperAdmin account logged in is a serious security problem, logging out will remove all authorizations and authentications from the client. **-Manage Admins:** list all admins, edit a particular admin, view the details of a particular admin, create a new admin account and delete an existing admin. **-Manage Customers:** list all customers, edit a particular customer, view the details of a particular customer, create a new customer account and delete an existing customer. **-Manage Bicycle Types:** List all bicycle types, edit a particular type, view the details of a particular bicycle type, create a new bicycle type and delete an existing bicycle type. **-Manage Bicycles:** List all bicycles, edit a particular bicycle, view the details of a particular bicycle, create a new bicycle, associate a bicycle type to it and delete an existing bicycle. **-Manage Subscription Plans:** List all subscription plans, view the details of a particular plan, create a new subscription plan, edit a particular subscription plan, enable or disable a plan and delete an existing plan. **-Manage Subscriptions:** List all subscriptions of a particular customer, view the details of an existing subscription and create a new subscription for a particular customer. **-Manage Coupon Types:** List all coupon types, view the details of a particular type, create a new type or delete an existing coupon type. **-Manage Coupons:** List all coupons of a particular customer and view the details of a particular coupon. **-Manage Reservations:** list all reservations, view the details of a particular reservation, edit a particular reservation, create a new reservation, confirm or cancel an existing reservation. **-Manage Transactions:** list all transactions, view the details of a particular transaction, edit a particular transaction, delete an existing transaction, create a new transaction or mark a current one as ended (returning a bicycle)
Admin	-	**-Login:** An Admin receives their authorizations and roles when authenticated after logging into the system through the login action. **-Logout:** Leaving a client with an Admin account logged in is a serious security problem, logging out will remove all authorizations and authentications from the client.
	Admins	**-Manage Admins:** list all admins, view the details of a particular admin and delete an existing admin.

	Customers	**-Manage Customers:** list all customers, edit a particular customer, view the details of a particular customer, create a new customer account and delete an existing customer. **-Manage Subscription Plans:** list all subscription plans, view the details of a particular plan **-Manage Subscription:** list all subscriptions of a particular customer, view the details of an existing subscription and create a new subscription for a particular customer. **-Manage Coupon Types:** list all coupon types, view the details of a particular type. **-Manage Coupons:** list all coupons of a particular customer and view the details of a particular coupon.
	Bicycles	**-Manage Bicycle Types:** list all bicycle types, edit a particular type, view the details of a particular bicycle type, create a new bicycle type and delete an existing bicycle type. **-Manage Bicycles:** list all bicycles, edit a particular bicycle, view the details of a particular bicycle, create a new bicycle, associate a bicycle type to it and delete an existing bicycle. -**Manage Bicycle Contracts:** list all bicycle contracts, view the details of a particular one, confirm or deny it.
	Reservations	**-Manage Reservations:** list all reservations, view the details of a particular reservation, edit a particular reservation, confirm or cancel an existing reservation.
	Transactions	**-Manage Transactions:** list all transactions, view the details of a particular transaction, edit a particular transaction, delete an existing transaction, create a new transaction, mark a current one as ended (returning a bicycle) or cashing in a transaction

Customer		**-Register:** All users are free to create a customer account with a unique username and email address. **-Login:** A customer will be authenticated and will receive their permissions when logged into the system. **-Logout:** Leaving a client with a customer account logged in is a security issue for the customer, logging out will remove all authorizations and authentications from the client. **-Manage personal User Information:** View the details of the connected account, edit those details and change the password. **-Browse Bicycle Types:** list all the bicycle types and view the details of a particular bicycle type. **-Manage Personal Bicycle Contracts:** list all the contracts the customer has created, view the details of a particular contract, cancel it, or create a new contract. **-Manage personal Reservations:** list all current and canceled reservations related to the same account, view the details of a particular reservation, create a new reservation and cancel an existing reservation. **-Manage personal Transactions:** list all current transactions related to the same account, view the details of a particular transaction, create a new transaction (through the store terminal). **-Browse Subscription Plans:** list all subscription types and view the details of a particular subscription type. **-Browse Coupon Types:** list all coupon types, view the details of a particular coupon type and redeem a coupon of a particular type. **-Manage personal Subscriptions:** list all active subscriptions of the same account, view the details of a particular subscription and delete an existing subscription. **-Manage personal Coupons:** list all coupons of the same account and view the details of a particular one.

3.1.2.3. Use Case Diagrams

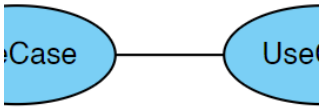
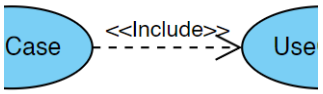
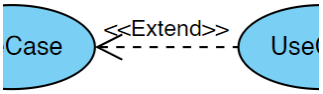
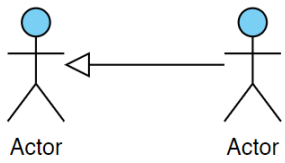
A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system[34].

Functional requirements: also known as the Use Cases. They describe an action that an actor can perform. A use case is depicted as an ellipse in a use case diagram.

Actors: any entity that interacts with the application, it could be a human, a machine, an organization or any other program. They are depicted with a stickman figure in the use case diagram.

Relationships: Used to represent the direct or indirect relationship between actors and use cases, or between actors. They are shown as arrows with specific directions, bodies and names. The following table summarizes the used relationships[35].

Table 3 Relationship Types in a Use Case Diagram

Relationship	Symbol	Description
Association		In UML models, an association is a relationship between two classifiers, such as classes or use cases, that describes the reasons for the relationship and the rules that govern the relationship.
Include		In UML modeling, an include relationship is a relationship in which one use case (the base use case) includes the functionality of another use case (the inclusion use case). The include relationship supports the reuse of functionality in a use-case model.
Extend		In UML modeling, you can use an extend relationship to specify that one use case (extension) extends the behavior of another use case (base). This type of relationship reveals details about a system or application that are typically hidden in a use case.
Generalization		In UML modeling, a generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent). Generalization relationships are used in class, component, deployment, and use-case diagrams to indicate that the child receives all of the attributes, operations, and relationships that are defined in the parent.

We can now illustrate in the following section, the various use case diagrams that govern the user interaction design of our web application. After that, we will give the textual

descriptions of these use cases, where we explain how actors can reach each action within the application, and what are the expected behaviors of the system.

The following figure represents the Use Case Diagram of the Admin actor with all the roles available. Removing a "role" results in the restriction in the access to the use cases related to that specific role. Some exceptions exist regarding the display actions such as "List" and "Details", where an admin without a specific role can still access those display actions.

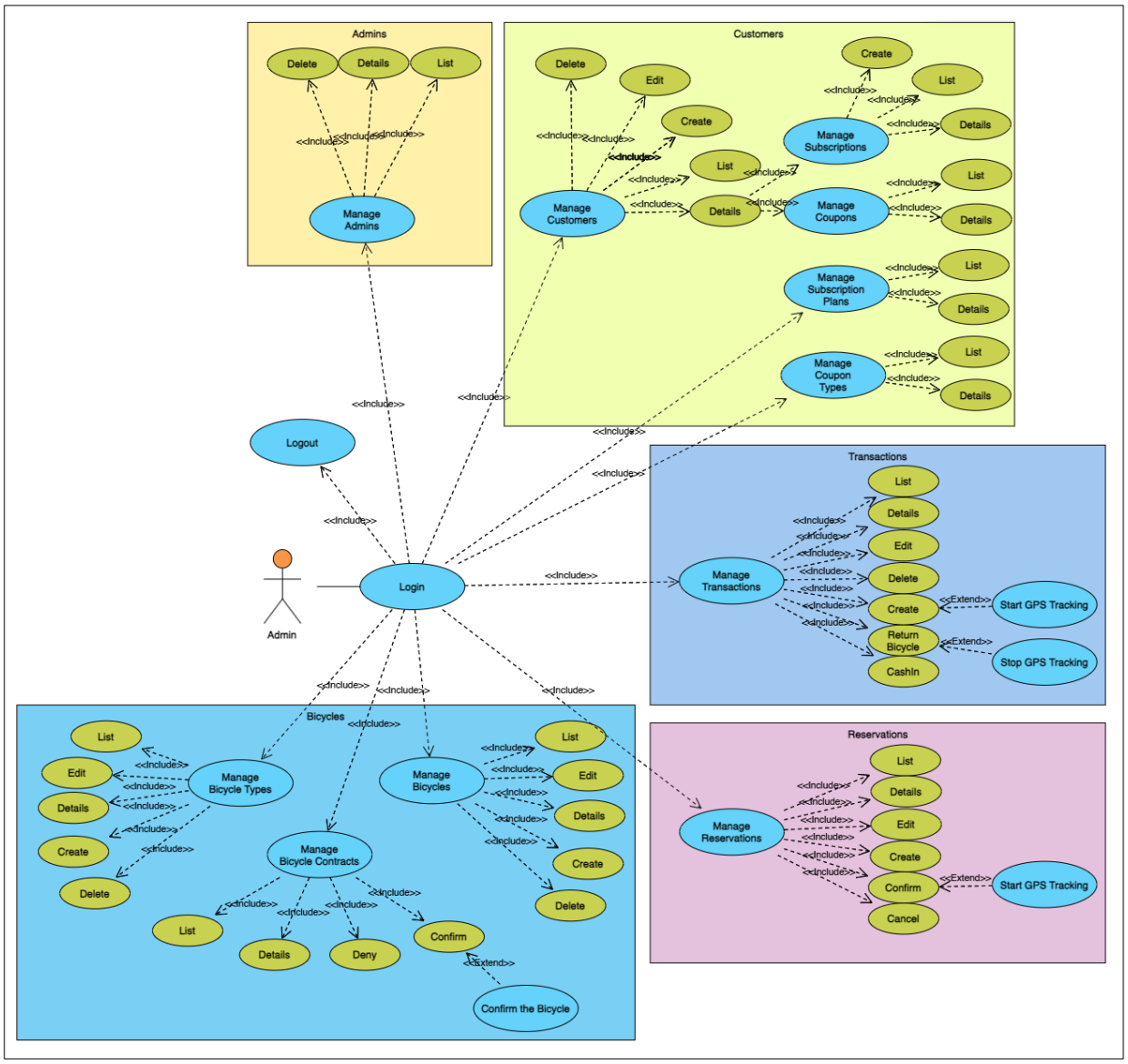


Figure 23 Admin with Roles Use Case Diagram

This figure represents the Use Case Diagram for the Customer Actor. Customers have a single role and thus there is no need to separate Use Cases.

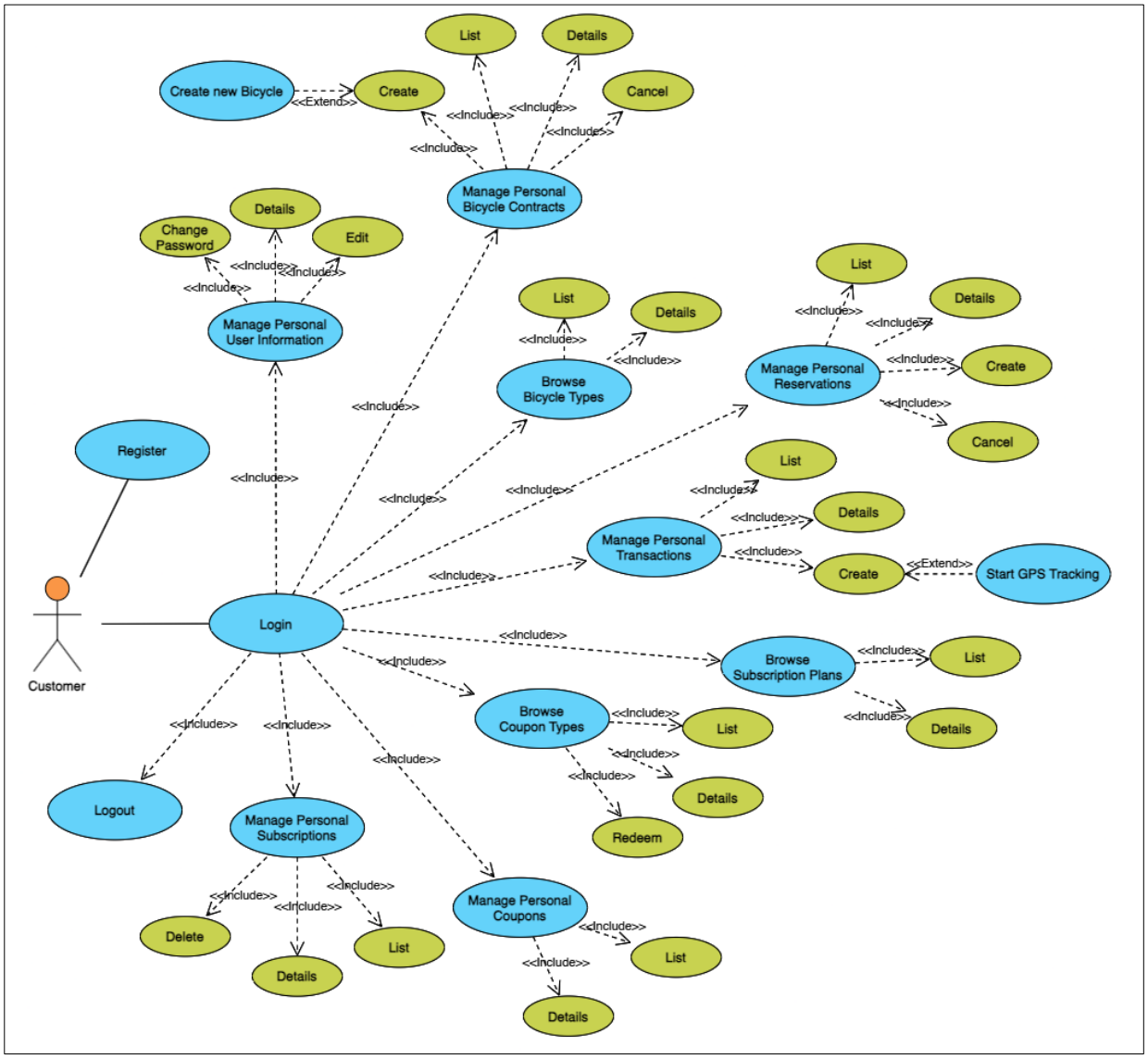


Figure 24 Customer Use Case Diagram

The following figure represents the Use Case Diagram for the SuperAdmin actor. SuperAdmins are a Generalization of an Admin having all roles.

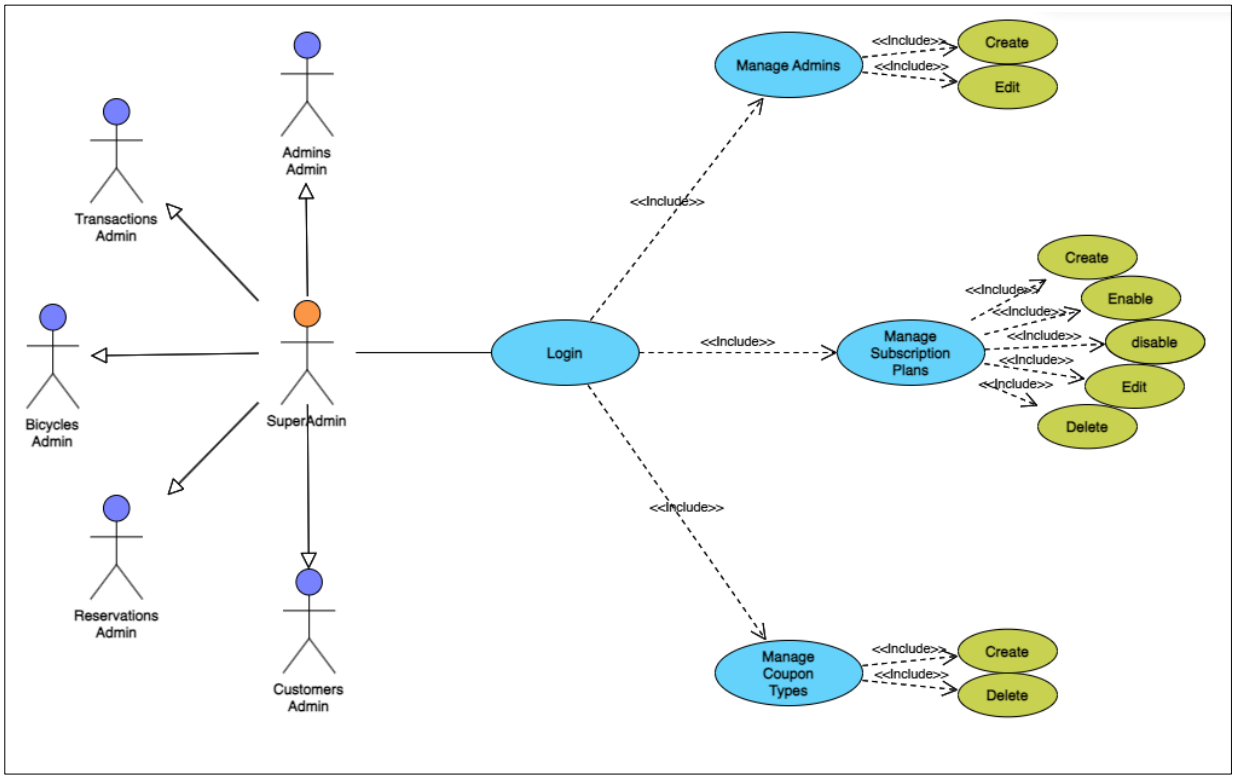


Figure 25 SuperAdmin Use Case Diagram

3.1.2.4. Textual Description of Use Cases

Use Case 01: Login

Table 4 Textual Description for the use case "Login"

Use Case name	Login
Actor	SuperAdmin, Admin, Customer
Objective	Authentication to receive permissions to use otherwise restricted functionalities of the application.
Precondition	User is Registered.
Scenario	1.The user clicks on “Login” via the navigation bar at the top. 2.The user enters account information (username and password) and clicks on “Login”. 3.The server receives the entered information and checks if the username exists. 4.The server will encrypt the password and verify if it matches the stored password in the database. 5.The server will encapsulate some user details in the authentication cookie and will log the user in the application. 6.The user is now logged in and can access all the previously restricted functionalities.
Alternative	-The entered username is either empty or does not exist in the database. (Return to 2 from 3) -The entered password is either empty or does not match the one stored in the database. (Return to 2 from 4)

Use Case 02: Register

Table 5 Textual Description for the use case "Register"

Use Case name	Register
Actor	Customer
Objective	Create a new customer account in order to use the web application.
Scenario	1.User clicks on “Register” via the navigation bar at the top. 2.The user enters personal information (first name, last name, username, password, date of birth, email) and clicks on “Confirm” 3.The browser will verify that there is no missing information and all fields have valid data. 4.The server receives the entered information and checks if the username and email do not already exist in the database. 5.The server will encrypt the password. 6.The server saves the new customer account. 7.The user is sent back to the home page.
Alternative	-The browser will display error messages under the fields with wrong data. (Return to 2 from 3) -Username or Email already exist in the database, the server sends an error. (Return to 2 from 4)

Use Case 03: Manage Admins

Table 6 Textual Description for the use case "Manage Admins"

Use Case name	Manage Admins
Actor	SuperAdmin, Admin
Objective	List, Details, Create, Edit, Delete Admins
Scenario	1.User clicks on “Admins” via the navigation bar at the top. 2.Server receives the Http Get request and sends the Index page for admins. 3.User clicks on “Details” next to any admin on the list and the server sends the Details page for the requested admin. 41. User clicks on “Edit” next to any admin on the list, the server receives the get request and first checks the role of the admin, then sends the edit form of the requested admin.

	4.2. User fills the edit form and clicks on confirm, the server verifies the data and updates the existing admin in the database. 5.1. User clicks on “Delete” next to any admin on the list, the server receives the get request and first checks the role of the admin, then asks for confirmation from the user. 5.2. User clicks on “Confirm” and the server will mark the admin as removed. 6.1. User clicks on “Add Admin” at the top of the list in the Index page, the server receives the get request and first checks the role of the admin, then sends the creation form. 6.2. User fills the creation form and clicks on “Create”. The server will receive the information and make all necessary verifications. 6.3. If all verifications are complete, the new admin is added to the database and the user is sent back to the admins Index page.
Alternative	-If the admin doesn’t have the necessary role “Admins”, the server will send a forbidden error code 403. (Return to 2 from 4.1) -If the edit data is erroneous, the server will resend the Edit form with the appropriate error messages. (Return to 4.1 from 4.2) -If the admin doesn’t have the necessary role “Admins”, the server will send a forbidden error code 403. (Return to 2 from 5.1) -If the user is not a SuperAdmin, the server will send a forbidden error code 403. (Return to 2 from 6.1) -If the create data is erroneous, the server will resend the Create form with the appropriate error messages. (Return to 6.1 from 6.2)

Use Case 04: Manage Customers

Table 7 Textual Description for the use case "Manage Customers"

Use Case name	Manage Customers
Actor	SuperAdmin, Admin
Objective	List, Details, Create, Edit, Delete Customers
Scenario	1.User clicks on “Customers” via the navigation bar at the top. 2.Server receives the Http Get request and sends the Index page for customers. 3.User clicks on “Details” next to any customer on the list and the server sends the Details page for the requested customer.

	4.1. User clicks on “Edit” next to any customer on the list, the server receives the get request and first checks the role of the admin, then sends the edit form of the requested customer. 4.2. User fills the edit form and clicks on confirm, the server verifies the data and updates the existing customer in the database. 5.1. User clicks on “Delete” next to any customer on the list, the server receives the get request and first checks the role of the admin, then asks for confirmation from the user. 5.2. User clicks on “Confirm” and the server will mark the customer as removed. 6.1. User clicks on “Add customer” at the top of the list in the Index page, the server receives the get request and first checks the role of the admin, then sends the creation form. 6.2. User fills the creation form and clicks on “Create”. The server will receive the information and make all necessary verifications. 6.3. If all verifications are complete, the new customer is added to the database and the user is sent back to the customers Index page.
Alternative	-If the admin doesn’t have the necessary role “Customers”, the server will send a forbidden error code 403. (Return to 2 from 4.1) -If the edit data is erroneous, the server will resend the Edit form with the appropriate error messages. (Return to 4.1 from 4.2) -If the admin doesn’t have the necessary role “Customers”, the server will send a forbidden error code 403. (Return to 2 from 5.1) -If the admin does not have the appropriate role “Customers”, the server will send a forbidden error code 403. (Return to 2 from 6.1) -If the create data is erroneous, the server will resend the Create form with the appropriate error messages. (Return to 6.1 from 6.2)

Use Case 05: Manage Bicycle Types

Table 8 Textual Description for the use case "Manage Bicycle Types"

Use Case name	Manage Bicycle Types
Actor	SuperAdmin, Admin
Objective	List, Details, Create, Edit, Delete Bicycle Types
Scenario	1.User clicks on “Bicycle Types” via the navigation bar at the top. 2.Server receives the Http Get request and sends the Index page for bicycle types. 3.User clicks on “Details” next to any bicycle type on the list and the server sends the Details page for the requested bicycle type. 4.1. User clicks on “Edit” next to any bicycle type on the list, the server receives the get request and first checks the role of the admin, then sends the edit form of the requested bicycle type. 4.2. User fills the edit form and clicks on confirm, the server verifies the data and updates the existing bicycle type in the database. 5.1. User clicks on “Delete” next to any bicycle type on the list, the server receives the get request and first checks the role of the admin, then asks for confirmation from the user. 5.2. User clicks on “Confirm” and the server will mark the bicycle type as removed. 6.1. User clicks on “Add Bicycle Type” at the top of the list in the Index page, the server receives the get request and first checks the role of the admin, then sends the bicycle type creation form. 6.2. User fills the creation form with all the bicycle type details and the pricing scheme details, then clicks on “Create”. The server will receive the information and make all necessary verifications. 6.3. If all verifications are complete, the new bicycle type is added to the database and the user is sent back to the bicycle types Index page.
Alternative	-If the admin doesn’t have the necessary role “Bicycles”, the server will send a forbidden error code 403. (Return to 2 from 4.1) -If the edit data is erroneous, the server will resend the Edit form with the appropriate error messages. (Return to 4.1 from 4.2) -If the admin doesn’t have the necessary role “Bicycles”, the server will send a forbidden error code 403. (Return to 2 from 5.1)

	-If the admin does not have the appropriate role “Bicycles”, the server will send a forbidden error code 403. (Return to 2 from 6.1) -If the create data is erroneous, the server will resend the Create form with the appropriate error messages. (Return to 6.1 from 6.2)

Use Case 06: Manage Bicycles

Table 9 Textual Description for the use case "Manage Bicycles"

Use Case name	Manage Bicycles
Actor	SuperAdmin, Admin
Objective	List, Details, Create, Edit, Delete Bicycles
Scenario	1.User clicks on “Bicycles” via the navigation bar at the top. 2.Server receives the Http Get request and sends the Index page for bicycles. 3.User clicks on “Details” next to any bicycle on the list and the server sends the Details page for the requested bicycle. 4.1. User clicks on “Edit” next to any bicycle on the list, the server receives the get request and first checks the role of the admin, then sends the edit form of the requested bicycle. 4.2. User fills the edit form and clicks on confirm, the server verifies the data and updates the existing bicycle in the database. 5.1. User clicks on “Delete” next to any bicycle on the list, the server receives the get request and first checks the role of the admin, then asks for confirmation from the user. 5.2. User clicks on “Confirm” and the server will mark the bicycle as removed. 6.1. User clicks on “Add Bicycle” at the top of the list in the Index page, the server receives the get request and first checks the role of the admin, then sends the bicycle creation form with all the existing bicycle types. 6.2. User fills the creation form with all the bicycle details and selects the appropriate bicycle type then clicks on “Create”. The server will receive the information and make all necessary verifications. 6.3. If all verifications are complete, the new bicycle type is added to the database and the user is sent back to the bicycle types Index page.
Alternative	-If the admin doesn’t have the necessary role “Bicycles”, the server will send a forbidden error code 403. (Return to 2 from 4.1)

	-If the edit data is erroneous, the server will resend the Edit form with the appropriate error messages. (Return to 4.1 from 4.2) -If the admin doesn't have the necessary role "Bicycles", the server will send a forbidden error code 403. (Return to 2 from 5.1) -If the admin does not have the appropriate role "Bicycles", the server will send a forbidden error code 403. (Return to 2 from 6.1) -If the create data is erroneous, the server will resend the Create form with the appropriate error messages. (Return to 6.1 from 6.2)

Use Case 07: Manage Bicycle Contracts

Table 10 Textual Description for the use case "Manage Bicycle Contracts"

Use Case name	Manage Bicycle Contracts
Actor	SuperAdmin, Admin
Objective	List, Details, Deny, Confirm
Scenario	1. User clicks on "Contracts" in the top navigation bar. 2. Server receives the Http Get request and sends the Index page for the bicycle contracts. 3. User clicks on "Details" next to any contract on the list, the server receives the get request. It first verifies the role of the connected admin, then retrieves the list of bicycle contracts from the database. It sends back the "Details" page for the requested contract. 4.1. In the details page, the user clicks on either "Confirm" or "Deny" buttons. 4.2. The server receives the Http Get request and sends a confirmation prompt to allow the user to confirm their action. User clicks on "Confirm" or "Deny" (after filling the denial information). 4.3. The server receives the Http Post request and updates the instance of bicycle contract that is modified in the database. It finally sends the user back to the index page.
Alternative	-If the admin does not have the necessary role "Bicycles", the server sends a forbidden error code 403. (Return to 2 from 3)

Use Case 08: Manage Subscription Plans

Table 11 Textual Description for the use case "Manage Subscription Plans"

Use Case name	Manage Subscription Plans
Actor	SuperAdmin, Admin
Objective	List, Details, Create, Edit, Delete Subscription Plans
Scenario	1.User clicks on “Plans” via the navigation bar at the top. 2.Server receives the Http Get request and sends the Index page for subscription plans. 3.User clicks on “Details” next to any subscription plan on the list and the server sends the Details page for the requested plan. 4.1. User clicks on “Edit” next to any subscription plan on the list, the server receives the get request and first checks the role of the admin, then sends the edit form of the requested subscription plan. 4.2. User fills the edit form and clicks on confirm, the server verifies the data and updates the existing subscription plan in the database. 5.1. User clicks on “Delete” next to any subscription plan on the list, the server receives the get request and first checks the role of the admin, then asks for confirmation from the user. 5.2. User clicks on “Confirm” and the server will mark the subscription plan as removed. 6.1. User clicks on “New Plan” at the top of the list in the Index page, the server receives the get request and first checks the role of the admin, then sends the subscription plan creation form. 6.2. User fills the creation form with all the plan details, selects all the appropriate days of the week and corresponding starting and ending times, then clicks on “Create”. The server will receive the information and make all necessary verifications. 6.3. If all verifications are complete, the new subscription plan is added to the database and the user is sent back to the plans Index page.
Alternative	-If the admin doesn’t have the necessary role “Customers”, the server will send a forbidden error code 403. (Return to 2 from 4.1) -If the edit data is erroneous, the server will resend the Edit form with the appropriate error messages. (Return to 4.1 from 4.2) -If the admin doesn’t have the necessary role “Customers”, the server will send a forbidden error code 403. (Return to 2 from 5.1)

	-If the user is not a SuperAdmin, the server will send a forbidden error code 403. (Return to 2 from 6.1) -If the create data is erroneous, the server will resend the Create form with the appropriate error messages. (Return to 6.1 from 6.2)

Use Case 09: Manage Subscriptions

Table 12 Textual Description for the use case "Manage Subscriptions"

Use Case name	Manage Subscriptions
Actor	SuperAdmin, Admin
Objective	List, Details, Create Subscriptions
Preconditions	Customer Details (Use Case 04, scenario 3)
Scenario	1.User clicks on “Subscriptions” under the details section of the customer. 2.Server receives the http get request and first verifies the role of the admin. The server will then send the Index page for the subscriptions of the specific Customer. 3.User clicks on “Details” next to any subscription on the list, server receives the get request and sends the Details page of the requested subscription. 4.1. User clicks on “Add Subscription” at the top of the list in the Index page. Server receives the get request and verifies the role of the admin. The server will then send the creation form for the Subscription alongside all the existing subscription plans. 4.2. User fills the creation form with all the subscription details, selects the desired plan from the customer, then clicks on “Create”. The server will receive the information and make all necessary verifications. 4.3. If all verifications are complete, the new subscription is added to the database, the customer will have the related reservations with the subscription added, and the user is sent back to the subscriptions Index page.
Alternative	-If the admin doesn’t have the necessary role “Customers”, the server will send a forbidden error code 403. (Return to 1 from 2) -If the admin doesn’t have the necessary role “Customers”, the server will send a forbidden error code 403. (Return to 1 from 4.1)

	-If the create data is erroneous, or there are no available bicycles for the subscription to be fulfilled, the server will resend the Create form with the appropriate error messages. (Return to 4.1 from 4.2)

Use Case 10: Manage Coupon Types

Table 13 Textual Description for the use case "Manage Coupon Types"

Use Case name	Manage Coupon Types
Actor	SuperAdmin, Admin
Objective	List, Details, Create, Delete Coupon Types
Scenario	1.User clicks on “Coupon Types” via the navigation bar at the top. 2.Server receives the Http Get request and sends the Index page for coupon types. 3.User clicks on “Details” next to any coupon type on the list and the server sends the Details page for the requested type. 4.1. User clicks on “Delete” next to any coupon type on the list, the server receives the get request and first checks the role of the admin, then asks for confirmation from the user. 4.2. User clicks on “Confirm” and the server will mark the coupon type as removed. 5.1. User clicks on “New Coupon Type” at the top of the list in the Index page, the server receives the get request and first checks the role of the admin, then sends the coupon type creation form. 5.2. User fills the creation form with all the type details then clicks on “Create”. The server will receive the information and make all necessary verifications. 5.3. If all verifications are complete, the new coupon type is added to the database and the user is sent back to the coupon types Index page.
Alternative	-If the user is not a SuperAdmin, the server will send a forbidden error code 403. (Return to 2 from 4.1) -If the user is not a SuperAdmin, the server will send a forbidden error code 403. (Return to 2 from 5.1) -If the create data is erroneous, the server will resend the Create form with the appropriate error messages. (Return to 5.1 from 5.3)

Use Case 11: Manage Coupons

Table 14 Textual Description for the use case "Manage Coupons"

Use Case name	Manage Coupons
Actor	SuperAdmin, Admin
Objective	List, Details of Coupons
Preconditions	Customer Details (Use Case 04, scenario 3)
Scenario	1.User clicks on “Coupons” under the details section of the customer. 2.Server receives the http get request and first verifies the role of the admin. The server will then send the Index page for the coupons of the specific Customer. 3.User clicks on “Details” next to any coupon on the list, server receives the get request and sends the Details page of the requested coupon.

Use Case 12: Manage Reservations

Table 15 Textual Description for the use case "Manage Reservations"

Use Case name	Manage Reservations
Actor	SuperAdmin, Admin
Objective	List, Details, Edit, Confirm, Cancel
Scenario	1.User clicks on “Reservations” via the navigation bar at the top. 2.Server receives the Http Get request and first verifies the role of the admin, then sends the Index page for reservations. 3.User clicks on “Details” next to any reservation on the list, the server will initially verify the role of the admin, then refreshes the validity of the reservations, then finally sends the Details page for the requested reservation. 4.1. User clicks on “Edit” next to any reservation on the list, the server receives the get request and first checks the role of the admin, then sends the edit form. 4.2. User fills out the edit form and clicks on the “Confirm” button to send the http post request. 4.3. The server receives the form and verifies the new information regarding the reservation, then will apply the changes in the database.

	5.1. User clicks on “Confirm” next to any reservation in the list, the server receives the get request and first checks the role of the admin, then sends the confirmation page. 5.2. User clicks on the “Confirm” button and the server will mark the reservation as confirmed and create a new transaction equivalent to the reservation. 6.1. User clicks on “Cancel” next to any reservation in the list, the server receives the get request and first checks the role of the admin, then sends the confirmation page. 6.2. User clicks on the “Confirm” button and server will mark the reservation as canceled.
Alternative	-If the admin doesn’t have the necessary role “Reservations”, the server will send a forbidden error code 403. (Return to 1 from 2) -If the admin doesn’t have the necessary role “Reservations”, the server will send a forbidden error code 403. (Return to 1 from 3) -If the admin doesn’t have the necessary role “Reservations”, the server will send a forbidden error code 403. (Return to 1 from 4.1) -Server will send back the edit form if any information is incorrect (Return to 4.1 from 4.3) -If the admin doesn’t have the necessary role “Reservations”, the server will send a forbidden error code 403. (Return to 2 from 5.1) -Server will send back the edit form if any information is incorrect (Return to 4.1 from 4.3) -If the admin doesn’t have the necessary role “Reservations”, the server will send a forbidden error code 403. (Return to 2 from 6.1)

Use Case 13: Manage Transactions

Table 16 Textual Description for the use case "Manage Transactions"

Use Case name	Manage Transactions
Actor	SuperAdmin, Admin
Objective	List, Details, Create, Edit, Delete, Return, CashIn
Scenario	1.User clicks on “Transactions” via the navigation bar at the top. 2.Server receives the Http Get request and first verifies the role of the admin, then sends the Index page for transactions.

	3.User clicks on “Details” next to any transaction on the list, the server will initially verify the role of the admin, then sends the Details page for the requested transaction. 4.1. User click on “New Rental” at the top of the list, the server receives the get request and checks the role of the admin, then sends the creation form. 4.2. User fills out the creation form and clicks on the button “Create” to send an Http Post request. 4.3. The server receives and verifies the data then creates a new transaction in the database. At the same moment, the control flag in the real-time database is set to allow the GPS tracker to begin tracking the bicycle. 5.1. User clicks on “Edit” next to any transaction on the list, the server receives the get request and first checks the role of the admin, then sends the edit form. 5.2. User fills out the edit form and clicks on the “Confirm” button to send the http post request. 5.3. The server receives the form and verifies the new information regarding the transaction, then will apply the changes in the database. 6.1. User clicks on “Delete” next to any transaction in the list, the server receives the get request and first checks the role of the admin, then sends the confirmation page. 6.2. User clicks on the “Delete” button to confirm deleting the transaction. The server will mark the transaction as deleted and update all the related data. 7.1. User clicks on “Return Bicycle” at the top of the transactions’ Index page, the server receives the get request and first checks the role of the admin, then requests the user to provide the username of the customer. 7.2. User enters the username and clicks on “Find Transaction”. The server will search for the only active paid transaction of the customer and send the confirmation form. 7.3. User fills the return date and time of the transaction and clicks on “Return”. The server receives the post request, performs the necessary verifications regarding the return date and time. 7.4. The server marks the transaction as completed and resets the control flag in the real-time database in order to stop the GPS tracking. The user is sent back to the transactions’ Index page. 8.1. User clicks on “Cash In” at the top of the transactions’ Index page, the server receives the get request and first checks the role of the admin, then requests the user to provide the username of the customer.

	8.2. User enters the username and clicks on “Find Transaction”. The server will search for the only unpaid active transaction of the customer and send the confirmation form. 8.3. User fills the paid amount and clicks on “Confirm”. The server receives the post request, performs the necessary verifications regarding the paid amount and sends the user back to the index page.
Alternative	-If the admin doesn’t have the necessary role “Transactions”, the server will send a forbidden error code 403. (Return to 1 from 2) -If the admin doesn’t have the necessary role “Transactions”, the server will send a forbidden error code 403. (Return to 1 from 3) -If the admin doesn’t have the necessary role “Transactions”, the server will send a forbidden error code 403. (Return to 1 from 4.1) -Server will send back the creation form if any information is incorrect or violates the logic of transactions. (Return to 4.1 from 4.3) -If the admin doesn’t have the necessary role “Transactions”, the server will send a forbidden error code 403. (Return to 2 from 5.1) -Server will send back the edit form if any information is incorrect (Return to 5.1 from 5.3) -If the admin doesn’t have the necessary role “Transactions”, the server will send a forbidden error code 403. (Return to 2 from 6.1) -If the admin doesn’t have the necessary role “Transactions”, the server will send a forbidden error code 403. (Return to 2 from 7.1) -If the username does not exist or the customer did not rent any bicycle, the server will send back the search form with the appropriate error message. (Return to 7.1 from 7.2) -If the return date is invalid, the server will send back the return form with the appropriate error message. (Return to 7.2 from 7.3) -If the admin doesn’t have the necessary role “Transactions”, the server will send a forbidden error code 403. (Return to 2 from 8.1) -If the username does not exist, the customer did not rent any bicycle or the customer has paid their transaction, the server will send back the search form with the appropriate error message. (Return to 8.1 from 8.2) -If the paid amount is invalid, the server will send back the cash-in form with the appropriate error message. (Return to 8.2 from 8.3)

Use Case 14: Manage personal User Information

Table 17 Textual Description for the use case "Manage Personal User Information"

Use Case name	Manage personal User Information
Actor	Customer
Objective	Details, Edit, Change Password
Scenario	1.User clicks on “Details” in the top navigation bar, sending an Http Get request to the server asking for the Details page with the user’s id. 2.Server receives the request and verifies if the connected customer is requesting their own details, then sends back the Details page. 3.1. User clicks on “Edit” at the bottom of the Details page in order to send an Http Get request to the server asking for the Edit page. 3.2. Server receives the request and verifies if the connected customer is requesting to edit their own information, then sends back the Edit page. 3.3. The form comes pre-filled with the current details of the customer. The user modifies the fields they wish to modify and click on the “Edit” button at the bottom of the form. Server receives the Http Post request. 3.4. Server verifies the new data then updates the existing instance of the customer in the database. The user is sent back to the Details page. 4.1. User clicks on “Change Password” at the bottom of the Details page in order to send an Http Get request to the server asking for the Change Password page. 4.2. Server receives the request and verifies if the connected customer is requesting to change their own password, then sends back the requested page. 4.3. The user fills the fields with the new password and clicks on the “Confirm” button at the bottom of the form. Server receives the Http Post request. 4.4. Server hashes the new password then updates the existing password of the customer in the database. The user is sent back to the Details page.
Alternative	-If the requested customer’s id and the connected user id do not match, the server will send a forbidden error code 403. (Return to 1 from 2) -If the requested customer’s id and the connected user id do not match, the server will send a forbidden error code 403. (Return to 1 from 3.2) -If data is erroneous the server will send back the editing form with the appropriate error messages. (Return to 3.1 from 3.4)

	-If the requested customer's id and the connected user id do not match, the server will send a forbidden error code 403. (Return to 1 from 4.2)

Use Case 15: Browse Bicycle Types

Table 18 Textual Description for the use case "Browse Bicycle Types"

Use Case name	Browse Bicycle Types
Actor	Customer
Objective	List, Details
Scenario	1.User clicks on "Our Bicycles" in the top navigation bar, sending an Http Get Request to the server. 2.The server receives the request and sends back the Index page for bicycle types. 3.1. User clicks on "Details" under any bicycle type in the list, sending an Http Get request to the server. 3.2. The server receives the request and sends back the Details page for the requested bicycle type.

Use Case 16: Manage personal Bicycle Contracts

Table 19 Textual Description for the use case "Manage Personal Bicycle Contracts"

Use Case name	Manage personal Bicycle Contracts
Actor	Customer
Objective	List, Details, Create, Cancel
Scenario	1. User clicks on "Contracts" in the top navigation bar. 2. Server receives the Http Get request and sends the Index page for the bicycle contracts with only the contracts related to the connected user. 3. User clicks on "Details" next to any contract on the list, the server receives the get request. It retrieves the bicycle contract from the database. It sends back the "Details" page for the requested contract. 4.1. In the details page, the user clicks on either "Cancel" button. 4.2. The server receives the Http Get request and sends a confirmation prompt to allow the user to confirm their action. User clicks on "Cancel".

	4.3. The server receives the Http Post request and updates the instance of bicycle contract that is modified in the database. It finally sends the user back to the index page.

Use Case 17: Manage personal Reservations

Table 20 Textual Description for the use case "Manage Personal Reservations"

Use Case name	Manage personal Reservations
Actor	Customer
Objective	List, Details, Create, Cancel
Scenario	1.User clicks on “Reservations” via the navigation bar at the top. 2.Server receives the Http Get request then sends the Index page for the reservations of the current customer. 3.User clicks on “Details” next to any reservation on the list, the server will initially verify that the connected user and the reservation’s customer are matching, then refreshes the validity of all the existing reservations, then finally sends the Details page for the requested reservation. 4.1. User clicks on “Reserve a bike” via the navigation bar at the top. 4.2. Server receives the Http Get request then sends the Create page for reservations. 4.3. User fills out the creation form of the reservation and clicks on confirm sending the Http Post request to the server. 4.4. The server receives the request with the information. After verifying the data, the server will create a new reservation and apply all the necessary changes to the related entities (bicycle, customer …) in the database. The user is sent back to the Index page for reservations. 5.1. User clicks on “Cancel” next to any reservation in the list, the server receives the get request and first checks that the connected user and the reservation’s customer are matching, then sends the confirmation page to cancel the reservation. 5.2. User clicks on the “Confirm” button and the server will mark the reservation as canceled.
Alternative	-If the connected user and the reservation’s customer are not matching, the server will send a forbidden error code 403. (Return to 1 from 3) -Server will send back the creation form if any information is incorrect with the appropriate error messages (Return to 4.1 from 4.4)

	-If the connected user and the reservation's customer are matching, the server will send a forbidden error code 403. (Return to 1 from 5.1)

Use Case 18: Manage personal Transactions

Table 21 Textual Description for the use case "Manage Personal Transactions"

Use Case name	Manage personal Transactions
Actor	Customer
Objective	List, Details, Create
Preconditions	The customer must be present at the store terminal in order to create a transaction.
Scenario	1.User clicks on "Transactions" via the navigation bar at the top. 2.Server receives the Http Get request then sends the Index page for the transactions of the current customer. 3.User clicks on "Details" next to any transaction on the list, the server will initially verify that the connected user and the transaction's customer are matching, then sends the Details page for the requested transaction. 4.1. User clicks on "Rent a Bicycle" on the main screen of the store terminal. 4.2. Server receives the Http Get request then sends the login page. 4.3. User fills out their username and password and clicks on login sending an Http Post request to the server. 4.4. The server receives the request with the login credentials. After verifying the data, the server will send the Create page for transactions. 4.5. User fills out the information on the creation form and clicks on "Create" sending an Http Post request. 4.6. The server verifies the data of the creation form and creates a new transaction. User is sent back to the main screen of the store terminal.
Alternative	-If the login credentials are wrong the server sends back the login form with the appropriate error messages. (Return to 4.1 from 4.3) -Server will send back the creation form if any information is incorrect with the appropriate error messages (Return to 4.5 from 4.6)

Use Case 19: Browse Subscription Plans

Table 22 Textual Description for the use case "Browse Subscription Plans"

Use Case name	Browse Subscription Plans
Actor	Customer
Objective	List, Details
Scenario	1.User clicks on “Plans” via the navigation bar at the top of the page, sending an Http Get Request to the server. 2.The server receives the request and sends back the Index page for subscription plans. 3.1. User clicks on “Details” under any subscription plan in the list, sending an Http Get request to the server. 3.2. The server receives the request and sends back the Details page for the requested subscription plan.

Use Case 20: Browse Coupon Types

Table 23 Textual Description for the use case "Browse Coupon Types"

Use Case name	Browse Coupon Types
Actor	Customer
Objective	List, Details, Redeem
Scenario	1.User clicks on “Rewards” via the navigation bar at the top of the page, sending an Http Get Request to the server. 2.The server receives the request and sends back the Index page for coupon types. 3.1. User clicks on “Details” under any coupon type in the list, sending an Http Get request to the server. 3.2. The server receives the request and sends back the Details page for the requested coupon type. 4.1. User clicks on “Redeem” in the Details page of a particular coupon type in order to turn customer points into a coupon. 4.2. The server receives an Http Get request and sends the confirmation page. 4.3. User clicks on the “Confirm” button. The server will verify the customer points of the concerned customer, then updates all the relevant data in the database. The user is sent back to the Index page of coupon types.

Alternative	-If the customer does not have enough customer points to redeem the chosen coupon type, the server will send back the confirmation page with the appropriate error message. (Return to 4.2 from 4.3)

Use Case 21: Manage personal Subscriptions

Table 24 Textual Description for the use case "Manage Personal Subscriptions"

Use Case name	Manage personal Subscriptions
Actor	Customer
Objective	List, Details, Delete
Scenario	1.User clicks on “Subscriptions” in the Details page of the customer. 2.Server receives the Http Get request, verifies if the connected user and the customer match, then sends the Index page for the subscriptions of the current customer. 3.User clicks on “Details” next to any subscription on the list, the server will verify that the connected user and the subscription’s customer are matching, then sends the Details page for the requested subscription. 4.1. User clicks on “Delete” in the details page of the subscription. 4.2. Server receives the Http Get request, verifies that the connected user and the subscription’s customer are matching, then sends the confirmation page. 4.3. User clicks on “Confirm” sending an Http Post request to the server. 4.4. The server receives the request and deletes the subscription and all the related reservations.
Alternative	-If the connected user and the reservation’s customer are matching, the server will send a forbidden error code 403. (Return to 1 from 2) -If the connected user and the reservation’s customer are matching, the server will send a forbidden error code 403. (Return to 1 from 3) -If the connected user and the reservation’s customer are matching, the server will send a forbidden error code 403. (Return to 4.1 from 4.2)

Use Case 22: Manage personal Coupons

Table 25 Textual Description for the use case "Manage Personal Coupons"

Use Case name	Manage personal Coupons
Actor	Customer
Objective	List, Details
Scenario	1.User clicks on “My Coupons” in the Details page of the customer. 2.Server receives the Http Get request, verifies if the connected user and the customer match, then sends the Index page for the coupons of the current customer. 3.User clicks on “Details” next to any coupon on the list, the server will verify that the connected user and the subscription’s customer are matching, then sends the Details page for the requested coupon.
Alternative	-If the connected user and the reservation’s customer are matching, the server will send a forbidden error code 403. (Return to 1 from 2) -If the connected user and the reservation’s customer are matching, the server will send a forbidden error code 403. (Return to 1 from 3)

3.1.3. Sequence Diagrams

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction that takes place in a collaboration that either realizes a use case or an operation. They also illustrate high-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams) [36].

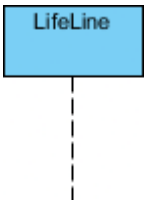
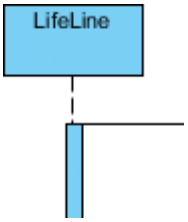
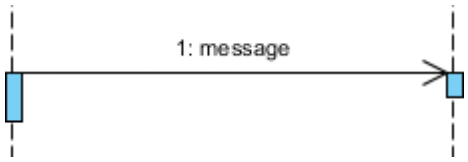
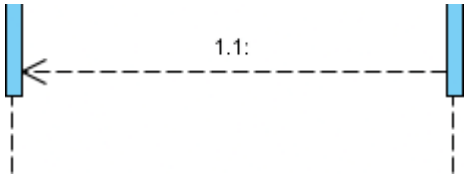
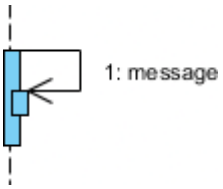
Sequence Diagrams show the interactions between elements of the project (horizontal axis) through time (vertical axis).

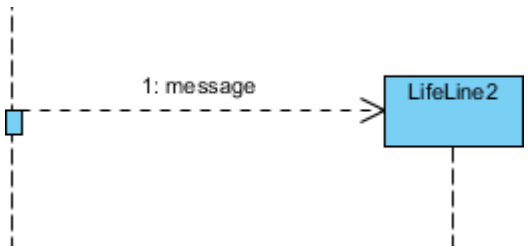
Building a Sequence Diagram requires the understanding of two main building block types: The Elements (or the notations) and the Sequence Fragments.

3.1.3.1 Elements of Sequence Diagrams

They are the blocks that represent the objects that exist within the project and the messages that occur between them.

Table 26 Elements of Sequence Diagrams

Element	Description	Symbol
Actor	An actor represents roles played by human users, external hardware, or other subjects.	
Lifeline	A lifeline represents an individual participant in the Interaction.	
Activations	A thin rectangle on a lifeline) represents the period during which an element is performing an operation.	
Call Message	A message defines a particular communication between Lifelines of an Interaction. Call message is a kind of message that represents an invocation of operation of target lifeline	
Return Message	Return message is a kind of message that represents the pass of information back to the caller of a corresponding former message.	
Self Message	Self message is a kind of message that represents the invocation of a message of the same lifeline.	
Recursive Message	Recursive message is a kind of message that represents the invocation of a message of the same lifeline. Its target points to an activation on top of the activation where the message was invoked from.	

Create Message	Create message is a kind of message that represents the instantiation of (target) lifeline.	

3.1.3.2. Sequence Fragments

A sequence fragment encloses a portion of the interactions within a sequence diagram. It is represented as a box with an operator (in the top left corner) that indicates the type of fragment.

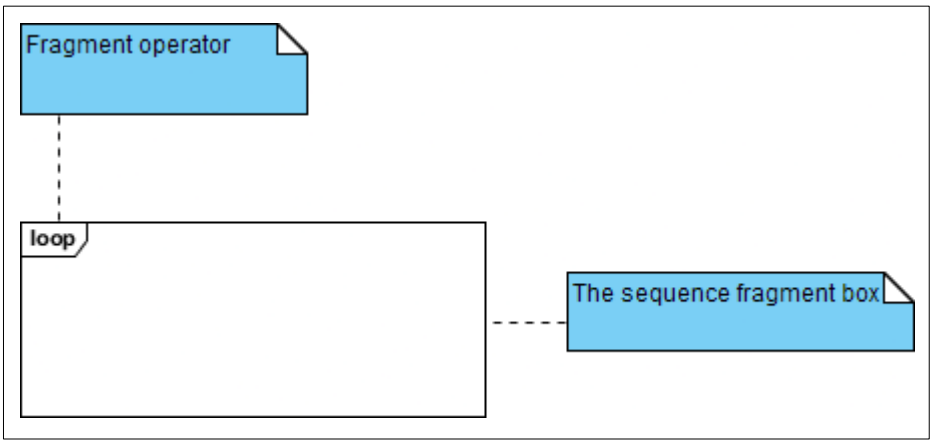


Figure 26 Sequence Fragment

Table 27 Fragment Type Operators

Operator	Fragment Type
alt	Alternative multiple fragments: only the one whose condition is true will execute.
par	Parallel: each fragment is run in parallel.
loop	Loop: the fragment may execute multiple times, and the guard indicates the basis of iteration.
ref	Reference: refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.

3.1.3.3 Actions used in Sequence Diagrams

We have previously stated that controllers control the flow of the application, they call actions depending on the request sent by the client. Each model has a controller with a set of actions where each action performs a specific task using the set of available functions in the service layer.

In order to create a readable and maintainable code, all controllers share a similar set of actions that allow them to perform the same basic operations but they also have some actions that are unique to them which allows them to perform specific operations on their respective models.

It is also important to note that most of these actions could exist twice within the same controller **(Respecting the function signatures in programing)** where one action is responsible for answering Http Get requests and the second is responsible for handling Http Post requests.

Shared Actions are the functions that we find within all controllers:

- ***Index:*** Recovers the entire instances of a model from the database.
- ***Details:*** Recovers a single entity of a model from the database based on their ID.
- ***Create:*** Creates a new instance of the model and saves it in the database.
- ***Edit:*** Edits of an existing instance of the model and updates it in the database.
- ***Delete:*** Removes an existing instance of the model from the database. In the cases where the database relations do not allow the complete removal of entities, we use a technique called the Soft-Delete, where we use a field in the model where we mark the entity to be deleted or not.

Unique Actions are the functions that belong to unique controllers:

Bicycle Contracts Controller has three extra actions:

Confirm: SuperAdmin or Admin confirms a bicycle contract with a customer, adding a new bicycle to our inventory.

Cancel: SuperAdmin, Admin or Customer that requested a contract cancels the request.

Deny: SuperAdmin or Admin denies the request and provides a justification to why the rejection occurred.

Coupon Controller has two extra actions:

Redeem Coupon: Customer redeems their customer points into a coupon.

Confirm: This action is a confirmation barrier which requests a second confirmation from the user in case of a mistake in the first request.

Customer Controller has two extra actions:

Redeem Coupon: This action is the first step in Redeeming a coupon for Customers, where at the end of the function, the program is redirected to Coupon Controller.

Change Password: Customer changes their password.

Reservation Controller has two extra actions:

Confirm: SuperAdmin or Admin confirms a reservation creating a transaction.

Cancel: SuperAdmin, the Admin or the Customer that created the reservation cancels a reservation.

Subscription Plan Controller has two extra actions:

Enable: SuperAdmin enables a disabled plan.

Disable: SuperAdmin disables an enabled plan.

Transaction Controller has four extra actions:

Get Transaction: SuperAdmin or Admin searches for an active transaction using the username of a customer.

Mock Login: Verifies the credentials of a customer at the store terminal.

Cash In: SuperAdmin or Admin receives the paid amount from the customer.

Return Bicycle: SuperAdmin or Admin terminates a transaction.

We will use the previous functions in the sequence diagrams to better visualize the flow of the application.

Sequence Diagram 01: Login

In order to benefit from the application's functionalities that are offered for both registered customers and admins, users should log in.

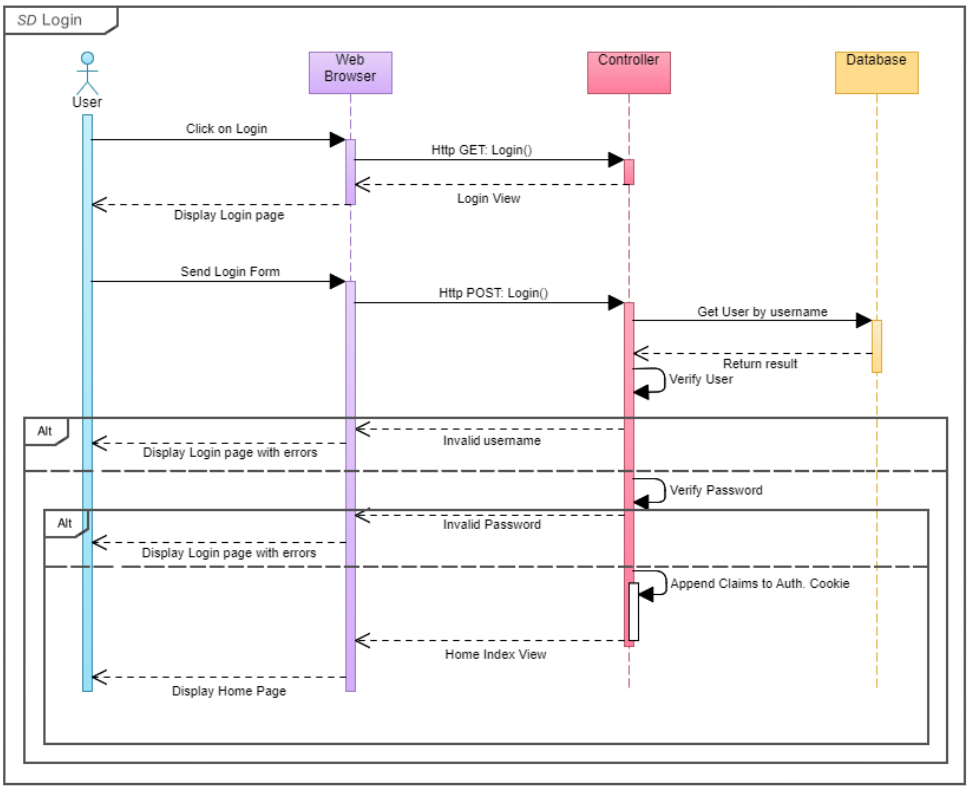


Figure 27 Login Sequence Diagram

Sequence Diagram 02: Logout

When the user is done using the application and to avoid unwanted interactions with their account, they can logout from the platform.

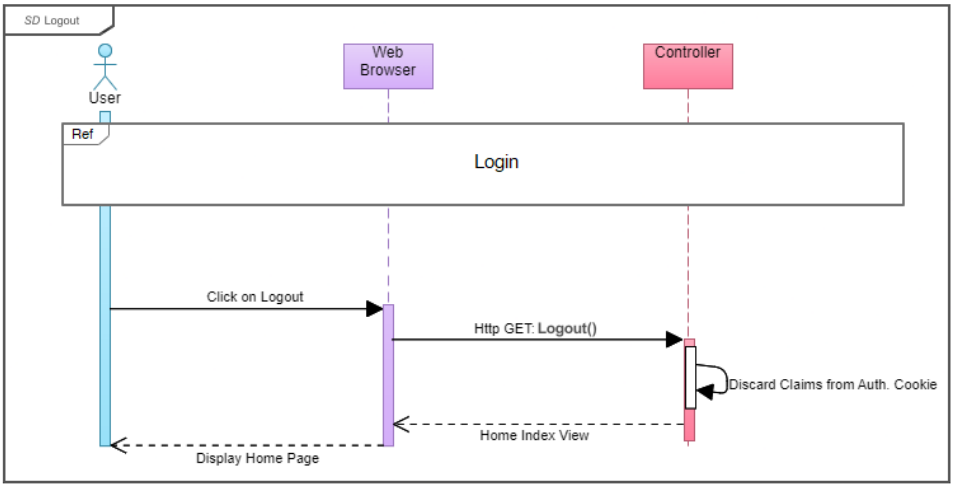


Figure 28 Logout Sequence Diagram

Sequence Diagram 03: Mock Login

Similar to login, this function allows the verification of user credentials. The only difference is that we do not login the user and do not append any information to the authentication cookie.

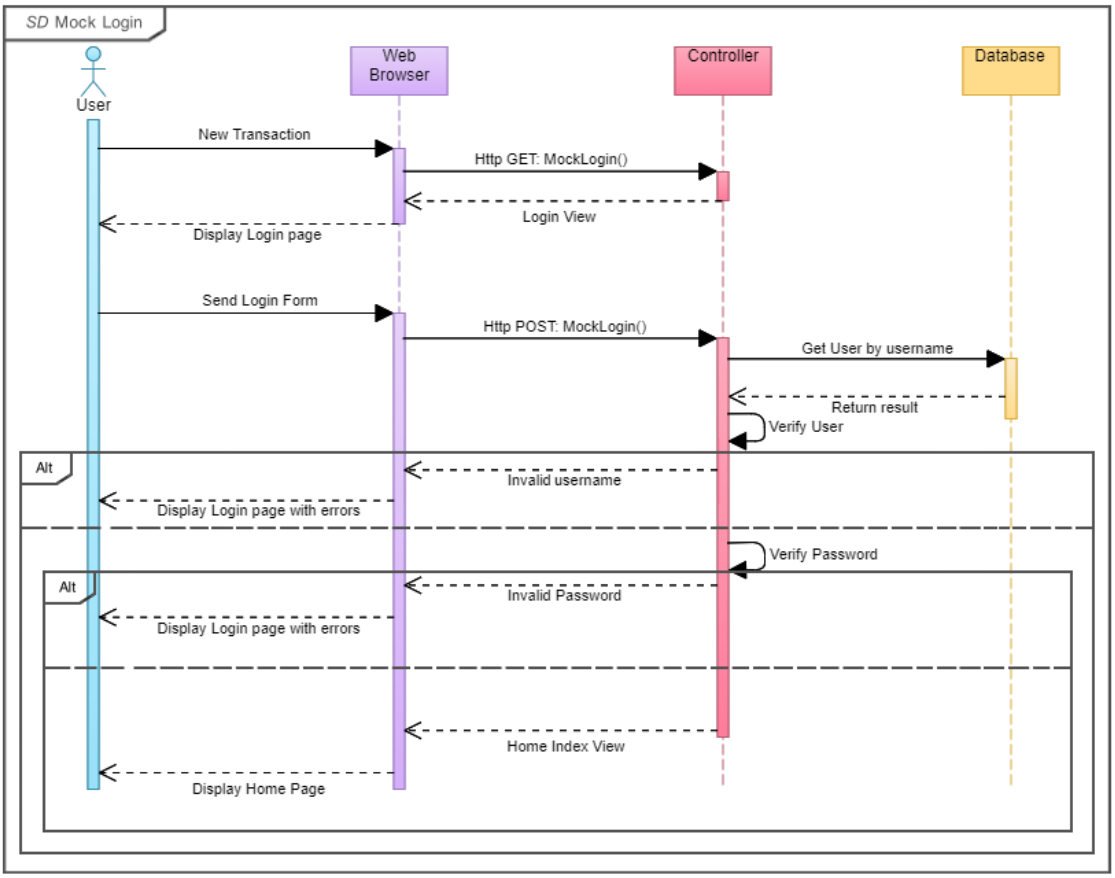


Figure 29 Mock Login Sequence Diagram

Sequence Diagram 04: List all items

After the user is logged in, they can list the items of any entity type if they have the authorizations to do so.

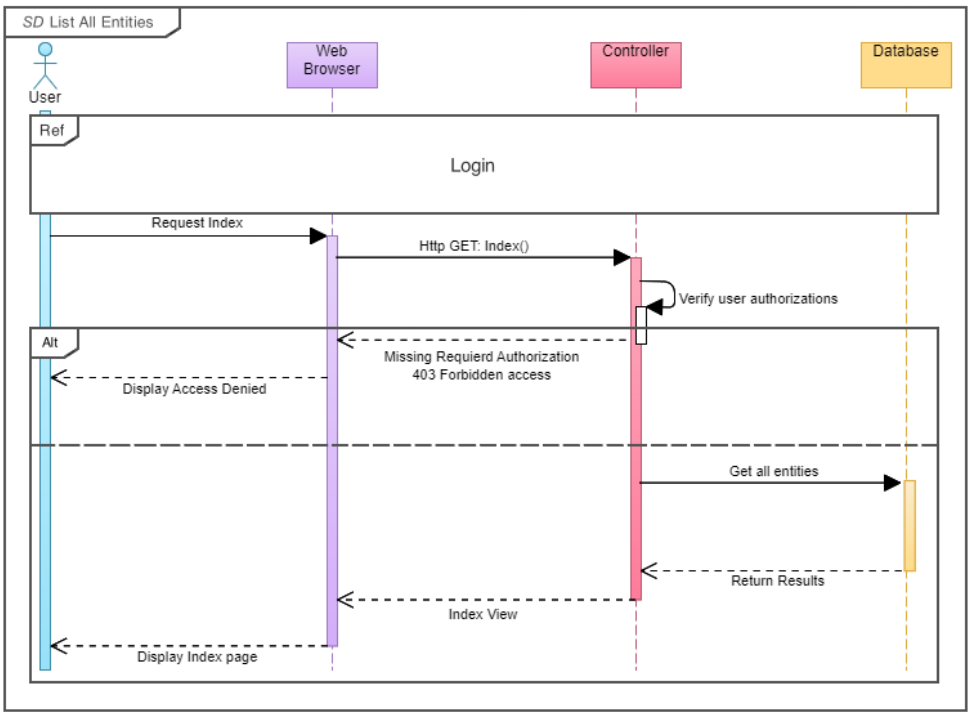


Figure 30 List All Entities Sequence Diagram

Sequence Diagram 05: View Details

After the user is logged in, they can view the details of a particular entity if they have the authorizations to do so.

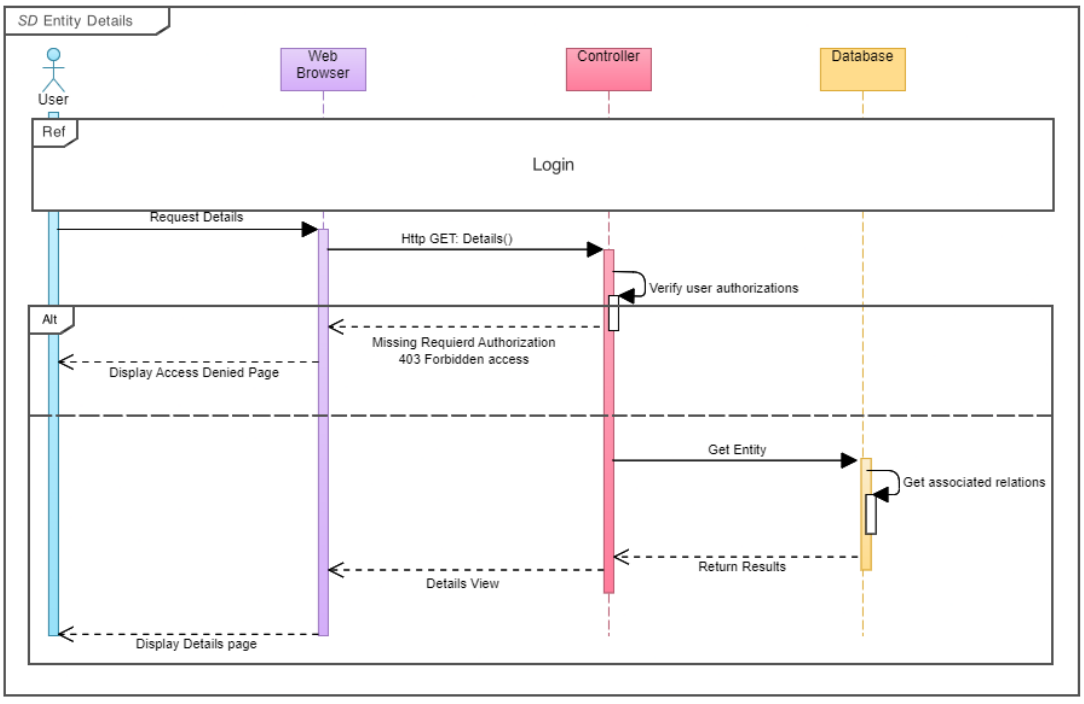


Figure 31 Entity Details Sequence Diagram

Sequence Diagram 06: Edit item

After the user is logged in, they can edit the information of a particular entity if they have the authorizations to do so.

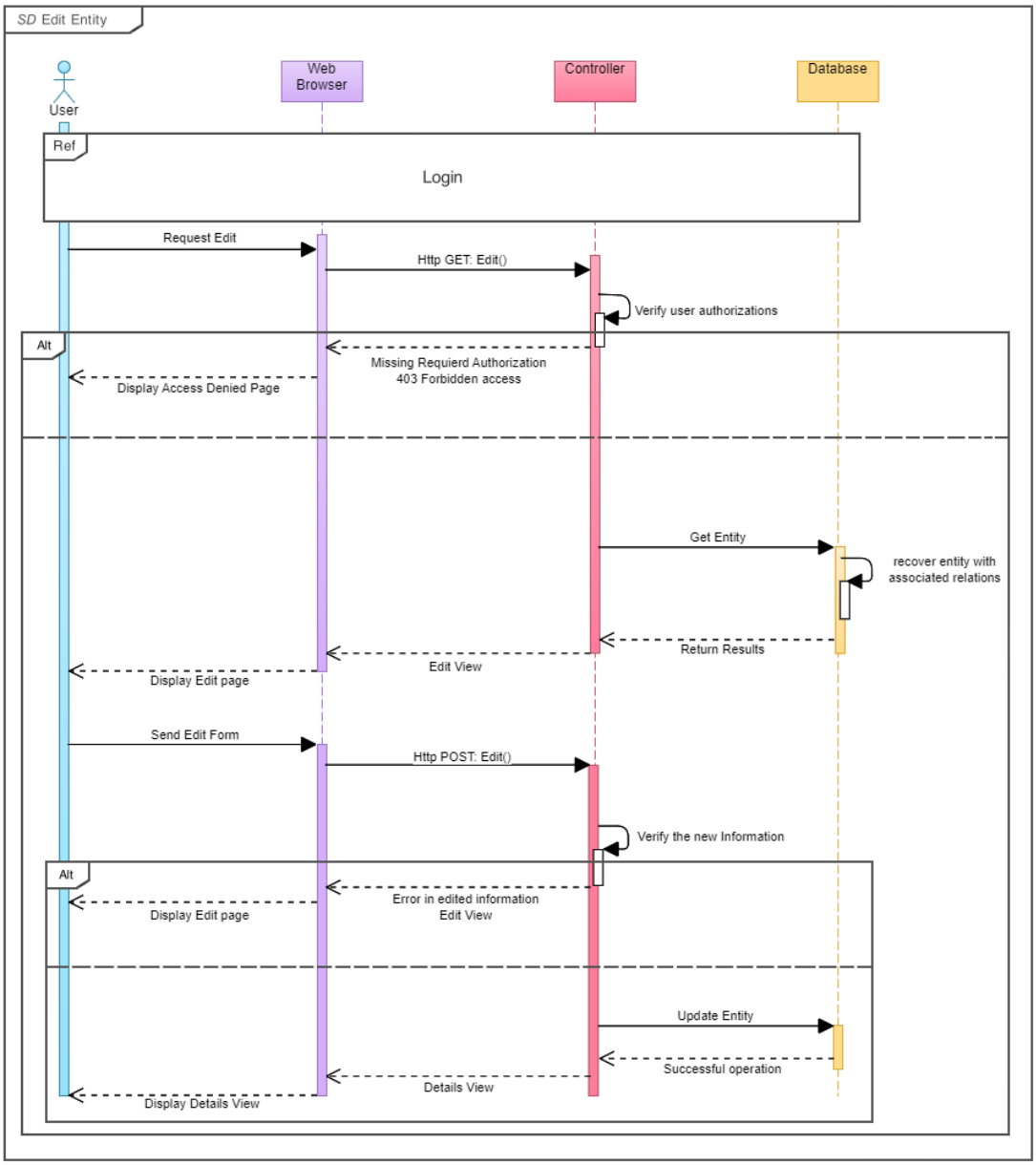


Figure 32 Edit Entity Sequence Diagram

Sequence Diagram 07: Delete/Remove item

After the user is logged in, they can remove a particular entity if they have the authorizations to do so.

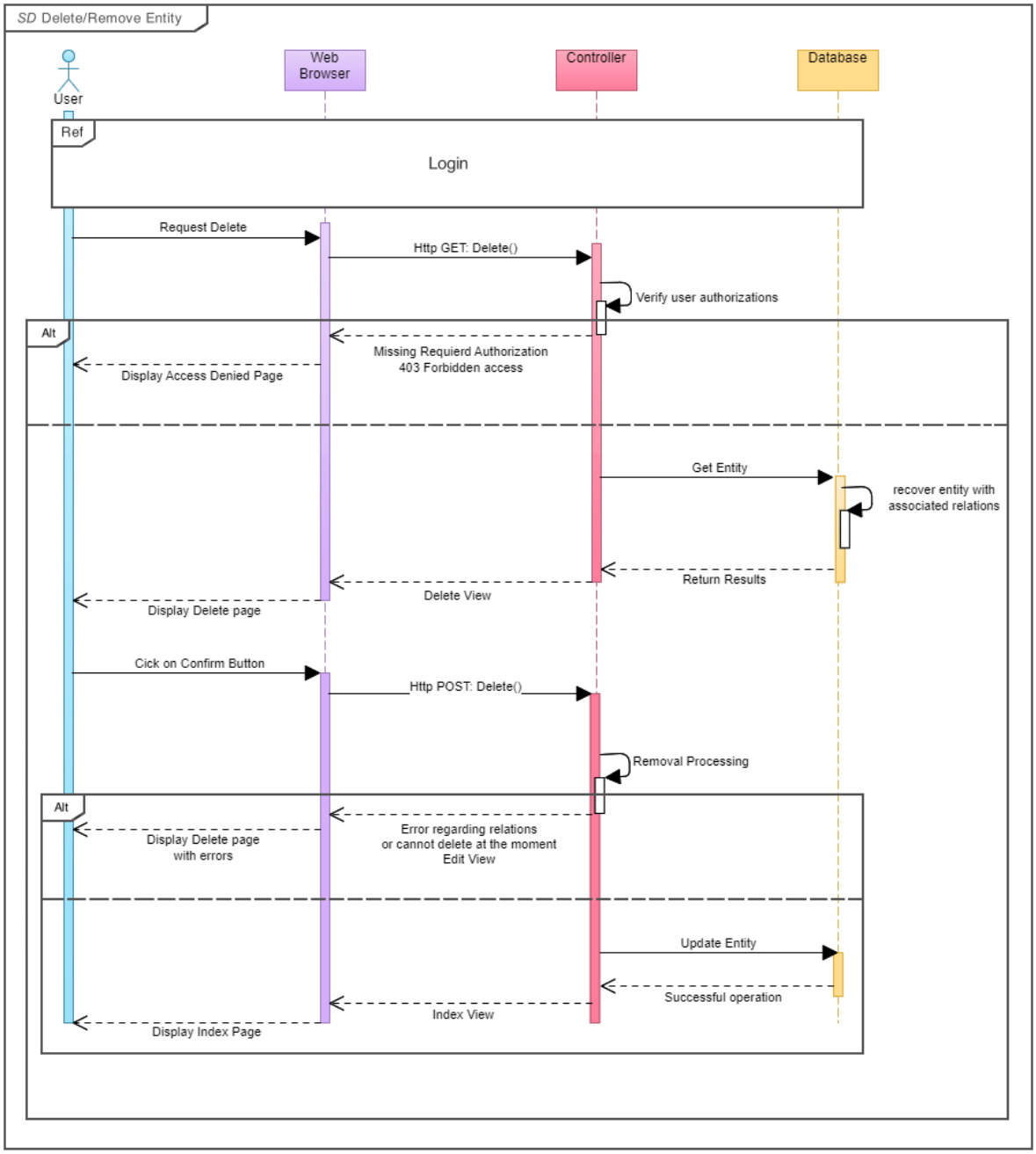


Figure 33 Delete/Remove Entity Sequence Diagram

Sequence Diagram 08: Confirm/Cancel Reservation

Once a customer has created a reservation, it can either be confirmed or canceled. It is important to note that a customer cannot confirm their own reservation, they can only cancel it.

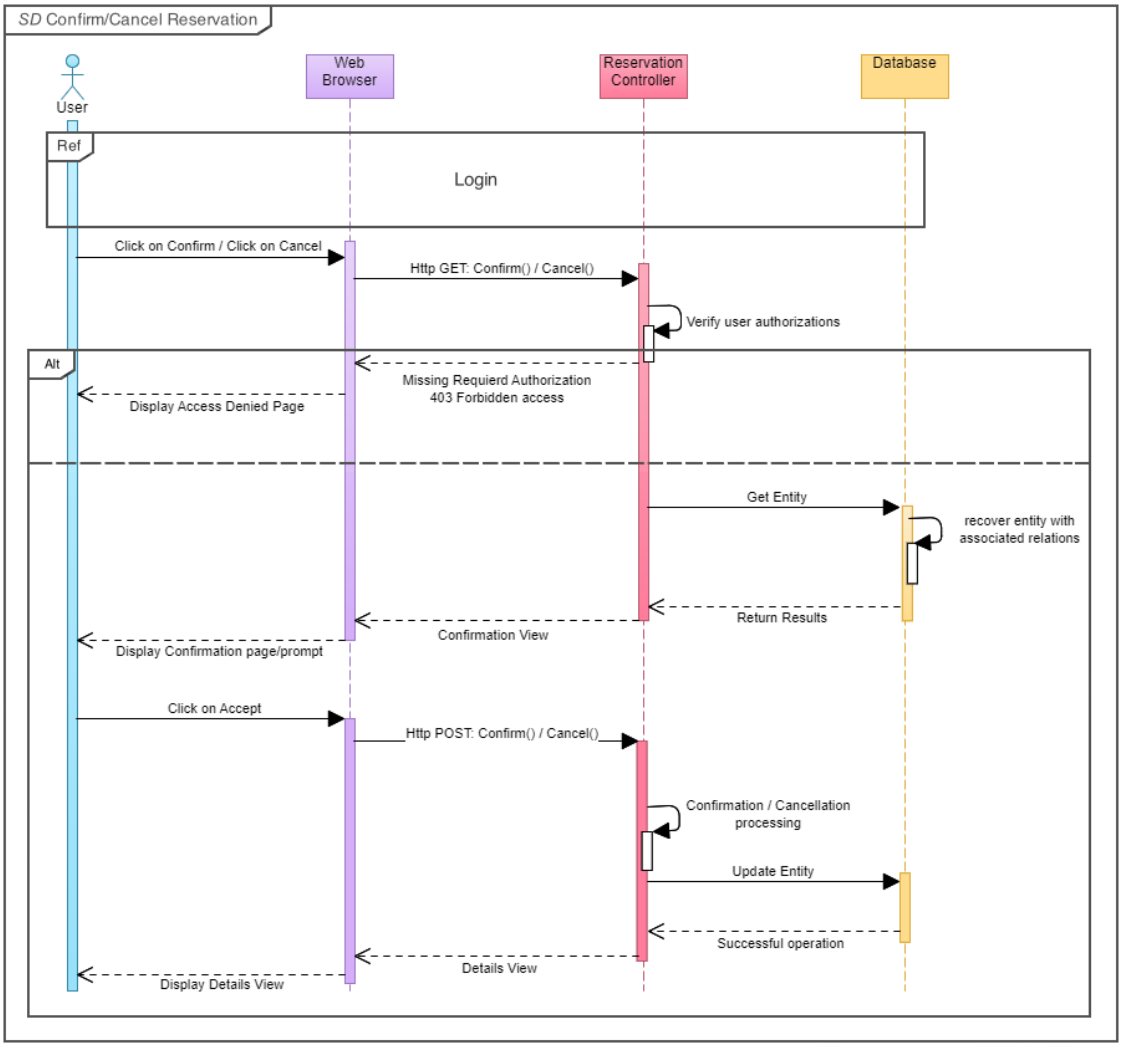


Figure 34 Confirm/Cancel Reservation Sequence Diagram

Sequence Diagram 09: Return Bicycle

Once a transaction has been created, it is possible to end it via the Return Bicycle process. Only Admins and SuperAdmins can perform this operation.

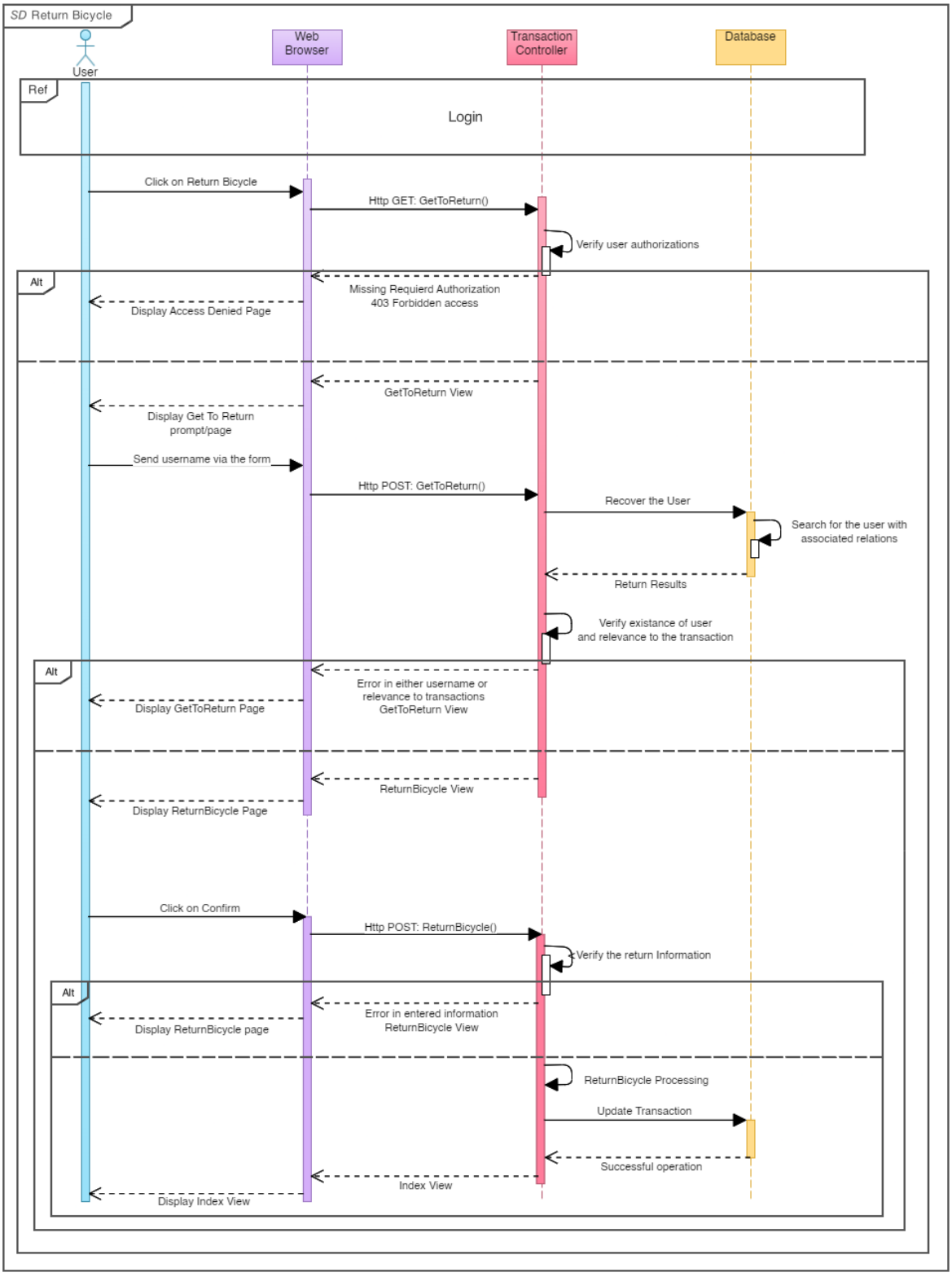


Figure 35 Return Bicycle Sequence Diagram

Sequence Diagram 10: Cash In

Once a transaction has been created, it is necessary for the customer to pay in order to be able to take out their bicycle. This is done via the Cash In process. Only Admins and SuperAdmins can perform this operation.

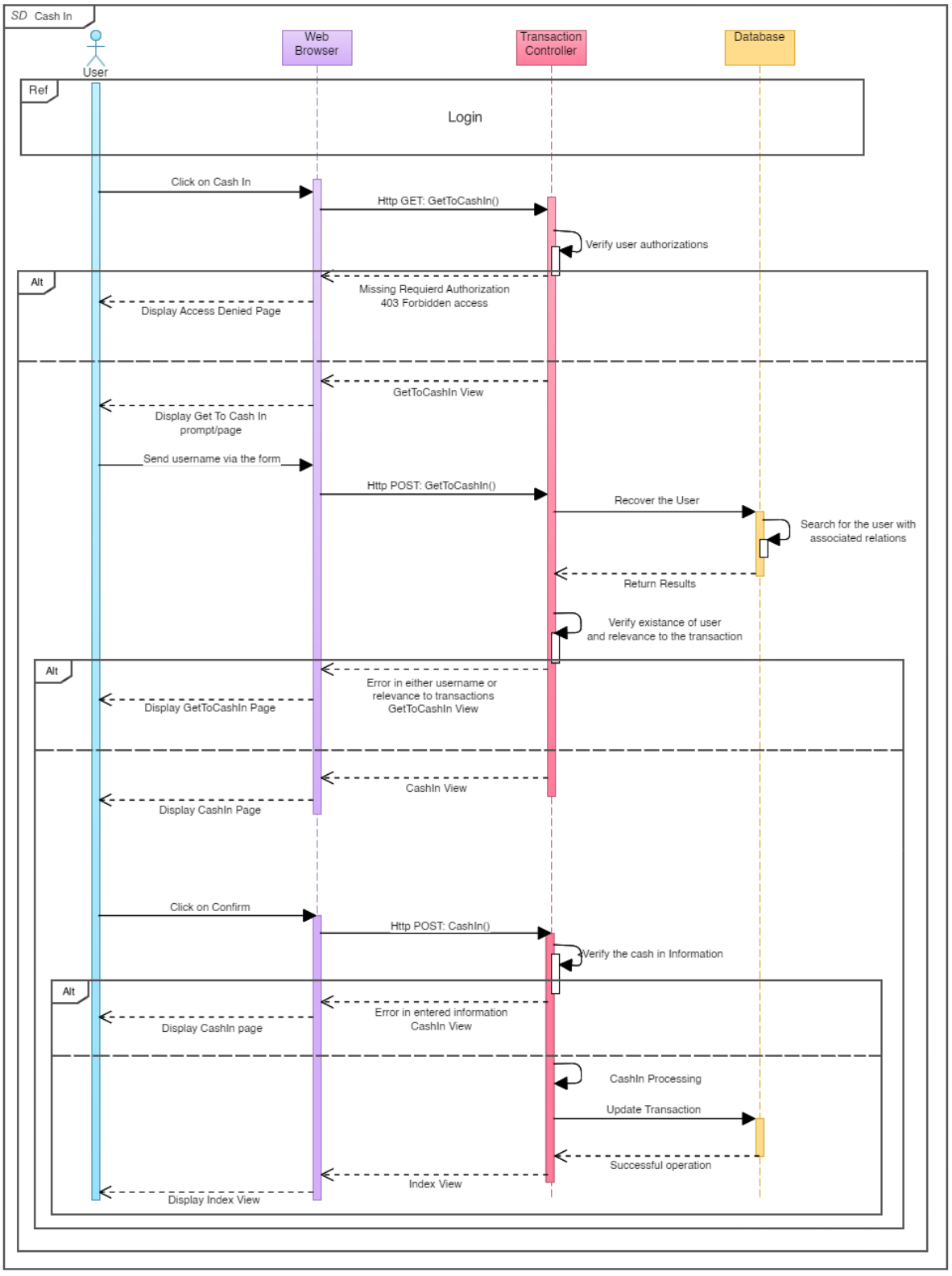


Figure 36 Cash In Sequence Diagram

Sequence Diagram 11: Redeem Coupon

A customer can redeem their customer points into coupons, allowing them to have reductions in their future transactions.

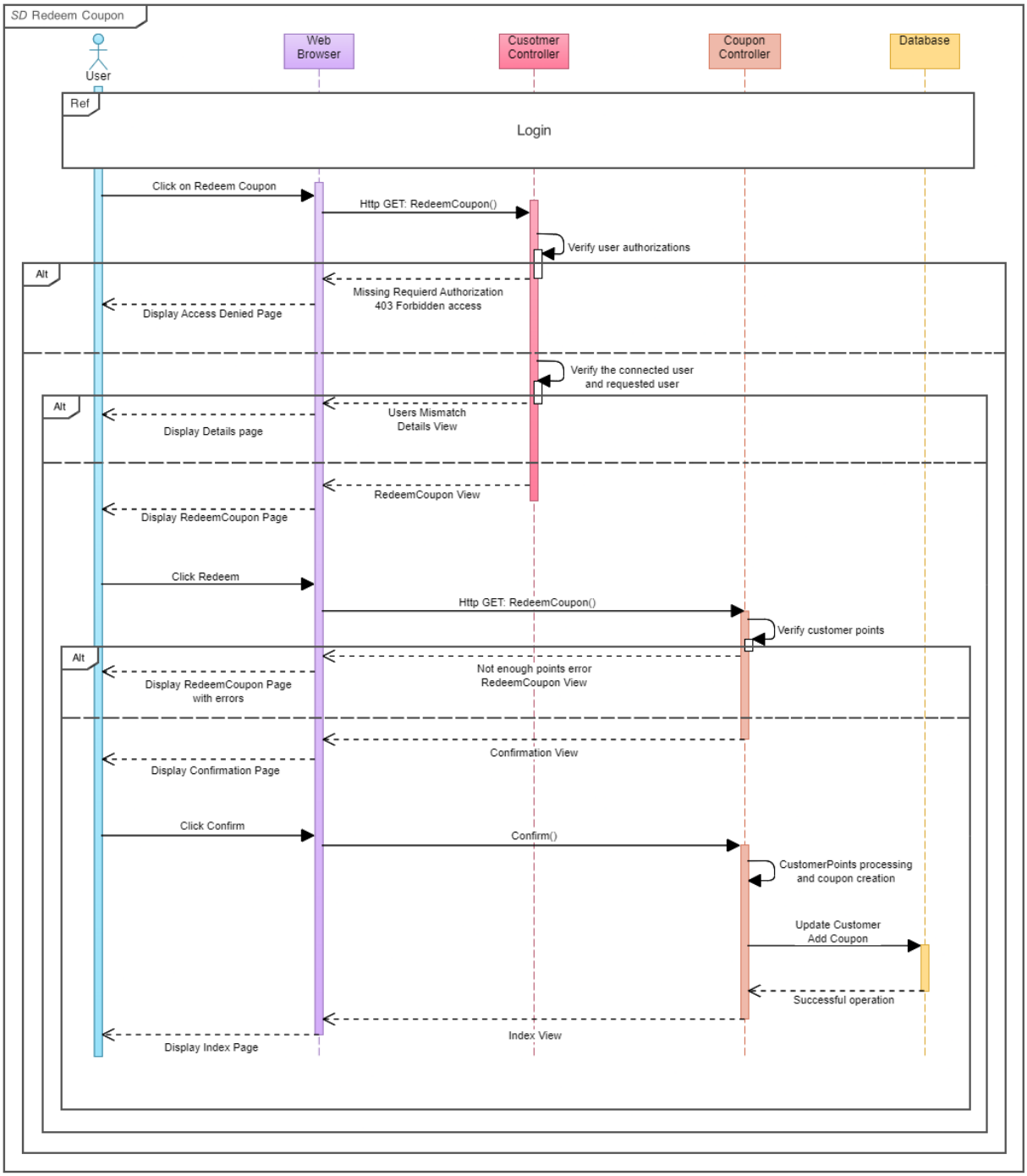


Figure 37 Redeem Coupon Sequence Diagram

Sequence Diagram 12: Enable/Disable Subscription Plan

To avoid completely deleting the plan from the database, a SuperAdmin has the ability to Enable or Disable a Subscription Plan.

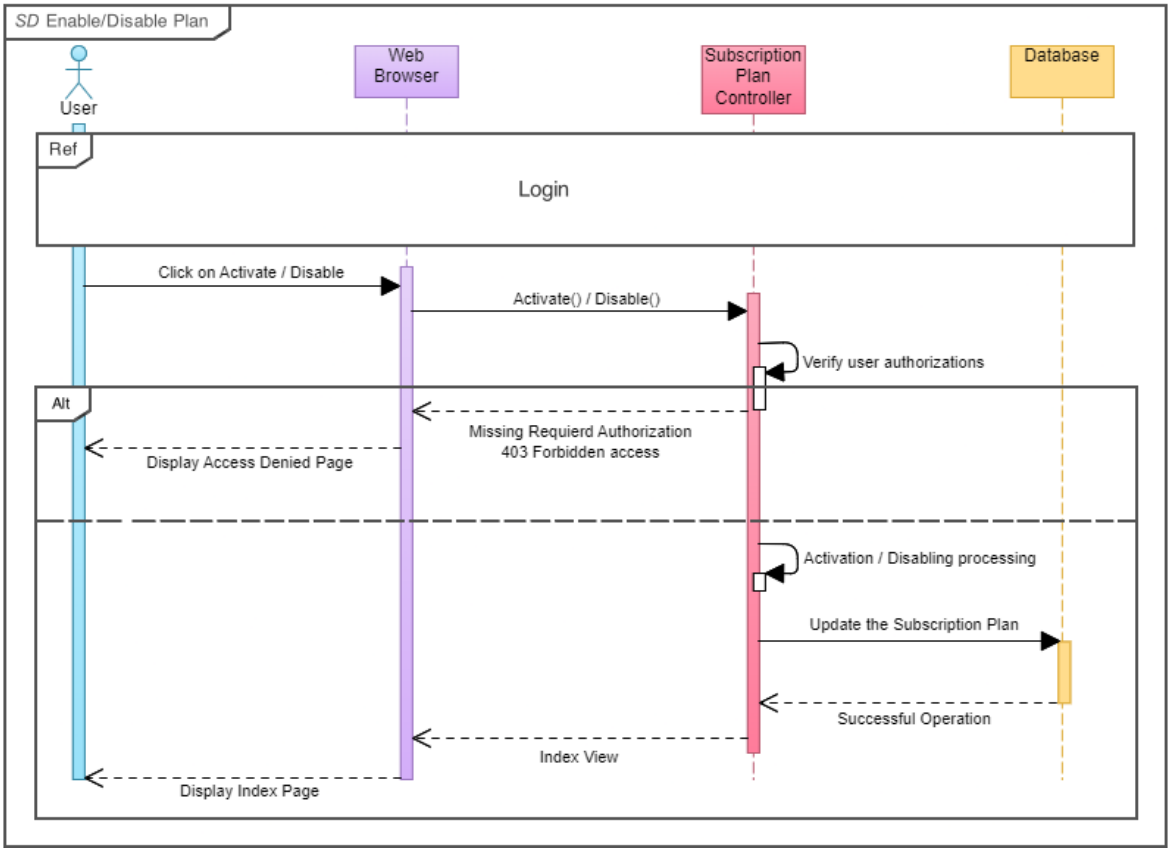


Figure 38 Enable/Disable Plan Sequence Diagram

Sequence Diagram 13: Confirm/Deny Bicycle Contract

When a customer creates a Bicycle Contract, a SuperAdmin or an Admin with the appropriate role would review the contract. When a decision is made, the user should be able to Confirm or Deny the contract.

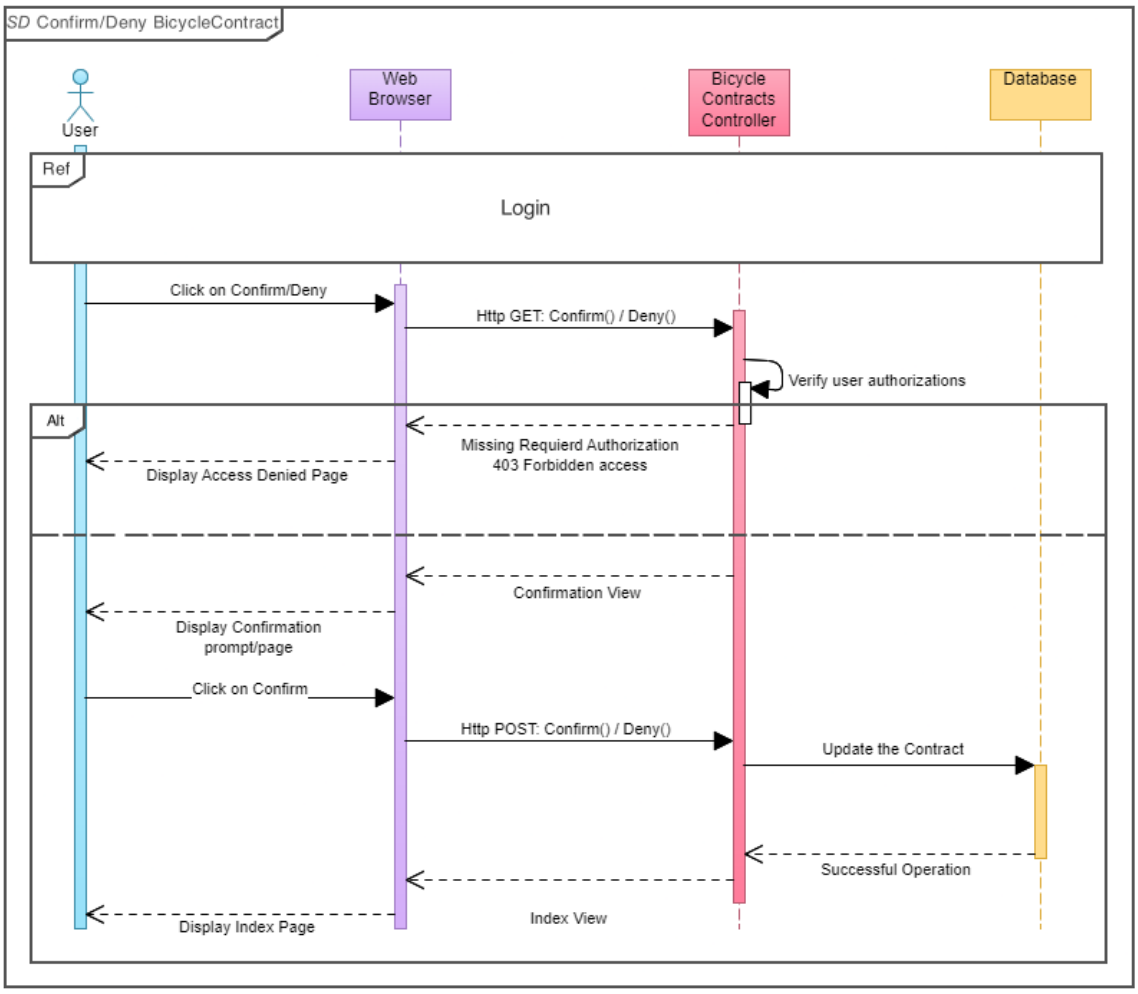


Figure 39 Confirm/Deny BicycleContract Sequence Diagram

3.2. Relational Database Design

3.2.1. Approach Types

Using Entity Framework Core ORM, we are able to use two different methods in order to design and implement the Relational Database.

Database First: This is how we used to develop our applications before entity framework i.e., we used to design a database first and then code our application to talk to this database. You use this approach when you already have a database designed and want to build your application around it[37].

Code First: The entity framework will create the database with tables as per the entity classes defined in the application. So, code is created first and this code is responsible for creating the database using commands available in the EF Core tools package for the creation of the database[38]. The entity classes talked about are the classes in the Model layer. There exist two implicit ways to configure and control the attributes of the tables created in the database from the code, Data Annotations and Configuration Methods.

- *Data Annotations:* They are simple tags that are used directly in the model class that set a certain constraint on the data field. They are few and have limited control. The main reason to use them is their simplicity and quick application.
- *Configuration Methods:* They are methods that Entity Framework apply at the creation of the database. They are more precise and have a wider range of control. They are more complex to use compared to data annotations.

Seeding the database with initial entities is an example that shows the difference between the two configuration techniques, since it is impossible to seed the database with data annotations, but it is convenient to use configuration methods.

In our web application we used the Code First technique, since it saves us from making any design mistakes. It is also very convenient because all the queries are carefully optimized by the ORM.

It is not recommended to implicitly modify the database after using the code first method because it can lead to miscommunication errors between the ORM and the Database. In the case of updates or maintenance, all modifications should be applied to the code only.

3.2.2. Migrations

In real world projects, data models change as features get implemented: new entities or properties are added and removed, and database schemas need to be changed accordingly to be kept in sync with the application. The migrations feature in EF Core provides a way to incrementally update the database schema to keep it in sync with the application's data model while preserving existing data in the database. When a data model change is introduced, the developer uses EF Core tools to add a corresponding migration describing the updates necessary to keep the database schema in sync. EF Core compares the current model against a snapshot of the old model to determine the differences, and generates migration source files[38].

3.2.3. Entity Relationship Diagram

Entity Relationship Diagrams are used in software engineering in order to design and simplify the debugging process of databases. In our representation, each rectangle represents a table in the database. The rows of each rectangle represent the attributes (or columns) of that table. An attribute is part of the primary key if there is a key icon next to it.

The following table illustrates the different relation types and their symbols.

Table 28 Relations Symbols

Relation Name	**Symbol**
One To One	
One To Many	
Many To Many	

To better visualize the relationship between Actors and the entities that exist in our web application, we will present the following ERD (Entity Relationship Diagram) which represents our relational database layout.

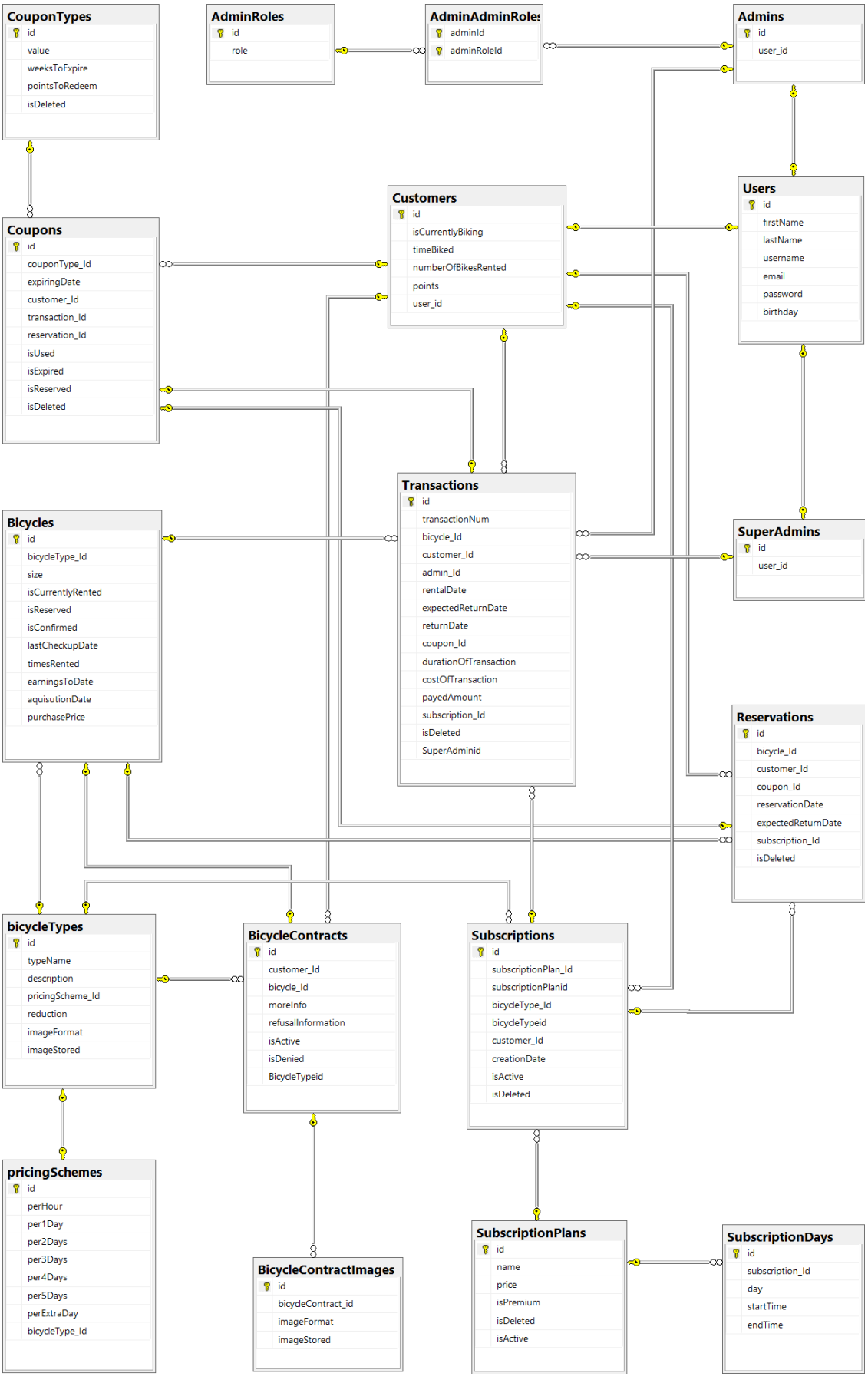


Figure 40 Entity Relationship Diagram

4. GPS Tracker Design

We have previously presented the pin diagrams of each element we are using to build our GPS Tracker. In this section we will suggest our implementation for the Microcontroller based GPS Tracker.

4.1. Circuit Diagram

The following figure shows the circuit for our GPS tracker

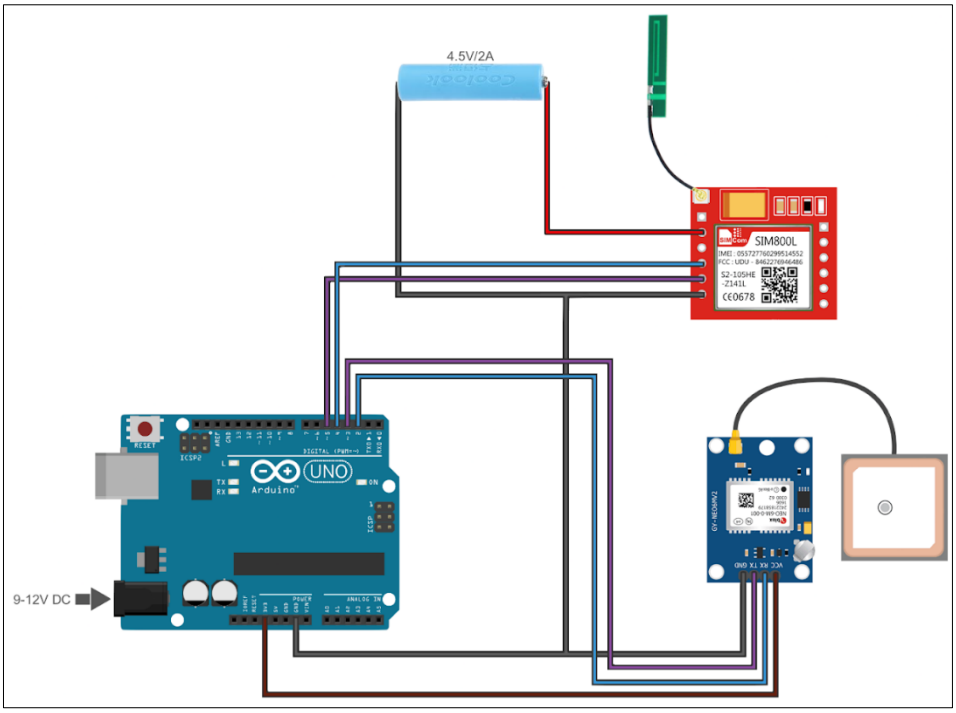


Figure 41 Circuit Diagram

The Microcontroller is powered by a 9-12V DC power supply. Any source with a higher voltage will cause irreversible damage to the power circuit of the board.

We use the 3.3V supply of the Arduino to power the NEO6MV2 module. Configuring the chip to use the UART communication protocol, we connect RX and TX to the pins 2 and 3 of the microcontroller respectively.

We use an external 4.5V/2A power source in order to power the SIM800L. The reason being that the GPRS module requires a peak of 2A current at the moment of transmission of the data. The Arduino 5V output pin does not provide such current. We configure the chip to use the UART communication protocol, we connect RX and TX to the pins 4 and 5 of the microcontroller respectively.

All GND pins are commonly connected to the same ground pin of the Microcontroller.

4.2. Software Design

For the previous circuit to properly function, we have to correctly configure every component through the software that the microcontroller will execute.

It is necessary to note the following:

- A Bicycle ID is hardcoded into the program of the microcontroller. The only way to modify this ID is to update the software of the microcontroller.
- The GPS Tracker communicates with the Firebase Realtime Database only. It is not linked to the web application in any other way. More appropriate design choices exist but because of the hardware limitations, this is the solution we chose to implement.

4.2.1 NEO 6MV2

This module initializes all of its components (Real Time Clock, GPS, Speed, Altitude… etc.) using the data received from the GPS satellites network. Nevertheless, we are required to initialize the Baud rate and the serial pins used by the microcontroller.

The format of the output returned by the previous function is as follows:

the date format: “dd/mm/yyyy”

the time format: “hh:mm:ss”

the location format: “latitude longitude”

4.2.2. SIM800L

The initialization process of this module is different from the previous one, since we are supposed to ensure the connection to the mobile service provider. But prior to that, we initialize the UART communication with the appropriate baud rate and the proper serial pins. In order to communicate with the chip, we use the previously presented AT Commands. We first initialize the module before connecting it to the mobile network. We can then either read from or write to the realtime database.

4.2.3. Setup Function

The Setup function is a section of the code that is executed only once by the microcontroller. As its name suggests, it is used to mainly setup and initialize the different modules and peripherals that are used in our system.

We present the following flowchart which shows the flow of the program within the Setup Function of the microcontroller.

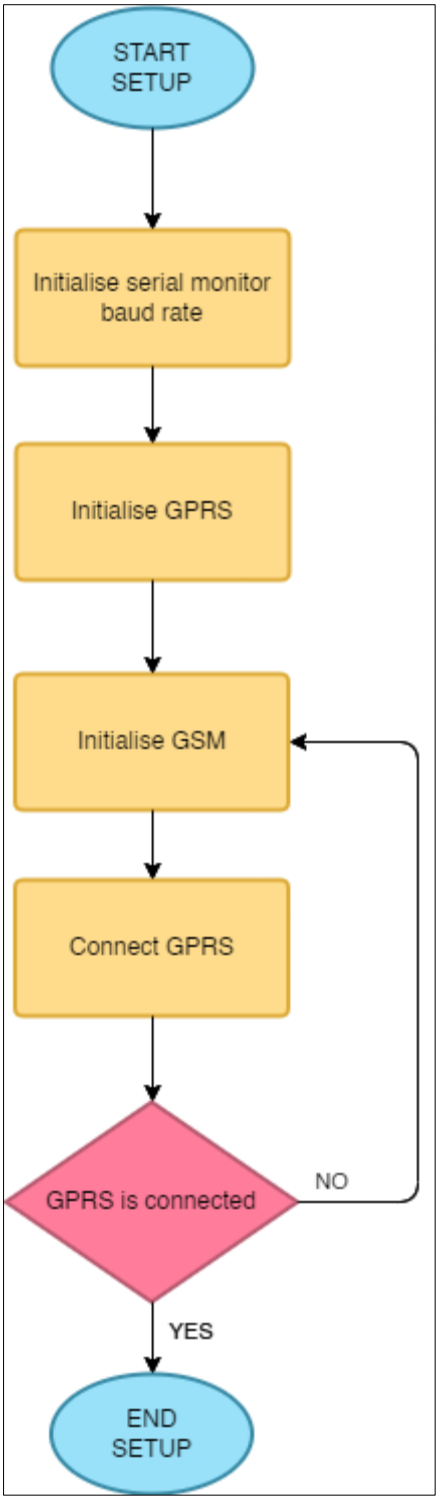


Figure 42 Atduino Setup Function Flowchart

4.2.4. Main Loop

In embedded systems, the main processing units (ex. microcontroller) execute an infinite loop called the Main Loop which contains the main software of the system. They are used to communicate in a real-time manner with the previously configured peripherals (in the Setup Function) and other parts of the system.

We present the following flowchart which shows the flow of the main loop within the microcontroller.

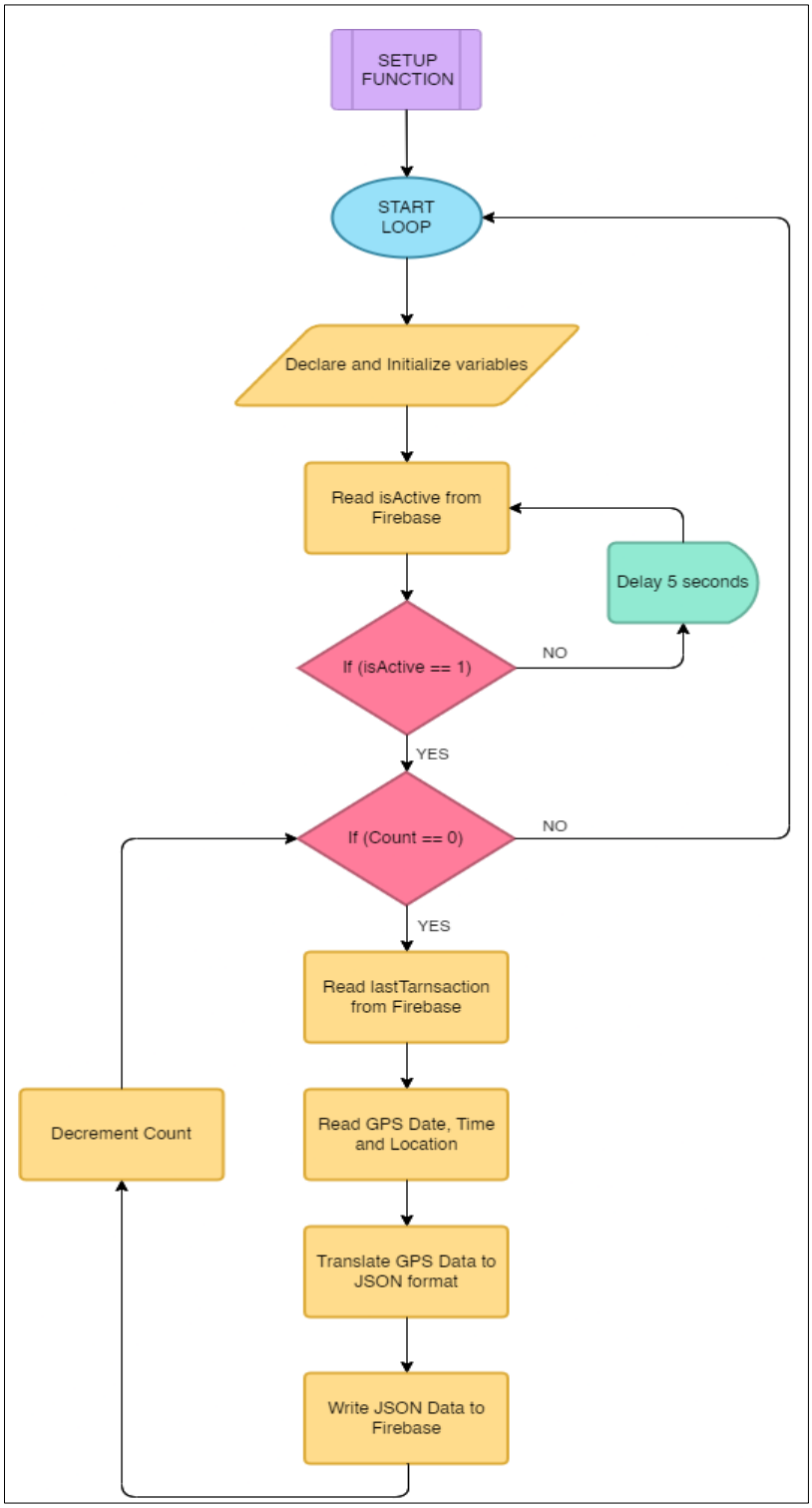


Figure 43 Arduino Loop Function Flowchart

4.3. Firebase real-time database

Even if NoSQL databases allow data storage without any limitations regarding organization. It is necessary to have a well-organized system that groups relevant data types together for optimal querying of the database.

We will present the way we organized our real-time data in a data-diagram, which contains the following elements

Table 29 NoSQL Database Element Symbols and Descriptions

Element	Symbol	Description
Collection		A collection holds a number of documents and other collections.
Document		Holds a set of Key-Value Elements and Collections.
Key-Value		A pair of Key and its value, they hold the actual data we are storing.
Contains		An arrow that shows that a collection or a document contains a certain set of elements

We decided to organize the data as shown in the following diagram:

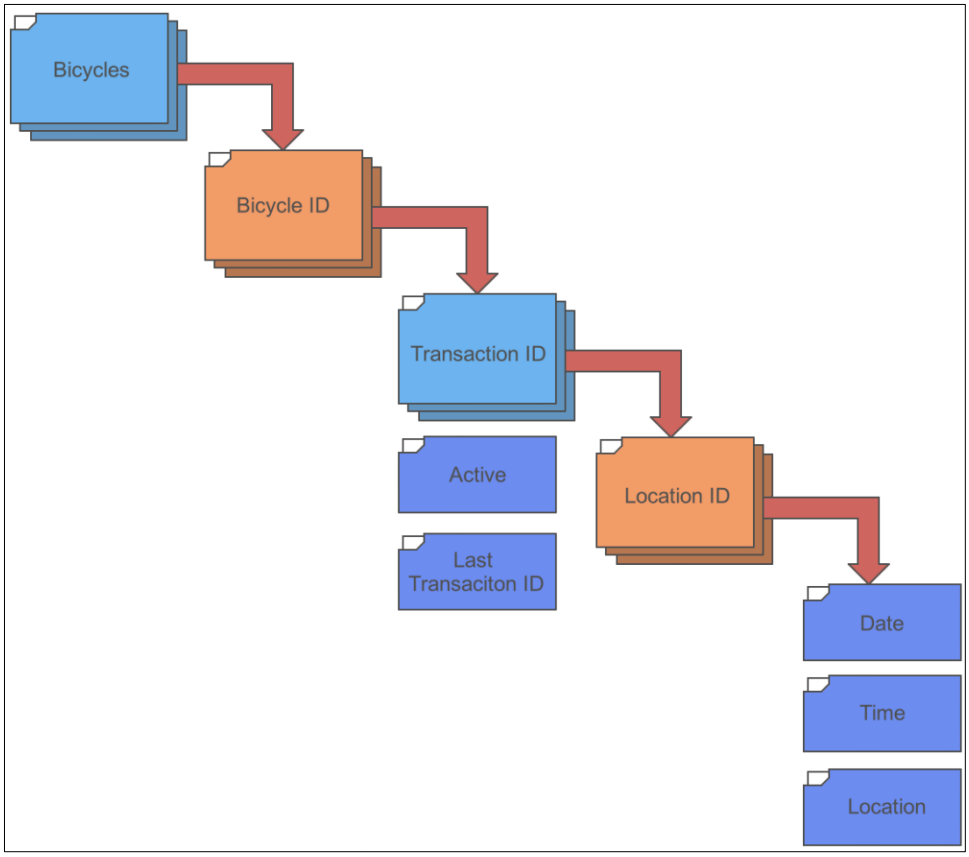


Figure 44 Firebase Realtime Database Data Organization Diagram

We can view how the data is actually stored within the database in the following figure:

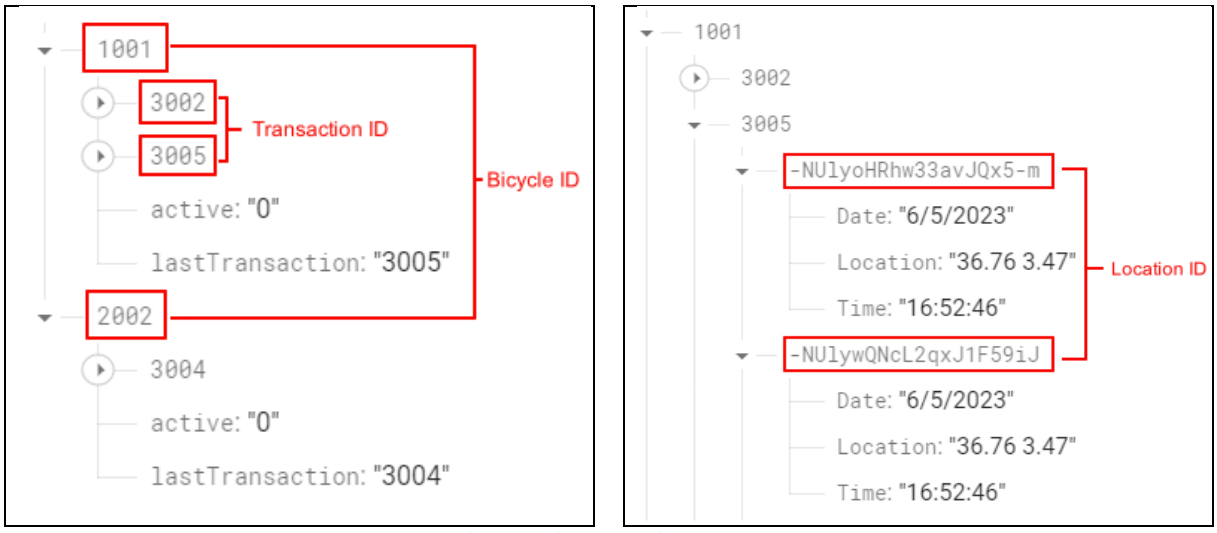


Figure 45 Firebase Realtime Database Data Organization

The date, time and GPS location are encapsulated within a location document. These documents are grouped under a single Transaction ID. At this level we can find the two control fields which are managed by the web application and read by the microcontroller. All of these are then stacked under a single Bicycle ID which identifies the microcontroller.

This design choice greatly reduces the complexity of the database and simplifies the path that allows us to access any kind of data, whether reading the control fields from the microcontroller, or accessing the GPS locations from the web application.

Conclusion

In this chapter we presented the design procedures we followed in the implementation of our system. In the next chapter, we will display and discuss the results of our implementation. We will review the limitations we faced and explore the future works and improvements that can be added to this project.

CHAPTER IV
Results and Analysis

CHAPTER IV Results and Analysis

In this concluding chapter, we will present the resulting web application. We will first start by demonstrating the various views and core functionalities, as well as an analysis of its user-experience. Furthermore, we will conclude this chapter by discussing the acquired results and outlining potential future works.

1. Web Application Results

The following results are a showcase of the product we propose to solve the introduced problem at the beginning of this report. A complementary video guide is available at the following link: https://youtu.be/7WNNMAlkMY4

We have succeeded in releasing our work and hosting our web application on a web hosting service. Since the plan we used was a free trial, the website is then accessible from the following link: http://phoenix35-001-site1.atempurl.com/, for the period of 60 days from release, or until July 22nd, 2023.

1.1. Layout

The previous chapters revolved around the framework utilized for our application along its backend tools. In terms of the frontend, Razor provides a powerful method called 'RazorPage.RenderBody' that facilitates dynamic rendering of the content. By defining the navigation bar and the footer only once in the Layout page, the need to repetitively copy and paste the two elements in each individual view is completely avoided, ensuring consistency and less redundancy.

Figure 46 Render Body Layout

The navigation bar, when a user has not logged in yet, includes the application's name and logo, accompanied by the register and login buttons.

As for the footer, it contains information related the copyright and privacy policy.

1.2. Home Page

- *The Presentation Section:* This section gives the visitor a clear understanding of the services that the application offers.

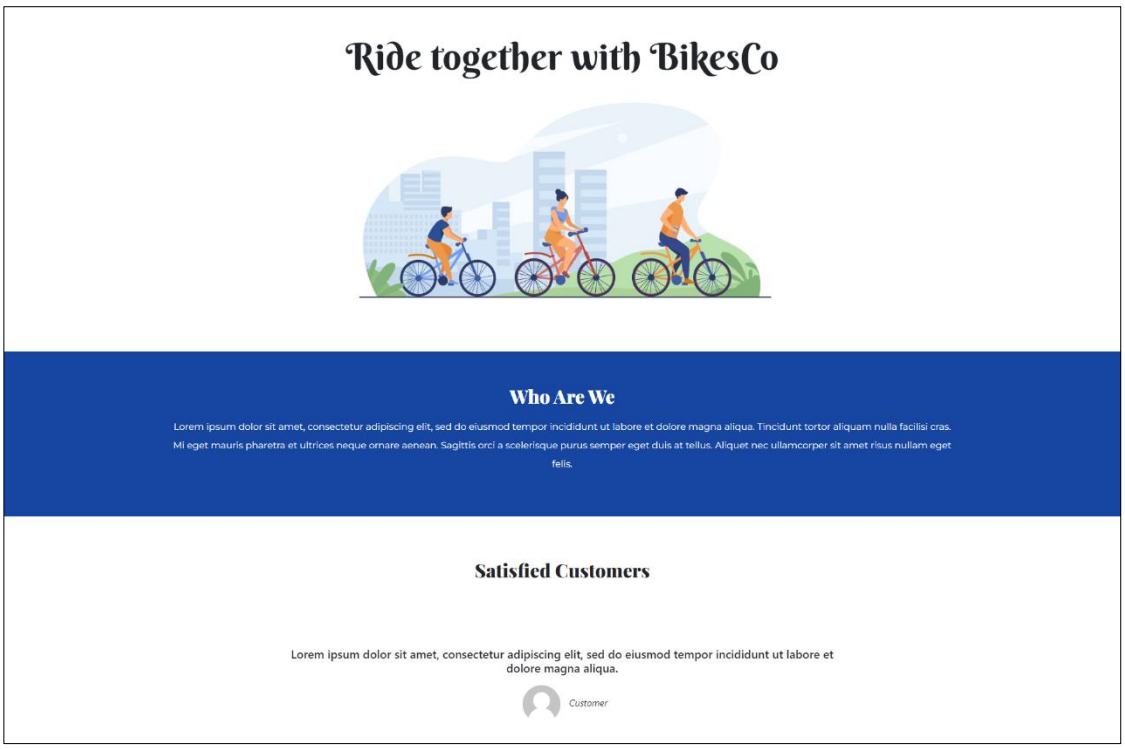


Figure 47 The Presentation Section

- *The Sign-Up Section:* This section encourages the visitor to register for an account membership. The "Sign Up" button, when clicked, will redirect the user to the Register View (refer to **1.3. Register**) where they can create their account.

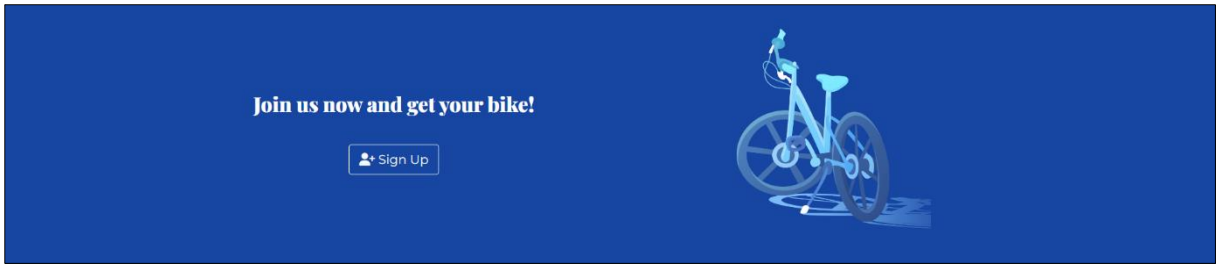


Figure 48 The Sign-Up Section

- *The Affiliate Section:* This section caters to any visitor that are interested in becoming affiliated with the company where the user can submit their bicycle for review. If the contract is accepted, and an administrator approves of the bicycle for potential rental, the customer will gain a commission for each rental transaction that occurs with their bicycle. The visitor is redirected to *Submit Bicycle View* (refer to **1.6.5. Contracts**).

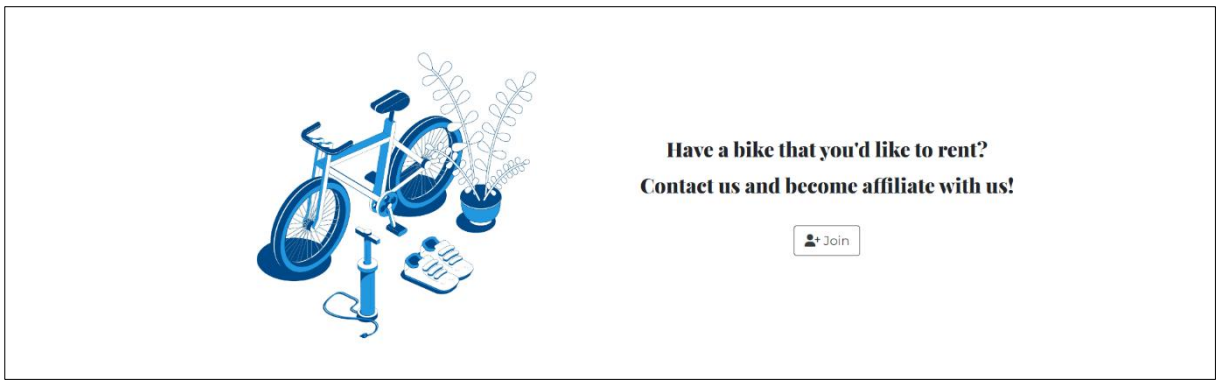


Figure 49 The Affiliate Section

1.3. Register

The following figure represents the **Register** vie where future customers fill out the form with their personal information.

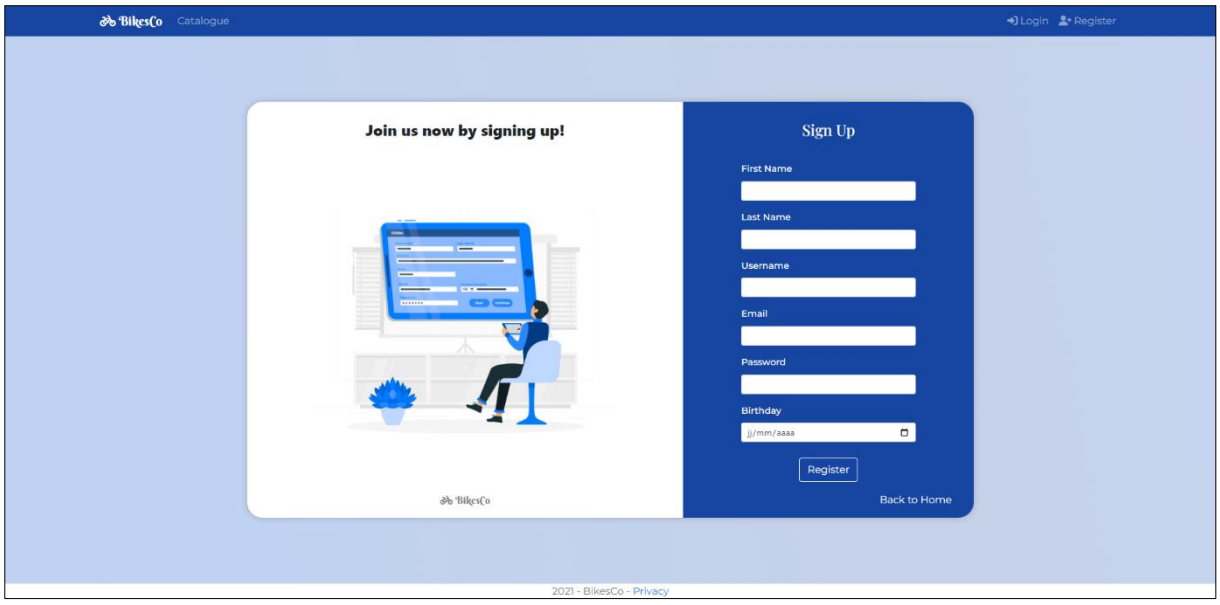


Figure 50 Register View

1.4. Login

Users who are already registered as customers or administrators have the ability to access the website by logging in. The following figure represents the **Login** view.

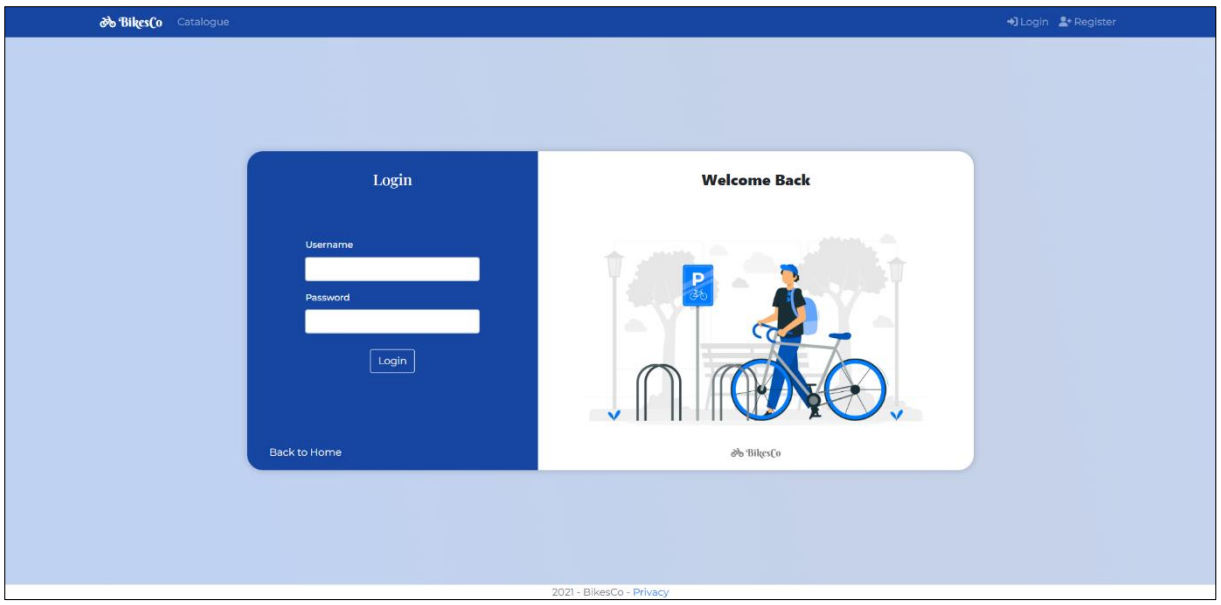


Figure 51 Login View

1.5. Catalogue

This view presents the various bicycle types that are readily available for rental. Each bicycle is accompanied by its corresponding image, title, starting price (per hour) and a "Details" button. This page allows the customer to browse through the available options.

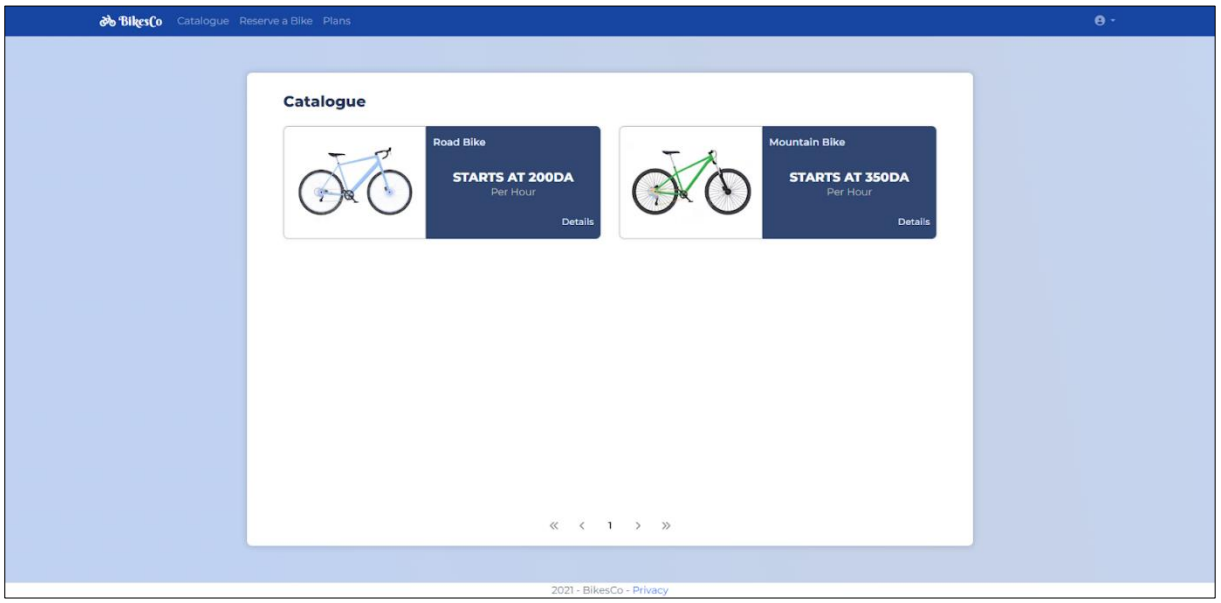


Figure 52 Catalogue View

When the "Details" button is clicked, a popup containing all the necessary information concerning the bicycle is displayed. Such as a detailed description and its rental prices details.

Figure 53 Catalogue Card

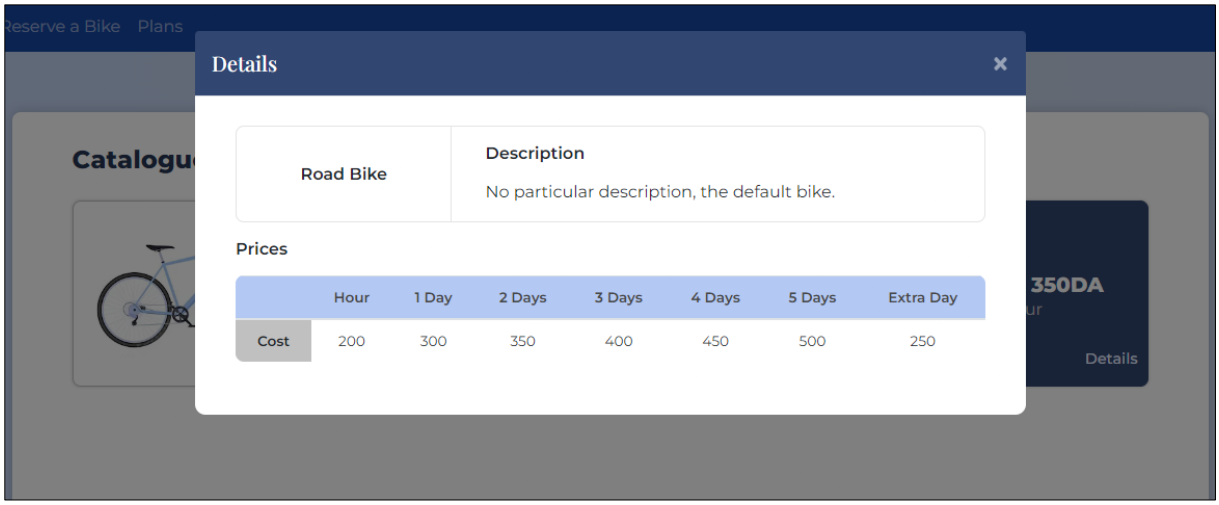


Figure 54 Catalogue Details

1.6. Customer

Upon successful login, the customer is redirected to the home page with an updated navigation bar. The updated navigation bar includes the following new links:

- *Reserve a Bike:* the customer will be able to make a reservation.
- *Plans:* the customer will be able to browse through the different plans.

By selecting the profile icon, a dropdown menu unfolds presenting the following options:

- *Profile:* the customer will be able to access their profile where they can edit, view their profile information, and other features that will be discussed later in Profile.
- *Reservations:* the customer will view their list of reservations.
- *Rentals:* the customer will view their list of transactions.
- *Contracts:* the customer will view their list of contracts.
- *Logout button:* the customer can logout from their account.

Figure 55 Customer's Navigation Bar

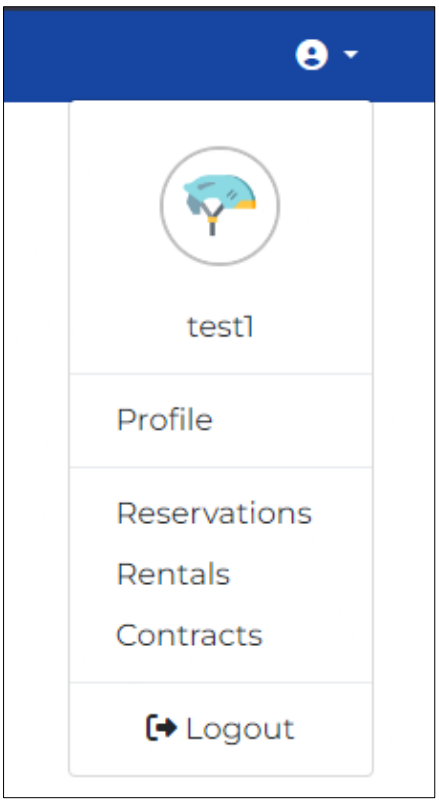


Figure 56 Customer Drop Down Menu

1.6.1. Making a Reservation

Once the customer has identified the bicycle of their choice, they are prepared to make a reservation.

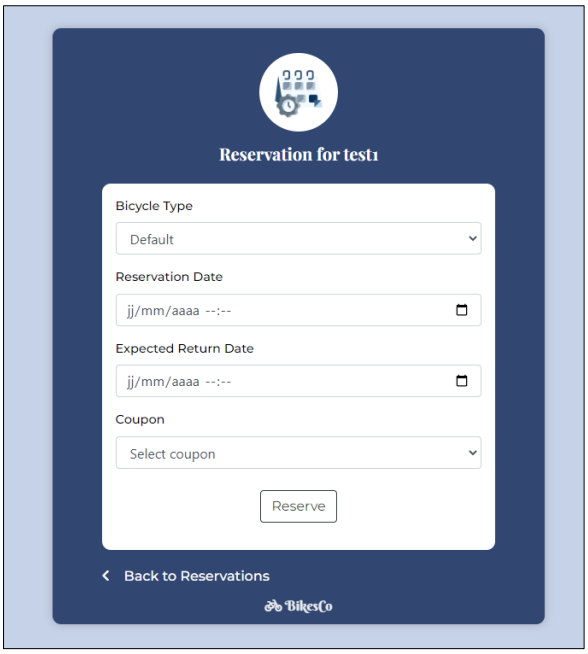


Figure 57 Make A Reservation View

The customer is prompted to select their desired bicycle, specify the reservation date and time at which the bicycle will be picked up, the expected return time and date, and optionally apply a coupon for the transaction.

1.6.2. Reservations

After completion of the reservation process, the customer can view their reservation in their list of reservations.

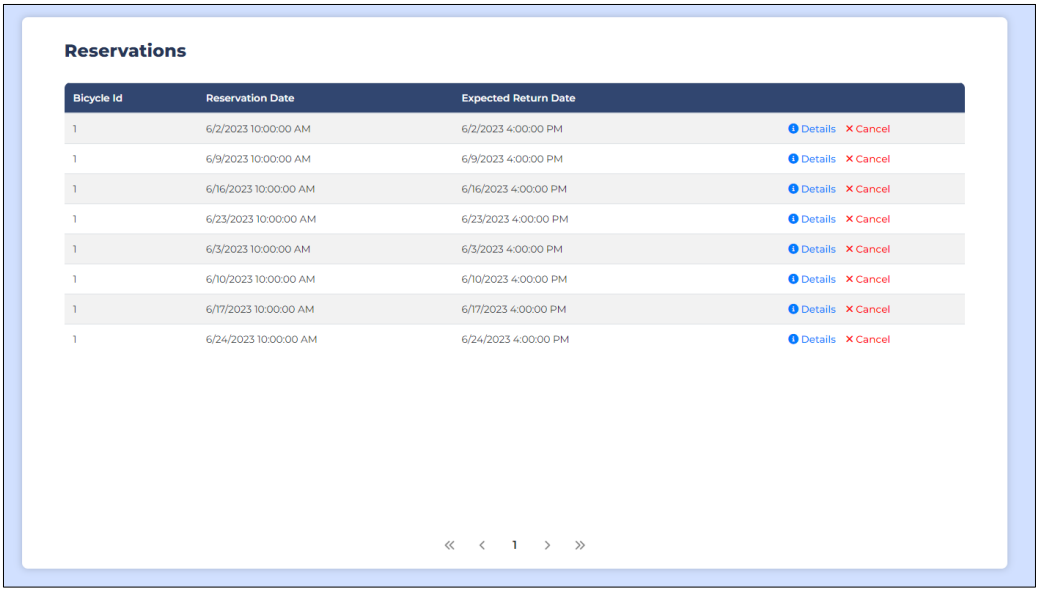


Figure 58 Customer's List of Reservations View

The page is structured as a table list with the essential details of the customer's reservations: the rented bicycle's ID, the reservation date, and the expected return date. For each reservation, the table provides both a "Details" and a "Cancel" button.

The "Details" button will trigger a modal that provides extra information concerning the reservation.

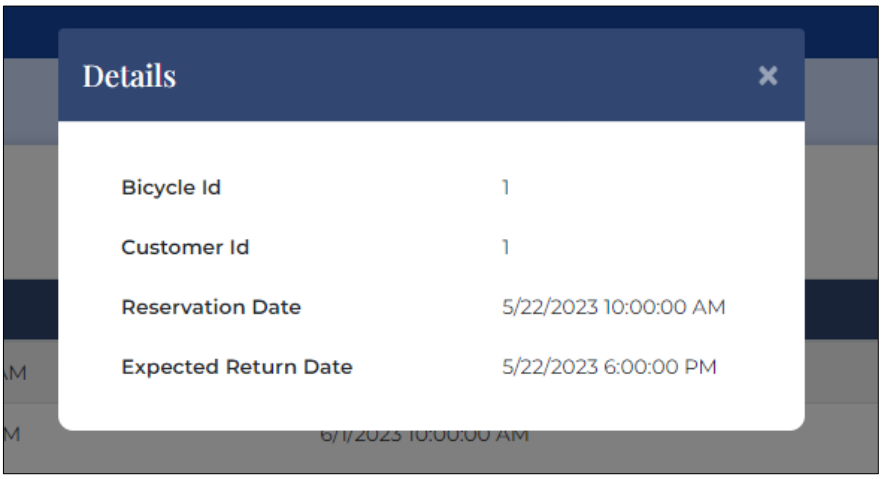


Figure 59 Customer's Reservation's Details Popup

The “Cancel” button will trigger a modal that prompts the customer whether to proceed with the cancelation of their reservation or not.

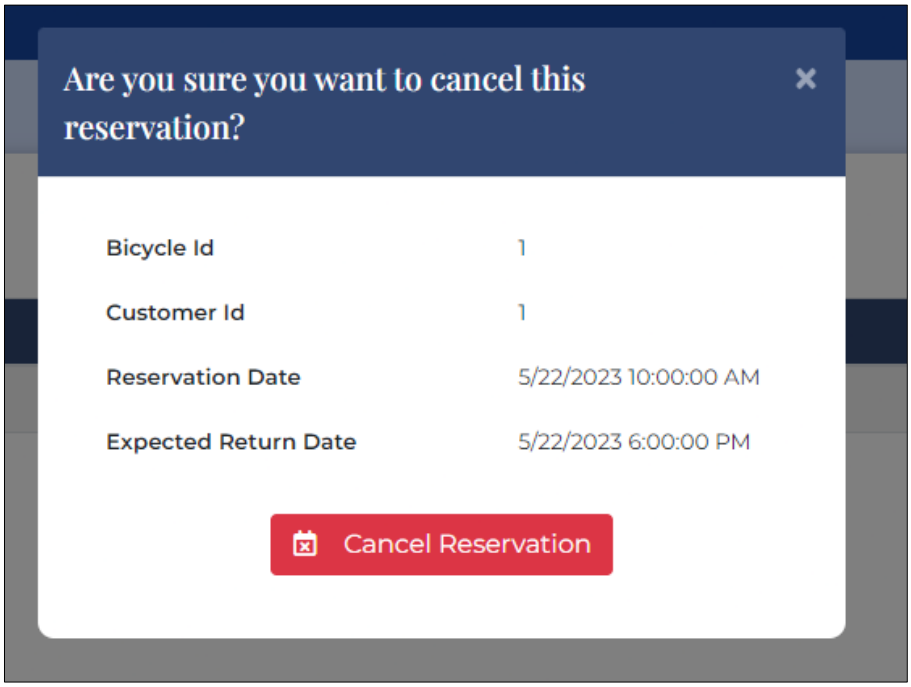


Figure 60 Customer’s Cancel Reservation Popup

1.6.3. Transactions

The customer can access their past and on-going transactions on this page, which is structured in a similar fashion as the Reservations view. The table presents information such as the transaction’s number, rental and expected return date, the actual return date (which remains empty until a transaction is completed), and the corresponding price.

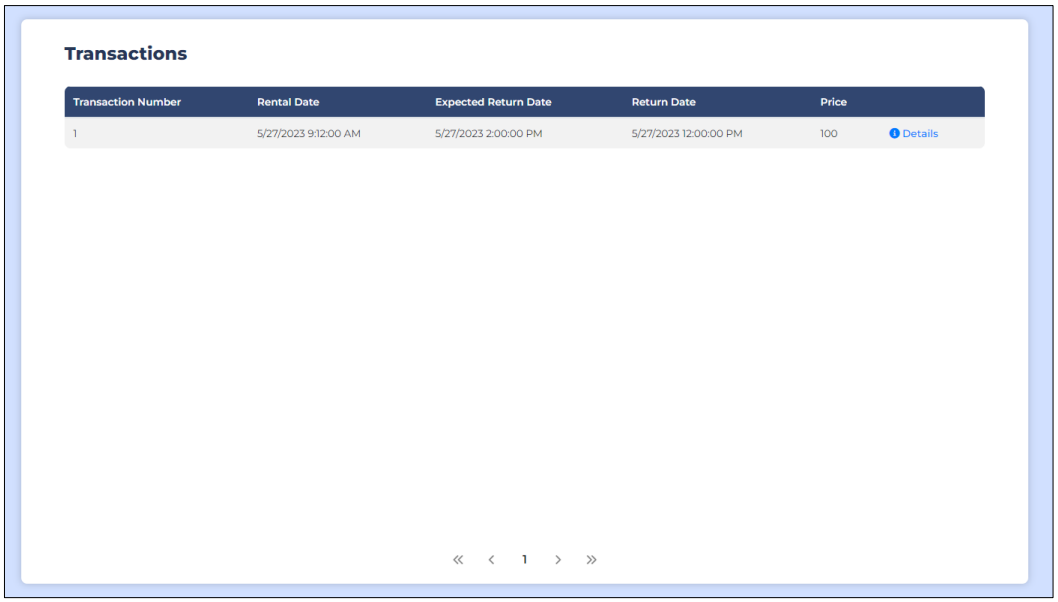


Figure 61 Customer’s List of Transactions View

The “Details” button directs the customer into the Transaction Details view. The remaining information concerning the transaction is displayed (such as the admin that confirmed the transaction and the actual paid price) accompanied by a GPS tracking path illustrating the route taken by the bicycle.

Unfortunately, the path coordinates will not align precisely with the road due to the limitations of the map APIs utilized in our application, Leaflet and ThunderStorm.

Note: The path displayed in the figure bellow was recorded during the testing process of our GPS Tracker.

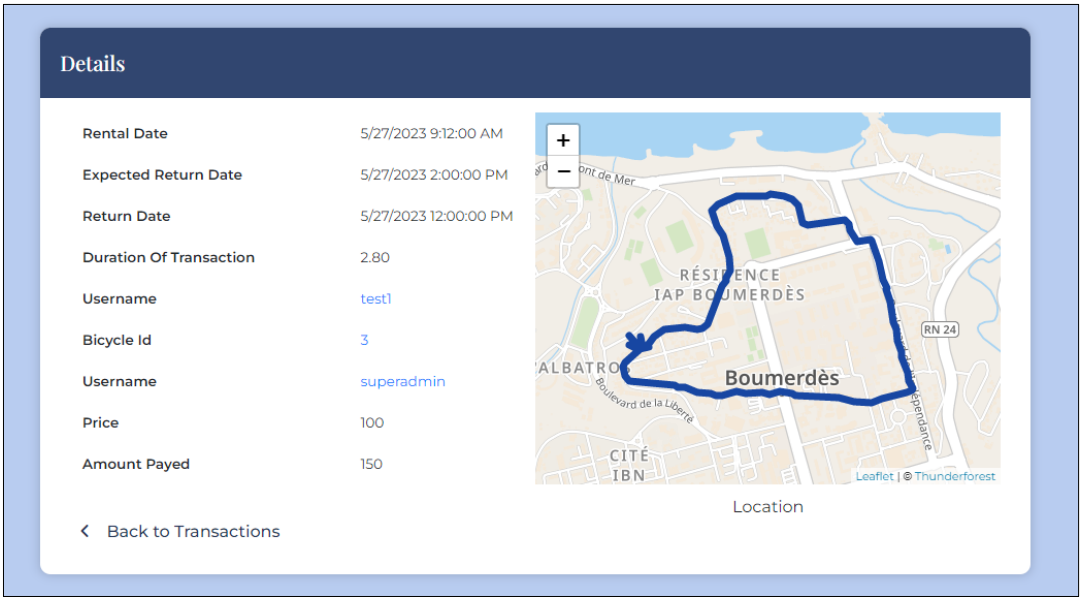


Figure 62 Transaction’s Details View

1.6.4. Profile

This page facilitates the viewing of the profile details and offers a side menu that enables navigation through various options. The customer has the ability to:

- Edit their profile details and password,
- Redeem points for coupons,
- View their active subscriptions,
- View their coupon inventory.

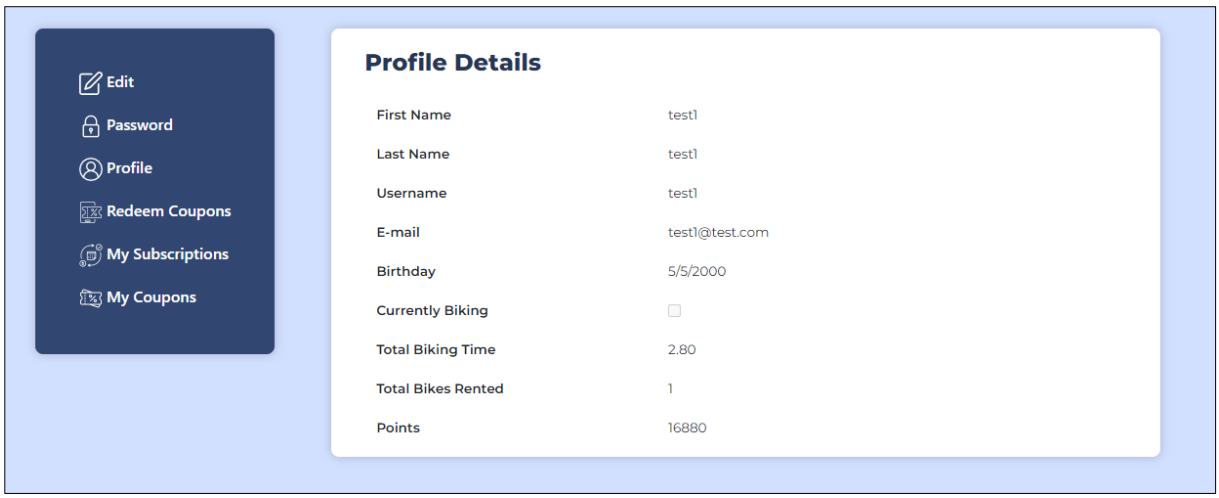


Figure 63 Customer's Profile View

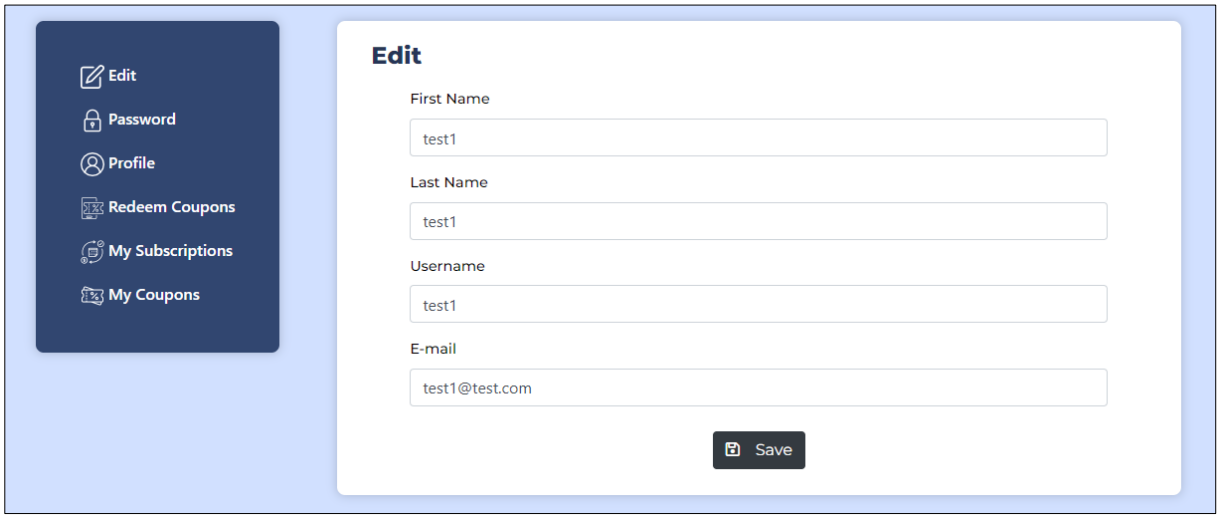


Figure 64 Customer's Edit Profile View

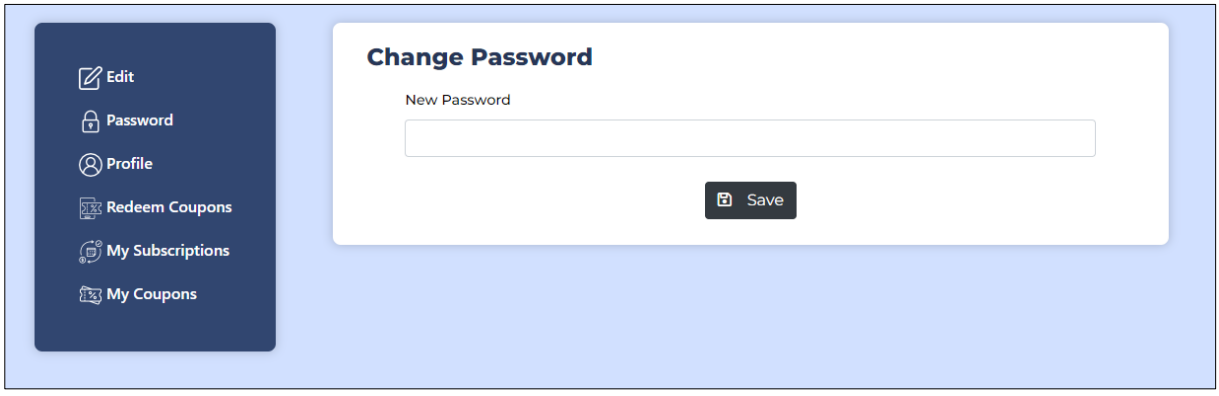


Figure 65 Customer's Change Password View

1.6.4.1. Redeem Coupons

The application incorporates a points system, allowing customers to accumulate points based on their activities. These points can be later exchanged for coupons that can be applied during transactions.

The following figure showcases the list view.

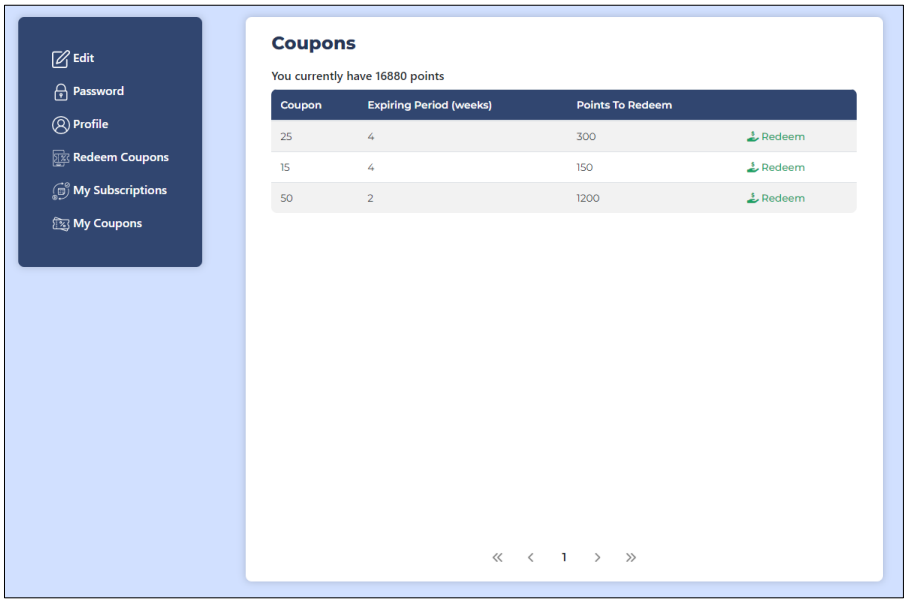


Figure 66 Customer's Redeem Coupons View

At the very top, below the title of the view, the customer can observe their current balance. Beneath it, a table displays the list of available coupons that the customer can retrieve in exchange of points.

When a customer selects a coupon, by clicking on the "Redeem" button, a popup is activated and prompts the customer for the confirmation to proceed with the conversion.

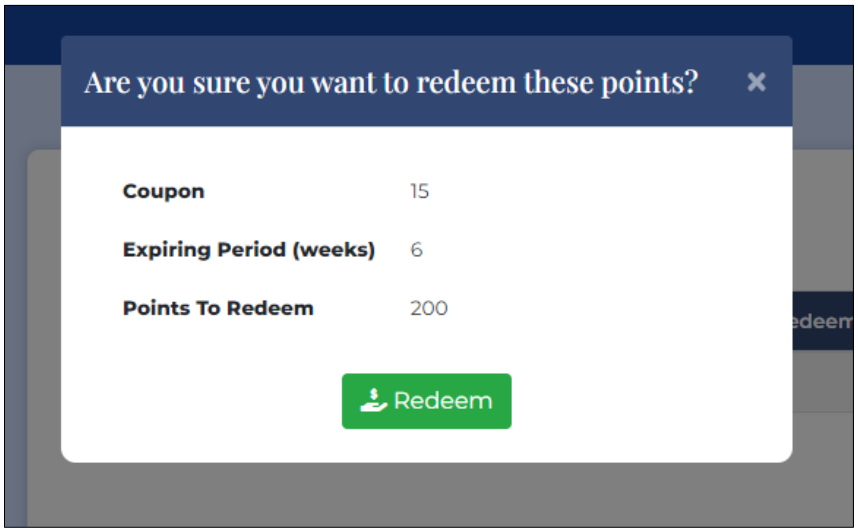


Figure 67 Customer's Confirm Redeem Coupons Popup

Upon completion of the operation, the customer can utilize the selected coupon. To browse through the coupon inventory, the customer can navigate to Coupons.

1.6.4.2. Coupons

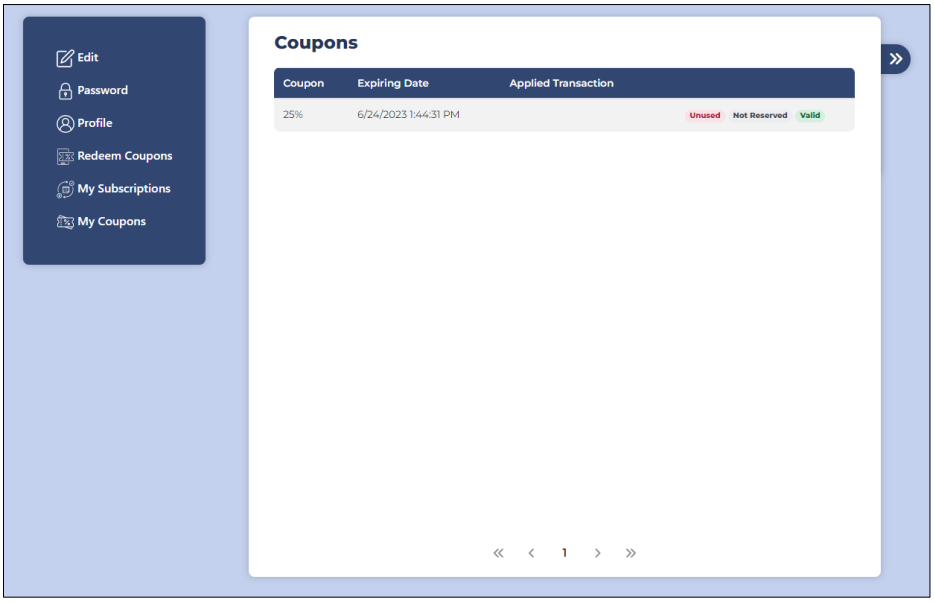


Figure 68 Customer's List of Coupons View

The customer has the ability to manage their active and usable coupons in this view, with access to information regarding each coupon's value, expiration date, and the transaction to which it was applied.

If the customer desires to see their expired, inactive or deleted coupons, they can simply click on the little arrow at the top right corner of the page. A menu, encompassing navigational links for each list, will slide to the right with a sliding animation.

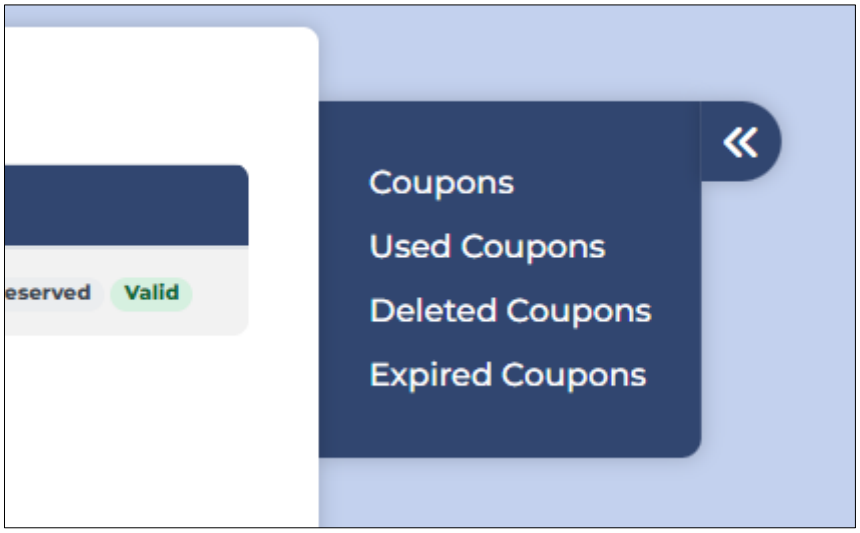


Figure 69 Customer's Coupons Sliding Menu

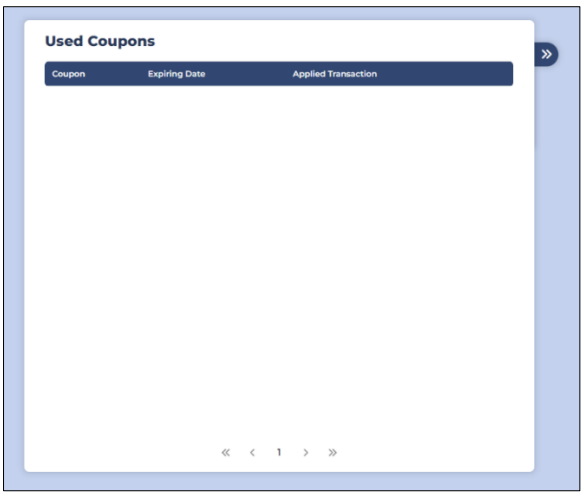


Figure 70 Customer's List of Used Coupons View

Figure 71 Customer's List of Deleted Coupons View

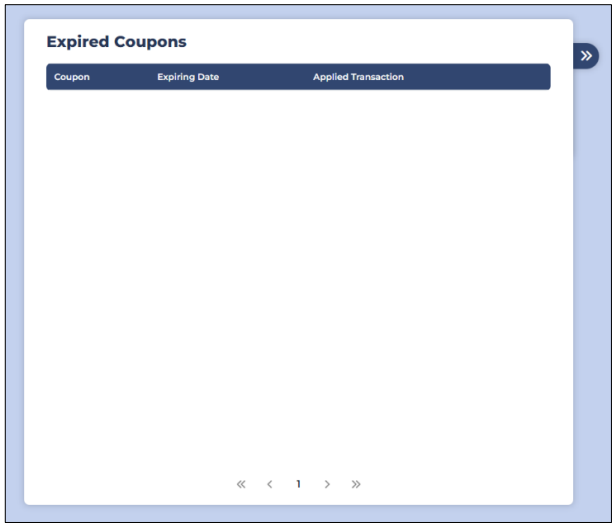


Figure 72 Customer's List of Expired Coupons View

1.6.4.3. Subscriptions

The final option in the menu is the Subscriptions list. The customer can utilize it to view and manage their subscriptions. Selecting the "Details" button will issue a popup with specific details related to the subscription, while the "Delete" button will trigger another popup allowing the customer to delete the subscription.

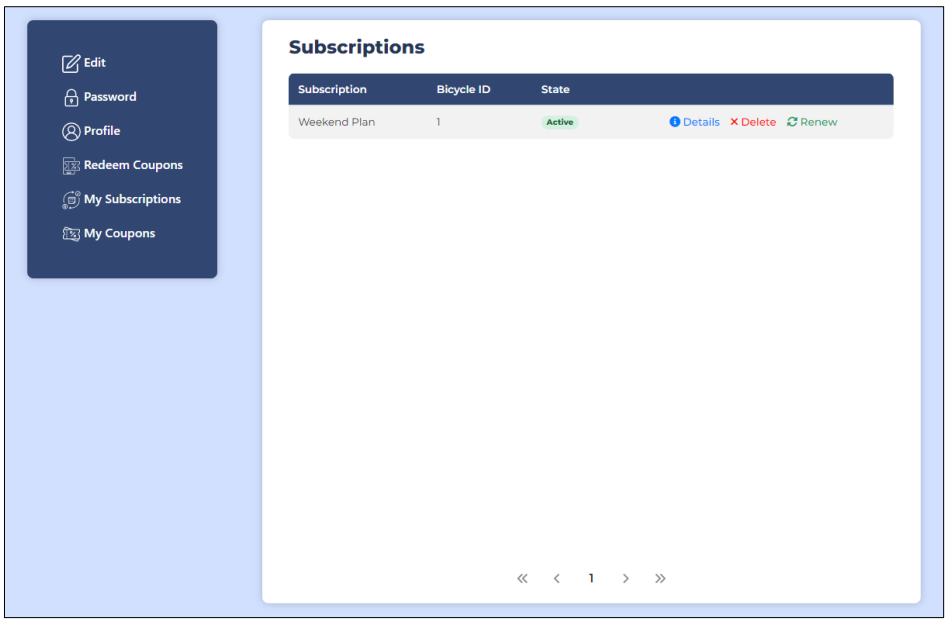


Figure 73 Customer's List of Subscriptions View

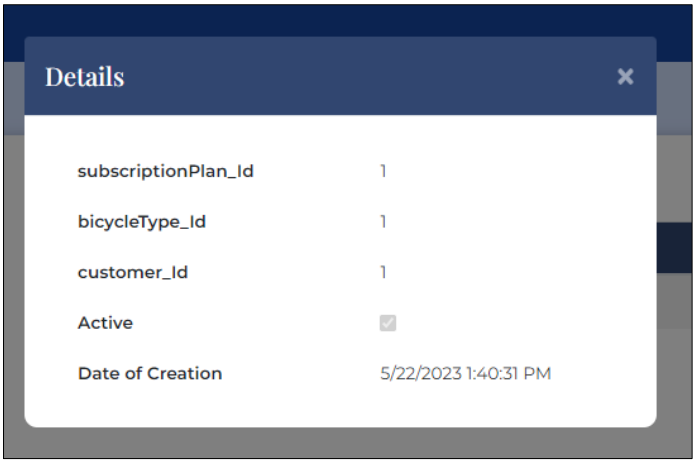


Figure 74 Customer's Subscription Details Popup

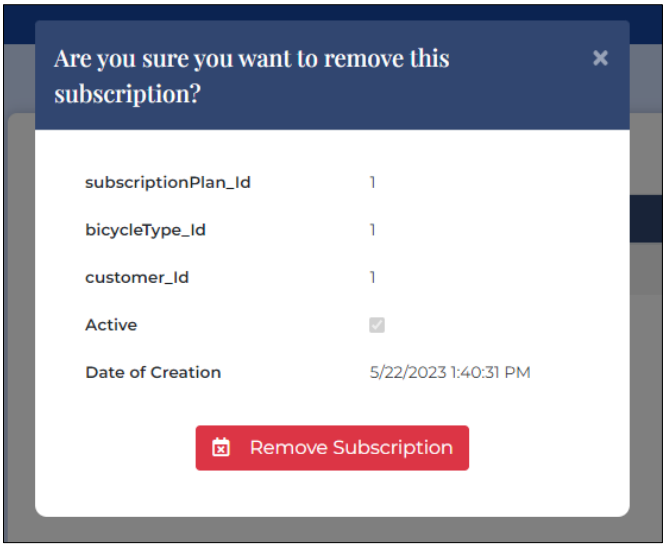


Figure 75 Customer's Remove Subscription Popup

1.6.5. Contracts

The customer can manage their contracts in this view. The list displays the list of active contracts the customer currently has.

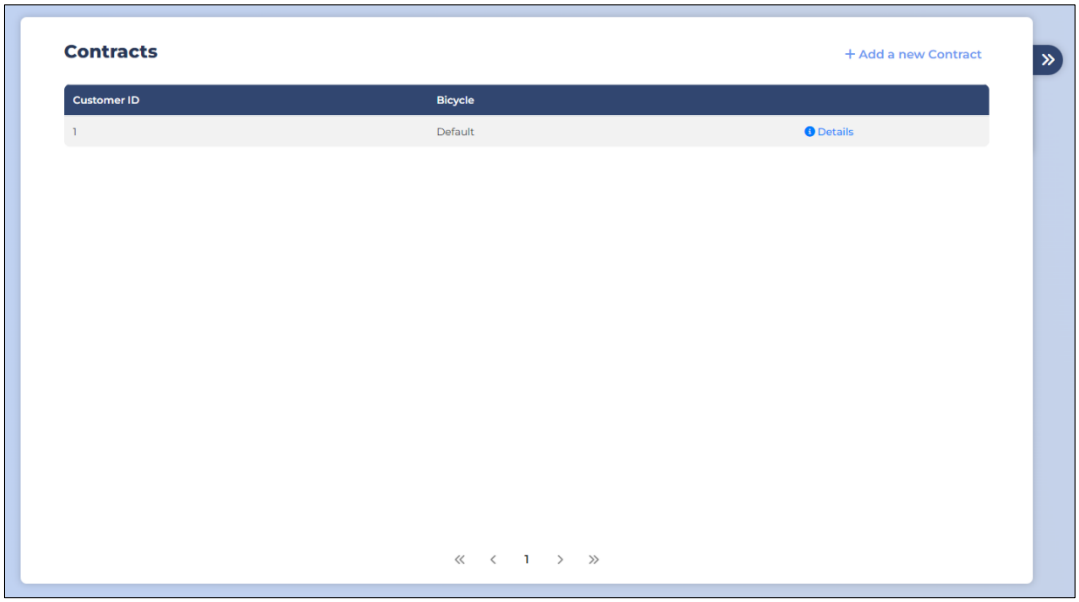


Figure 76 Customer's List of Contracts View

The "Details" button provides additional information by displaying a popup. Within the Details popup, an accessible gallery is presented, offering two navigation buttons that enable the admin to browse through and view the various images provided.

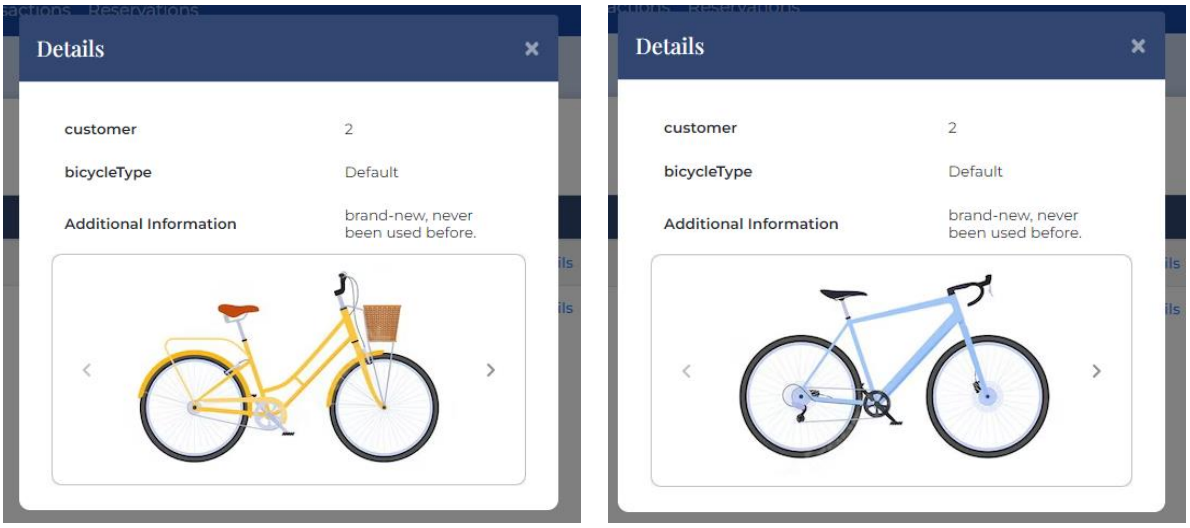


Figure 77 Customer's Contract Details Popup

If the customer wishes to submit a bicycle for an affiliated contract, they can either access the submission page through the homepage (as mentioned above in **1.2 Home Page**, *Affiliate Section*) or by clicking on the "Add a new Contract" button at the top right corner.

The button will display a modal with a form requiring the customer to submit their bicycle details for future review by the admins.

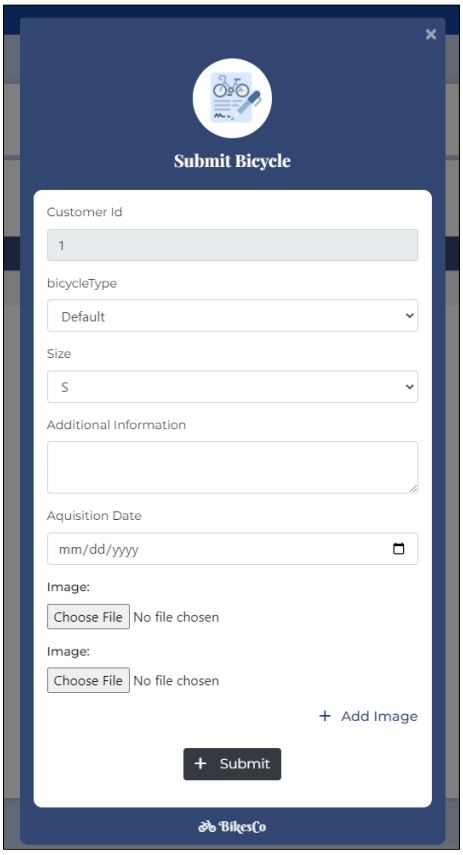


Figure 78 Submit a Bicycle Contract Popup

The “Add Image” button will add another “Image:” section if the customer has more than two pictures to submit.

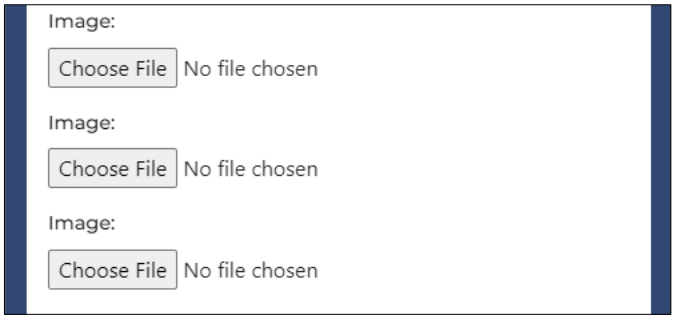


Figure 79 Add an Image Bicycle Contract

Similar to the Coupons view, a menu is available to the customer from the arrow button at the top right corner. This menu provides the ability to navigate through the customer’s lists of inactive and denied contracts.

Figure 80 Contracts Sliding Menu

1.6.6. Plans

A customer has the opportunity to take advantage of the plans offered by the application, which can be accessed through the Plans link. The following figure represents the view.

Figure 81 Plans View

Each Plan is represented by its title, price per month, and an icon that indicates whether it is a premium or a regular plan. A “Details” button is positioned on each Plan card for any further information, such as the subscription’s timetable.

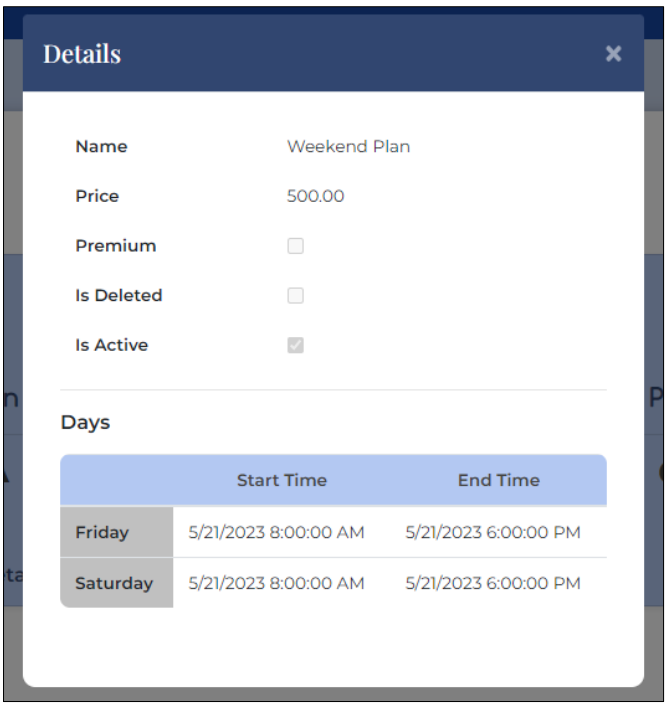


Figure 82 Customer's Plans Details

1.7. Admin

It was previously mentioned that the admins have different roles, where each admin had access to a specific set of views. For the sake of simplicity, we will show the entirety of the pages on the admins' side and not mention the roles.

Similar to a customer, when an admin logs in, they will be directed to the Home Page with an updated navigation bar. Notably, an admin's navigation bar offers more functions.

Figure 83 Admin's Navigation Bar

- *Bicycles:* provides a list of all the available bicycles.
- *Bicycle Types:* provides a list of the different bicycle types.
- *Customers:* provides a list of the registered customers.
- *Admins:* provides a list of admins along their respective roles.
- *Transactions:* provides a list of ongoing and past transactions.
- *Reservations:* provides a list of reservations.

As for the drop-down profile customer menu icon, the admin has the following new options:

- *Coupons:* provides a list of the available coupons.
- *Plans:* provides a list of the available, active, and disabled subscriptions.
- *Contracts:* provides a list of the appending and confirmed contracts.

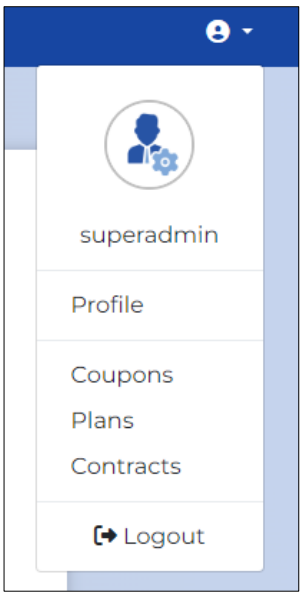


Figure 84 Admin's Drop-Down Menu

1.7.1. Bicycles

This view displays a list of bicycles owned by the company along with their corresponding ID, type, and state. The state information indicates whether a bicycle is available, reserved or currently being rented.

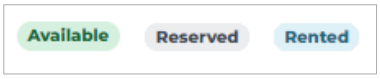


Figure 85 Bicycle's State

Furthermore, three buttons are provided for each bicycle. An "Edit" button that allows the admin to edit the bicycle's details, a "Details" button that allows the view of the bicycle's information, and a "Delete" button that allows the admin to delete the bicycle. Upon clicking these buttons, a popup will be triggered, presenting the admin relevant tasks corresponding to each button's function.

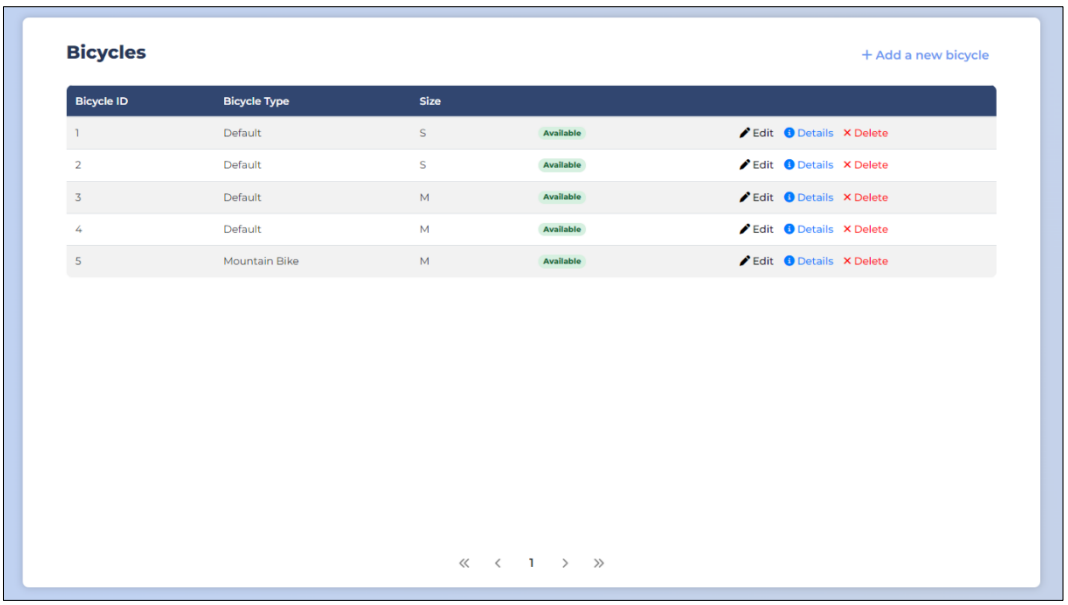


Figure 86 List of Bicycles View

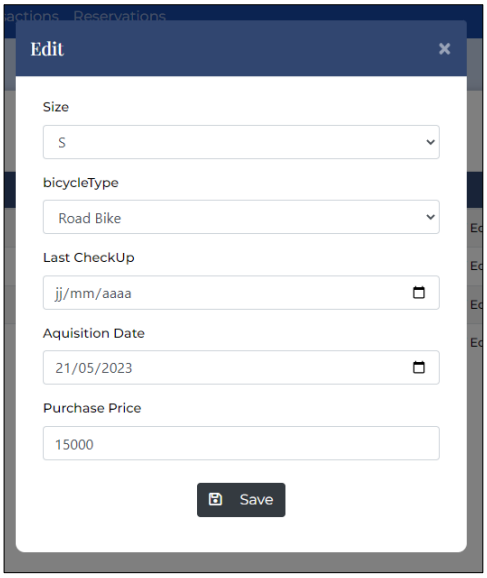


Figure 87 Bicycle's Edit Popup

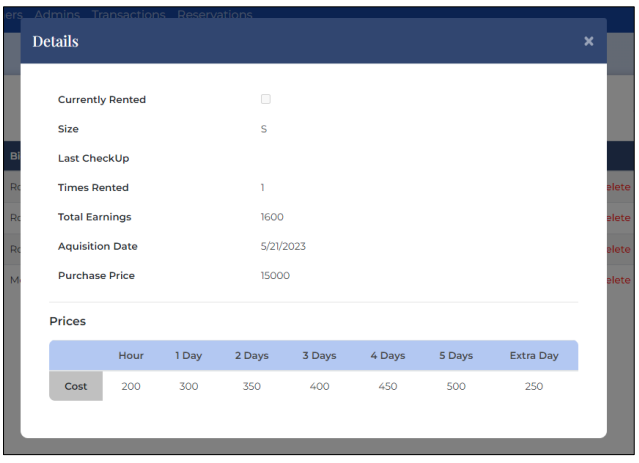


Figure 88 Bicycle's Details Popup

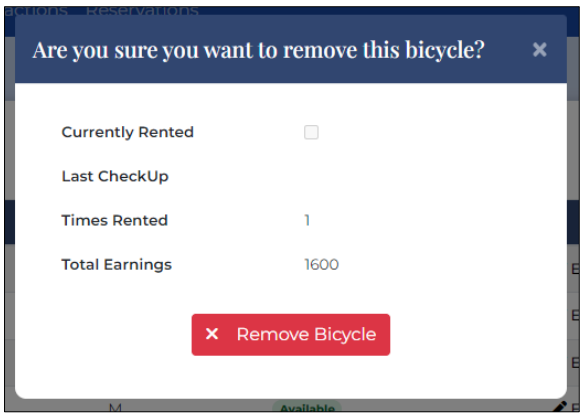


Figure 89 Bicycle's Delete Popup

An additional button, "Add a new bicycle" is located at the top right corner of the page. The button serves the purpose of adding a new bicycle to the inventory. When clicked the following modal is triggered:

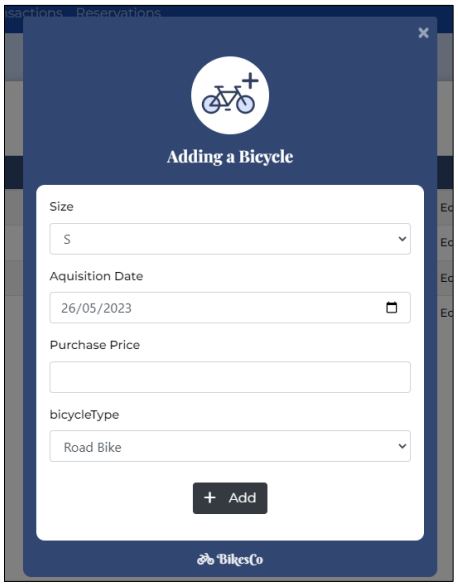


Figure 90 Adding a Bicycle Popup

1.7.2. Bicycle Types

Bicycle Types, or Catalogue, is the same **Catalogue** view seen in **1.6. Customer**. The only distinction lies in the inclusion of two other buttons. "Add a new bicycle", for adding a new type of bicycle to the catalogue, and "Edit", for editing the details' information of a selected bicycle type. Clicking either of them will trigger their respective modal for the desired action.

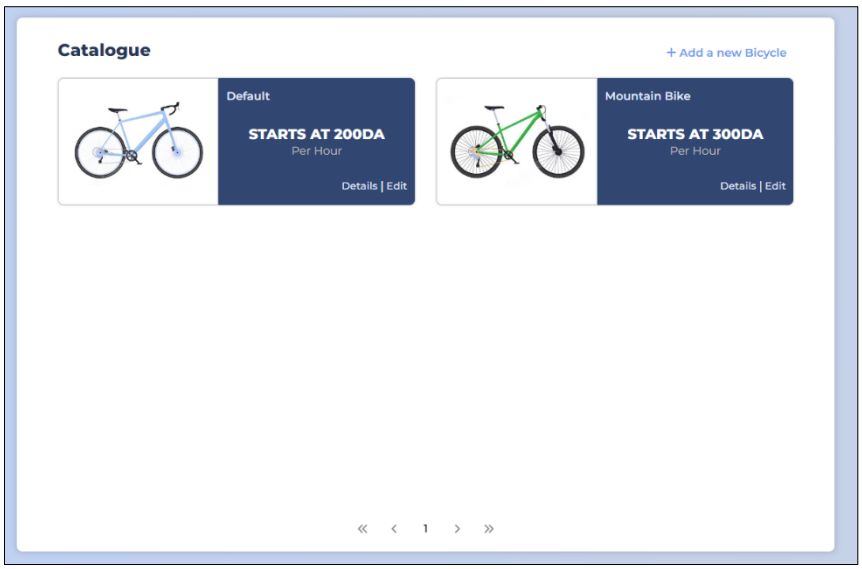


Figure 91 Admin's Catalogue View

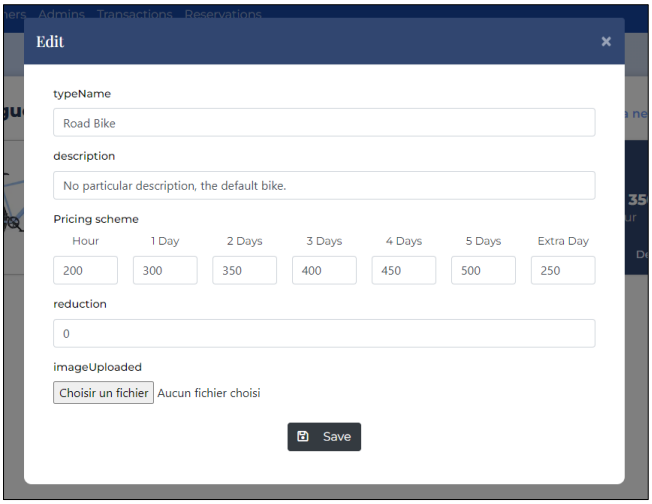


Figure 92 Catalogue's Bicycle Edit Popup

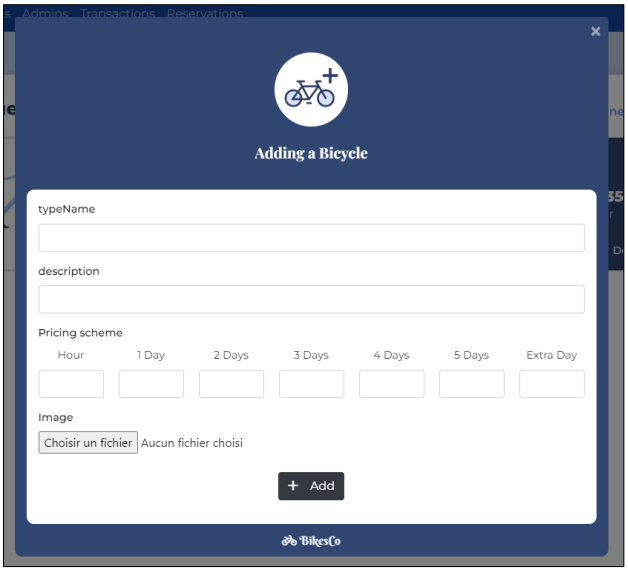


Figure 93 Adding a Bicycle Type Popup

1.7.3. Customers

The customers' list view provides admins with the ability to access a list of all registered customers, including their username, ID, full name, state (indicating if they are currently biking or not). Two additional buttons, "Details" and "Delete", are available for each customer, triggering their respective popup upon clicking.

The "Add a new customer" button at the top right corner serves the purpose of registering a new customer. When clicked, the button will redirect the admin to the register page (refer to **1.3. Register**).

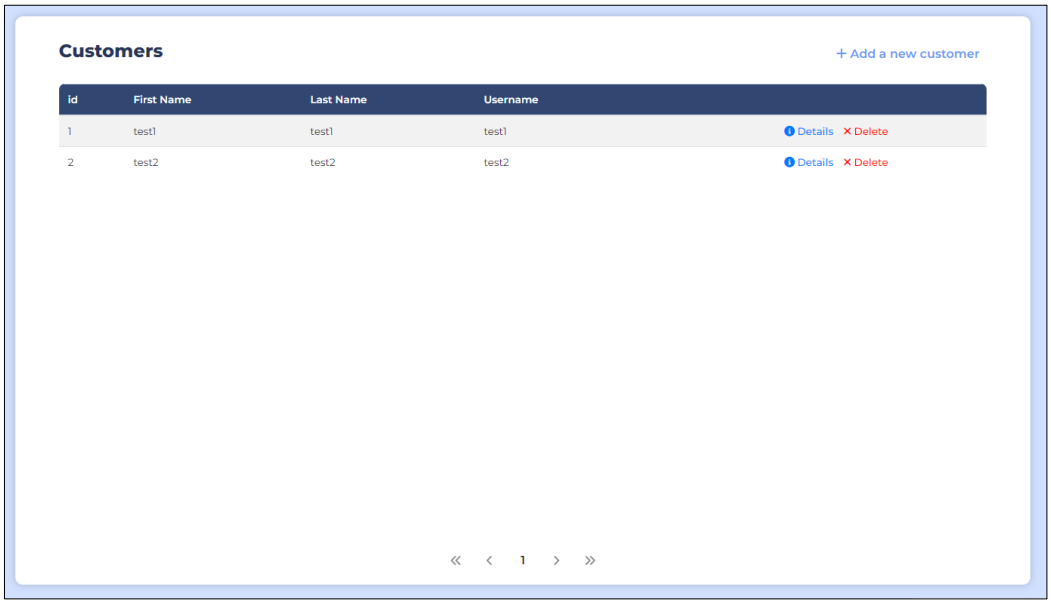


Figure 94 List of Customers View

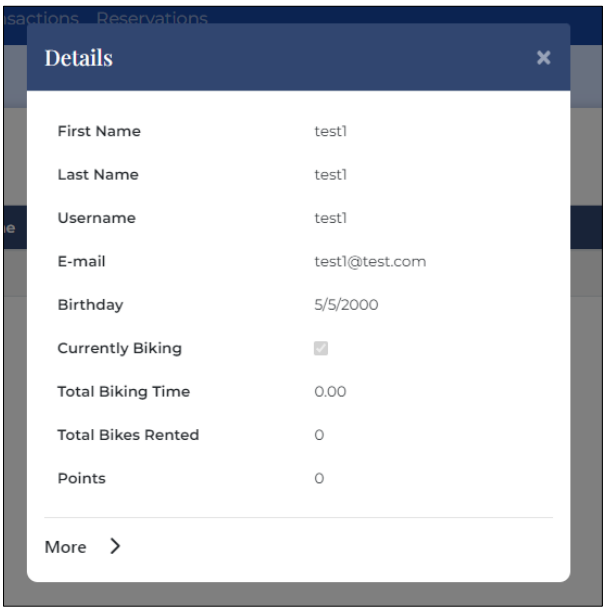


Figure 95 Customer's Details Popup

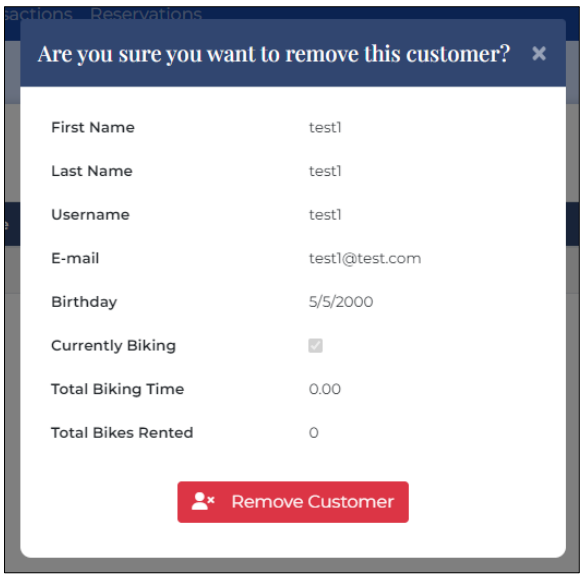


Figure 96 Customer's Delete Popup

At the bottom of the modal, a "More" button is present. Upon clicking, the following menu slides down, providing the admin additional functions:

Figure 97 Customer's More Button

Upon selecting one of the provided functions, the admin will be redirected to their corresponding view, based on their functionality.

To facilitate the navigation between these options, a floating menu, similar to the one found in the customer's views, is provided. Additionally, a "Back to Customers" button is available in the menu. The button redirects the admin back to the **Customers**' list.

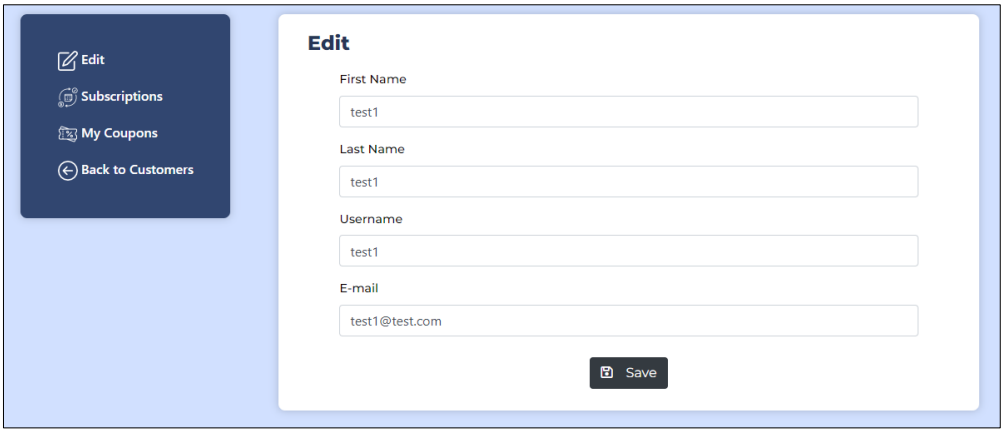


Figure 98 Admin's Customer Edit View

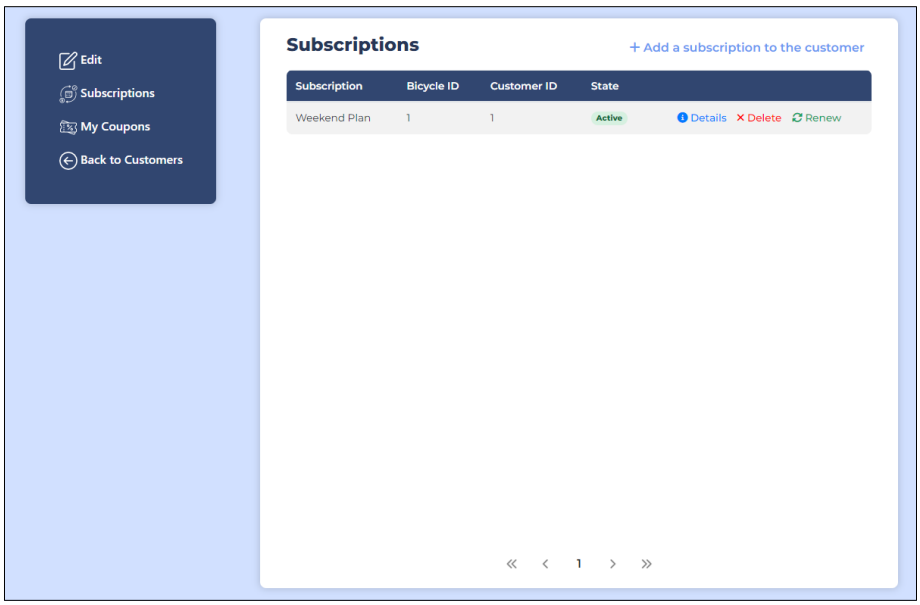


Figure 99 Admin's Customer List of Subscriptions View

For Subscriptions, the view remains identical to the view in **1.6.4.3. Subscription**. It only differs in two additional buttons:

"Renew" will trigger a modal that prompts the admin to confirm the renew of the customer's subscriptions.

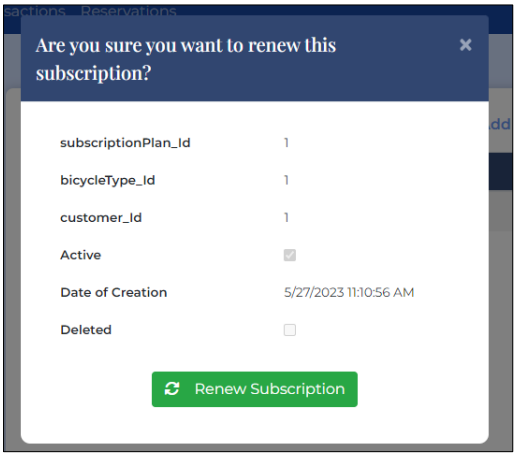


Figure 100 Renew Subscription Popup

"Add a subscription to the customer" will trigger a modal where the admin can add a subscription to the selected customer. The "Customer ID" field is automatically filled with the customer's ID.

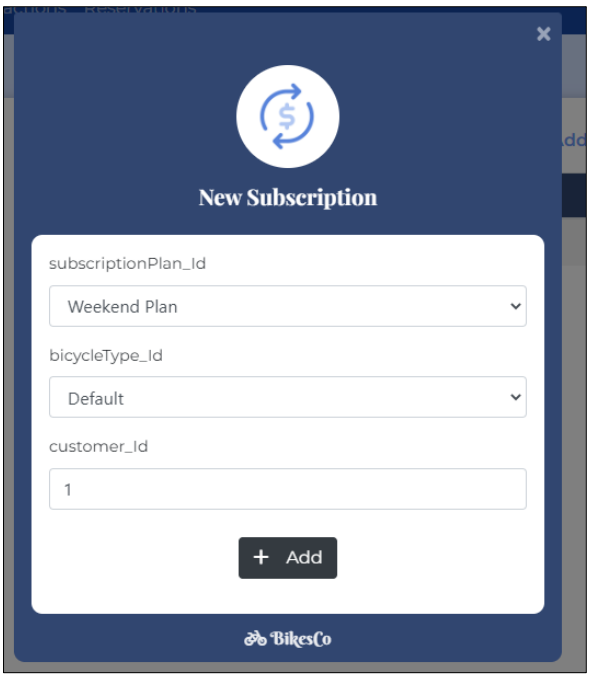


Figure 101 Add a Subscription to Customer Popup

The Coupons view is the exact same as the one found in **1.6.4.2. Coupons**.

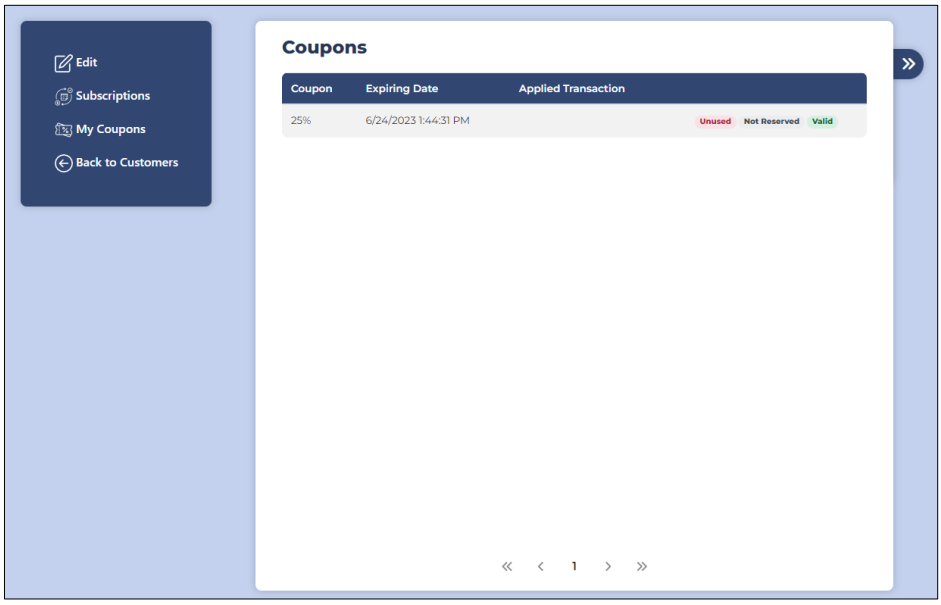


Figure 102 Admin's Customer Coupons View

1.7.4. Admins

This page is fairly similar to **Customers**, with the difference being it presents a list of admins. Each entry has an "Edit", "Details", and "Delete" buttons. The functionality of these buttons is the same as the previously mentioned ones.

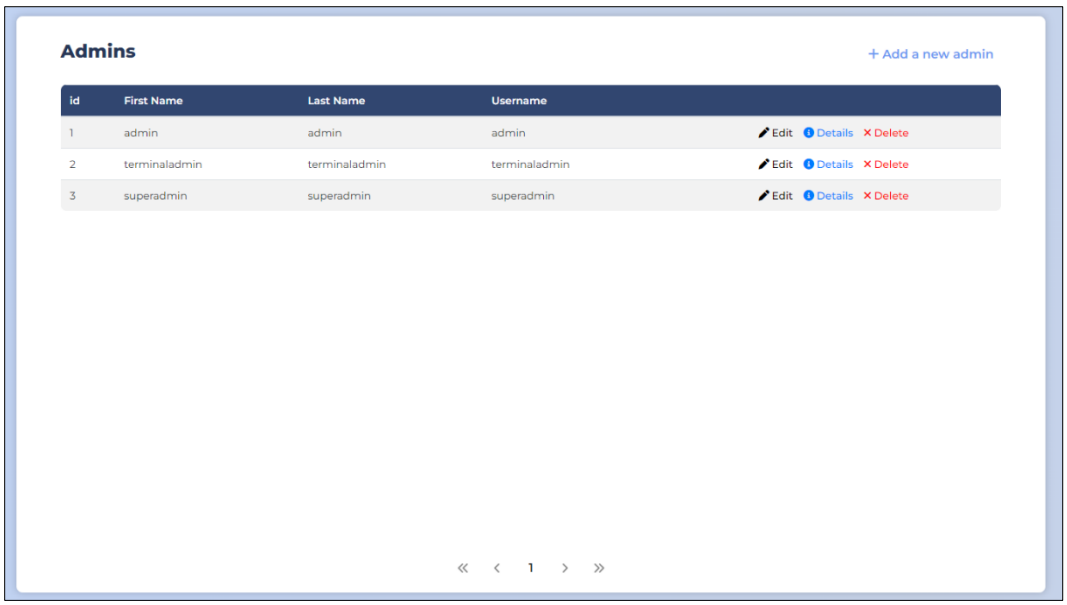


Figure 103 List of Admins View

The "Edit" modal provides an additional functionality that allows for the modification of an admin's role. This can be achieved by simply checking or unchecking the corresponding box associated with each role.

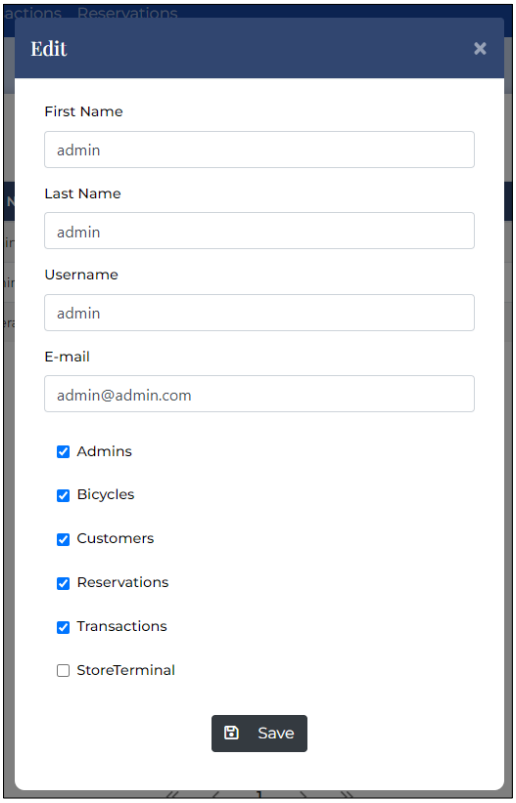


Figure 104 Admin's Edit Popup

The “Add a new admin” button activates the popup shown below. It allows an admin to register a new admin by providing their personal information, as well as selecting their designated role(s).

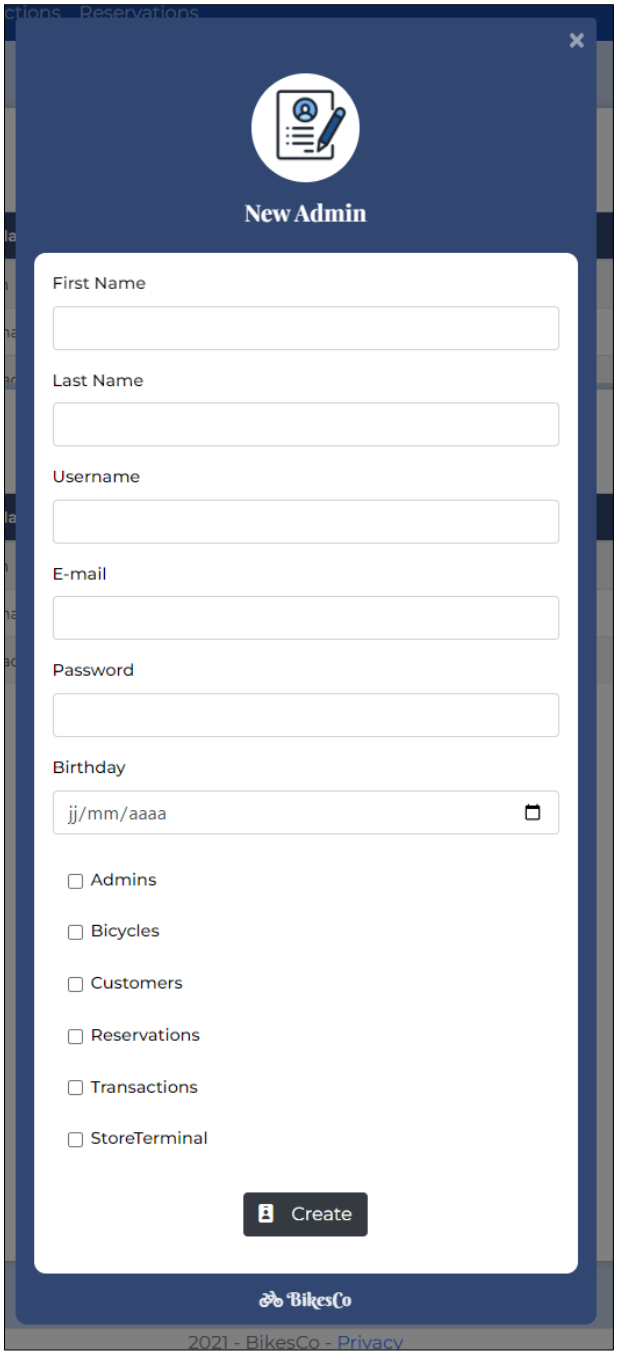


Figure 105 New Admin Popup

1.7.5. Reservations

The reservations page presents a list of all reservations, with each entry accompanied by an “Edit”, “Details”, “Confirm”, and “Deny” buttons. The “Edit” button allows for the modification of reservation’s details while the “Details” button provides a view of the

reservation's details. On the other hand, "Confirm" and "Deny" prompts the admin to confirm or deny a reservation respectively.

Each of the aforementioned buttons trigger a popup.

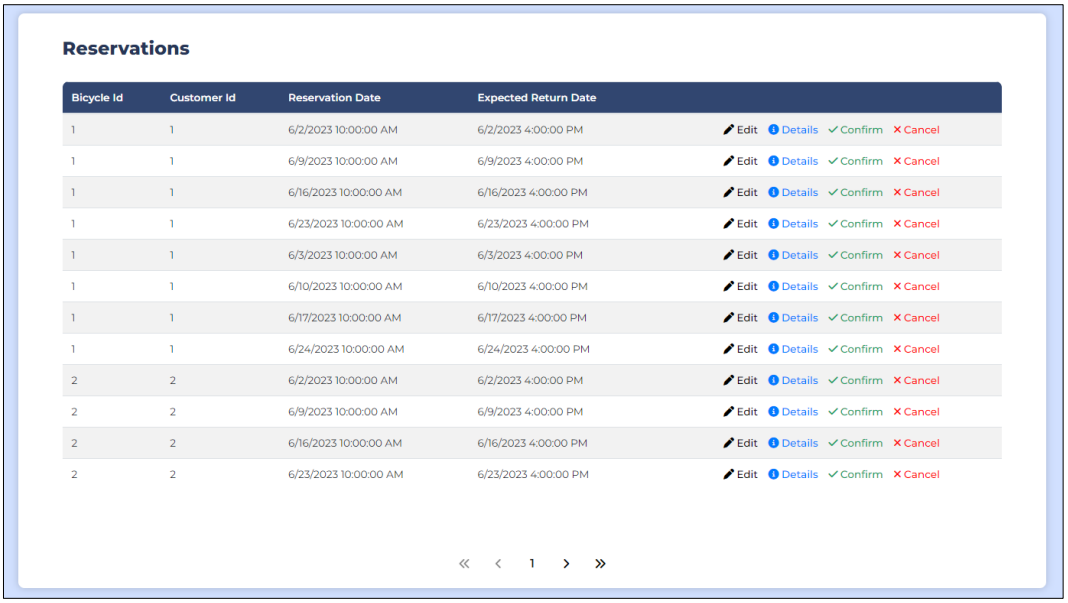


Figure 106 List of Reservations View

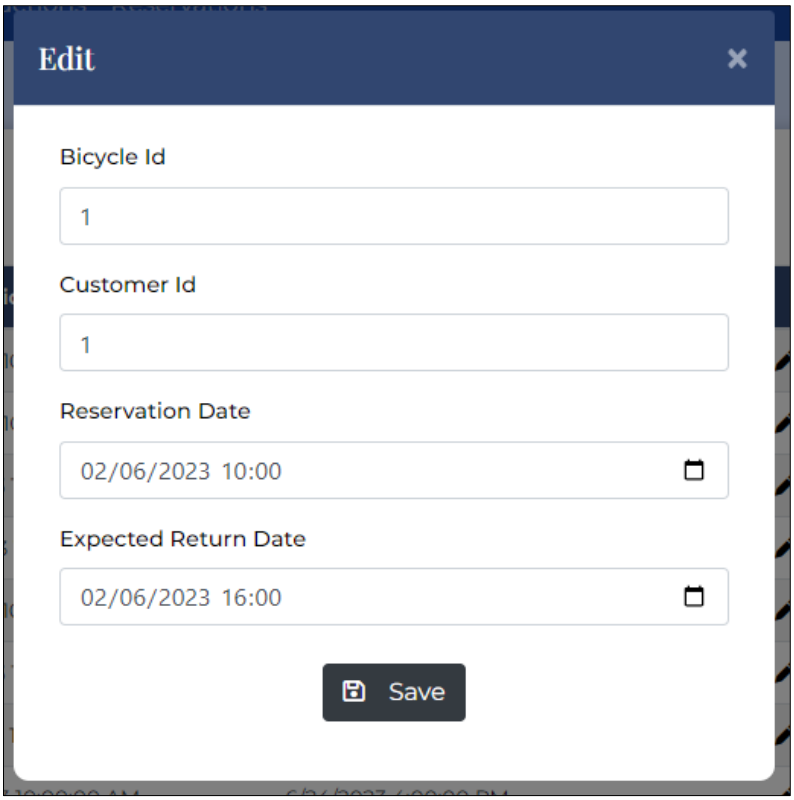


Figure 107 Reservation's Edit Popup

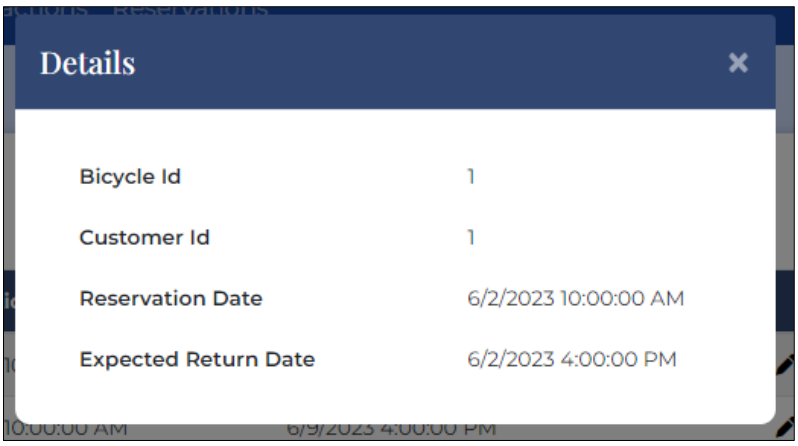


Figure 108 Reservation's Details Popup

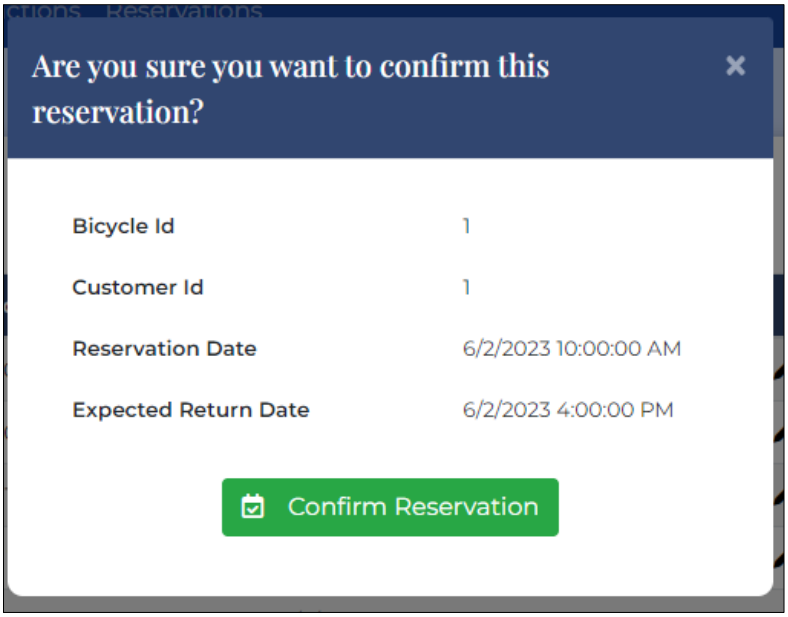


Figure 109 Reservation's Confirm Popup

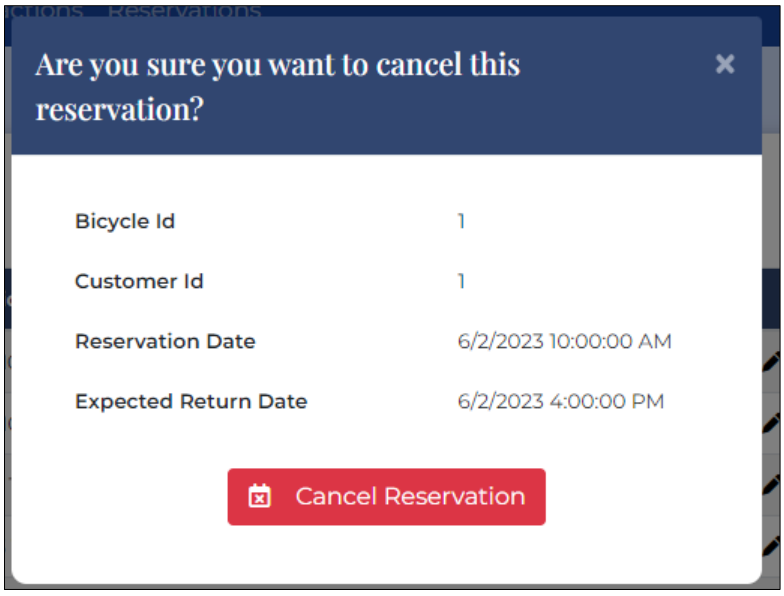


Figure 110 Reservation's Cancel Popup

1.7.6. Transactions

The **Transactions** view provides the admin a comprehensive list of both past and on-going transactions.

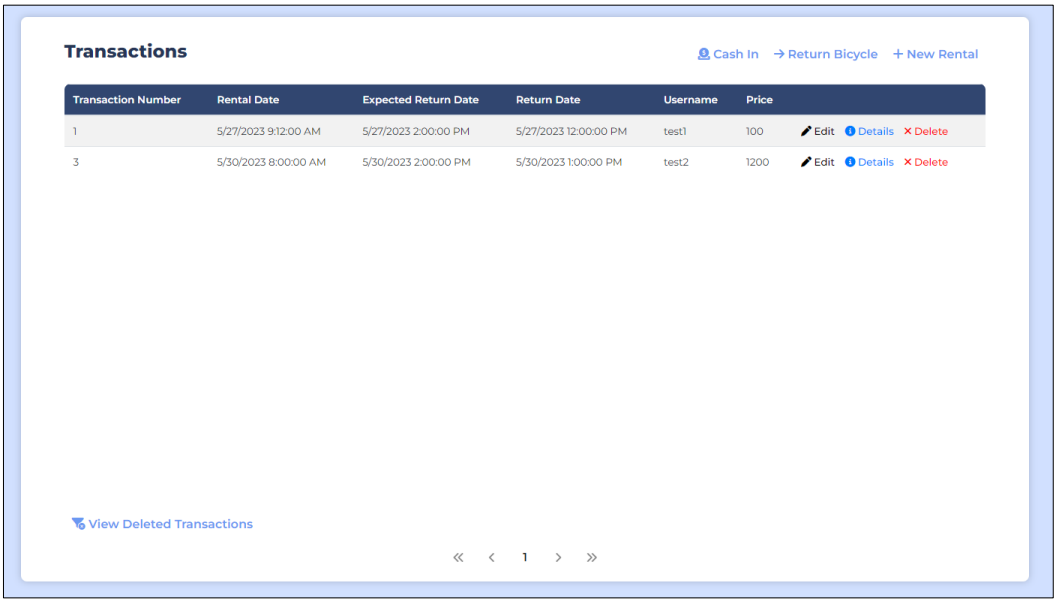


Figure 111 List of Transactions View

The standard set of buttons: "Edit", "Details" and "Delete" can be found in each row in the list. "Edit" and "Delete" will trigger a modal, while "Details" will redirect the admin to the Details view, which corresponds to the same view mentioned in **1.6.3. Transactions**.

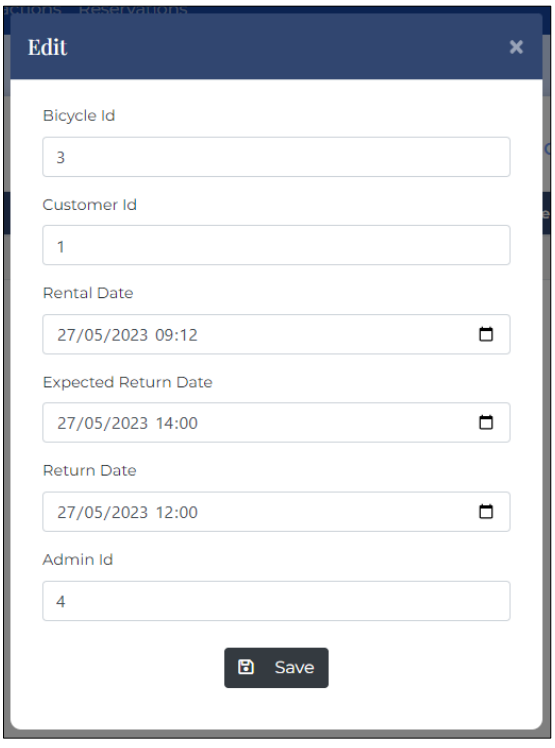


Figure 112 Transaction's Edit Popup

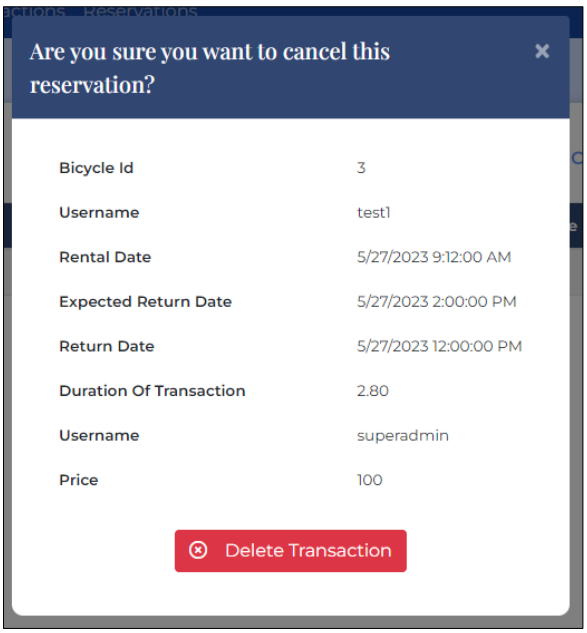


Figure 113 Transaction's Delete Popup

At the top right corner, three buttons are available. "New Rental", "Return Bicycle", and "Cash In".

- "New Rental" will trigger a popup for an immediate transaction for a customer.

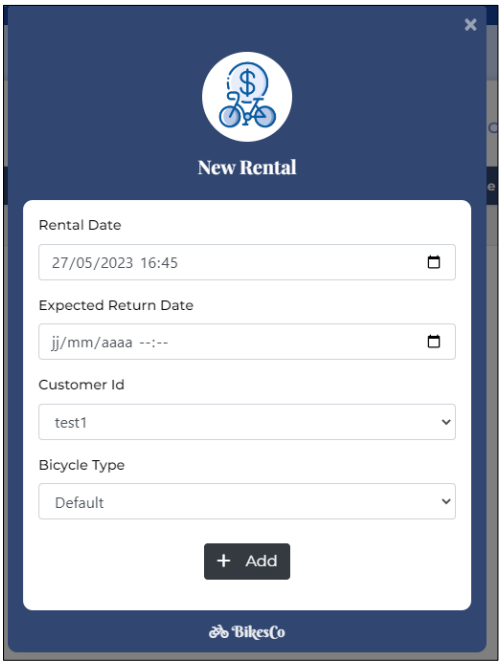


Figure 114 New Rental Popup

- "Cash In", will trigger a popup that asks the admin to enter the customer's username. The admin will be redirected to another page featuring a pre-filled form. Within this form, the

admin can input the “Amount Paid” field and proceed to submit the form. When the form is submitted, the admin will be redirected back to **Transactions**.

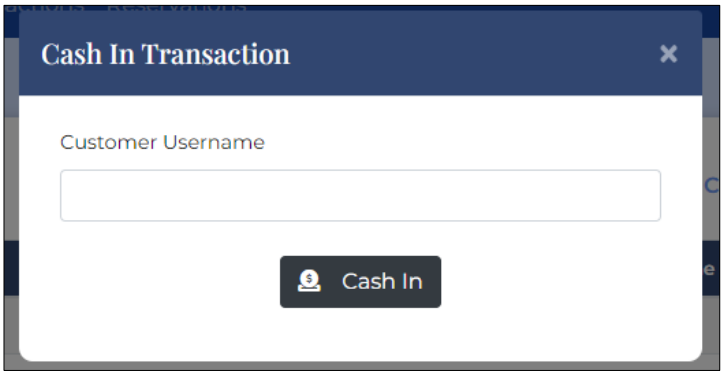


Figure 115 CashIn Popup

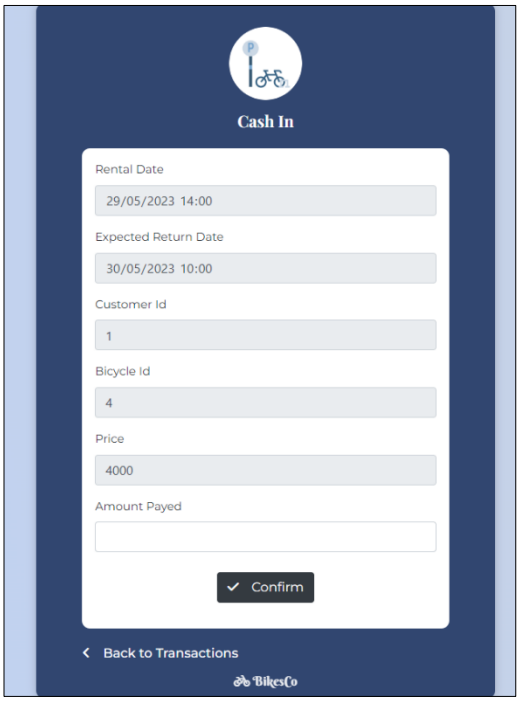


Figure 116 CashIn View

- “Return Bicycle”, similarly to “Cash In”, will trigger a popup that prompts the admin to enter the customer’s username who returned their bicycle. The admin is then redirected to another page where a form has been prefilled with essential transaction details. Such as the reservation date, customer’s username, expected return date, transaction price, and amount paid. The only field requiring input is the return date, which the admin is prompted to fill in. When the form is submitted, the admin is redirected back to **Transactions**, allowing them to view the updated transaction information.

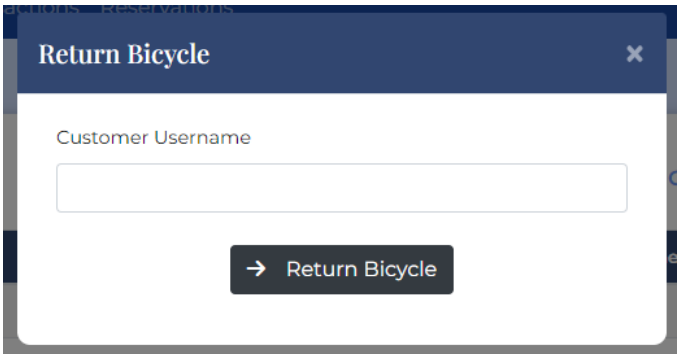


Figure 117 Return Bicycle Popup

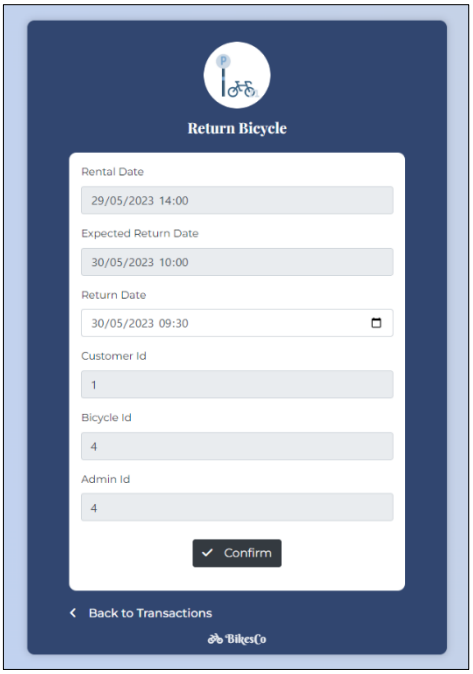


Figure 118 Return Bicycle View

At the bottom left corner, a "View Deleted Transactions" is available. This button serves the purpose of displaying a list of deleted transactions, styled in the same fashion as **Transactions.**

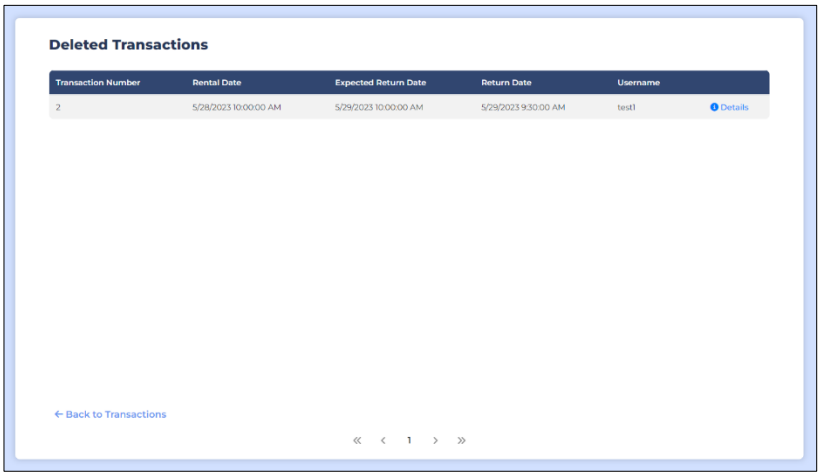


Figure 119 List of Deleted Transactions View

1.7.7. Coupons

The **Coupons** view simply provides the admin with a list of the available coupons that customers can obtain by redeeming their earned points (refer to **1.6.4.2. Coupons**). Each entry displays the coupon's value, its validity duration, price in points, and includes a "Delete" button for removing the coupon from the list.

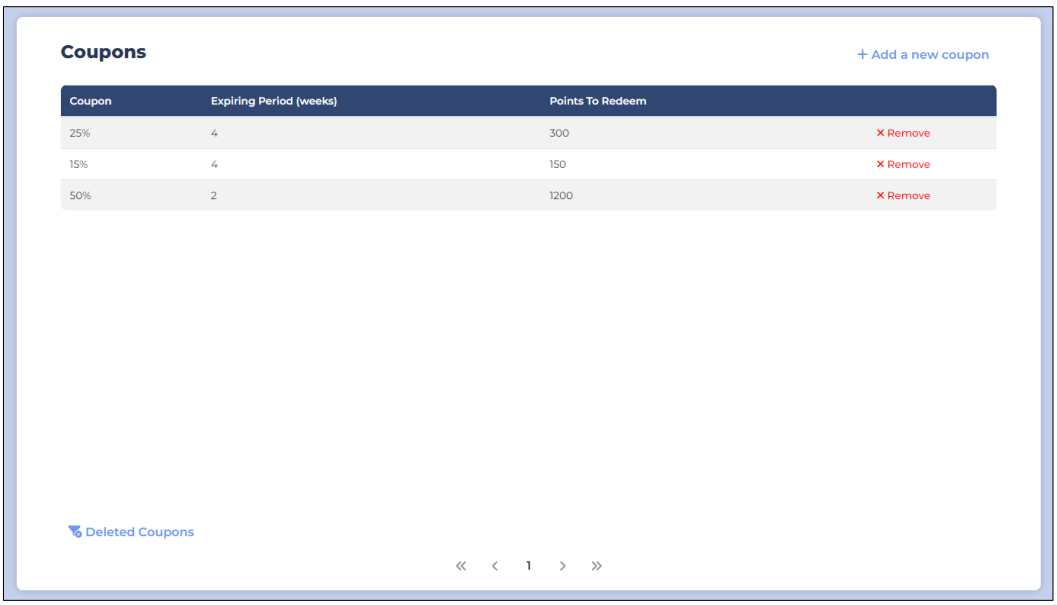


Figure 120 List of Coupons View

The "Add a new coupon" positioned in the top corner activates a popup that allows the admin to create a new coupon and add it the inventory.

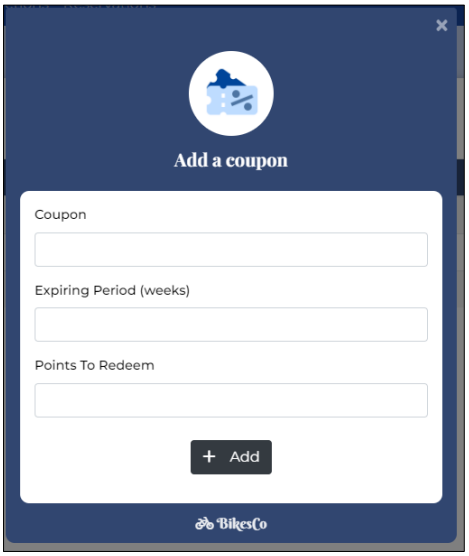


Figure 121 Add a Coupon Popup

As for the "Deleted Coupons" at the bottom left, its purpose to display the list of deleted coupons, along with a "Back To Coupons" button.

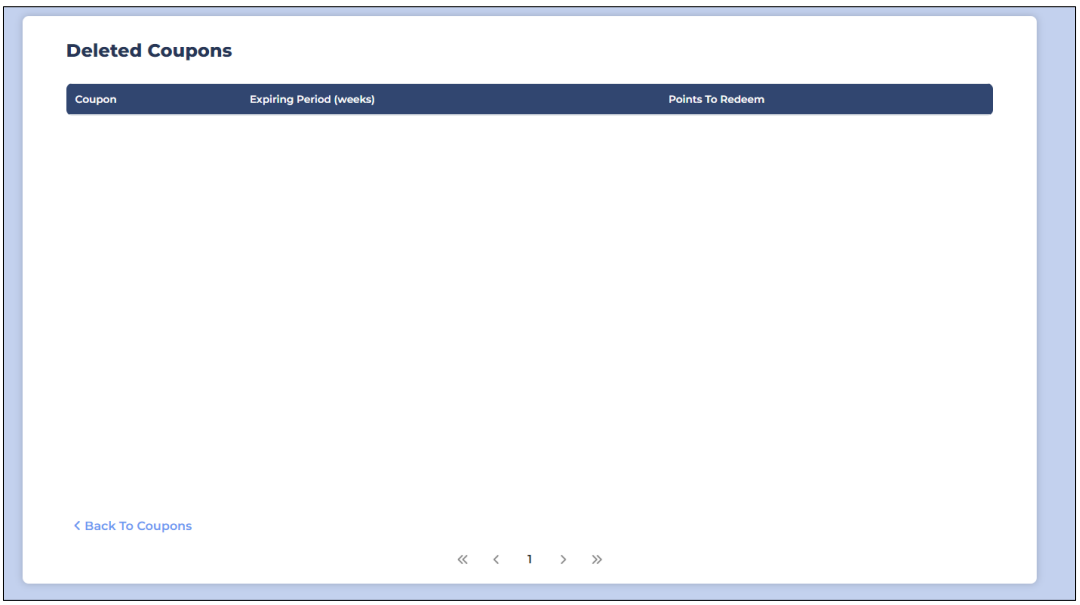


Figure 122 List of Deleted Coupons View

1.7.8. Plans

This view presents a list of plans, showcasing their respective name, price, and state (indicating whether they are active or disabled) in each entry. It also includes a "Details" button that activates a modal that provides the necessary information regarding the specific plan.

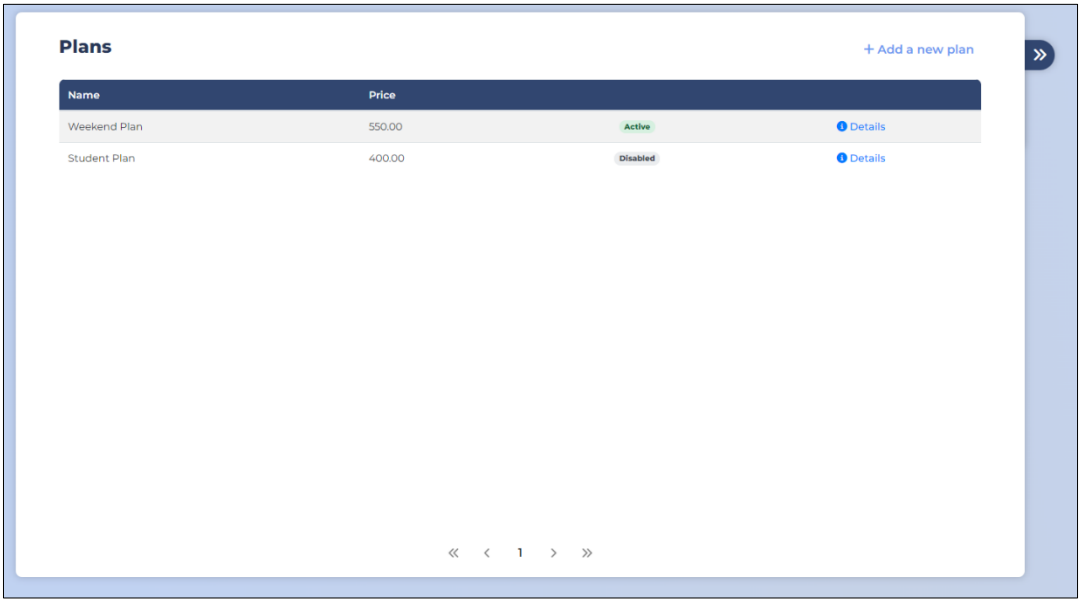


Figure 123 List of Plans View

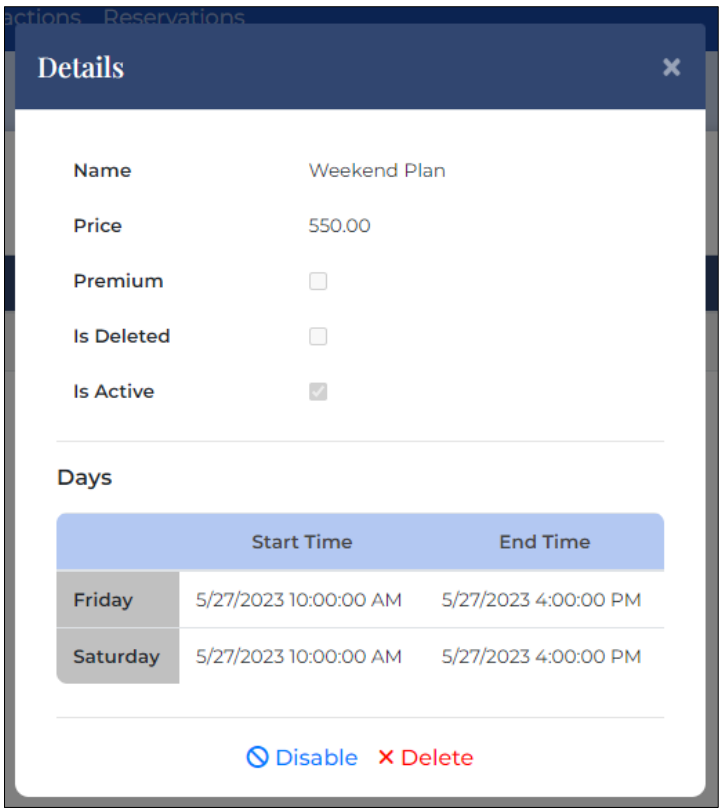


Figure 124 Admin's Plan Details Popup

In the Details modal, two buttons are situated at the bottom: "Disable" and "Delete". "Disable" serve the purpose of deactivating an active plan, rendering it inaccessible to customers in the **Plans** view. "Delete", on the other hand, removes the plans from the list.

If a plan is already disabled, which occurs when it is newly added to the list, the "Disable" button is replaced with the "Activate" button. This allows the admin to activate (or reactivate), enabling its accessibility in the **Plans** view.

Figure 125 Plan's Buttons

The state of each plan is also updated in the list.

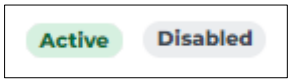


Figure 126 Plan's State

At the right top corner of the page, two buttons can be visible.

The "Add a new plan" button will redirect the admin into a new page where the admin is presented with a form. The form allows the admin to input the plan's name, price, days and hours, and whether the plan is classified as premium or not.

Figure 127 New Subscription Plan View

The time fields are only fillable if their corresponding day is checked first.

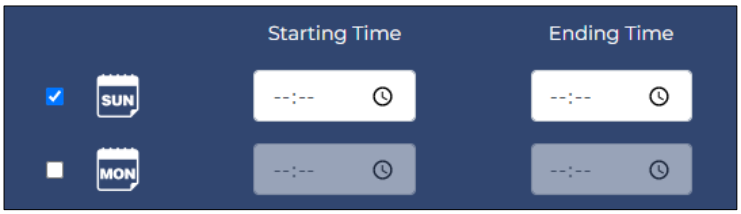


Figure 128 New Subscription Plan Time Fields

The arrow button functions as a sliding menu, the same one found in **1.6.4.2. Coupons**. It offers the admin the possibility to navigate between different lists, including **Plans**, **Inactive Plans**, and **Active Plans**. The **Active Plans** corresponds to the same view as the one found in **1.6.6. Plans**.

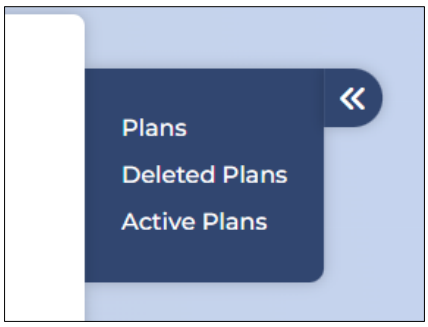


Figure 129 Plan's Sliding Menu

1.7.9. Contracts

The **Contracts** page showcases the list of currently active contracts. Each entry is provided with the customer's ID and bicycle type associated with the contract, and a "Details" button that triggers a popup which displays additional information.

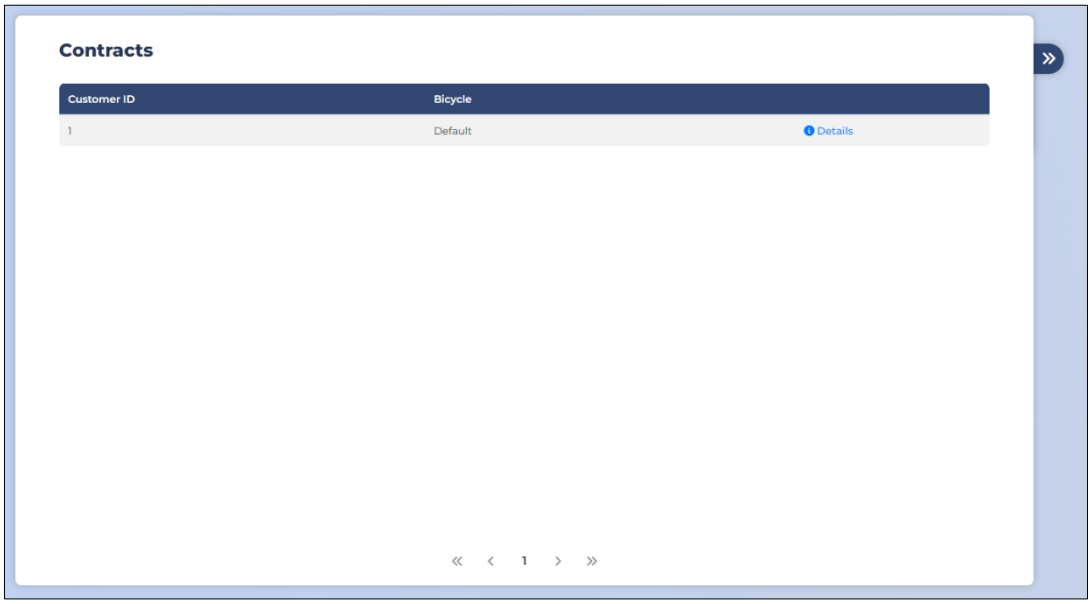


Figure 130 List of Contracts View

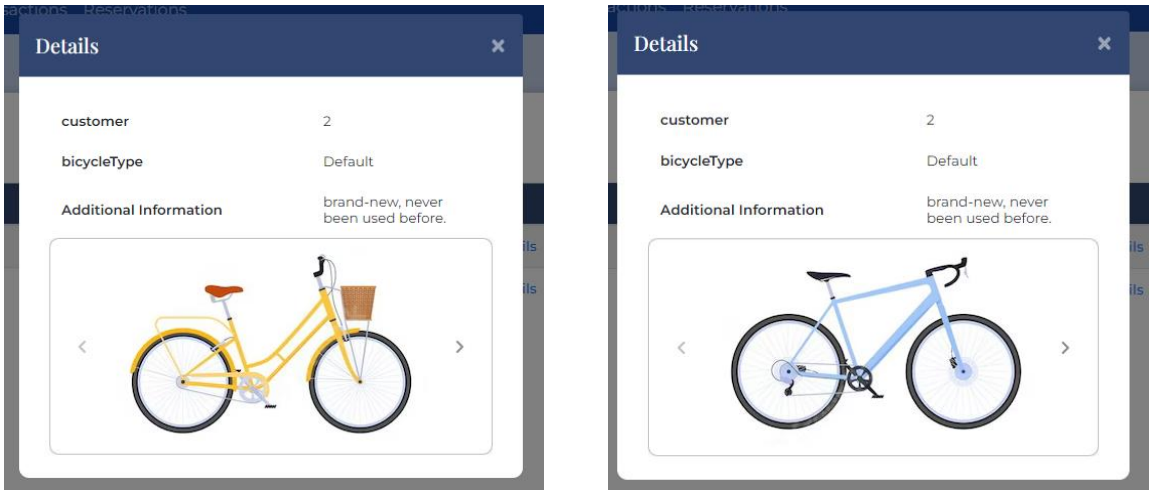


Figure 131 Contract's Details

This view includes the same sliding menu seen in the previous section.

Figure 132 Contracts Sliding Menu

The **Inactive Contracts** provides the admin a list of submitted contracts that are awaiting either confirmation or denial. The “Confirm” and “Deny” buttons, each triggering a popup, serve that purpose.

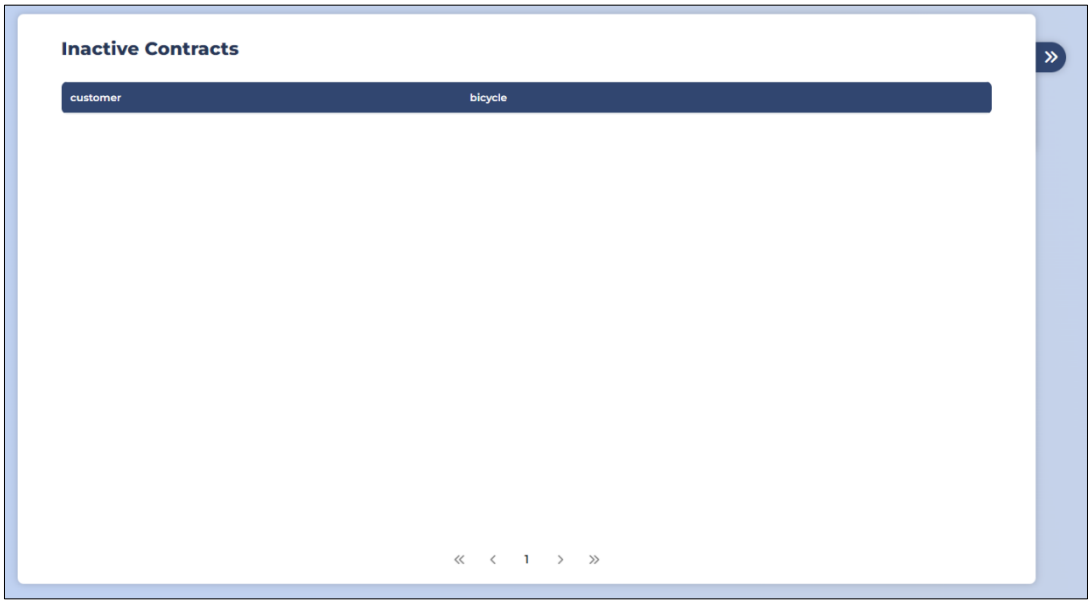


Figure 133 List of Inactive Contracts View

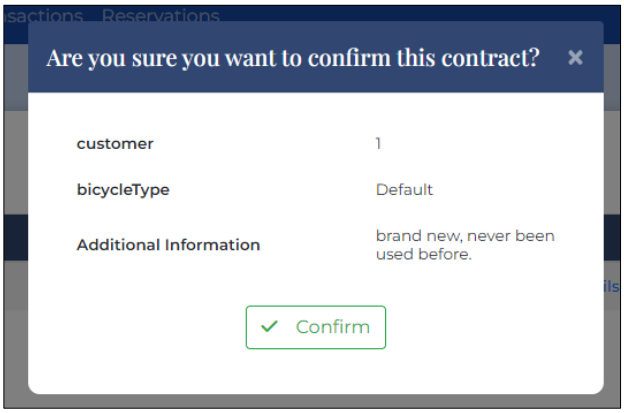


Figure 134 Contract’s Confirm Popup

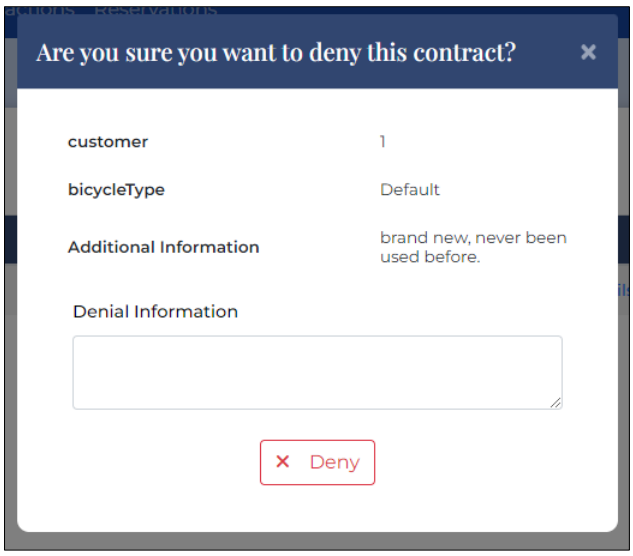


Figure 135 Contract's Delete Popup

The **Denied Contracts** view displays a list of contracts that have been denied.

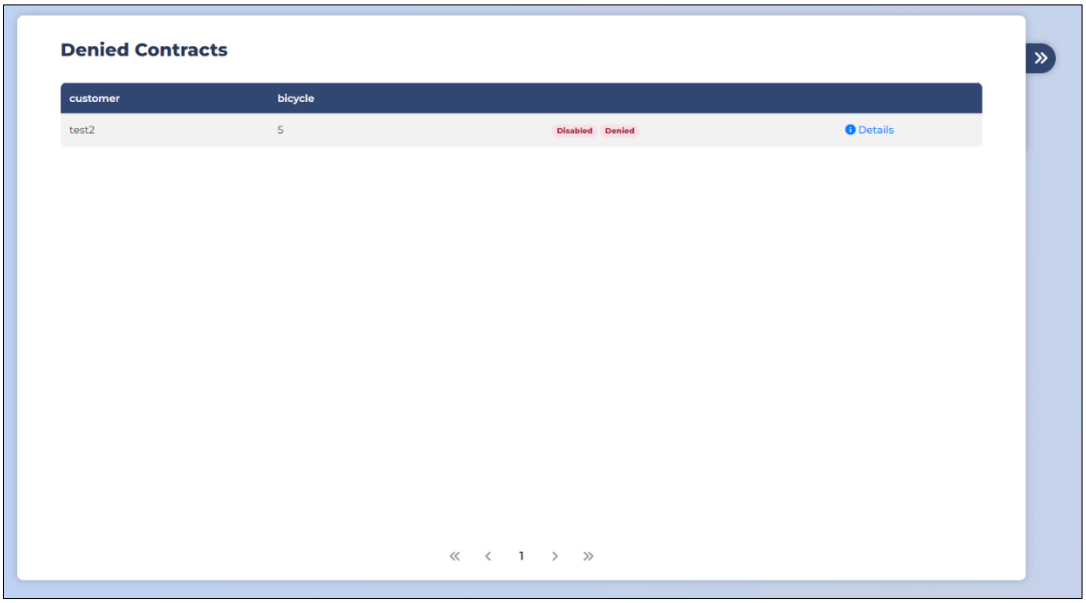


Figure 136 List of Denied Contracts

1.8. Terminal Admin

This view is specifically designed for the store terminal, catering to customers who visit the physical store location. The view offers two options: either confirming a reservation or renting a bicycle.

If a customer already has a reservation, they can click on the "Confirm Reservation" button. They are then prompted to log in and are directed to their reservations for the day. They can proceed with the confirmation process.

Alternatively, if a customer does not have a reservation but wishes to rent a bicycle, they can click on the “Rent a Bicycle” button. This action also prompts them to log in and the customer is provided with a form to fill with the necessary transaction details.

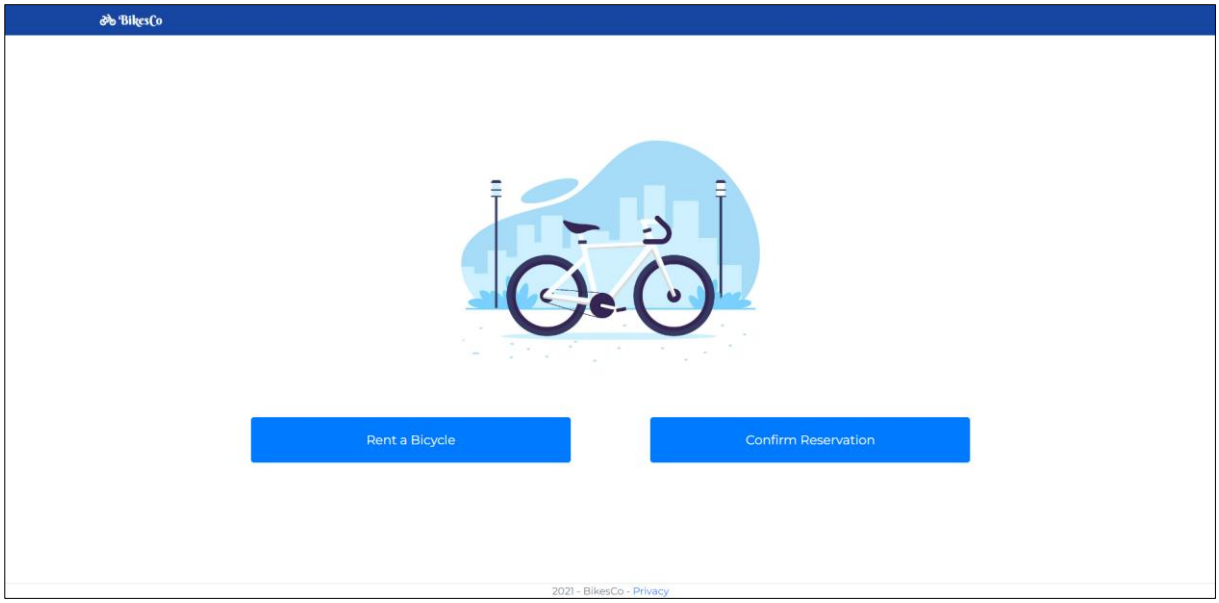


Figure 137 Store Terminal View

1.9. Mobile Responsiveness

Although our web application is originally designed to be used on desktop devices, we have also taken into consideration the importance of ensuring its responsiveness on mobile devices. As a result, users can seamlessly access and navigate our application on their preferred device, enhancing their overall experience.

The following figures showcase a selection of views as they appear on a mobile device:

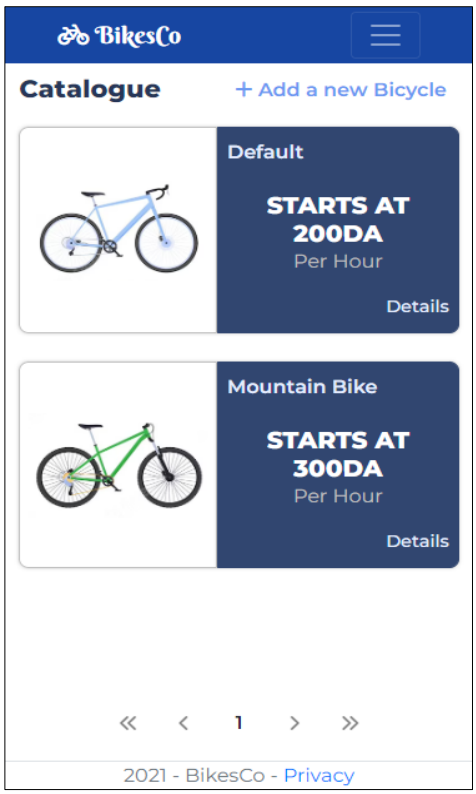


Figure 138 Catalogue Mobile View

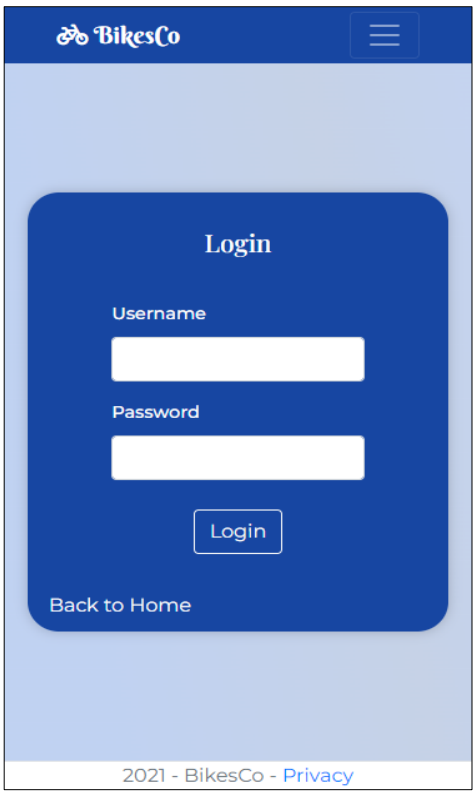


Figure 139 Login Mobile View

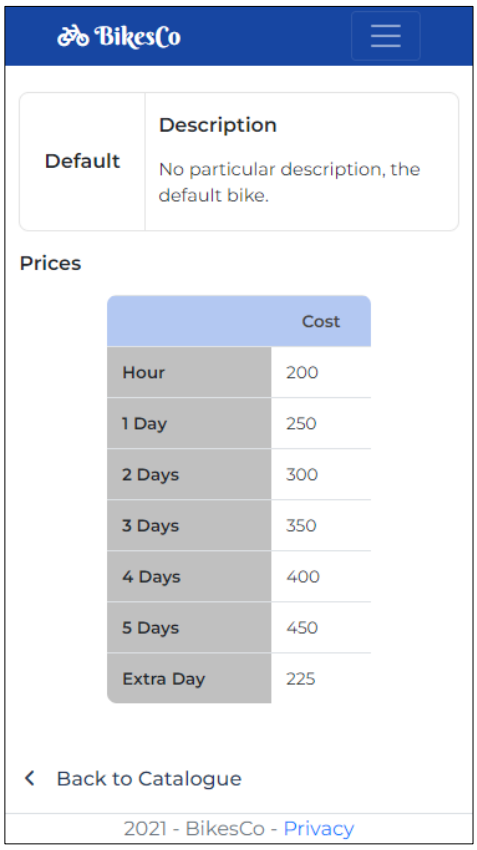


Figure 140 Bicycle Type Details Mobile View

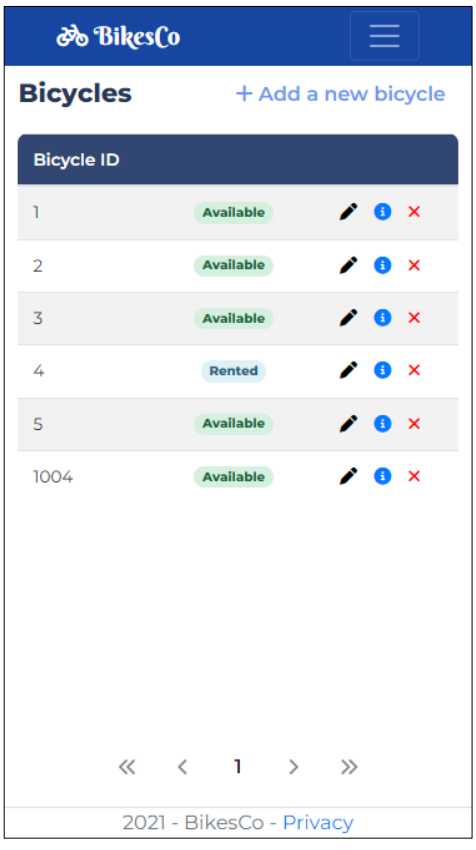


Figure 141 Bicycles List Mobile View

2. GPS Tracker Results

Before implementing the complete system, we test the functionality of each subsystem and their different operations.

2.1. GPS Output

After building the microcontroller-based system and implementing the software design presented in the previous chapter, we can test and fine tune the designs until we reach satisfactory results. The initial results of the GPS tracker were the following:

```
Date: 27/5/2023 , Time: 10:18:14 , Latitude: 36.76 , Longitude: 3.46
Date: 27/5/2023 , Time: 10:18:52 , Latitude: 36.76 , Longitude: 3.46
Date: 27/5/2023 , Time: 10:18:52 , Latitude: 36.76 , Longitude: 3.46
```

Figure 142 GPS Output Logs

We noticed that the GPS data is too inaccurate for tracking displacement. Having only two digits after the decimal point is too slim to register any change in position.

We figured that using a higher resolution floating point would resolve the issue. The results were as expected, but this will cause a problem when implementing the entire system.

```
Date: 27/5/2023 , Time: 10:18:56 , Latitude: 36.760590 , Longitude: 3.467131
Date: 27/5/2023 , Time: 10:18:59 , Latitude: 36.760590 , Longitude: 3.467131
Date: 27/5/2023 , Time: 10:19:17 , Latitude: 36.760532 , Longitude: 3.467316
Date: 27/5/2023 , Time: 10:19:17 , Latitude: 36.760532 , Longitude: 3.467316
Date: 27/5/2023 , Time: 10:19:21 , Latitude: 36.760719 , Longitude: 3.467132
Date: 27/5/2023 , Time: 10:19:21 , Latitude: 36.760719 , Longitude: 3.467132
Date: 27/5/2023 , Time: 10:19:36 , Latitude: 36.760788 , Longitude: 3.466979
Date: 27/5/2023 , Time: 10:19:36 , Latitude: 36.760788 , Longitude: 3.466979
Date: 27/5/2023 , Time: 10:19:40 , Latitude: 36.760777 , Longitude: 3.466984
Date: 27/5/2023 , Time: 10:19:59 , Latitude: 36.760628 , Longitude: 3.467227
Date: 27/5/2023 , Time: 10:20:07 , Latitude: 36.760628 , Longitude: 3.467227
Date: 27/5/2023 , Time: 10:20:09 , Latitude: 36.760628 , Longitude: 3.467227
Date: 27/5/2023 , Time: 10:20:10 , Latitude: 36.760628 , Longitude: 3.467227
Date: 27/5/2023 , Time: 10:20:12 , Latitude: 36.760628 , Longitude: 3.467227
Date: 27/5/2023 , Time: 10:20:13 , Latitude: 36.760628 , Longitude: 3.467227
```

Figure 143 GPS Output Logs with higher precision

Having six digits increases the accuracy of the readings from 50 meters to 10-5 meters. The tradeoff resides in the size of the data since we have larger floating points to process.

2.2. GPRS Output

2.2.1. Read Data

In order to start the tracking, the GPRS should be able to read the control fields (active, lastTransaction) from Firebase Realtime Database.

The following is the log output of the controller's serial port:

```
initializing serial GPRS
initializing GPRS
AT
OK

AT+CPIN?
+CPIN: READY
AT+CFUN=1
OK

AT+CMEE=2
OK

AT+CBATCHK=1
OK

AT+CREG?
+CREG: 0,

connecting GPRS

AT+SAPBR=3,1,"Contype","GPRS"
OK

AT+SAPBR=3,1,"APN",internet
OK

AT+SAPBR=3,1,"USER",ooredoo
OK

AT+SAPBR=2,1
+SAPBR: 1,1,"10.164.111.103"
OK

AT+CGATT?
+CGATT: 1

connected

AT+HTTPINIT
OK

AT+HTTPSSL=1
OK

AT+HTTPPARA="URL","https://rtdbgpstracker-default-rtdb.europe-
west1.firebasedatabase.app/1001/active.json"
OK

AT+HTTPACTION=0
OK

+HTTPACTION: 0,200,3
AT+HTTPREAD=0,50

+HTTPREAD: 3
```

```
Is Active: "1" failure counter: 19

AT+HTTPINIT
OK

AT+HTTPSSL=1
OK

AT+HTTPPARA="URL","https://rtdbgpstracker-default-rtdb.europe-
west1.firebasedatabase.app /1001/lastTransaction.json"
OK

AT+HTTPACTION=0
OK

+HTTPACTION: 0,200,6
AT+HTTPREAD=0,50

AT+HTTPTERM
OK
Last Transaction Id: 3006
```

Figure 144 GPRS Reading Data Output Logs

The first section (from “AT” to “connected”) is the initialization process of the GPRS module where we properly configure the peripheral and connect it to the mobile network.

The second section (from “AT+HTTPINIT” to “Is Active: "1" failure counter: 19”) is the fetching process of the “active” field. We initiate the HTTP connection and enable SSL for an HTPPS communication. We then provide the URL path of the wanted field and finally send the HTTP GET request. We wait for a response, and we notice “+HTTPACTION: 0,200,3”. Http Action 200 is the code for successful operation. 3 stands for the length of the data received.

The third section (from “AT+HTTPINIT” to “Last Transaction Id: 3006”) is the fetching process of the “lastTransaction” field. The operation is similar to the previous fetch.

Errors might occur at “AT+HTTPACTION=0” where the response would be different from 200. The list of possible responses is available in the AT Commands Manual. We simply resend the request until we receive a successful response.

2.2.2. Write Data

Similar to reading data from the database, we first initiate a secure HTTP connection. We then provide the URL path which describes the location where the data should be stored. After that, we send the actual data to be stored. Finally, we send an HTTP POST request to Firebase Realtime Database and wait for a successful response.

The following displays the serial port’s output:

```
initializing GPRS
AT+HTTPINIT

OK
AT+HTTPSSL=1
```

```
OK
AT+HTTPPARA="CID",1

AT+HTTPPARA="URL"," https://rtdbgpstracker-default-rtdb.europe-
west1.firebasedatabase.app /1001/3006.json"

OK
AT+HTTPPARA="REDIR",
AT+HTTPPARA="CONTENT
"application/json"

AT+HTTPDATA=70,10000

DOWNLOAD

OK

+HTTPACTION: 1,200,31

AT+HTTPREAD

+HTTPR
AT+HTTPTERM

OK
```

Figure 145 GPRS Writing Data Output Logs

This operation might fail because of various factors, such as unstable network connection, server overload, invalid data type and controller memory overload.

The first reasons are out of our reach and cannot do much to rectify them. We can, in the other hand, properly format the GPS data to fit the request and avoid overflowing the microcontroller's buffers. We also have to optimize both the data and the program code to safely execute the process.

2.3. System Results

The results stored in the non-relational database are shown in the following figure.

Figure 146 Firebase Realtime Database Stored GPS Data

After being successfully stored, these locations are retrieved by the web application and conveniently displayed on a map in the Details page as shown previously.

Conclusion

In this final chapter, we have showcased the results reached by our project. We have seen the finalized views of our web application, from the various simple forms to the complex maps and overlaying procedures. We have presented the outputs of our GPS tracker and discussed the key factors that directed our design.

General Conclusion and Future Works

Throughout this report, we have presented the work we invested in creating a bicycle rental system with GPS tracking. We initially introduced web applications and their various aspects. We realized that it is a rich field with thriving technologies and various frameworks that assist both novice and veteran developers. We chose to build our application under the Microsoft development stack, with ASP.NET Core and the vast ecosystem built around it. The enthusiastic community behind this stack and the documentation provided by Microsoft proved to be efficient and helpful at multiple points of the development process.

We agreed to implement the GPS tracker using a microcontroller-based system. The Arduino Uno was chosen for its cheap price and wide range of applications. We finally selected the NEO6MV2 and the SIM800L to allow GPS reading and GPRS communication. When we talk about low cost, we generally imply lower accuracies, limited functionalities and slower devices. Our case is no stranger to this phenomenon. The proposed design had to always maintain a balance between the functionality to implement and the limited hardware (speed and memory). Using higher-end elements and devices offers more functionalities, safer protocols and an overall better experience.

As a conclusion to our work, we can claim that we have succeeded in realizing the objectives set at the beginning of this project. We have successfully built a web-based application that handles bicycle rentals which communicates with a GPS tracking device. This system is ready to be tested and verified in order to be introduced to the interested public and upgraded as needed, since we have polished our work in a way to keep it functional and maintainable for future improvements and updates.

The entirety of the work and any future updates will be available in the following GitHub repository: https://github.com/DeltaPhoenix35/BikesCorp.

Developing an application might seem an endless task, more functionalities can always be added and users dissatisfied. For this, we propose the following future works that can be implemented:

- Expanding the scope of the application to handle more than a mere shop, and manage multiple stations.
- Manage multiple vehicle types at the same time.
- The ability to rent accessories with the bicycle, such as water bottles, helmets, gloves, protection gear …etc.
- Add the possibility of e-payments with credit card or Baridi-Mob.

- Implement the maps to follow the bicycles in real time.
- With a better microcontroller, implement a more reliable communication between the tracker and Firebase Realtime Database.
- Implement a smart lock which unlocks the moment a transaction is cashed-in.

A famous saying states "*An application with no updates is a dead application"*. The real challenge behind software development starts after the first deployment of the application. Developers must always listen to user reviews and bug reports in order to maintain their application "alive", functional and up to date.

References

[1] https://www.stackpath.com/edge-academy/what-is-a-web-application/ [Accessed March 14, 2023]

[2] https://www.indeed.com/career-advice/career-development/what-is-web-application [Accessed March 14, 2023]

[3] https://lvivity.com/single-page-app-vs-multi-page-app [Accessed March 14, 2023]

[4] https://hackr.io/blog/what-is-frameworks [Accessed March 14, 2023]

[5] https://www.heavy.ai/technical-glossary/client-server [Accessed March 15, 2023]

[6] https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/#:~:text=HTTPS%20is%20HTTP%20with%20encryption,far%20more%20secure%20than%20HTTP. [Accessed March 15, 2023]

[7] https://doc.oroinc.com/api/http-methods/#:~:text=The%20primary%20or%20most%20commonly,or%20CRUD)%20operations%2C%20respectively. [Accessed March 15, 2023]

[8] https://www.heavy.ai/technical-glossary/embedded-systems [Accessed March 16, 2023]

[9] https://www.tutorialspoint.com/microprocessor/microcontrollers_overview.htm [Accessed March 16, 2023]

[10] https://www.techtarget.com/whatis/definition/GPS-tracking [Accessed March 16, 2023]

[11] https://developer.mozilla.org/en-US/docs/Glossary/MVC [Accessed March 16, 2023]

[12] https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0 [Accessed March 16, 2023]

[13] https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/dependency-injection?view=aspnetcore-5.0 [Accessed March 16, 2023]

[14] https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-5.0#endpoints [Accessed May 10, 2023]

[15] https://www.oracle.com/database/what-is-database/ [Accessed March 19, 2023]

[16] https://www.techtarget.com/searchdatamanagement/definition/database [Accessed March 19, 2023]

[17] https://techmonitor.ai/what-is/what-is-sql-server [Accessed March 19, 2023]

[18] **Entity Framework Core IN ACTION** by Jon P Smith, First ed. July 2018, Page 3.

[19] **Entity Framework Core IN ACTION** by Jon P Smith, First ed. July 2018 Table 1.1

[20] https://firebase.google.com/docs/database [Accessed March 19, 2023]

[21] https://github.com/ziyasal/FireSharp [Accessed March 19, 2023]

[22] https://en.wikipedia.org/wiki/Markup_language [Accessed May 15, 2023]

[23] https://www.techtarget.com/whatis/definition/bootstrap [Accessed May 21, 2023]

[24] https://leafletjs.com/ [Accessed May 21, 2023]

[25] https://www.syncfusion.com/faq/blazor/general/what-is-the-difference-between-blazor-and-razor [Accessed March 19, 2023]

[26] https://www.arduino.cc/reference/en/language/functions/communication/serial/ [Accessed March 27, 2023]

[27] https://components101.com/modules/neo-6mv2-gps-module [Accessed March 27, 2023]

[28] “SIM800L Hardware Design”, version 1.00. Release Date 2013-08-20. Page 11.

[29] https://www.elprocus.com/at-commands-tutorial/ [Accessed March 27, 2023]

[30] “SIM800 Series AT Command Manual”, version 1.09. Release Date 2015-08-03. Page 22.

[31] https://learn.saylor.org/mod/book/view.php?id=32962&chapterid=12787 [Accessed April 1, 2023]

[32] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ [Accessed April 1, 2023]

[33] https://www.visual-paradigm.com/VPGallery/datamodeling/EntityRelationshipDiagram.html#:~:text=An%20entity%20relationship%20diagram%20(ERD,a%20reply%20or%20a%20transaction. [Accessed April 1, 2023]

[34] https://www.javatpoint.com/uml-use-case-diagram [Accessed April 12, 2023]

[35] https://www.ibm.com/docs/en/rational-soft-arch/9.5?topic=diagrams-relationships-in-use-case [Accessed April 12, 2023]

[36] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ [Accessed April 27, 2023]

[37] https://procodeguide.com/csharp/entity-framework-code-first/#:~:text=The%20entity%20framework%20code%20first%20or%20migrations%20is%20the%20most,database%20design%20is%20already%20available. [Accessed April 24, 2023]

[38] https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli [Accessed April 24, 2023]