

**Undergraduate Final Year Project**

**Motorcycle Rental Management System using Spring Boot and ReactJS**

**NGUYEN PHUC TAI**

**001199358**

**Bachelor of Science with Honours in Computing**

Table of Contents

[**Chapter 1: Introduction** 1](#_Toc133269183)

[1.1. Introduction about the project subject 1](#_Toc133269184)

[1.2. Project objectives 2](#_Toc133269185)

[1.3. Project plan 2](#_Toc133269186)

[1.4. Project outcomes 5](#_Toc133269187)

[1.5. Project evaluation 5](#_Toc133269188)

[**Chapter 2: Literature Review** 7](#_Toc133269189)

[2.1. Web application 7](#_Toc133269190)

[2.2. Interaction Design 8](#_Toc133269191)

[2.3. HTTP protocol 10](#_Toc133269192)

[2.4. MVC model 11](#_Toc133269193)

[2.5. RESTful API 12](#_Toc133269194)

[2.6. Version Control 13](#_Toc133269195)

[**Chapter 3: Technology and Tools** 14](#_Toc133269196)

[3.1. Spring Boot 14](#_Toc133269197)

[3.2. ReactJS 18](#_Toc133269198)

[3.3. MySQL 23](#_Toc133269199)

[3.4. SCSS 25](#_Toc133269200)

[3.5. Firebase 26](#_Toc133269201)

[3.6. Git 27](#_Toc133269202)

[3.7. IntelliJ 29](#_Toc133269203)

[3.8. Visual Studio Code 30](#_Toc133269204)

[3.9. Postman 31](#_Toc133269205)

[**Chapter 4: Software Product Requirements** 32](#_Toc133269206)

[4.1. Overview of other similar products 32](#_Toc133269207)

[4.2. User Stories – Product Backlog 33](#_Toc133269208)

[4.3. Use Case Diagram 38](#_Toc133269209)

[4.4. ERD 41](#_Toc133269210)

[4.5. Sitemap 43](#_Toc133269211)

[**Chapter 5: Review of Software Development Methodologies** 45](#_Toc133269212)

[5.1. Waterfall Model 45](#_Toc133269213)

[5.2. Spiral 47](#_Toc133269214)

[5.3. RAD (Prototyping) 49](#_Toc133269215)

[5.4. Agile 51](#_Toc133269216)

[5.5. Our selection of a software development methodologies and our justification 53](#_Toc133269217)

[**Chapter 6: Design and Implementation of our demo product** 54](#_Toc133269218)

[6.1. Product Analysis and Design 54](#_Toc133269219)

[6.2. Features include with screenshots 61](#_Toc133269220)

[6.3. Product Implementation 71](#_Toc133269221)

[6.4. Evaluation of our product (good/bad) 107](#_Toc133269222)

[**Chapter 7: Conclusions** 110](#_Toc133269223)

[7.1. What you have learned in this project? 110](#_Toc133269224)

[7.2. What is the result of this project? 111](#_Toc133269225)

[7.3. Further development of this project 111](#_Toc133269226)

[**REFERENCES** 114](#_Toc133269227)

**Chapter 1: Introduction**

**1.1. Introduction about the project subject**

Rental services for motorcycles are becoming more and more well-liked as a practical and economical option for people to travel or commute inside their city. In recent years, these services have been boosted by the introduction of Industry 4.0 technologies, which enable better data management and consumer experiences. Technology has completely changed how businesses manage their data. Manual paper-based data management is no longer used since it was time-consuming, prone to error, and ineffective. Organizations may now handle their data in a more streamlined and effective manner because to the advancement of digital technologies.

Businesses may store, process, and analyze data more quickly and correctly by using software applications, cloud-based storage solutions, and automation tools. As a result, they are able to make better decisions and perform better as a whole. With modern enterprises producing an increasing amount of data, manually managing it can rapidly become overwhelming. Adopting technology may help organizations stay organized, compliant, and competitive in an increasingly data-driven environment.

Vietnam is known for having one of the largest populations of motorbike users in the world, making motorbike rental services a thriving industry in many tourist destinations. However, it is common to find many of these motorbike rental service providers still relying on outdated and manual methods to manage their data. This can lead to several challenges, including inefficient processes, potential errors, and limited access to accurate information. By adopting modern technology solutions for data management, these businesses can streamline their operations and improve their customer service. With technology, rental providers can easily track their bikes, manage rental schedules, and automate tasks such as invoicing and customer communication. Additionally, it can help businesses to gain valuable insights through real-time data analytics, leading to better decision-making and a more competitive edge. Embracing technology is no longer a luxury but a necessity to remain relevant in today's rapidly evolving business landscape. Therefore, it is essential for motorbike rental service providers in Vietnam to adopt modern technology solutions to help manage their data effectively and remain competitive.

Based on the aforementioned factors, I made the decision to work on this project to create a website management system that enables small enterprises or individuals to control all data as well as the motorcycle rental procedure. ReactJS and Spring Boot, two well-known technologies that will be used in this project, will give me the opportunity to study and practice them.

**1.2. Project objectives**

Here are some possible objectives for a project to build a website data management system for motorbike rental field:

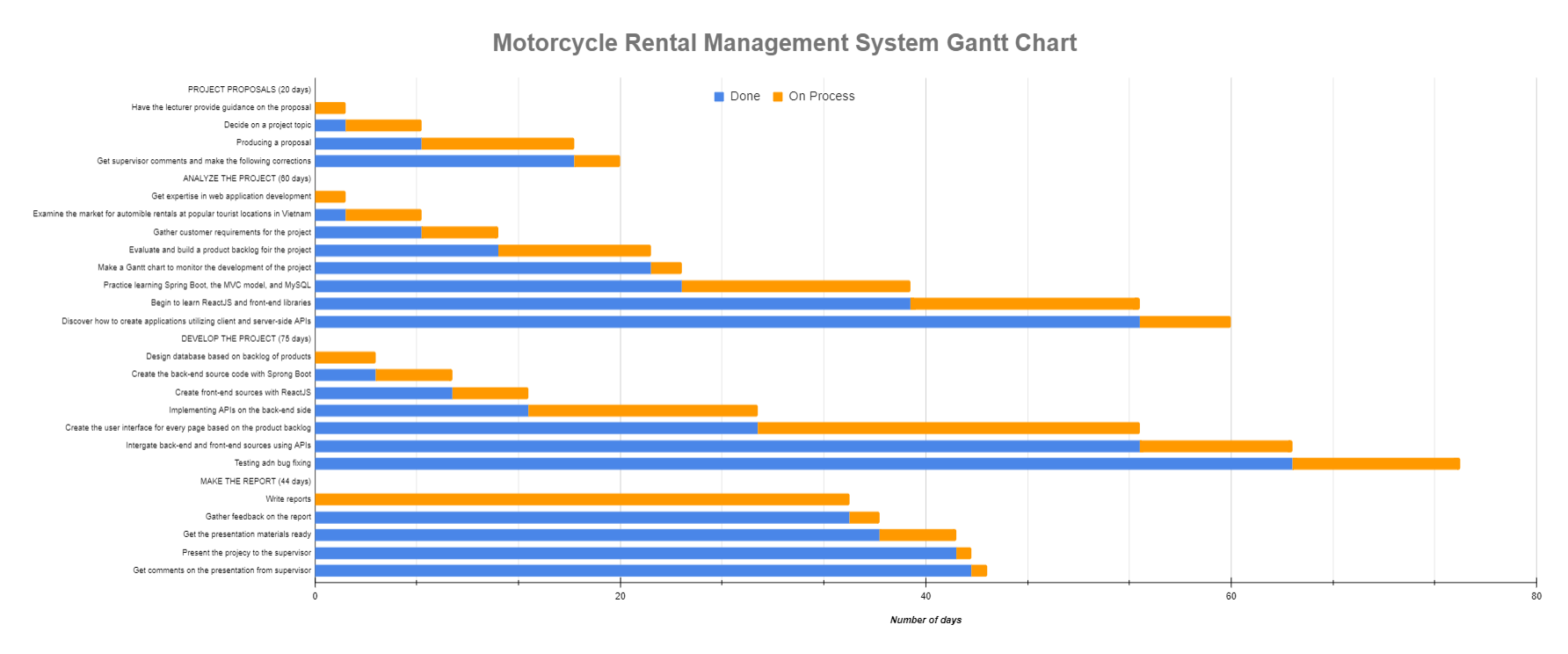
* Simplify the booking process: The website data management system should make it simple for customers (admin) to book motorcycles, check availability, and securely pay.
* Enhance customer experience: The website should have an easy-to-use interface and make personalized recommendations based on the customer's preferences and rental history.
* Ensure data security and compliance: The system should be designed with strong security measures to protect customer data (who rent motorcycle) also admin information.
* Increase data accuracy: By automating data entry and providing accurate reports on the dashboard, the software should reduce errors and improve data accuracy for admin.
* Enable real-time reporting: To assist admin in viewing dashboard, the application should provide real-time reporting and analytics.
* Scale the business: The system should be able to accommodate future growth and expansion by supporting multiple locations, integrating with other systems, and providing tools for marketing and customer engagement.

The objectives listed above are required for a project of mine to run smoothly. As we can see, these criteria can be used to assess the project when it is finished.

**1.3. Project plan**

Planning is an important step in any motorcycle rental project because it helps determine the key activities and resources needed to achieve the project's objectives. To properly plan this project, I used a Gantt Chart to represent each task. This is a tool that can show a manager a visual representation of the project's progress. A Gantt Chart is a timeline that outlines all of the tasks and activities involved in the project as well as their dependencies. During this planning phase, using a Gantt Chart is an effective way to manage a motorbike rental project by helping me monitor the time of each job to reduce the risks in the project.

For the aforementioned reason, I will show the Gantt Chart in this section, which lists the tasks that need to be accomplished during the Motorcycle Rental Management System project's development phase:

*Figure 1: Gantt Chart of Motorcycle Rental Management System*

We will have 4 primary phases for the main procedure, which will include:

**PROJECT PROPOSALS**: 20 days. My supervisor will help me choose a topic for the project at this point. After giving it some thought, I decided to write about creating a website application to handle automobile rentals. At this stage, I typically compose the proposal that the supervisor has requested for this topic.

**ANALYZE THE PROJECT**: Because I have a lot of business-related duties to complete, this may be considered one of the most essential phases of the project. The current tasks can be divided into the following three groups.

* The initial step is to gather customer requirements. I discovered a family that offers motorbike rentals in Nha Trang (This is one of the famous tourist destinations in Vietnam). Presently, they keep running the motorbike rental business by hand-calculating the money and writing on billboards and in their personal notebooks. After discussing, this family realized I was creating a website to assist them manage more effectively. They consented to assist and fulfill my customer's requirement. From there, I gained an understanding of the vehicle rental process as well as a list of needs for a rental management system.
* The second crucial step is to examine the requirements in order to create a project backlog. A project backlog is a prioritized list of the tasks, requirements, features, and improvements that must be finished in order to meet the objectives of the project. This data collection and analysis took me roughly 15 days in total.  In addition, I refer to other similar applications and compare them to the customers' requirements. We then filtered out the project's requirements as well as scenarios to complete the Project Backlog. Then, I created a Gantt Chart to display the project progress once I had a list of requirements and priorities. Looking at the Gantt Chart, we can easily see each task's timetable.
* The final task in this phase is to research and understand the technologies, processes, and associated expertise that will be required while developing a management system via a website. This is the stage when I will learn more about ReactJS and Spring Boot, as well as the process of developing a system from beginning to end. Because it is the learning period, the tasks are almost often very long, lasting more than a month. During this time, I largely studied on the internet. By the conclusion, I had learned the fundamentals of deploying a tiny application utilizing new technologies.

**DEVELOPE THE PROJECT**: The implementation phase is a critical stage in building software because it involves transforming the design and specifications into an actual functioning system. During this phase, the development team codes, integrates, and tests the software to ensure it meets the specified requirements and functions as intended. In the coding section, I'll go over the task:

* The first step is to create two source code projects: ReactJS (front-end) and Spring Boot (Back-end). I generated the two most basic sources in order to run a small project and configure other relational libraries. For Spring Boot, I focused on configuring security and connecting to the database (MySQL). In addition, I researched and added libraries for ReactJS that support interface creation, data validation, API connection, and so on.
* After ensuring that two sources were stable and capable of communicating with one another, I proceeded to complete each function on the product backlog. This is the most time-consuming part of the project. The following is the work order: First, I'll analyze the screen and create the necessary entities (tables in the database). When I have the entities, I'll start coding the back-end API to handle data processing for that website. Then I began designing the database-driven interface. For ReactJS, I also create small reusable components for web pages with similar functionality. After creating the interface, I proceed to integrate the data from the back-end and front-end.
* The final step is testing and debugging. When a function or a page is finished, I will test it in a variety of scenarios. When an issue arises, I will resolve it immediately and ensure that it does not occur again.

**WRITE REPORT**: This is the final stage after the product has been completed. I spend most of my time completing the report requested by the supervisor. When I finish the report, I'll start working on the presentation material. As a result, the project came to a successful conclusion.

**1.4. Project outcomes**

Certain criteria are required for every project. Based on those criteria, we can gradually improve the project. Following that, I will share the outcomes of the Motorcycle Rental Management System project.

* Completed product: The successful delivery of a product to users is the most important outcome of a software development project. My customer will receive a product with all of the basic functions for managing motorbike rentals at the end of this project.
* Improve my knowledge: When this project is finished, I will have a better understanding of new frameworks such as ReactJS and Spring Boot. Aside from technical knowledge, I will also learn how to manage projects, collect and analyze requirements, create detailed plans, draw charts, use project management tools, and so on. This project, it can be said, has created a favorable environment for me to develop myself in all aspects of information technology.
* Product quality: Products must satisfy customers in terms of user experience. Furthermore, every function must be error-free and accurate. It is bad if the product contains numerous errors while in use; it is an extremely unpleasant experience. More than that, I want the software to be simple to use and accessible to all users after just one tutorial. Furthermore, I will design the application so that it can be accessed by customers on any platform, such as phones or computers. With so many functions and criteria, each function must be completed with care and polish.
* The best user interface: I needed to make sure the interface was vibrant and colored to match the tones provided by the customer because it was built and handcrafted by me. To tailor the interface to the user's satisfaction, I will need to select a large number of theme templates for the "Motorcycle rental" theme. I am aware that my self-created interface will not look as good as the templates available online, so I must carefully study the layouts, colors, details, and images to perfect them.
* Documentation and Presentation: At the end of this project, I must complete the report and proposal. To meet the supervisor's criteria, I must include all project-related information in the report section. In addition, I must prepare the presentation using the PowerPoint tool.

**1.5. Project evaluation**

In this section, I will go over the Motorcycle Rental Management System project in detail. I will divide the evaluation into two parts: Project development process and final product evaluation.

**Project development process evaluation:**

* Overall, the project schedule management process appears to have gone smoothly and efficiently. The first is about meeting project goals; it can be seen that all of the client's and my goals have been completely met. The customer received a fully functional product on time. In terms of new technologies, I've learned about ReactJS, Spring Boot, and related libraries. As a full-stack developer, I've gained enough knowledge to create a complete website.
* The second factor is customer and supervisor satisfaction. The supervisor determined that the product met all of the basic requirements for the system to function. Customers believe that the product works well, but the interface is unimpressive because the entire interface is built by hand and does not use any free templates.
* Third, I will assess the project management procedure. The project was completed on time thanks to a well-thought-out plan. Except for the development phase, every stage of the product development process runs smoothly. Because the customer made numerous change requests during the coding process, some tasks were delayed. Furthermore, because the project was completed concurrently with my learning new technology, fixing bugs took a long time. Despite the fact that there were numerous such issues during the development phase, the product was completed on time and all processes were free of major issues.
* I'll then go over risk management. As I previously stated, because I have a clear plan, I have covered the majority of the potential risks, from the timeline to the product, as well as other objective factors. There is only the risk of work progress being halted due to a change in the customer's requirements, but I have it under control.
* Finally, because the product is handmade by me, I only use Word to document the project's information. These doc files are stored on my laptop. I only keep the source code on git. This is also a constraint because if my laptop fails, I will lose all of my documents. On this point, I believe the document management is inadequate.

**Final product evaluation:**

* First, I'll assess the functionality: The website already has all of the necessary functions to work, but no advanced functions have been added. CRUD is at the core of all major functions (Create - Read - Update - Delete).
* Second, I will evaluate User Experience: After demoing the product to customers, they confirmed that every function is simple to use. However, a tutorial is required for those who are just getting started with it. The interface of the website is quite rudimentary and unprofessional. This UI will be temporarily acceptable if used for a household or a small business. However, if used for a large corporation, the interface must be significantly upgraded.
* The third point is about security: Because security is configured from Spring Boot and ReactJS uses decentralization, the system can be described as decentralized and extremely secure.
* Fourth, this website has been designed to run on both computers and phones using browser platforms such as Chrom, Coc Coc, Microsoft Edge, and others.
* The fifth important factor is performance: Because the application only contains basic functions and a small amount of data, the performance is excellent. The loading of images is currently the website's weak point because the image data is saved by firebase, so it is dependent on the response speed of firebase.
* The following section is about analytics: The system already includes analytics in the dashboard, which displays all of the information required by customers. The analysis is illustrated with colorful charts.
* Finally, there is maintenance and support: Because it is a small system with no advanced functionality, it is very simple to maintain and support the customer. I optimized the lines of code so that any developer could read and upgrade them.

**Chapter 2: Literature Review**

**2.1. Web application**

First, we'll go over the definition of a Web application:

A web application is a software application that runs on a web server and can be accessed via the internet using a web browser. With the benefit of being accessible from any location with an internet connection, it is made to offer functionality and a user interface that are comparable to those of a desktop application.

Web applications can be accessed through any web browser-enabled device, including desktop computers, laptop computers, tablets, and smartphones. They are frequently built to be highly scalable, allowing them to easily handle large numbers of users and data. Aside from that, we can build with a variety of programming languages and frameworks, and they typically rely on server-side technologies like PHP, Java, Ruby, or Python to manage backend logic and database interactions.

As a result, we can see that web applications are widely used in today's society, particularly during the 4.0 period. It enables businesses, organizations, and individuals to use the internet's power to deliver services and information to everyone quickly, efficiently, and easily. In this section, I will discuss some of the factors that distinguish web applications:

* Digital Transformation: With the introduction of 4.0 technology, an increasing number of businesses are looking to digitize their operations and processes in order to improve efficiency and competitiveness. Web applications can assist them in making this transition possible by providing tools and services that streamline workflows and improve communication between teams and stakeholders.
* Remote Work: Because of the COVID-19 pandemic, which has fueled the trend of remote working, web applications have become critical in enabling remote communication and collaboration. Employees can access the information they require using web applications from any location with an internet connection, making it easier to work from home or other remote locations.

Overall, web apps are an important part of today's digital 4.0 landscape. It provides businesses and organizations with the tools they need to remain competitive and agile in an ever-changing business environment. After discussing the current importance of web applications, we will look at how they work:

* To begin, the user will launch a web browser such as Chrome, Microsoft Edge, or Safari and enter the URL or click a link to the web application.
* The browser sends the request to the web hosting server.
* The request is received by the web server and forwarded to the appropriate application server.
* The application server processes the request and returns a response.
* The response is returned to the web server, which in turn returns it to the browser.
* The response is received by the browser, which then displays the web page to the user.

We now have a better understanding of the web application's workings as well as its significance. As a result, the Motorcycle Rental Management System project will create such a system to assist customers in managing the process and motorbike rental invoices. I will also create a source that is divided into two parts: the front-end, which receives requests from the browser, and the back-end, which processes the requests and sends responses to the other side.

**2.2. Interaction Design**

Interaction Design is the design of the interaction between the user and the product. Typically, when people discuss interaction design, they are referring to software products such as apps or websites.

Interaction Design is a critical component of developing a web application, so we must thoroughly understand it. It is the process of designing interactive digital products such as websites, mobile applications, and software interfaces with the goal of creating engaging and user-friendly experiences. Interaction designers create the look, feel, and functionality of digital products, which includes visual design, information architecture, navigation, and user flow. To create effective and engaging user experiences, they employ a variety of design tools and methods such as wireframe, prototyping, user testing, and design thinking.

Following that, I will discuss five dimensions of Interaction Design. They are more than just a useful concept to keep in mind when developing a digital product. We'll look at these parameters to get a full picture of how users interact with digital products and what interaction design entails.

* Word: The meaning of words is represented by this dimension. Words are extremely powerful in any field. They can assist users in quickly grasping the meaning and influencing them through word expressions. As a result, words must be familiar and easy to understand, communicated in a tone appropriate to the context, and used consistently throughout the product to convey information to the end user. In order to prevent misunderstandings among users when they use our product, we must carefully select vocabulary and proofread the language.
* Visual representations: This dimension discusses visual elements that the user interacts with, such as typography, diagrams, symbols, or images. These elements are just as effective as "Words" because they quickly convey meaning to the user. In certain situations, this improves the user experience. However, we must exercise control over the use of these elements. When using our software, avoid overloading to avoid confusing users.
* Physical objects or space: This dimension refers to the physical objects with which users interact while using the product. For example, using a mouse or touchpad on a laptop, a finger on a smartphone, and so on. Apart from physical objects, this dimension also refers to a type of physical space in which users interact with software. For example, users can use the laptop at work or at home. All of these factors have an impact on how people interact with products.
* Time: This is a unique factor because it is related to the amount of time the user spends interacting with the previous three dimensions. It entails creating designs for specific situations, such as the first interaction, repeated use, or when the user encounters an error. Furthermore, it is regarded as a criterion by which users can measure and evaluate their progress. Nobody wants a design that is responsive or takes a long time to manipulate. In today's world, time is synonymous with money, so optimizing and saving time is a top priority for businesses or corporations looking to improve their products.
* Behavior: This final dimension consists of actions, reactions, activities, and presentations that are adaptable and understandable to all users. They frequently include questions such as the following: How do users react to product actions? How do customers interact with the product? Developers must learn and design their own products based on societal trends to ensure that users, both new and old, can easily use their applications.

We now fully comprehend all interaction design concepts. The dimensions will be used in my Motorcycle Rental Management System project. With the first dimension, I will prioritize the use of basic English words so that users can understand the function and meaning of the website when they visit it. With the Visual representations element, I will consult some websites that provide bike rental services in order to select the appropriate fonts, colors, images, and charts for this project. Regarding the third factor, which is an objective factor, I can't completely control the user's space, so I built an application that can run on both laptops and phones to diversify users' needs so that they can use the product everywhere. In the fourth dimension, time, I will attempt to optimize the interaction time with the website, ensuring that all functions respond quickly enough to satisfy the basic user. Finally, with dimension behavior, I will listen to the user's request in conjunction with some other products to build appropriate and reasonable interactions for each function, ensuring user satisfaction.

**2.3. HTTP protocol**

In this section, I will discuss the protocol that was used in this project. Let's start with an explanation of what HTTP is:

* HTTP (Hypertext Transfer Protocol) is a standard protocol for sending data across the internet. It serves as the foundation for data communication on the World Wide Web.
* HTTP operates on a client-server model, in which the client sends a request message to the server, and the server responds with a message containing the requested content or an error message if the request cannot be fulfilled.
* HTTP is a stateless protocol, which means that each request is handled independently, with no knowledge of previous requests. Web applications use cookies or other methods to store user data between requests in order to maintain session state.
* HTTP is a basic protocol that allows data to be exchanged over the internet, allowing users to access and interact with web content from anywhere in the world.

After we've grasped the fundamentals of the HTTP protocol, we'll compare it to a more specialized version of HTTPS.

* HTTPS (Hypertext Transfer Protocol Secure) is a version of HTTP that employs encryption to ensure the privacy and security of data transmitted over the internet. It is commonly used for online transactions, e-commerce, and secure communications between web servers and clients.
* HTTPS secures data sent between a web server and a client by combining the SSL (Secure Sockets Layer) or TLS (Transport Layer Security) protocol and an encryption algorithm. When a user accesses a website using HTTPS, their browser establishes a secure connection with the web server, encrypting all data sent between them.

So, what's the distinction between HTTP and HTTPS? The primary distinction between HTTP (Hypertext Transfer Protocol) and HTTPS (Hypertext Transfer Protocol Secure) is that HTTPS encrypts data transmitted between the server and client using a secure socket layer (SSL) or transport layer security (TLS) protocol, whereas HTTP does not.

This means that when we use HTTPS to access a website, the data transmitted between our browser and the web server is encrypted, making it much more difficult for someone to intercept and read. This is critical for websites that handle sensitive data like credit card numbers, login credentials, and personal information.

In contrast, when we use HTTP to access a website, the data sent between our browser and the web server is sent in plain text and can be easily intercepted and read by anyone with network traffic access. HTTP is thus less secure than HTTPS.

Another distinction between HTTP and HTTPS is the port used. HTTP uses port 80, while HTTPS uses port 443. When we use HTTPS to access a website, our browser automatically establishes a secure connection with the web server via the SSL/TLS protocol, which encrypts the data being transmitted.

In general, the main distinction between HTTP and HTTPS is that HTTPS provides encryption and thus increased security when transmitting sensitive data over the internet. Due to the fact that the Motorcycle Rental Management System project is being used for a small household, we are currently only using HTTP protocol to transmit request and response between Client and Server. If the project grows in size, I will update using HTTPS.

**2.4. MVC model**

Knowing the MVC pattern is one of the basic elements of a web application product in the project. Model-View-Controller, or MVC, is a software development design pattern that divides an application's concerns into three interdependent parts: the model, the view, and the controller.

Diagram

Description automatically generated

*Figure 2: The MVC model*

* The Model represents the application's data and business logic. It is in charge of data retrieval, manipulation, and storage.
* The View represents the application's user interface. It is in charge of displaying data to the user and allowing the user to interact with the application.
* The Controller acts as a go-between for the Model and the View. It receives user input through the View, processes the user's requests, and updates the Model as needed. It also updates the View with any changes made to the Model.

The MVC pattern allows programmers to create code that is simpler to maintain, test, and modify by dividing an application's concerns into these three parts. I believe it is quite appropriate and provides numerous benefits when used in web applications. Here are a few reasons why the MVC model is important for web applications:

* Concerns Separation: The MVC pattern divides the concerns of the application into three distinct components, each with their own set of responsibilities.
* Greater Code Organization: The MVC pattern encourages developers to write organized and modular code to easily maintain, test, and modify code when the application's concerns are separated.
* Testing Simply: MVC makes it easier to test individual application components. Separating the Model from the View and Controller during testing can make the process easier and more efficient.
* User Experience Improved: By dividing the application's logic and presentation, the MVC pattern contributes to an improved user experience. Therefore, we can focus on making the user interface more user-friendly and responsive by keeping the application's business logic separate from its user interface.

At the Motorcycle Rental Management System project, I will use the MVC model with View as the Front-end source code (ReactJS) used to send requests to or receive responses from the Controller in the Back-end source. In addition, I have logically structured the Back-end source, including folders "Controller", "Entity", "Model", "Service", and "Specification" in accordance with the standard Spring Boot project structure. This structure follows the MVC pattern.

**2.5. RESTful API**

A RESTful API is a way to make web services that follow the rules of REST. RESTful APIs are easy to use, lightweight, and can be made bigger if needed. This is why they are a common choice for new web applications.

In a RESTful API, everything has a unique URL. We can use standard methods like GET, POST, PUT, and DELETE to work with things at those URLs. These methods are for getting, making, changing, or deleting resources.

RESTful APIs also use standard formats like JSON or XML to show data. This makes it easier for different apps and languages to work with the API in the same way.

Following that, we'll take a closer look at the RESTful API's components:

* API (Application Programming Interface): a set of rules and mechanisms that allow one application or component to interact with another. The API can return data for our application in common data types such as JSON or XML.
* REST (Representational State Transfer) is a data structure transformation technique and an architectural style for developing APIs. It facilitates machine-to-machine communication by using simple HTTP methods. Instead of using a URL to handle some user information, REST sends an HTTP request to a URL to process the data, such as GET, POST, DELETE, and so on.

For a web application, RESTful APIs become important for several reasons:

* First, because RESTful APIs are lightweight and scalable, they are ideal for developing large-scale web applications that must handle a high volume of requests.
* Second, RESTful APIs are adaptable and simple to integrate with current systems because they can be used with any platform or programming language.
* Third, RESTful APIs are modular in the sense that they can be easily divided into smaller, reusable components, making them simple to maintain and update over time.
* The next reason is interoperability. RESTful APIs use standard data formats or XML to represent data like JSON, allowing different applications and programming languages to interact with the API in a standardized manner.
* Finally, there is the security factor. To protect sensitive information and guarantee that only authorized users have access to the API, standard authentication and authorization mechanisms, such as OAuth or API keys, can be used to secure RESTful APIs.

In this project, I used RESTful API to send requests and responses between the back-end and front-end. Almost every function - API of this project revolves around two HTTP methods, POST and GET. Even with the update and delete functionality, I still use POST to simplify the process of creating and managing APIs. In accordance with current web application standards, I have set the data's sending format to JSON. In addition, I tested each API using a program called POSTMAN. When a user takes an action on the website, ReactJS (client side) encodes the data into JSON format and sends it to the Back-end at the Controller via the preconfigured APIs. The backend will receive this JSON data and encode it so that Spring Boot can read it and convert it to Objects in the Java source. This principle also applies when data processed on the server side is sent to the client side. As a result, in order for this project to work, I must always run two sources simultaneously. Each source has its own role and objective, demonstrating the project's genuine professionalism.

**2.6. Version Control**

Version control is the management of changes made to the source code, documentation, and other assets of a software project. It is an essential procedure in software development that enables developers to keep track of code modifications and work productively together on a project.

Version control also makes it easier for multiple developers working on the same project to collaborate. It enables developers to work on the same codebase without overwriting each other's changes, and it enables developers to merge changes made by different developers into a single coherent codebase.

There are numerous version control programs accessible, including Git, Subversion, and Mercurial. Branch management, tagging, and merging are just a few of the features offered by these systems for managing and recording changes to software projects.

In software development initiatives, version control is crucial for a number of reasons:

* First is collaboration: multiple developers can collaborate on the same codebase at once without running into negotiation thanks to version control. Developers can share their changes, combine them into a unified codebase, and keep track of who made which changes using the mechanism it offers.
* Second is history/backup; version control systems keep track of all project modifications, enabling developers to examine and contrast various codebase versions. By doing this, programmers can find bugs, go back to previous iterations, and retrieve lost or deleted code.
* Third is accountability; version control makes it simpler to pinpoint who introduced a bug or a particular feature by keeping track of the changes made over time by various workers. This team member encourages members to take ownership of their actions and assume responsibility.
* The next factor is experimental features. By using version control, developers can test out various approaches to an issue, make new branches to work on experimental features, and merge those branches back into the main codebase once they are complete.
* Finally, version control is an essential component of a continuous integration workflow, in which changes to the codebase are immediately tested and integrated into the primary codebase. This makes sure that the codebase is always in release-ready condition and helps to find bugs.

I also used Git, a tool that will be described in more detail later, to manage this project's front-end and back-end sources. I'll upload that version to Git each time I complete a function or resolve a bug. Because the project is solely mine, all source code is presently stored in a single branch, master. I know it's not ideal and doesn't resemble large outside initiatives, but it saves me time when controlling source code in Git. In the future, if the project scale is expanded with a large team, I suggest using other branches to guarantee the product is fully controlled.

**Chapter 3: Technology and Tools**

**3.1. Spring Boot**

**3.1.1. What is Spring Boot**

Spring Boot is an open-source Java framework that enables the creation of standalone, production-grade Spring-based applications with minimum configuration. It is built on top of the popular Spring Framework and seeks to simplify and accelerate development by providing default configurations, embedded servers, and a broad variety of starter dependencies.

Spring Boot makes it simple to build Spring-based apps that can be deployed as standalone executables or as micro services running on cloud platforms. Among its many features are auto-configuration, which does away with boilerplate code, production-ready metrics, health checks, and tracking, as well as a potent command-line interface for controlling application development and deployment.

Spring Boot is highly modular and works with a broad range of popular databases, web servers, and other third-party libraries, making it an excellent option for developing scalable and high-performance applications. Its emphasis on convention over configuration and ease of use has made it a popular option among developers worldwide.

Today, Spring Boot is gaining more and more traction in all projects, big or small, for Java programming. Spring Boot has emerged as one of the top frameworks for all Java programmers. Some of the characteristics listed below will demonstrate why Spring Boot has become so popular:

* Modular and Lightweight: Because the Spring Framework is highly modular and lightweight, it is easy to integrate with other frameworks and technologies.
* Inversion of Control (IoC): The Spring Framework follows the Inversion of Control (IoC) principle, which makes managing dependencies between objects and components simple.
* AOP (Aspect-Oriented Programming): The Spring Framework also supports Aspect-Oriented Programming (AOP), which allows for the separation of cross-cutting concerns and improves code modularity.
* Simplified Development: Data access, security, testing, and web development are just a few of the tools and features available in the Spring Framework to help us get started faster.
* Flexibility: The Spring Framework is extremely adaptable and supports a variety of programming styles, including procedural, object-oriented, and functional programming.
* Community Support: The Spring Framework has a sizable and active developer community that offers a wealth of resources such as documentation, tutorials, and support.

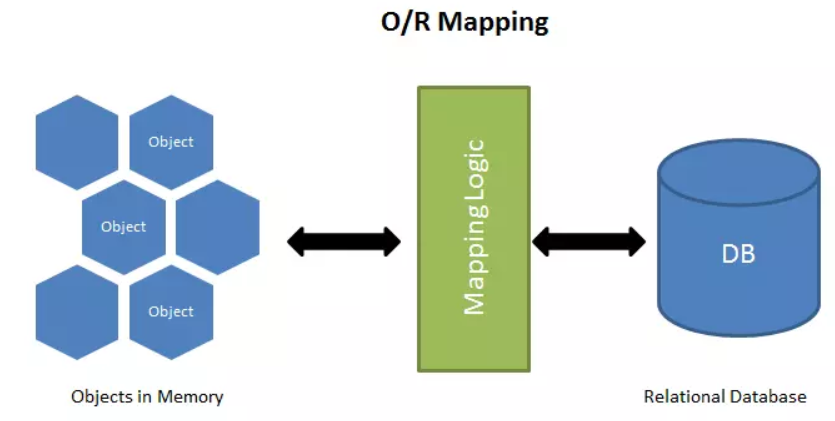
**3.1.2. Differentiate between Spring Boot and Spring Framework.**

Some individuals will mistakenly refer to Spring Boot as Spring framework when discussing it. Consequently, I'll discuss how these two ideas vary from one another. Spring Boot and Spring Framework are both Java-based frameworks created by the Spring community, but they differ in several ways. Some of the most significant variations are as follows:

* Configuration: The Spring Framework requires extensive configuration to set up an application, whereas Spring Boot uses annotations, properties files, and default configurations to simplify configuration.
* Opinionated and Flexible: Spring Boot has a stronger point of view than the Spring Framework. It provides a set of defaults and best practices that are suitable for the majority of projects, whereas the Spring Framework is more flexible and requires more configuration.
* Dependency Management: Spring Boot includes a dependency management system that simplifies dependency management, whereas the Spring Framework requires manual dependency management.
* Embedded Servers: Spring Boot includes embedded servers such as Tomcat, Jetty, and Undertow, making it simple to develop and test applications without requiring an external server. By default, the Spring Framework does not include an embedded server.
* Microservices: Because of its lightweight, modular architecture, Spring Boot is frequently used to develop microservices, whereas the Spring Framework is better suited to developing large-scale, monolithic applications.

**3.1.3. Introduce about ORM**

Currently, the project's back-end source code is built from a Spring Boot project with entities, controllers, dependencies, etc. The feature I like about Spring Boot is the use of the ORM (Object Relational Mapping) framework. This is the technique used to map objects to database tables and vice versa. ORM frameworks enable developers to work with databases using an object-oriented paradigm, making it easier to write and maintain database code. In addition, ORM also provides a set of APIs that allow developers to interact with databases through the use of object-oriented concepts. From there, it eliminates the need to write boilerplate code for database interactions, reducing the amount of code that developers need to write and maintain. Besides, ORM also improves performance by optimizing database queries and caching data.



*Figure 3: ORM (Object Relational Mapping)*

**3.1.4. Introduce about Spring Data JPA**

Spring Boot, in addition to ORM, provides the concept of Spring Data JPA, which has greatly aided me in implementing queries to extract data from the database. Spring Data JPA is a Spring Data project module that provides support for implementing JPA (Java Persistence API) repositories using the Spring Framework. Spring Data JPA in Spring Boot provides a quick and efficient way to work with databases by minimizing the amount of boilerplate code required to interact with the database.

JPA is a Java object-relational mapping (ORM) specification. It enables the mapping of Java objects to relational databases and vice versa. JPA is supported by a number of ORM frameworks, including Hibernate, EclipseLink, and OpenJPA.

Spring Data JPA extends JPA by providing a repository abstraction layer that simplifies data access logic implementation. It provides a set of interfaces and default implementations for defining a repository for a specific domain entity. Developers can define custom repository methods simply by adding method signatures to the repository interface. Spring Data JPA generates the required SQL queries based on the method signatures. Here are some of the project's great features:

* Automatic CRUD Operations: Spring Data JPA includes default implementations for common CRUD (Create, Read, Update, Delete) operations that can be used as-is or customized.
* Query Methods: Spring Data JPA enables developers to define repository methods using naming conventions, which can then be used to create complex queries without the need for SQL code.
* Sorting and pagination: Spring Data JPA includes pagination and sorting support for query results.

As we can see, the two factors mentioned above have greatly aided my project in handling logic in the Service layer and data extraction. Because the current Motorcycle Rental Management System project only has a few basic functions, the queries are all initialized with JPA statements. If new and more complex functions are added later, we will have the opportunity to expose more advanced knowledge in using this library.

**3.1.5. Introduce about Specification of Spring Boot**

In addition to utilizing the fundamental query phrases offered by Spring Data JPA, one of the skills I used for this project was specification. In Spring Boot, a specification is a set of rules or specifications that can be used to filter data from a database. JPA and Spring Data JPA frequently work together with specifications to generate dynamic queries that can be built at runtime based on user input.

Spring Data JPA provides a Specification interface through which developers can define a set of rules for data filtering. A specification defines a set of criteria that can be combined using logical operators such as AND and OR. Each criterion is defined by a Predicate, which can be thought of as a boolean expression that evaluates to true or false.

Developers can create their own Specification implementations by implementing the Specification interface and supplying the required Predicate expressions. They can then use these Specifications to construct queries dynamically at runtime.

Here are some reasons why we should use Specification in a Spring Boot project alongside Spring Data JPA:

* Dynamic Queries: Specifications enable developers to create dynamic queries based on user input at runtime. Developers can use Specifications to create queries that are more flexible and can handle a broader range of scenarios.
* Code Reusability: Specifications can be thought of as reusable components that can be applied to multiple queries.
* Type Safety: Specifications enable the type-safe definition of query criteria in Java code.

In this project, I used Specification to handle queries that joined multiple tables together. This has significantly aided me in simplifying data extraction. I don't need to write a complicated query when joining multiple tables, especially in the project "Bike" entity. Every query concerning this object can be described as quite complex. Specification was extremely helpful to me when handling bike logic.

**3.1.6. Introduce about Spring Security – JWT token**

One of Spring Boot's final outstanding features is Spring Security. A framework for authentication and authorization called Spring Security is part of the Spring Boot project. For online applications, it provides a full range of security capabilities, including access control, authorization, and authentication.

On top of the Spring Framework, there is a security framework called Spring Security that offers a number of security capabilities that are simple to integrate into Spring Boot applications. These are the primary features of Spring Security:

* Authentication: Spring Security includes a number of authentication methods, such as form-based authentication, OAuth2, and JWT.
* Authorization: Spring Security includes a fine-grained authorization mechanism that allows developers to define access control rules at the URL or method level.
* Session Management: Spring Security includes features for managing user sessions, such as session fixation protection and session timeout configuration.
* Integration with Other Spring Modules: Spring Security integrates easily with other Spring modules like Spring MVC, Spring Boot Actuator, and Spring Data.

In this project, I used Spring Security to decentralize accounts using a JWT token. JWT (JSON Web Token) is a widely used standard for securely representing claims between two sources. In addition, JWT is frequently used as a means of authentication and authorization in the context of Spring Security.

I will discuss this term in more detail. The JWT tokens can be used in a Spring Security configuration to validate a user's identity and grant or deny access to specific resources within an application. I will show some basic flow of using JWT with Spring Security follows these steps:

* The user sign in with your input request.
* The request are validated and a JWT token is generated by the server.
* The server returns the JWT token to the user.
* Based on the contents of the token, the server validates the JWT token and grants or denies access to each API.

To sum up, using JWT tokens to a Spring Security setup can help create a secure and adaptable way to authenticate and authorize users on a web app.

**3.2. ReactJS**

**3.2.1. What is ReactJS?**

ReactJS (also known as React) is a well-known open-source JavaScript library for developing user interfaces (UI) for web applications. This library allows developers to produce reusable UI components that can be used to quickly and easily build complex web applications. It employs a declarative syntax that allows developers to describe the desired UI state, with React handling DOM (Document Object Model) updates as needed.

React has a large user community, so I can easily find help to solve problems that arise during use. Furthermore, it supports a large number of libraries and frameworks that support the development of system interfaces.

Following, I will list the advantages of this open-source:

* Improved performance: ReactJS employs a virtual DOM, which aids in the performance of web applications by reducing the number of direct manipulations of the actual DOM.
* Reusable components: Developers can use ReactJS to create reusable components that can be used throughout the application. This reduces the amount of code that must be written, resulting in faster and more efficient development.
* Simple to learn: ReactJS has a simple API, making it simple for developers to learn and use.
* Integration with other libraries: ReactJS integrates easily with other libraries and frameworks.

In general, ReactJS is a strong and flexible library that gives developers the resources they need to create scalable and fast web applications.

**3.2.2. Introduce about JSX**

One of the new term acquired while working with ReactJS is JSX. In this section, I will introduce this concept. JSX (JavaScript XML) is a syntax extension for JavaScript that is used with ReactJS. It makes it easier to construct and work with the UI components of a React application by allowing developers to write JavaScript code that resembles HTML. Actually, the React compiler converts JSX code into regular JavaScript code that the browser can interpret.

Although JSX syntax is similar to HTML syntax, it is not the same. Instead, it's a syntax extension that lets developers write JavaScript code that generates React elements. However, it still enables us to use a familiar HTML-like syntax while utilizing JavaScript and ReactJS to their full potential.

I'll explain why it's preferable to use JSX instead of HTML in a ReactJS source code. In fact, a React application can be created without the use of JSX. Therefore, the developers can create React components in either pure JavaScript or plain HTML templates.

Using JSX will provide some of the following advantages:

* Improved developer experience: JSX enables developers to create components using familiar HTML-like syntax.
* Full JavaScript power: JSX enables developers to use the full power of JavaScript to create dynamic and interactive components.
* Performance: To improve performance in React applications, JSX can be compiled. React can effectively update only the portions of the user interface (UI) that have changed rather than having to re-render the entire page by using a virtual DOM.

In conclusion, we can make React components by using plain HTML or pure JavaScript. Using JSX can give us some benefits that can make it simpler and faster to create great apps.

**3.2.3. Introduce about Component**

A component in ReactJS is a part of a website or app that you can use again and again. It has the look and behavior of something on the screen. A component might use HTML, CSS, and JavaScript and can get or give data to other parts.

Components are the basic pieces of ReactJS apps and an important part of the ReactJS plan. They help keep the code organized and make it easier to understand. We can create big and complex things by using smaller components.

Also, we can use the same components again in other projects or pages. This saves us time and stops us from making mistakes in the code.

To sum up, components are important for ReactJS projects. They help make the code organized, easy to keep up, and let me create big things using smaller parts.

**3.2.4. Introduce about Props and State**

**3.2.4.1 What is Props in ReactJS project?**

Props (short for "properties") are a data transfer mechanism in ReactJS that allows data to be passed from a parent component to a child component. Props are read-only and the child component is unable to change them.

Props are essential in ReactJS projects because they enable developers to create reusable and modular components. A component can be customized to meet the needs of the parent component by passing data down through props. Because each component can be made to handle a particular set of data, it is simpler to design complex user interfaces with numerous components.

Using props also allows us to keep data flow in their application unidirectional, making it easier to reason about the application's state. Data flows from parent components to child components via props, and a child component can update the data by invoking a function passed down as a prop from the parent component.

Props are an important part of ReactJS projects because they enable the creation of reusable, modular components that can be easily customized and combined. We can create complex user interfaces that are easy to reason about and maintain by passing data down through props.

**3.2.4.2. What is State in ReactJS project?**

In ReactJS, state is an object that stores the data that a component requires to render and manage its behavior. Unlike props, which are read-only and are passed down from parent components, state is managed internally by the component and can be modified by the component.

Because it allows components to manage user interactions and react to changes in the application, the state is important for ReactJS projects. A form component, for example, could use state to store the values of the input fields and update them as the user types.

State is also important in ReactJS projects because it enables efficient user interface updates. The reason is that ReactJS will automatically re-render a component when its state changes, updating only the portions of the user interface (UI) that have changed.

In general, the state is an important factor of ReactJS projects because it enables components to be interactive and dynamic, responding to user input and changes in the application state.

**3.2.4.3. Difference between Props and State:**

Props and state are both important concepts in ReactJS, but they serve different functions. Therefore, we must be able to distinguish between the two in order to develop the application more effectively.

* Ownership: Props are owned by the parent component and passed down to child components, whereas state is owned and managed by the component itself.
* Mutability: Props are read-only and cannot be changed by the child component, whereas state can be altered by the component.
* Initialization: Props are passed into a component when it is created, whereas state is initialized within the component's constructor or in a lifecycle method.
* Usage: Props are used to pass data from parent to child components, whereas state is used to manage internal data and handle user interactions.

As we can see, the props and state are used in a variety of ways within a ReactJS application, each serving a distinct purpose. Understanding the distinctions between the two is critical for developing effective and efficient components.

**3.2.5. Introduce Lifecycle methods of ReactJS**

Lifecycle methods in ReactJS are special methods that are called at specific points in a component's lifecycle. These methods enable we to perform specific actions at various stages of a component's existence, such as when it is first mounted, updated, or unmounted from the DOM.

In ReactJS, lifecycle methods fall into three broad categories:

* Mounting methods: When a component is first created and added to the DOM, these methods are invoked. This category includes constructor(), static getDerivedStateFromProps(), render(), and componentDidMount() .
* Updating methods: These methods are called when the state or props of a component change and the component needs to be re-rendered. Static getDerivedStateFromProps(), shouldComponentUpdate(), render(), getSnapshotBeforeUpdate(), and componentDidUpdate() are examples of methods in this category.
* Unmounting methods: When a component is removed from the DOM, these methods are invoked. The componentWillUnmount() is the only method in this category.

In addition to these main categories, there are a few less commonly used lifecycle methods, such as static getDerivedStateFromError(), componentDidCatch(), and shouldComponentUpdate().

Lifecycle methods are important in ReactJS because they give we hooks to perform actions at specific points in the lifecycle of a component. ComponentDidMount(), for example, can be used to retrieve data from an API when a component is first mounted, whereas componentWillUnmount() can be used to clean up any resources used by the component before it is unmounted.

We can control the behavior of our components at various points in their lifecycle using lifecycle methods, making it easier to create efficient and responsive user interfaces.

**3.2.6. Introduce about Hooks**

Hooks is a new thing in version ReactJS 16.8. We can use it to make functional components act like class components. It's useful because it makes it easier to manage state, side effects, and other things that class components could do. Hooks are good for ReactJS projects because they:

* Help organize the code: Hooks can combine related things into one function, making it easier to understand.
* Make things simpler: Hooks let us make one component do many things instead of using many components.
* Reuse things: Hooks let us use the same thing in many components instead of making new ones.

In general, Hooks are an important thing for ReactJS projects. They can help make better things for people and make apps work better.

**3.2.7. Introduce about Redux**

Redux is a library for managing states in JavaScript programs, such as ReactJS. It provides a centralized store for managing the state of an application and allows for predictable and efficient handling of state updates and data flow.

The application state is stored in a single "store" object in a Redux-based ReactJS project, which is then accessed as needed by individual components. Changes to the state are handled by "actions" that are sent to the store and then processed by "reducers" to update the state.

Then, I will discuss some benefits of redux:

* Managed by a centralized state: Redux makes application state management and maintenance simpler by allowing for a single source of truth.
* Predictable state updates: Redux makes it simpler to debug and predict the behavior of applications by adhering to a rigid pattern for handling state updates.
* Developer tools: Redux includes a number of developer tools and extensions to aid in debugging, testing, and maintaining the application.

As we can see, Redux is an important tool for ReactJS projects because it can help improve the application's scalability, maintainability, and predictability. While it may add some complexity to the development process, it has the potential to make it easier to build high-quality, complex applications with a clear separation of concerns and predictable behavior.

In the Motorcycle Rental Management System project, I used redux for some intermediate variables that run in real-time, such as saving tokens for authorization and loading menus based on whether or not the user is logged in.

**3.3. MySQL**

I want to present MySQL as one of the following ideas to everyone. It is one of the most important pieces of information for this endeavor. Before I get into MySQL, let me explain what a Relation Database is.

**3.3.1. Introduce about Relational Database**

A relational database is a type of database management system (DBMS) that stores data in rows and columns in tables. The key word "relational" refers to how data in a database is organized based on relationships between tables.

In relational database, each table represents a specific type of data, such as customer information, product information, or order information. In addition, each table is made up of rows that represent unique data instances and columns that represent certain data characteristics or characteristics.

The relationships between tables are defined by keys, which are used to connect data from one table to data from another. As we can see, the primary key is a unique identifier for each row in a table while the foreign keys are used to connect related rows across tables.

One of the primary benefits of a relational database is its ability to support complex queries and reporting, making it an best choice for applications that require data analysis. Moreover, Relational databases also provide a high level of data integrity and consistency, which ensuring that the data is accurate and reliable.

Popular relational database management systems include MySQL, Oracle, SQL Server, and PostgreSQL.

**3.3.2. What is MySQL?**

MySQL is a popular open-source relational database management system (RDBMS) that enables developers to efficiently store, organize, and retrieve data. I'll explain why MySQL has grown in popularity in the programming community now.

* Open Source: MySQL is an open-source database, which means it is free to use and modify.
* Reliability: MySQL is a very dependable database management system that can handle large amounts of data without slowing down.
* Simple to Apply: MySQL is simple to use and works with a variety of programming languages, including PHP, Python, and Java.
* Community Support: MySQL has a large and active developer community that contributes to its development and provides support to other developers.

As we can see, MySQL is an important and popular database management system because it is dependable, scalable, simple to use, and has a large and active developer community. Therefore, it is an excellent choice for web applications that must manage large amounts of data efficiently.

Despite using MySQL Server, I chose to create a non-relational database for the Motorcycle Rental Management System project when implementing project.

Non-relational databases are designed to expand horizontally, which means they can handle large amounts of data and high volumes of traffic by adding more servers to the database cluster. Furthermore, non-relational databases are schema-less, which means they do not require a predefined schema or data format.

Non-relational databases can be optimized for particular use cases and provide superior performance for certain types of queries or data access patterns in terms of performance. A key-value store database, for example, can provide extremely fast lookups for particular data items.

Finally, about availability, non-relational databases are intended to be extremely available, with built-in redundancy and failover mechanisms that ensure data is always accessible. As a result, they are an excellent option for applications requiring high uptime and dependability. After completing the project, I will configure MySQL's relational database to prevent data from being accidentally deleted.

**3.3.3. Distinguish between MySQL and SQL server.**

Many people will confuse MySQL server with SQL server because they both have the term "SQL" in their names. Although these two concepts are comparable in nature, they are two distinct types of RDBMS. (relational database management systems). Because SQL server is very common among programmers, I will distinguish between these two types of RDBMS.

* Ownership and Licensing: MySQL is an open-source RDBMS created by Oracle, whereas SQL Server is a proprietary RDBMS developed by Microsoft.
* Platform Support: MySQL is a cross-platform database system that can operate on Windows, Linux, and macOS. SQL Server, on the other hand, is mainly intended to run on Windows, with some limited support for Linux.
* Performance: Both MySQL and SQL Server are high-performance database platforms, but they have distinct advantages. MySQL is designed to execute basic queries quickly. SQL Server is designed for complex queries.

In conclusion, both MySQL and SQL Server are powerful RDBMS platforms that can be used in a variety of applications. For this project, I chose MySQL over SQL server because it is free and has strong support for integrating with Spring Boot projects.

**3.3.4. Introduce about MySQL Workbench**

Following that, I'll go over a well-known MySQL utility called MySQL Workbench. This is a visual database design and management tool for MySQL. It includes a graphical user interface (GUI) for designing, creating, and administering MySQL databases. MySQL Workbench includes the following popular features:

* Data Modeling: MySQL Workbench's graphical interface enables us to view design, create, and modify database schemas. The tables, relationships, and indexes can be created, and the schema can be viewed in a number of formats.
* SQL Development: MySQL Workbench contains a SQL editor with syntax highlighting, code completion, and query profiling. We can write and execute SQL statements, build and edit stored procedures, and handle triggers.
* Database Administration: MySQL Workbench enables us to handle user accounts, backup and restore databases, and monitor server status.
* Collaboration: MySQL Workbench contains features for collaborating with other developers, such as version control integration, team collaboration, and the ability to exchange database designs and models.

MySQL Workbench is a powerful tool for handling MySQL databases. I also use this utility because it is free and is the primary MySQL support tool.

**3.4. SCSS**

Following that, I will discuss SCSS, a language used to style webpages. This is a more advanced form of CSS.

**3.4.1. What is SCSS?**

Sassy Cascading Style Sheets, or SCSS, is a preprocessor programming language that creates CSS. It is basically a CSS extension that adds additional functionality and features to CSS, making it easier and more efficient to write and handle stylesheets.

Since SCSS is a superset of CSS, every legitimate CSS rule also applies to SCSS. SCSS, on the other hand, introduces features such as variables, nesting, inheritance, mixins, and more. These features make it easier to create reusable and maintainable stylesheets, particularly for big and complex projects.

**3.4.2. Advantages of SCSS**

The SCSS has the following advantages:

* Variables: We can use variables to store and reuse values, which can help us maintain consistency throughout our stylesheets. For example: $primary-color: #007bff;
* Nesting: SCSS enables us to nest CSS selectors within each other, making it easier to write and read complex stylesheets.
* Mixins: We can also create and reuse sets of styles using mixins, which can help us avoid duplicating code and make our stylesheets more consistent.
* Inheritance: SCSS allows us to share styles between selectors by using inheritance, which can help us avoid duplicating code and make our stylesheets more efficient.
* Modularity: We can divide our stylesheets into smaller modules. This help us organize our code and make it easier to maintain.

**3.4.3. Disadvantages of SCSS.**

The SCSS has the following disadvantages:

* Learning curve: SCSS is more difficult to learn than plain CSS because it needs knowledge of its syntax and features.
* Compilation: Before it can be used in a web browser, SCSS must be compiled into CSS, which can add an additional step to our workflow.
* Performance: Depending on the complexity of our stylesheets, using SCSS may result in larger file sizes and possibly slower page load times.
* Tooling: Using SCSS may necessitate the setup of extra tooling or build processes, which can complicate our development workflow.
* Browser support: Some older web browsers may not completely support all of SCSS's features, resulting in inconsistencies or errors in our stylesheets.

**3.5. Firebase**

This part will go over a specialized tool for storing images pertaining to cloud computing. Firebase is the name of this application. It is an essential tool in the Motorcycle Rental Management System project.

**3.5.1. What is Firebase?**

Firebase is a Google-owned mobile and web application development tool that offers a variety of backend services such as authentication, real-time databases, hosting, cloud storage, and messaging. It helps developers to create high-quality mobile and web apps rapidly, with minimal setup and maintenance.

Firebase provides a real-time database that can be used to store and synchronize data in real-time between numerous clients. It employs NoSQL technology, which makes it easier to store and retrieve data in a flexible way without requiring developers to build complex database schemas.

Firebase also provides authentication services, allowing developers to quickly integrate sign-in and sign-up features into their applications. Support for major social media platforms such as Google, Facebook, and Twitter is included.

Furthermore, Firebase offers hosting services, making it simple to deploy web applications rapidly and securely. It also provides cloud storage for storing and serving information, as well as messaging services for sending tailored messages to users.

In the end, Firebase is a powerful platform for developing mobile and web apps that has grown in popularity among developers due to its ease of use and extensive feature set.

**3.5.2. Some benefits of Firebase**

I will discuss some popular benefits of Firebase in this project:

* Accessibility: Firebase enables us to view our stored images from any application or web browser.
* Security: Firebase includes built-in security features to safeguard our stored data, such as access controls and encryption.
* Integration: Firebase integrates easily with other Firebase services, such as authentication and real-time databases, making it easier to create a comprehensive application that utilizes multiple Firebase services.

**3.6. Git**

Git is one of the finest tools for developers to use to manage source code versions. Despite the fact that there are many tools to support this industry today, Git maintains a strong position in the developer community.

**3.6.1. What is Git?**

Git is a popular version control system (VCS) that enables software developers to manage and track changes to their code over time. It was developed by Linus Torvalds in 2005 and has since grown to be one of the most widely used VCS tools in the software development sector.

Git works by establishing a repository, or "repo," which is a central location where developers can store their code and monitor changes to it. Developers can make changes to their code locally on their own computers and then send those changes to the central repository. Other developers can then pull those modifications from the repository and incorporate them into their own work.

Git enables developers to collaborate on the same codebase even if they are spread out across various parts of the globe, which is one of its main advantages. It also includes branching and merging features, which enable developers to work on separate features or versions of code at the same time and then merge those changes back into the primary codebase.

**3.6.2. Benefits of Git**

Here are some of the benefits of using Git for source code management:

* Version control: Git enables developers to keep track of changes made to the codebase over time. This means that developers can easily revert to earlier versions of the code if necessary, as well as monitor who made changes and when.
* Branching and merging: Git has strong branching and merging capabilities, allowing developers to work on different features or versions of code at the same time and then merge those changes back into the primary codebase.
* Collaboration: Git allows developers to work together on the same codebase even if they are in different parts of the globe. It has tools like pull requests, code reviews, and commenting that aid in collaboration and ensure that code changes are correctly reviewed and tested.
* Open-source community: Git is an open-source tool, which means that it has a big developer community that contributes to its development and support.

**3.6.3. Some terms in Git**

When using Git, we must grasp some of the following concepts. These terms are practically used by every version control tool, making it accessible to the vast majority of individuals. These include:

* Repository: A repository (also known as a "repo") is a central location where developers can store their code and monitor changes to it.
* Commit: A commit is a collection of modifications made to the codebase that are saved in the repository. It contains a message that explains the changes.
* Branch: A branch is a separate version of the codebase that enables developers to work on different features or versions of the code at the same time.
* Merge: Merging is the process of integrating changes from distinct branches into the main codebase.
* Pull: Pulling is the process of obtaining modifications made to the repository by other writers and incorporating them into our local copy of the codebase.
* Push: Pushing is the process of uploading modifications made to our local copy of the codebase to the central repository.
* Conflict: When Git is unable to automatically merge changes made to different branches, a conflict arises. In this situation, developers must carefully resolve the conflict by reviewing the changes and deciding how to merge them.
* Fork: A fork is a copy of a repository that enables developers to make modifications to the codebase independently of the initial repository.
* Pull request: A pull request is a method for developers to send modifications made to a branch of a repository for review and possible merging into the main codebase.
* Gitignore: The “gitignore” file is a file that defines which files or directories should be ignored by Git when tracking changes to the codebase.

**3.6.4. Introduce about Git Desktop**

Git Desktop is a graphical user interface (GUI) for Git that enables developers to manage and monitor changes to their codebase visually. It makes it simple for us to perform typical Git chores like creating and cloning repositories, committing changes, and merging branches.

Although the command-line interface (CLI) and npm statements can be used to control the version of source code, Git Desktop can make the procedure simpler and more effective for me.

Git Desktop provides a visual interface that shows the current state of the codebase, including any changes made but not yet committed, and enables us to easily commit changes, create and merge branches, and view commit history for this project.

Git Desktop can, in general, assist programmer in managing and tracking changes to their codebase more quickly and efficiently, which can reduce errors and save time. It can be a helpful tool for developers who prefer a visual interface for managing their codebase or are less familiar with the Git CLI.

**3.7. IntelliJ**

Next, I'll discuss a unique tool for creating back-end code with Spring Boot support. IntelliJ is that utility. Out of the many IDEs that handle the Java language, I chose IntelliJ for this project.

**3.7.1. What is IntelliJ?**

JetBrains' IntelliJ IDEA is an integrated development environment (IDE). It is intended to boost developer efficiency by including features such as code completion, refactoring, debugging, and version control integration. It supports a number of computer languages, including Java, Kotlin, Groovy, Scala, and others.

IntelliJ IDEA supports version control systems such as Git, Mercurial, and Subversion, making it simple to manage code changes and work with other developers. It also has built-in integration with build tools such as Maven and Gradle, enabling us to build, test, and deploy our code from within the IDE.

One of the most important aspects of IntelliJ IDEA is its plugin architecture, which allows us to customize and expand the IDE's functionality. There are hundreds of plugins accessible that support different frameworks, libraries, and languages.

**3.7.2. Benefits of IntelliJ**

I'll outline some benefits that using IntelliJ offers to Java and Spring Boot projects:

* Productivity: IntelliJ IDEA allows developers to write code quicker and with fewer errors. Besides that, it has some features like code completion, code analysis, and automated refactoring that make development processing more effective.
* Integration: IntelliJ IDEA includes integration with a variety of build tools, version control systems, and third-party tools.
* Plugin ecosystem: IntelliJ IDEA has a big and busy plugin ecosystem that supports a variety of libraries, frameworks, and tools.
* Debugging: The powerful debugger in IntelliJ IDEA makes it easily to debug code and rapidly identify problems. It also provides remote debugging, breakpoints, and other advanced features that support developers in finding and resolving issues.

Because of its powerful features, productivity tools, and plugin ecosystem, IntelliJ IDEA is a popular option among Java developers. This IDE appeals to me because of its excellent design and easier debugging than Eclipse (another specialist IDE for Java projects). Back-end developers should think about utilizing IntelliJ, it has been suggested.

**3.8. Visual Studio Code**

I'll present a fantastic tool for ReactJS programming after we learn about IDEs for the back-end. This is a utility that is widely used by most people, free, and both of those things. This tool covers almost all programming languages in addition to ReactJS. Visual Code is the name of that tool.

**3.8.1. What is Visual Studio Code?**

Visual Code, also known as Visual Studio Code, is a famous source code editor created by Microsoft. It is a free and open-source application that works with Windows, macOS, and Linux.

In addition to supporting a broad range of programming languages, Visual Code provides a number of tools like syntax highlighting, code completion, debugging, source control integration, and extensions that can improve the editor's functionality. It is also very customizable, enabling users to change key bindings and other settings to suit their preferences.

Visual Code is a strong and adaptable tool for developers in general, especially for those who work with numerous languages or must cooperate with other developers on a project.

**3.8.2. Some features of Visual Studio Code**

I'll describe the features of this tool here.

* Code Editing: Numerous code editing tools are available in Visual Studio Code, such as syntax highlighting, auto-completion, and mistake checking. Additionally, a large number of computer languages are supported.
* Extensions: There is a sizable library of extensions available for Visual Studio Code that can improve its usefulness. These include add-ons for widely used frameworks, tools, and computer languages.
* Integrated Terminal: An combined terminal in Visual Studio Code allows us to use command-line tools right from the editor. Working with build tools or executing server-side scripts can make use of this in particular.
* Version Control: Git is already supported by Visual Studio Code, making it simple to handle version control inside the editor. Support for staging changes, committing changes, and examining diffs are all included in this.
* Snippets: Visual Studio Code contains a number of code snippets that can help us speed up our development process. By typing the proper shortcut and pressing Tab, we can view the snippets.

As we can see, with a wealth of tools to support development across a range of programming languages and frameworks, Visual Studio Code is a strong and adaptable code editor.

**3.8.3. How Visual Studio Code support ReactJS coding?**

I'll explain why this editor can support the ReactJS project in this section after we grasp the definition and are aware of the outstanding features of Visual Studio Code. ReactJS is supported by the following components in Visual Studio Code:

* Support for JSX: The syntax used by ReactJS to define components, JSX, is well supported by Visual Code. Additionally, auto-completion, syntax highlighting, and mistake checking are included.
* Extension Support: Visual Code has a large library of extensions that can be used to improve its functionality for ReactJS development. The ReactJS code snippets extension, which offers code snippets for React components.
* Integration of Git: Visual Code includes Git support, making it simple to handle version control from within the editor. Working on ReactJS projects, where teamwork is prevalent, can be especially advantageous in this regard.

I made the decision to start front-end code for this project right away based on the aforementioned considerations. One could say that Visual Code offers a strong and adaptable environment for ReactJS programming, supporting the entire spectrum of ReactJS-related technologies and tools. Additionally, this teaching method is used consistently in online classes. I've found a lot of assistance with the free.

**3.9. Postman**

We have covered the debut of the back-end and front-end source editors. I will now introduce everyone to a very significant and helpful tool for this endeavor. As I mentioned in the previous part, the Motorcycle Rental Management System project is constructed from two distinct sources for the client side and server side. Through APIs, these two sources will interact with one another. I employed an application named Postman to verify these APIs. We'll learn what the item is.

**3.9.1. What is Postman?**

Postman is a popular API development and testing tool that allows us to make HTTP requests and watch the responses in a graphical user interface. We can rapidly develop and test APIs with Postman, automate tests, and work together with our team. There are some key features of this application:

* Request builder: For creating HTTP requests, Postman offers a clear and user-friendly UI. Our requests can readily include headers, parameters, authentication, and other things.
* Environment and Collection: We can group your requests into groups and environments with Postman. For various phases of our development or testing, we can build up various environments and group related requests into collections.
* Automation and testing: Postman includes powerful testing and automation features such as the ability to write test scripts in JavaScript, set up automated test suites, and connect with continuous integration and delivery (CI/CD) tools.
* Collaboration: We can work together on developing and testing APIs with your team thanks to Postman. Our team members and I can collaborate in real time while sharing collections, environments, and test findings.
* Integrations: GitHub, Jira, Slack, and many other programs and services are among those that Postman works with. We can use these integrations to streamline our productivity and automate chores.

Postman is an all-around strong and flexible utility for developing and testing APIs that may assist us save time and raise the caliber of our APIs.

**3.9.2. Benefits of the Postman**

Here, I'll discuss some of the benefits Postman offers developers working on API-based applications.

* Efficient testing and debugging: By providing a straightforward and understandable interface for creating and sending HTTP requests, Postman makes it simple to test and troubleshoot APIs. Different endpoints, parameters, and headers can be rapidly tested, and we can see the results immediately.
* Time-saving: We can automate time-consuming chores with Postman, such as testing and documentation, which can help us save a ton of time and effort. Additionally, we can collaborate more effectively and with less duplication of effort by sharing collections and settings with our team members.
* improved teamwork: In order to create and test APIs, we can collaborate with other team members using Postman. We are able to work together in real time while sharing collections, environments, and test findings. This can help to enhance communication and reduce errors.
* Strong automatic and testing features: Writing test scripts in JavaScript, creating automated test suites, and integrating with CI/CD tools are all made possible by Postman's robust testing and automation capabilities. By doing so, we can guarantee the quality of your APIs and find bugs as early as possible.
* Versatility: REST APIs, GraphQL APIs, SOAP APIs, and many other types of APIs can all be developed and tested using Postman. It is a flexible tool for our development process because it integrates with a wide range of tools and services.

The Motorcycle Rental Management System project's APIs have all been stored in Postman. I'll use this program to try each potential match of the API each time a new function is implemented. When both the front end and the back end are operating smoothly, I simply begin to integrate the API between these two sources of code.

**Chapter 4: Software Product Requirements**

**4.1. Overview of other similar products**

In the first part of Chapter 4, I will discuss some real-world products that are comparable to the Motorcycle Rental Management System project. We all know that this project's two primary users are the Customer (Bike Renter) and the Administrator (Website Manager). Customers typically visit public websites, which are always available to us. On the other hand, private sites are those used for administration. We won't be able to examine these websites or learn about their features and interface. As a result, I only turn to customer websites as inspiration for the design of this project's public sites. I will actively collaborate with users to design the user experience and functionality of websites that admins use. I also make reference to a few management system templates, but the majority of them have nothing to do with renting cars and are internal products, so sharing them outside is not permitted. Here is a list of some of the public motorcycle rental services I refer to:

* EagleRider: One of the most popular motorcycle rental websites in the world is this one. The company rents out a variety of bikes, including Harley-Davidson, BMW, Honda, and others. EagleRider makes it simple to rent a motorcycle and face the open road with more than 100 locations worldwide. Site: <https://www.eaglerider.com/>
* Hertz Ride: High-end BMW motorcycles are available for hire through Hertz Ride, a motorcycle rental company. They have sites in South Africa, the USA, and Europe. They provide a variety of rental choices, including self-guided tours, guided tours, and one-way rentals. Site: <https://www.hertzride.com/>
* MotoQuest: Motorcycle rental company MotoQuest provides a variety of motorcycles for hire, including BMW, Harley-Davidson, and more. They also provide guided tours in a variety of places, including South America, Alaska, and Hawaii. Site: https://www.motoquest.com/
* Twisted Road: Twisted Road is a service that links riders looking to rent motorcycles with motorcycle owners. This service is ideal for people who want to experience a particular motorcycle variety or who want a different kind of adventure. Site: https://www.twistedroad.com/

In general, the above websites are big companies in the world, so the interface of these websites is very beautiful and eye-catching. It might provide me with some motivation as I work to create a user-friendly interface for clients. Nevertheless, as I previously mentioned. I am unable to create websites like this due to time, resource, and money limitations. I can only design the interface myself to accommodate Vietnamese clients. Second, I won't put too much emphasis on the interface because the project scope is quite small. I want the product's functionality to be polished and free of bugs as much as feasible. As a result, these websites are merely for reference and have little significance for me in this endeavor.

**4.2. User Stories – Product Backlog**

In this part, I'll go over two new concepts: User Stories and Product Backlog. When analyzing the project's requirements, these two ideas are always brought up in the context of a specific software development project.

Prior to discussing these two concepts, I'll discuss the following requirement given by a particular customer:

1. The user can access the Home Page to view information about each type of motorcycle.
2. The user can sort the material in the motorcycle list.
3. Admin must log in to use management features.
4. An administrator can examine the Dashboard to collect economic reports during the specified time period. They are able to observe the ranking of new customers in order to offer discounts.
5. The administrator has the ability to sort, add, delete, and edit details for the bike, color, and manufacturer. Because we only provide two primary motorbikes, Manual Transmission Motorcycle and Automatic Transmission Motorcycle, we can only update the price of these two motorcycles at the moment.
6. Admin can select which motorcycles to add to the cart.
7. Admin can save client information when there is no motorcycle in the cart.
8. Admin can create orders with complete cart information. Ordering a new creation will maintain the status pending.
9. Admin has the authority to close or cancel any pending order.
10. Here is the formula for determining motorcycle rental: A \* (((B - (B % 24)) / 24) + C)

There is:

* A represents the overall number of motorcycles in the cart for a rental day.
* B is the entire amount of time spent renting a computer motorcycle.
* C is our coefficient, which is computed using the excess time. (0 is less than 1 hour, 0.5 is from 1 to less than 7 hours, 1 is from 7 years or more)

For example:

A Customer X rented 2 automatic motorcycles and a manual motorbike during the expected 3 days. As a result, the expected cost is (100,000 x 2 + 70,000 x 1) x 3 = 810,000 VND.

* A = 100,000 x 2 + 70,000 x 1 = 270,000 VND
* B = 3 x 24 = 72 hours
* C will be calculated based on when the motorbike is returned. We will have three examples in particular:

Case 1: If a customer returns a motorcycle 73 hours in early (3 days + 1 hour), then C = 0. So, according to the official rate formula, the cost of rent is 810,000 VND.

Case 2: If the customer returns a motorcycle between 73 and 80 hours before getting the bike, C = 0.5. As a result, the rental fee for the motorcycle is 945,000 VND.

Case 3: If a customer returns a motorcycle after 80 hours or more since getting the vehicle, C = 1. As a result, the rental fee for the motorcycle is 1,080,000 VND.

1. The admin can create a maintenance to record expenses for a particular day. These expenditures could include the cost of purchasing materials or repairs to the vehicle. We need to know which motorbikes are involved in this Maintenance in order to calculate motor vehicle service expenses.
2. If multiple admins are using the system to create invoices at the same time, we must properly control the number of vehicles in each admin cart. It is not permitted to appear in many carts.
3. The interface must reflect the Customer's brand color, which is orange.
4. The interface must be basic, uncomplicated, and easy to use.
5. The system must respond quickly and accurately.
6. The system must be completely secure, particularly when dealing with customer data.
7. The features accept bug count must be less than 20%.

Once we have the requirements list, we will begin the analysis to build the project's product backlog and user stories. The first step is to comprehend these two ideas.

**4.2.1. What is Product Backlog?**

A product backlog is a prioritized list of product or software application features or needs. It is an essential component of agile software development, and it guides the development team in building and providing the most important features first.

The product owner is usually in charge of maintaining the product backlog, which includes working with stakeholders to understand their needs and priorities, as well as defining the features and requirements that will be included in the product. The product owner is also in charge of prioritizing backlog items based on their worth to the company or end users.

The product backlog is very necessary for software development project because of some following reason:

* Prioritization: The product backlog allows the development team to prioritize features or requirements based on their business value, allowing them to concentrate on the most important features first.
* Communication: The product backlog serves as a communication tool between the development team and the stakeholders, enabling everyone to understand what is being developed, why it is being developed, and when it will be delivered.
* Adaptability: The product backlog is a living document that can be updated and reprioritized in response to new information, altering company priorities, or user feedback.
* Transparency: The product backlog gives visibility into the work being done and progress being made.

I will build the product backlog for this project based on the requirements listed above:

1. Create Home Page with information about each type of motorcycle
2. Implement sorting functionality for the motorcycle list
3. Develop a login page for Admin to access management features
4. Create Dashboard for Admin to view economic reports and customer rankings
5. Implement CRUD (Create, Read, Update, Delete) functionality for bike, color, and manufacturer details
6. Allow Admin to select motorcycles to add to cart
7. Implement functionality to save client information when there are no motorcycles in the cart
8. Develop functionality to create orders with complete cart information and maintain status as pending
9. Allow Admin to close or cancel pending orders
10. Implement formula for determining motorcycle rental cost
11. Develop functionality to record maintenance expenses for a particular day and track the motorbikes involved
12. Implement cart control to prevent multiple admins from having the same motorcycles in their carts
13. Design the interface with the Customer's brand color (orange)
14. Create a basic and user-friendly interface
15. Ensure the system responds quickly and accurately
16. Implement security measures to protect customer data
17. Ensure the system has a low bug count (less than 20%) by testing and debugging regularly.

This product backlog contains a number of critical features and functions required to build a successful motorcycle rental management system. The backlog includes a variety of user stories, such as those concerning the user interface, usefulness, and security. The backlog also includes critical features such as CRUD functionality implementation and security measures to protect client data. Overall, this backlog of products offers a strong foundation for creating a dependable and user-friendly motorcycle rental administration system.

**4.2.2. What is User Stories?**

User stories are a method used in software development to describe a product's or application's requirements or features from the end user's point of view. They are usually written in a simple, informal manner that both technical and non-technical team members can understand.

A user story is usually written in the following format: "As a [user role], I want [goal] so that [reason or benefit]." For example, "As a customer, I want to be able to save items to my wish list so that I can purchase them later.".

User stories are frequently used in agile development methodologies to ensure that the development team is focused on developing features that are essential to the end user. It can be used to prioritize features and guide development, as well as to identify possible usability problems or gaps in functionality.

The following are some reasons why user experiences are crucial for software development projects:

* Prioritization: According to how important they are to the customer; features can be prioritized via user stories. By doing this, it is possible to make sure that the most crucial features are created first.
* Usability: User stories can be used to find possible usability problems or functional gaps. The development team can make sure the application is user-friendly and fulfills user requirements by concentrating on their requirements.
* Flexibility: User stories are frequently brief and straightforward, making it simple to revise or modify them as the project progresses.
* Communication: Both technical and non-technical team members can readily understand user stories' straightforward, informal language.

The following are user stories for the project's motorcycle rental management system:

1. As a user, I want to access the Home Page to view information about each type of motorcycle so that I can make an informed decision on which motorcycle to rent.
2. As a user, I want to be able to sort the material in the motorcycle list so that I can easily find the motorcycle that I want.
3. As an Admin, I want to be able to log in to use management features so that I can manage the system.
4. As an Admin, I want to be able to examine the Dashboard to collect economic reports during the specified time period and observe the ranking of new customers in order to offer discounts.
5. As an Admin, I want to be able to sort, add, delete, and edit details for the bike, color, and manufacturer so that I can keep the system up to date.
6. As an Admin, I want to be able to select which motorcycles to add to the cart so that I can make orders on behalf of customers.
7. As an Admin, I want to be able to save client information when there is no motorcycle in the cart so that I can easily access it later.
8. As an Admin, I want to be able to create orders with complete cart information and maintain status as pending so that I can keep track of orders and their progress.
9. As an Admin, I want to be able to close or cancel any pending order so that I can manage the system effectively.
10. As an Admin, I want to use the formula for determining motorcycle rental cost so that I can accurately calculate the cost of rentals.
11. As an Admin, I want to record maintenance expenses for a particular day and track the motorbikes involved so that I can keep track of expenses and maintain the motorbikes effectively.
12. As an Admin, I want to control the number of vehicles in each admin cart to prevent multiple admins from having the same motorcycles in their carts.
13. As a user, I want the interface to reflect the Customer's brand color (orange) so that I can easily recognize the brand.
14. As a user, I want the interface to be basic, uncomplicated, and easy to use so that I can navigate the system easily.
15. As a user, I want the system to respond quickly and accurately so that I can complete my tasks efficiently.
16. As a user, I want the system to be completely secure, particularly when dealing with my data, so that my data is protected.
17. As a user, I want the system to have a low bug count (less than 20%) by testing and debugging regularly so that I can use the system without issues.

The user stories provided encompass a broad range of motorcycle rental management system functionalities. It addresses the wants and requirements of users as well as administrators. The user stories cover both fundamental capabilities, such as accessing the home page, sorting, and using a user-friendly UI, as well as more sophisticated ones, such as controlling orders and costs, delivering economic reports, and upholding security. In general, it appears to be a thorough list, and each user narrative seems to be crucial to the overall functionality of the system.

**4.3. Use Case Diagram**

This part will introduce the term "Use Case" and display the project's Use Case for the Motorcycle Rental Management System. I'll start by providing details on use cases.

**4.3.1. What is Use Case?**

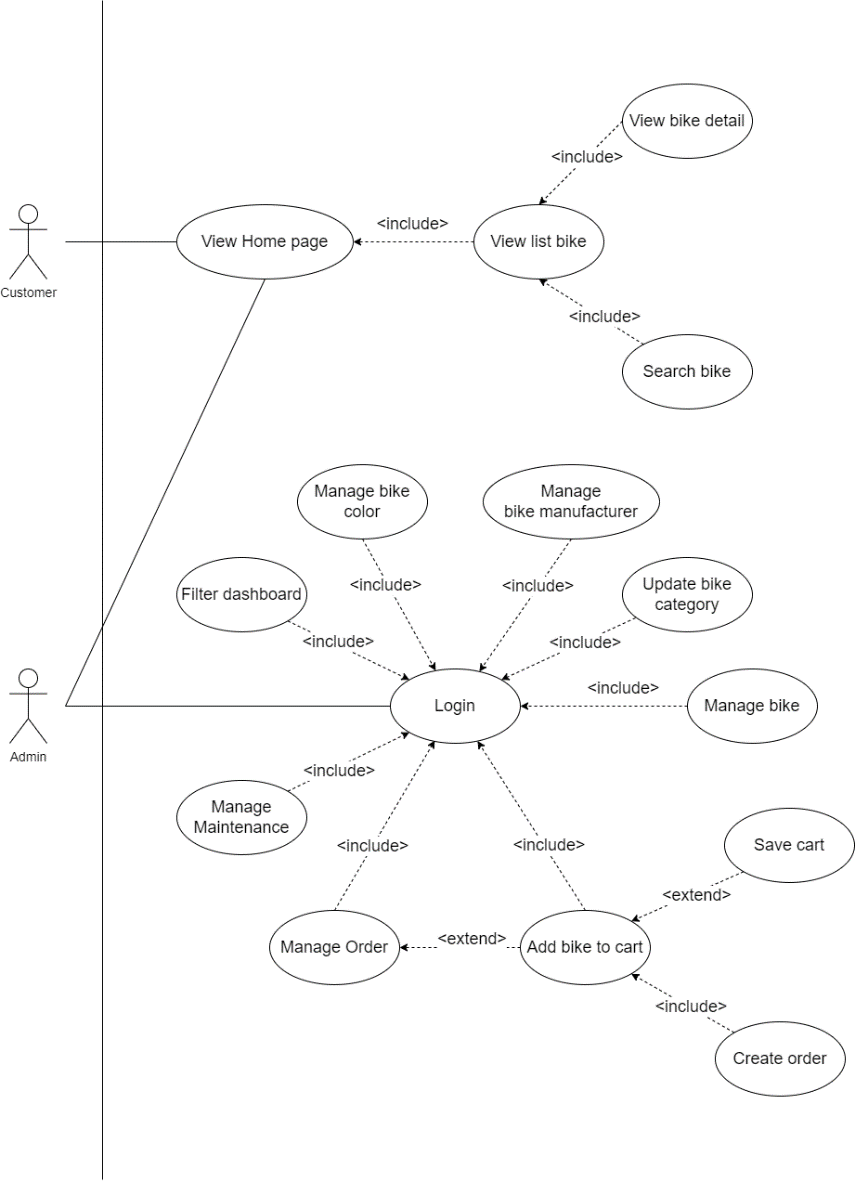
A use case diagram is a visual depiction of the interactions between actors (users or external systems) and a system under review. It is used to describe the system's functional requirements and to capture the requirements of a system as seen from the viewpoint of the user.

Use case diagrams include actors, use cases, and the relationships between them. A use case is a collection of actions or steps taken by a user or an external system to accomplish a goal. The outside parties that engage with the system are represented by actors.

Following are some key elements of a use case diagram that I will present here:

* Actor: An actor is an outsider who communicates with the system. Users, other networks, or devices can all be actors. In the use case diagram, they are shown as stick characters.
* Use case: A use case is a specific task or activity carried out by an actor in order to accomplish a system objective. It provides a user's perspective description of how the technology behaves.
* System boundary: A system boundary is a box that encompasses every use case in a diagram and serves as a representation of the system's extent. It displays what is both within and without the machine.
* Include Relationship: When demonstrating how one use case is a component of another, an include relationship is used. A solid line with an arrowhead going from the including use case to the included use case serves as a visual representation of it. Every time the including use case is executed, the included use case is also executed.
* Extension Relationship: When demonstrating how one use case may be optional or conditional on another use case, an extend relationship is used. It is shown as a dashed line with an arrowhead pointing from the extending use case to the expanded use case. The extending use case is only executed under certain circumstances or situations.

**4.3.2. Use Case of Motorcycle Rental Management System.**



*Figure 4: Use Case diagram of Motorcycle Rental Management System*

I'll go over this project's Use Case. First, we observe that the diagram has two primary actors: the customer and the administrator. An actor playing the role of the customer represents motorcycle renters. An administrator actor is a system user for managing vehicle rentals. Customers can only access publicly accessible websites, such as the homepage, bike list, and bike detail, to view images and comprehensive information about motorcycles. Administrators have full access to all webpages and all features, both public and private. After logging in as an admin, they can do almost anything on sites that require authentication. When the phrase "Manage" appears in a use case, it refers to all four CRUD actions (Create - Read - Update - Delete) for that site object. For instance, the admin has the ability to filter, add, remove, or modify a specific car color on the manage color page. Additionally, some pages do not include the word "Manage" because they lack the four features that the customer requires. For example, we can only update the details for the bike category on the manage category page (Specifically in the project is the price of that category). We can see that the arrows associated with Login all use the "include" relationship since almost every admin feature requires logging in in order to be used. Only a few actions in this project make use of the "extend" relationship.

**4.4. ERD**

The ERD is a critical chart for representing a database. We will learn more about this diagram and how it applies to the project in the sections that follow.

**4.4.1. What is ERD?**

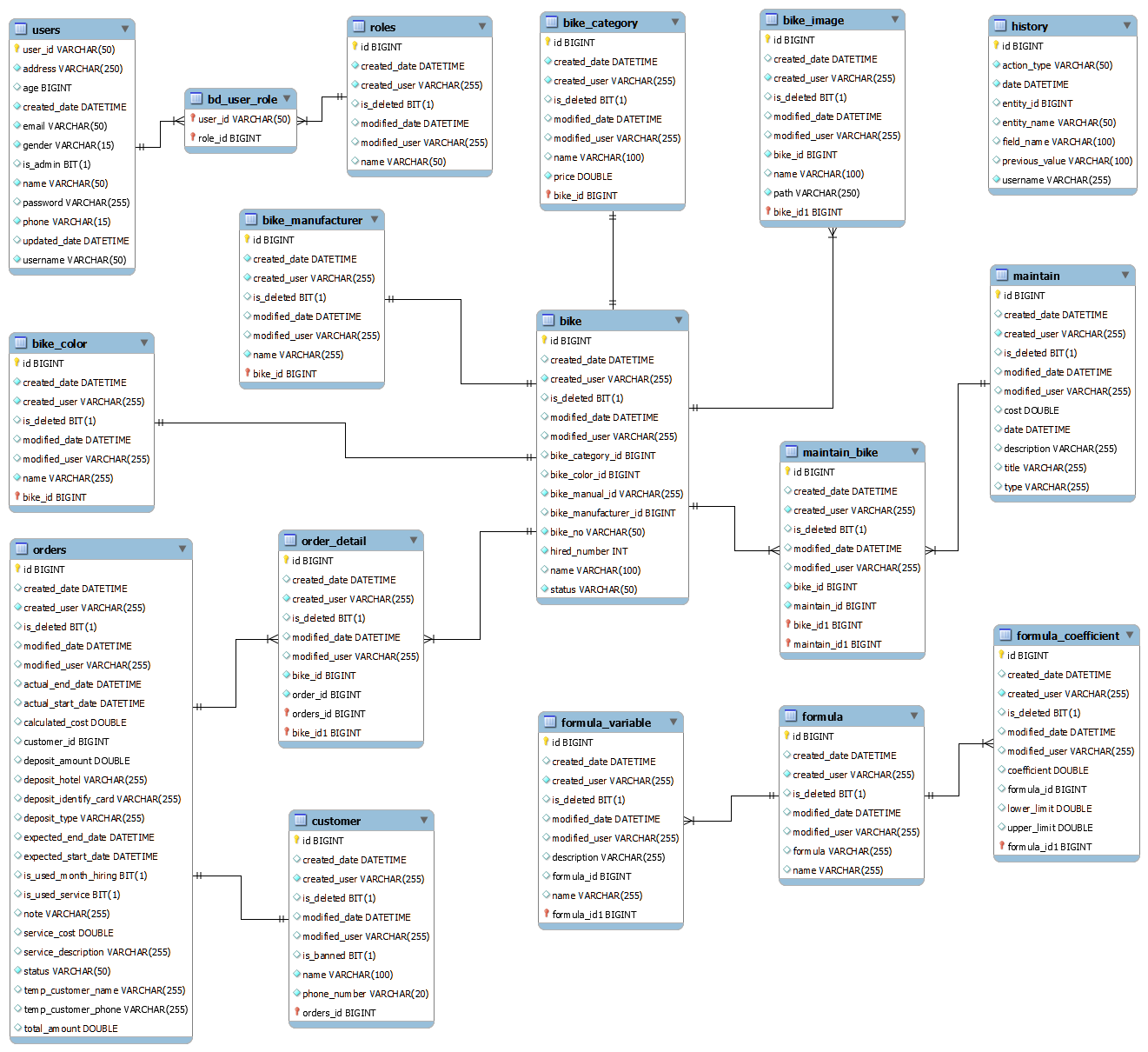
ERD is an acronym that stands for Entity-Relationship Diagram. It is a visual depiction of the relationships between entities in a database.

Entities are shown as rectangles in an ERD, whereas relationships between entities are shown as lines joining the rectangles. The types of relationships between the entities, such as one-to-one, one-to-many, or many-to-many, are indicated on the lines connecting them.

The attributes (or qualities) of each entity, the connections between entities, and the cardinality of those connections can all be determined using ERDs. This data can then be used to construct a database schema or to check that an existing database's architecture is accurate and fits the system's needs.

Here are some components of ERD

* Entity: A person, place, thing, event, or concept that is reflected in the database.
* Attribute: A feature or property of an entity.
* Primary Key: Each entity instance has a unique identifier that is used to differentiate one instance from another.
* Foreign Key: A field in one table that references to the primary key in another table, used to establish a relationship between the two tables.
* Relationship: The relationship between two entities, demonstrating how they are related to one another.
* One-to-One Relationship: A relationship between two entities in which each instance of one entity is connected with just one instance of the other entity.
* One-to-Many Relationship: A relationship between two entities in which each instance of one entity can be connected with multiple instances of the other thing.
* Many-to-Many Relationship: A relationship between two entities in which each instance of one entity can be connected with numerous instances of the other entity, and vice versa.

**4.4.2. ERD of Motorcycle Rental Management System project**

*Figure 5: ERD of Motorcycle Rental Management System*

This ERD, which contains 17 tables total, is broken down into four primary categories: independence, authentication, bikes, and formulas. Except for the tables in the group independence, all tables in each group will be connected together. We will explore to learn more about each group and to understand what each group means:

Group 1 - Authentication: users, roles, and bd\_user\_role are three of the tables in this group. These are the databases used to keep track of user accounts for system logins. We can see that the bd\_user\_role table is made up of users and roles in a many-to-many relationship. A design like this will aid in the growth of decentralization. Although this project has only one role per user, in the future we may be able to conduct additional duties such as an account with numerous roles.

Group 2 – Bike: There are 10 tables in this group, with the Bike table serving as its hub. These are the most crucial project entities since they hold consumer business information. I'll start by introducing the entity's color, manufacturer, category, and image. These entities are motorcycle attributes. A motorcycle will have a single manufacturer, one category, and one color. However, a bike can have numerous graphics or just one image. We can see that while the image and the bike entity have a one-to-many relationship, the other three tables and the bike entity have a one-to-one relationship. Following that, I'll discuss two entities: order and maintenance. These are two entities which are involved in the project's revenues and expenses. The bike and both of these things have a relationship many-to-many. From there, we can see that order\_detail and maintain\_bike, two more tables, have been established. Using a one-to-one relationship, the entity order alone will be linked to the entity customer. We only want one customer per order, for this reason.

Group 3 – formula: Three entities that are utilized to define and store the project's calculation formula are included in this group. We have yet to witness the value of these entities because the project only has one formula. We can use a single calculation in the code to determine the rental vehicle. However, using entities in the database allows us freedom in the way we may store, change, and use them. We may create a website that enables administrators to change or add new formulas through generating these three tables. I'll go into greater detail about these things. We are all aware of the formula used in section 4.2 to determine the cost of renting a motorcycle: "A \* (((B - (B% 24)) / 24) + C" This formula will be kept in the table formula. The variables A, B, and C are stored in the formula\_variable table, each with its own description. We have variable C as a coefficient that is calculated depending on the exceed time, thus we have table formula\_coefficient to hold the maximum and minimum of the exceed time.

Group 4 – independence: This group comprises independent tables that have no tie to any other tables. The history table is the only independent one currently present in the Motorcycle Rental Management System project. To keep track of every user action, including logins, data creation, deletion, and editing, I constructed this entity. This will assist the IT staff in both controlling data modification and helping in the recovery of outdated data during updating. In addition, IT may monitor user behavior to help the software become more secure.

**4.5. Sitemap**

Finally, I'll present a diagram that is crucial to any software development project in this chapter. That diagram is the sitemap.

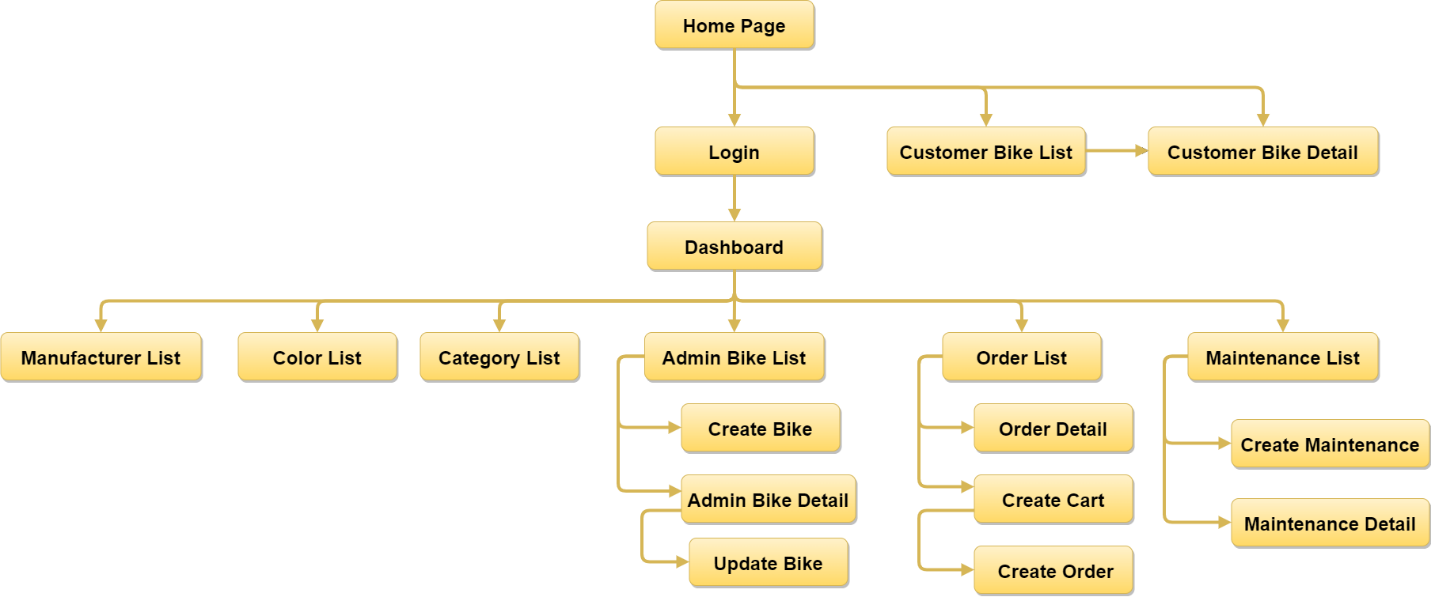
**4.5.1. What is Sitemap?**

A sitemap diagram for a software development project is a graphic depiction of the navigation and page structures within the project. The home page usually appears at the top of the sitemap diagram, followed by all the pages that are linked to it from other sites. The pages are typically grouped together based on their content or utility, and the diagram depicts the hierarchy of these pages and their relationships.

A basic sitemap will have some of the following components:

* Home page: This is the app's main page, which is typically the page users land on when they start the app.
* Parent page: This website has links to one or more of its child pages. The material that is covered by the child pages is typically summarized or given an overview on the parent page.
* Child page: This website is connected to a parent page. Child pages generally offer more in-depth details or functionality in relation to the subject matter or feature covered by the parent website.
* Navigation: This refers to the system of links or menus that users can use to navigate between pages in the program. There are two types of navigation: hierarchical (i.e., organized in a parent-child structure) and flat. (i.e., all pages are listed at the same level).

**4.5.2. Sitemap of Motorcycle Rental Management System.**



*Figure 6: Sitemap diagram of Motorcycle Rental Management System*

I'll describe this project's sitemap here. When a new project is launched, the Homepage is the first web page that is displayed. Both consumers and administrators are able to inspect this page. Customers can either log in or continue to view the three public Bike List and Bike Detail pages. Only administrators are currently given accounts to log into the system. The Dashboard is the first screen that the admin will see after successfully logging in. This is a website with charts displaying all of the project's material and processed data. Admins can continue to view various pages, the majority of which are list containers, from this point. All CRUD operations are managed within the three sections of Category, Color, and Manufacturer. There will be child pages to handle other extra tasks, particularly for pages that control other complex objects like Bike, Order, and Maintenance. Although the majority of functions also focus on CRUD, the information and processing required are more involved than in the first three pages. As a result, we have seen the Motorcycle Rental Management System project sitemap and are familiar with all of the project's formal websites.

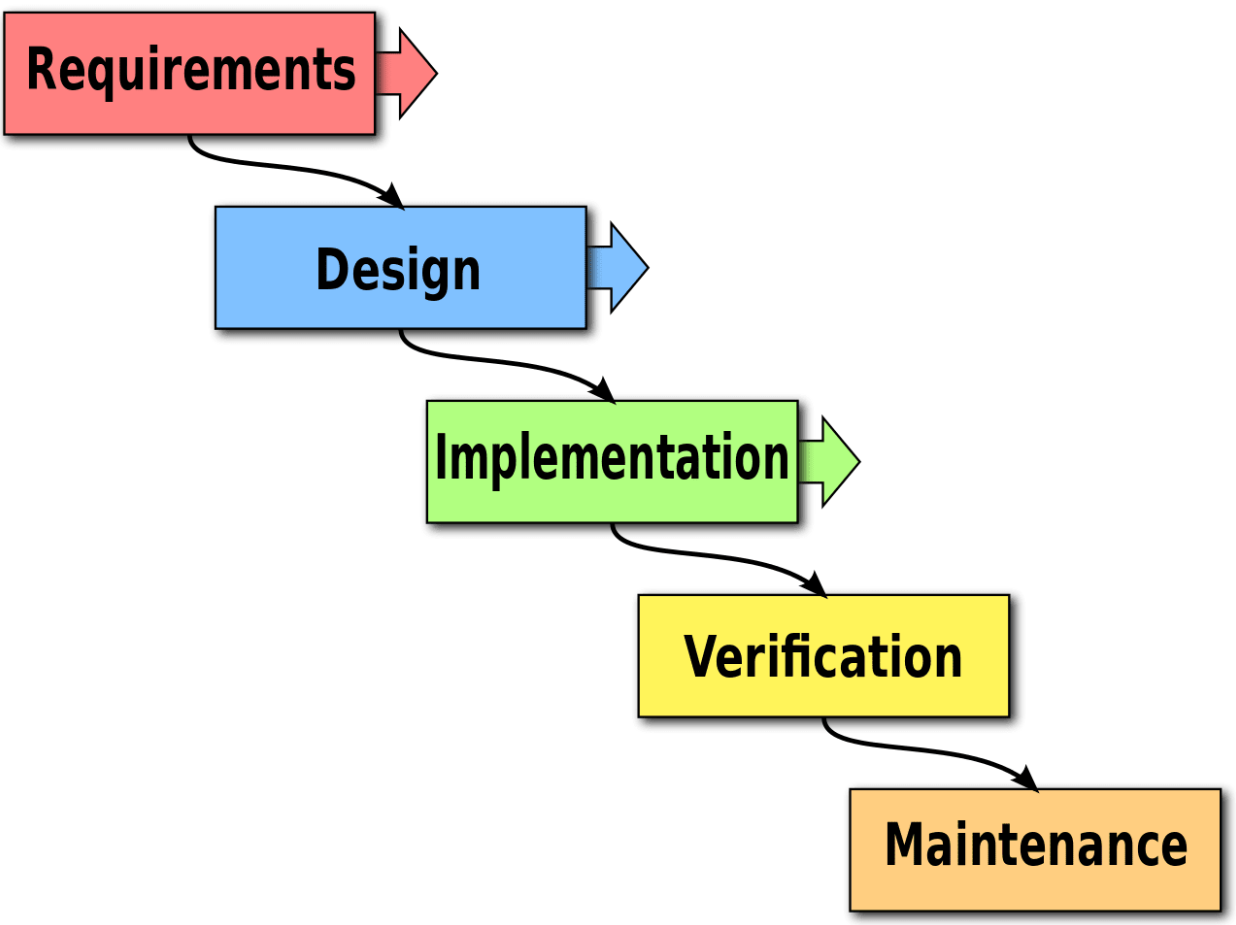
**Chapter 5: Review of Software Development Methodologies**

I'll present a few of the contemporary software development methodologies in this chapter. We will comprehend these methods' basic ideas as well as which ones were chosen for the Motorcycle Rental Management System project.

**5.1. Waterfall Model**

**5.1.1. What is waterfall model?**

The Waterfall methodology is a sequential software development procedure in which advancement occurs in a linear manner, much like a waterfall. This methodology, which has been in use for many years, was one of the first ones for software creation. The Waterfall model's strict linear method, in which each stage of the development process must be finished before moving on to the next, is its distinguishing characteristic.



*Figure 7: Waterfall model*

Waterfall is one of those long-standing models, and it also serves as the foundation for many other methods. We'll go over the phases of this waterfall model in the sections that follow.

* Gathering requirements: The project team collaborates closely with stakeholders during this period to identify and document the software system's requirements. This may entail performing interviews, surveys, or workshops to collect information about user needs, business requirements, and technical constraints. This step produces a detailed requirements document outlining what the software system should do and how it should work.
* Design: The design phase entails developing a comprehensive plan for the software system based on the requirements outlined in the previous phase. Technical specifications, system architecture, database design, and user interface design may be included in the blueprint. The design process may also include the creation of prototypes or mockups to assist stakeholders in visualizing the final product.
* Implementation: The software is produced in the implementation phase in accordance with the design specifications created in the prior phase. During this phase, code must be written, various software components must be integrated, and the software must be tested to make sure it complies with design requirements. Additionally, throughout this phase, developers might produce user guides and paperwork. A functioning software system that complies with the requirements and design standards is the phase's output.
* Testing/Verification: During the testing process, the software is examined to ensure that it complies with specifications and performs as intended. Unit testing, integration testing, system testing, and user acceptance testing are a few of the test types that might be included in this step. To expedite the testing procedure, developers can also use automatic testing tools. An extensively tested software system that satisfies the project's quality requirements is the result of this period.
* Maintenance: The software system is deployed and used in a production environment during the maintenance phase of the waterfall model, which is the last step of the software development life cycle. To make sure that the software continues to meet the requirements of users and stakeholders, the emphasis during this phase is on maintaining and supporting it.

**5.1.2. Advantages of waterfall model**

* Structured and well-defined: The waterfall model is a highly organized and sequential method of developing software. The method is well-defined, and each stage has clear deliverables. Planning, monitoring development, and resource management become simpler as a result.
* Simple to comprehend and manage: Even non-technical stakeholders can easily manage and comprehend the waterfall model. Since the method is linear, it is simple to monitor development and make sure the project doesn't veer off course.
* Clear documentation: The waterfall model has distinct deliverables for each step, which results in a well-defined and well-documented process for developing software. As a result, it is simpler to maintain the software over time and transfer it, if required, to new team members.
* Early problem detection: A testing phase follows the production phase in the waterfall model. This means that problems are found early on in the process, when they can be fixed more easily and cheaply.
* Cost-effective: The waterfall model is a more efficient way to create software because it is highly structured and well-documented. Cost overruns are less likely because it enables project teams to prepare and allocate resources more effectively.
* Ideal for small tasks with well-defined requirements: For small tasks with clear requirements, the waterfall model is perfect. Even for users who are not technically savvy, it is a straightforward and easy-to-understand method of developing software.

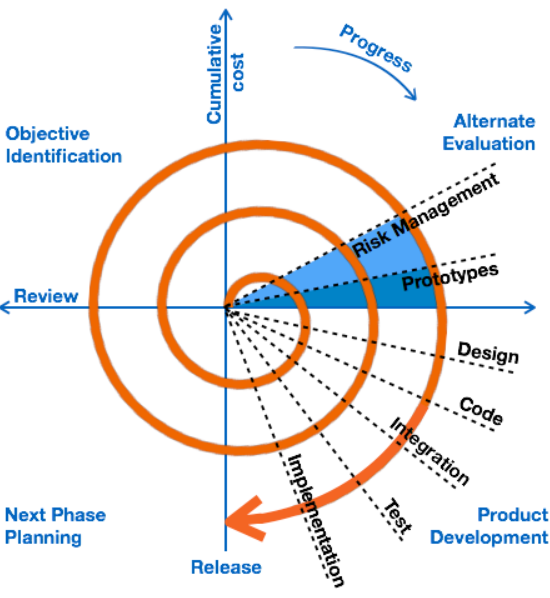
**5.1.3. Disadvantages of waterfall model**

* Rigid and inflexible: The waterfall model is a highly structured, sequential, and occasionally rigid method of software creation. It is challenging to make adjustments after a phase is finished without affecting other project phases.
* Limited scope for feedback: There is little opportunity for input from stakeholders or end users because the waterfall model is a linear process. As a result, problems or changes might not be discovered until much later in the project, when fixing them will be more difficult and costly.
* High risk of failure: If requirements are not clearly specified or if the project scope changes significantly while the project is being developed, the waterfall model is highly likely to fail. Since each project step depends on the one before it, a failure in one phase can have an impact on the entire project.
* Time-consuming: Software development using the waterfall model can take a long period, particularly for large and complex projects. Since the process is linear, each step must be finished before moving on to the next, which can cause delays and extend the development cycle.
* Creativity is restricted: The waterfall model's fixed structure and lack of flexibility can restrict software developers' ability to be innovative and creative. The requirements and design guidelines may impose restrictions on developers, which may stifle their ingenuity and reduce the software's potential.
* Adaptability is limited: The waterfall model is not suitable for tasks with changing or evolving requirements because it is a linear procedure. It might be difficult or even impossible to adjust the requirements if they change substantially throughout the project without incurring significant costs or delays.

**5.2. Spiral**

**5.2.1. What is Spiral model?**

The spiral methodology is a paradigm for the risk-driven software development process that combines aspects of iterative development and the waterfall model. The model was first put forth by Barry Boehm in 1986 and is frequently used in software development projects where there is a significant risk of failure.



*Figure 8: Spiral model*

The spiral methodology is founded on the idea of a spiral, with each loop signifying a stage of the software development process. The model has four major phases, which are as follows:

* Planning: The project objectives are established, the project's scope is established, and its requirements are determined during this period. A risk management strategy is created after a risk assessment of the undertaking.
* Analysis of Risk: The risks discovered during the planning phase are examined and assessed during this phase. Finding the risks with the highest priority and creating a strategy to mitigate them are the goals.
* Creating and Testing: This stage involves the iterative development and testing of the program. A better and more streamlined version of the software emerges after each round of testing and development.
* Evaluation: The software is assessed in this step to see if it satisfies the project's requirements and goals. The evaluation's findings are applied to the software's development and efficiency enhancement.

Following the evaluation phase, the project team goes on to the spiral's subsequent iteration, beginning anew with the planning phase. The spiral's iterations build on one another, taking input into account and addressing risks and problems as they come up.

The spiral model is especially effective for big, complicated undertakings with lots of risks and uncertainties. It enables project teams to address risks and issues early in the development process, reducing the chance of project delays or failures. The spiral model's continuous structure also enables project teams to adjust and react to evolving needs or conditions as the project advances.

**5.2.2. Advantages of Spiral model**

* Risk management: In terms of software development, the spiral model takes a risk-based strategy. It places a focus on early risk detection and reduction through a continuous process of prototyping, testing, and feedback. As a result, it works well for tasks that involve a lot of risk and uncertainty.
* Flexibility: The Spiral model is incredibly adaptable and enables changes to be made at any point during the creation process. It is the perfect choice for projects where specifications might be hazy or changeable because it is simple to adapt to changes in requirements or design.
* Effective resource management: The Spiral model permits a gradual and iterative growth process, which aids in the efficient use of resources. In addition to lowering the possibility of cost overruns or delays, this makes project cost and timetable management simpler.
* Stakeholder engagement has improved: The Spiral model incorporates routine review and feedback cycles, which can enhance stakeholder involvement and guarantee that the final product meets their needs and expectations.
* Product of higher quality: The Spiral model places a strong emphasis on testing and quality control throughout the entire production process. Products made as a result are of a better caliber, more dependable, and are less likely to malfunction in use.
* Better communication: Between team members, stakeholders, and end users, the Spiral model fosters frequent communication and cooperation. This makes it easier to make sure that everyone is on the same page and that problems and concerns are dealt with right away.

**5.2.3. Disadvantages of Spiral model**

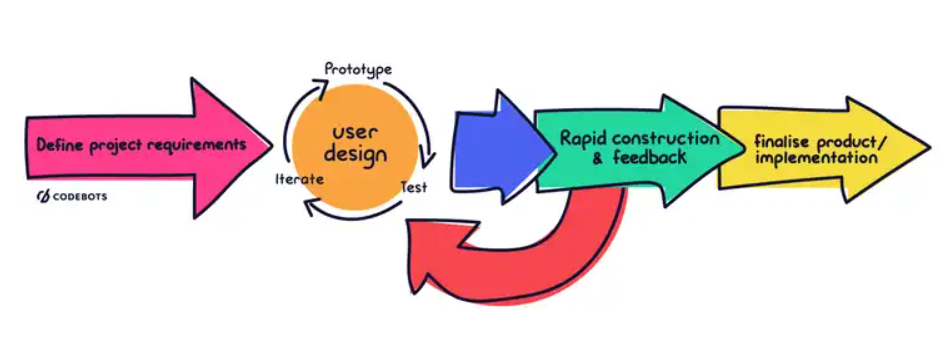
* Complexity: The Spiral model is a complicated procedure that can be challenging to handle, especially for smaller projects. It necessitates a high degree of risk management and project planning expertise, making it unsuitable for teams with little background or available resources.
* Time-consuming: The Spiral model is iterative, which can take time, especially if numerous iterations are needed. This could lead to lengthier development times and higher expenses, which might not be practical for all projects.
* Cost estimation challenges: The Spiral model's flexible and iterative approach can make it challenging to correctly estimate project costs and timelines. This could result in budget overruns or delays, which could be problematic for some initiatives.
* Dependence on risk evaluation: The accuracy and efficiency of risk analysis are crucial components of the Spiral model's performance. Project failures or delays may occur if risks are not accurately identified or successfully managed.

**5.3. RAD (Prototyping)**

**5.3.1. What is RAD model?**

The Rapid Application Development (RAD) model is a software development technique that prioritizes rapid prototyping and iterative development. It was created in reaction to the demand for more rapid software development cycles and resource efficiency.

Unlike the Spiral model, the RAD model stresses rapid prototyping and rapid response cycles, whereas the Spiral model focuses on risk management and includes more planning and review stages. Furthermore, this methodology is intended to save time and money in comparison to Spiral. The similarity between RAD and Spiral is that both are extremely adaptable and can take into account modifications to specifications or design as they occur during development.



*Figure 9: RAD model*

Similar to Spiral, RAD has the following 4 phrases:

* Planning for requirements: During this phase, the software requirements are identified and prioritized. Interviews with stakeholders and end consumers are frequently used to accomplish this.
* Rapid prototyping: The program is created in this phase as a functioning prototype. This prototype is used to collect input from stakeholders and end users and to fine-tune the software's specifications.
* Iterative development: During this phase, the software is created iteratively, with each iteration building on the one before. This provides for faster software development and testing, as well as quick responses to changing requirements or feedback.
* Deployment: The software is tested, implemented, and maintained during this period. This includes user training, documentation, as well as continuous assistance and maintenance.

**5.3.2. Advantages of RAD model**

* More rapid time to market: The RAD model's focus on rapid prototyping and feedback can speed up the development cycle and help bring a functional product to market more quickly.
* Increased stakeholder engagement: The RAD model promotes close cooperation among developers, stakeholders, and end users, which can result in a better grasp of requirements and greater buy-in from all parties.
* Reduced production costs: The RAD model can lower development costs by identifying and resolving problems early in the process by concentrating on rapid prototyping and iterative development.
* Greater flexibility: The RAD model allows for quick iteration and adaptation to shifting needs, making it ideal for projects that call for frequent updates and changes.
* Improved quality: The RAD model places a lot of stress on testing and feedback, which can enhance the quality of the software and lower the likelihood of bugs or errors.

**5.3.3. Disadvantages of RAD model**

* Lack of comprehensive documentation: The RAD model's rapid prototyping and iterative development approach can lead to a lack of thorough documentation, making it challenging to support and manage the software in the long run.
* High dependency on the expertise of developers: To deliver software on time, the RAD model needs highly skilled and experienced developers who can work quickly and effectively.
* Greater risk of scope creep: As a result of the RAD model's heavy reliance on stakeholder input and feedback, scope creep may become more likely as stakeholders ask for extra features or adjustments while the project is still being developed.
* Limited suitability for large-scale projects: The RAD model works best for shorter-term, iterative tasks that are of a smaller scope. It might be necessary to take a more structured strategy for bigger, more complicated projects.
* Increased risk of project failure: Rapid prototyping and iterative development can lead to a lack of stability and consistency, which, if not managed correctly, can raise the risk of project failure.

**5.4. Agile**

Unlike the previous models, Agile is considered as knowledge rather than a model. Instead of focusing on project management, this method focuses the value it contributes. We'll discover what Agile is right here.

**5.4.1. What is Agile methodology?**

Agile methodology is a method of project management that places a strong emphasis on adaptability, teamwork, and ongoing development. It was created in response to the limitations of conventional project management methods, which frequently include rigid planning, documentation, and an emphasis on sticking to a predetermined plan.

A set of ideals and principles emphasized in the Agile Manifesto that define the agile methodology include:

* Individuals and interactions over processes and tools: This value emphasizes the significance of individuals in the process of development. In contrast to following rigid procedures or depending solely on tools, agile methodology values teamwork and communication between members.
* Working software over comprehensive documentation: This value places a greater emphasis on delivering a functional product to the customer than on devoting time to producing copious documentation. According to agile methodology, documentation that might not add value to the finished product is valued over a product that satisfies the requirements of the customer and can be tested and evaluated.
* Customer collaboration over contract negotiation: This value emphasizes the worth of close customer collaboration throughout the development process. Instead of depending on a contract to specify the project's requirements, the agile methodology values regular communication with the customer to ensure that the product being created meets their needs.
* Responding to change over following a plan: This value points out the importance it is to be adaptable and flexible in the face of shifting demands. Instead of sticking to a rigid plan that might no longer be applicable, agile methodology values the ability to reorient and modify the development process in response to customer feedback or shifting market circumstances.

Agile methodology usually entails breaking down a project into smaller, more manageable tasks known as "sprints," which are finished in short periods of time. The team in charge of the project collaborates closely, working on each sprint and modifying their strategy as required in response to input from stakeholders.

The agile approach is frequently used in software development, but it can be used on any project that calls for adaptability and a commitment to ongoing improvement.

**5.4.2. Advantages of Agile methodology.**

* Flexibility: Agile methodologies are created to be adaptable and flexible to changes in project requirements, enabling teams to react rapidly to new information or changing needs.
* Enhanced collaboration: Collaboration between team members, clients, and stakeholders is encouraged by the agile methodology, which may result in a clearer grasp of the project objectives and a more effective workflow.
* More rapid time to market Agile methodology places a strong emphasis on providing functional software frequently, which can reduce time-to-market and hasten the delivery of value to customers.
* Quality improvement: Throughout the project, the agile methodology places a strong emphasis on testing and quality assurance, producing deliverables of better quality.
* Continuous improvement: The goal of the agile methodology is to promote continuous development through frequent feedback, retrospectives, and iteration, resulting in better results over time.
* Greater transparency: Agile methodologies encourage openness and dialogue both within the team and with stakeholders, which can help to prevent misunderstandings and make sure that everyone is on the same page regarding the objectives and status of the project.
* Enhanced client satisfaction: Agile methodology places a strong emphasis on customer feedback and collaboration, which can result in a greater comprehension of customers' requirements and a higher level of end-user satisfaction.

**5.4.3. Disadvantages of Agile**

* Limited predictability: Because the agile technique depends on flexibility and adaptation, it can be challenging to forecast project timelines or results.
* Requires skilled team members: Teams using the agile methodology need members with the right skills and expertise who can work well together and act quickly.
* Can be difficult to scale: Agile methodology depends on close collaboration and frequent communication, so scaling it to bigger teams or more complicated projects can be challenging.
* Can be disruptive: In some organizations, the agile methodology may be difficult to adopt because it can disrupt established workflows and processes.
* Risk of scope creep: Agile methodologies are prone to scope creep because changes to project requirements and objectives are easily accommodated.
* Reliance on customer availability: Because the agile methodology depends on customer collaboration and input, it can be challenging if customers are not accessible or responsive.

**5.5. Our selection of a software development methodologies and our justification**

We've gone over the four most prevalent types of methodologies used in software development projects. Now I'll show everyone the approach that I chose and used for the Motorcycle Rental Management System project. Waterfall is the model that I selected. I'll explain why I selected this fundamental model over others at a higher level, such as RAD or Spiral.

First, I selected the Waterfall model because the project scope is so small. In contrast to actual initiatives from companies, the Motorcycle Rental Management System was developed with the goal of learning and mastering cutting-edge technologies like Spring Boot and ReactJS. It's simple to see how long a project takes from the outside for a real job. Also, smaller projects are completed in one to two years. Mid-range projects last between three and ten years. In addition, big projects can go on for even longer than ten years. A significant number of employees are involved in these projects. Therefore, in order to effectively manage these initiatives, we will need to use higher-level and iterative methodologies to lower risks and maintain stability. Regarding staff size, I am the only person in control of all aspects of the project, including gathering requirements, analyzing, planning, creating products, testing, and writing reports. So it makes sense to use the Waterfall model for a project of this scale.

The second one is about the requirement aspect. unlike Agile or Spiral, where clients can continuously change requirements. I only gather client requirements once because I only have a limited amount of time. It is impossible to delete and restart the generated functions. They can only be updated to make them more comprehensive. As we can see, the project method moves from planning to testing in a single direction. I won't have enough time to accommodate all of the customer's requests. During the product testing stage, I can only provide additional interface updates or correct customer-reported defects. That is one of the objective considerations in selecting the Waterfall model.

A final point on time management. We all know that the Waterfall model is simple to use because it only has a few phases. The initial phase can be completed before beginning the following phase. So, when a phase ends, we can no longer worry about it and move on to the next. This significantly reduced the amount of time I needed to run this model. In order to initiate repeated phases for other models, such as RAD or Spiral, I must spend more time in scheduling and keeping track of them. I might use that squandered time to improve my product or fix bugs, for example. That's why it was crucial that I used the Waterfall model for this undertaking.

In summary, a lot will depend on the size, scope, and resources of the project when deciding on a model. To save time, money, and effort on a small project run by just one individual, the simplest models should be used. Nevertheless, using a waterfall model comes with a number of disadvantages, including the inability to manage risks and a lack of adaptability to shifting requirements. However, it enabled me in finishing the Motorcycle Rental Management System project on schedule and on budget.

**Chapter 6: Design and Implementation of our demo product**

This part will contain the product's design, actual interfaces, and source code. In addition, I'm going to provide my personal evaluation of the positives and negatives of this product.

**6.1. Product Analysis and Design**

I will start by discussing the project's UI design. We are all aware that User Interface (UI) design is an important component of software development because it has a big impact on the project's success. A well-designed user interface can improve the software's usability and user experience, which can increase user engagement and customer satisfaction. The user's wants and preferences must be taken into account when creating the UI because they have a direct impact on the software's overall usability and functionality. Designing the user interface (UI) with accessibility and usability for people with impairments in mind is crucial for inclusivity. A software product that invests in good UI design will likely be more successful and user-friendly.

**6.1.1. What is wireframe?**

A wireframe is a graphic depiction or skeletal framework of a user interface that shows the fundamental organization, structure, and operation of a design. It is often produced in the first phases of the design process to assist designers and stakeholders in visualizing and comprehending the overall structure and features of the user interface.

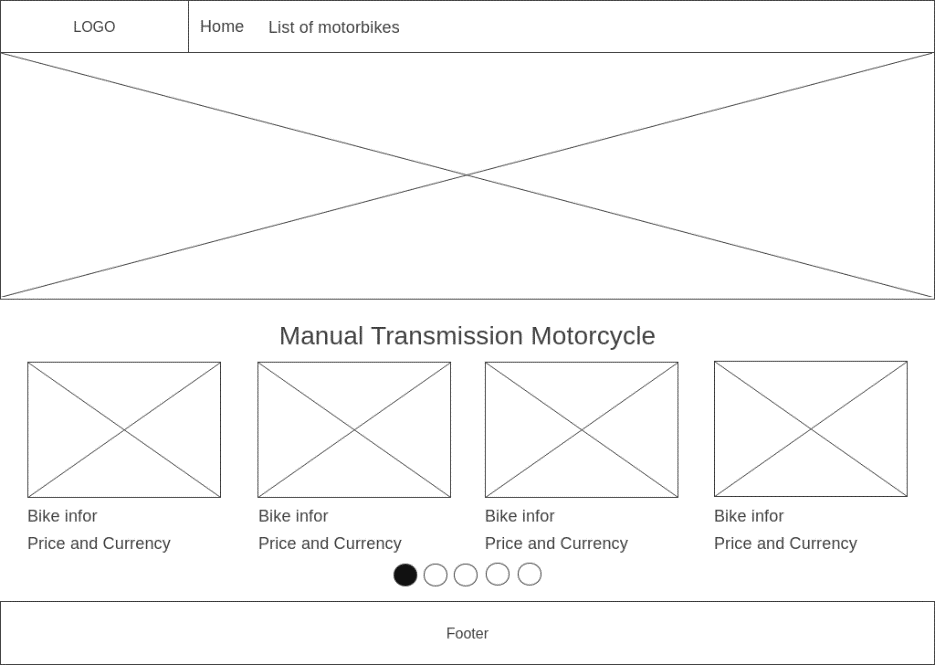
A wireframe is typically made out of simple forms and lines that represent the various interface elements, such as buttons, text, and images, without including visual design aspects such as color, typography, or branding. By using wireframes as a guide, the design team may more quickly decide where to put each element and how the interface should be used overall, ensuring that the finished product fits the needs and expectations of the user.

I will present some common terms of wireframe:

* Layout: The arrangement of interface elements, including their placement, size, and spacing.
* Grid: A collection of standards that assists designers in aligning objects on the interface and creating a consistent visual framework.
* Placeholder: A transitory graphic element used to represent future information such as text, photos, or icons.
* Elements of a Wireframe: Buttons, input fields, text blocks, and images are examples of wireframe building blocks.
* Annotations: Add descriptive comments or labels to the wireframe to explain how the UI elements work.
* Navigation: The system of links or buttons that allows users to navigate the interface and access various pages or sections.

**6.1.2. Wireframe of Motorcycle Rental Management System project.**

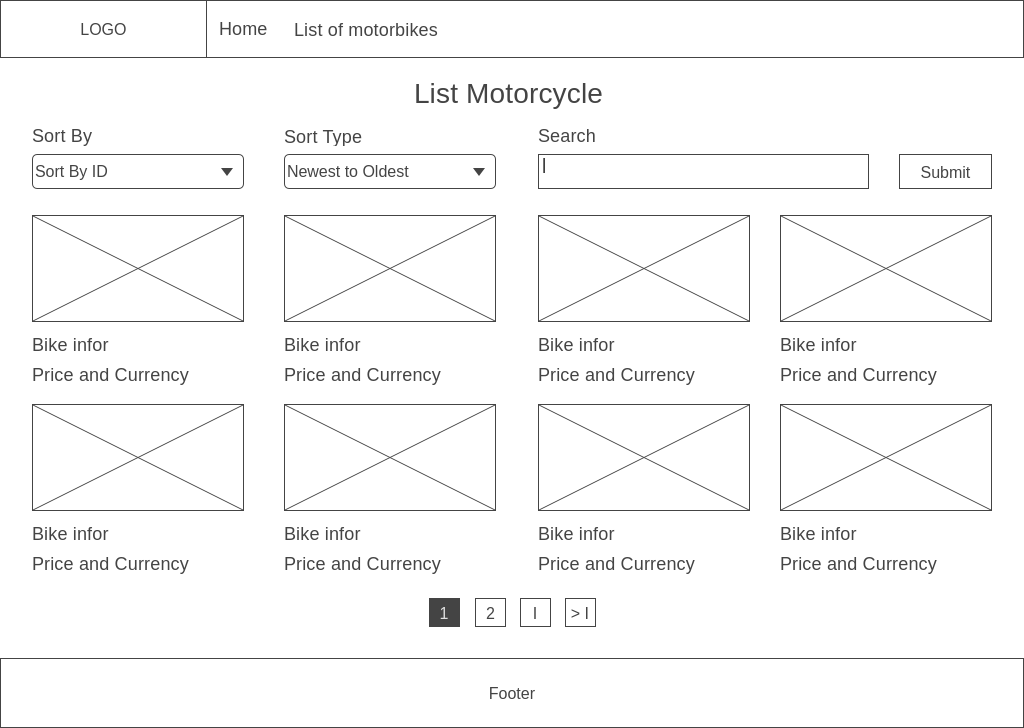
**6.1.2.1. Home Page**



*Figure 10: Homepage wireframe*

The homepage has three simple sections: a header, a body, and a footer. The menu will be in the header and utilized on all pages that are accessible to the public. A logo will be on the left side of the menu. There will be some elements in the Body section, including a sizable banner. In addition, there are few common motorbikes that can be rented. Finally, the Footer will have a contact list for renting a vehicle.

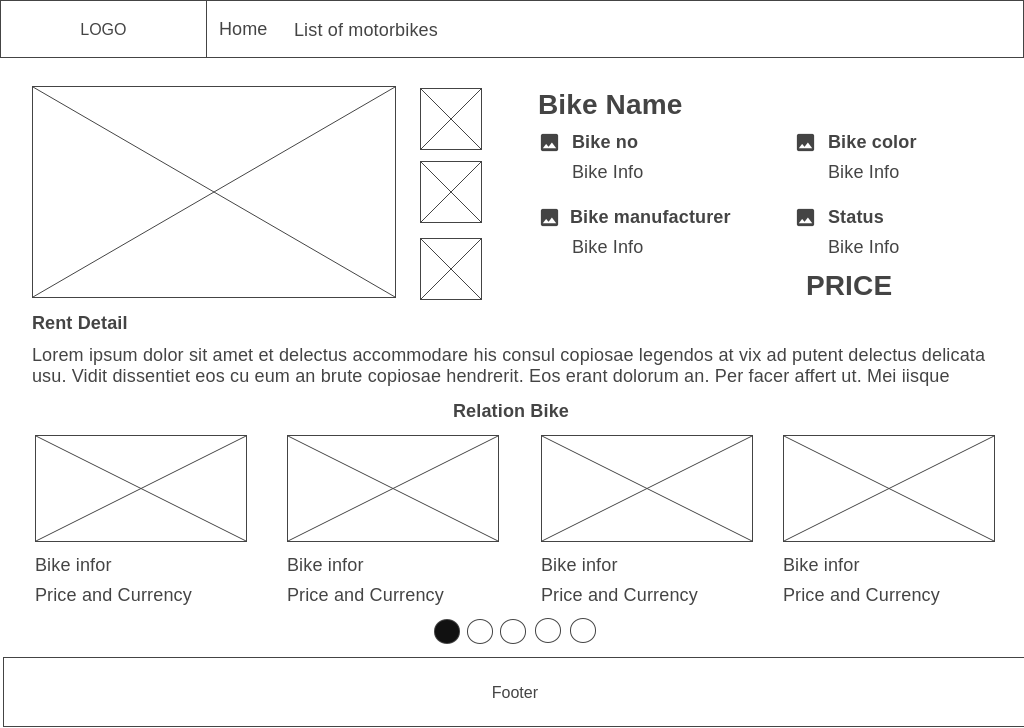
**6.1.2.2. Public Bike List Page**



*Figure 11: Public bike list wireframe*

This page lists motorcycles along with their prices. We can see that while the content will be more distinctive, the header and footer will be comparable to the site. Customers will be able to quickly filter the list of automobiles according to their preferences thanks to Body's sort and search style.

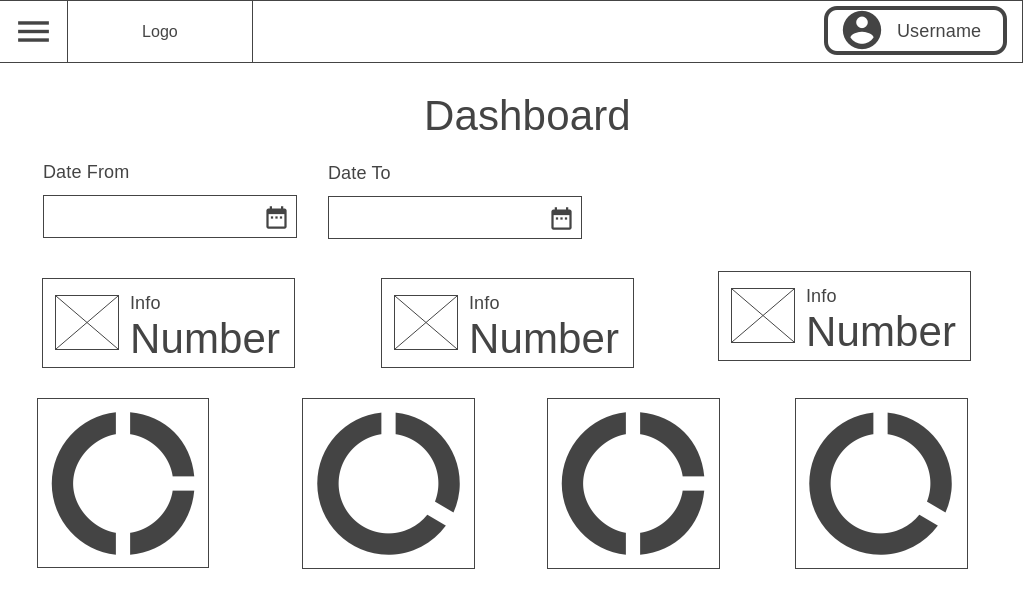
**6.1.2.3. Public Bike Detail Page**



*Figure 12: Public bike detail wireframe*

This is the web interface used to display a motorcycle's specifics. As we can see, the body is split into two sections that contain the motorcycle's information and related motorcycle information. With the motorcycle's actual photograph, all relevant details will be displayed above. A list of some motorcycles related to modern motorcycles is shown below, organized by motorbike type.

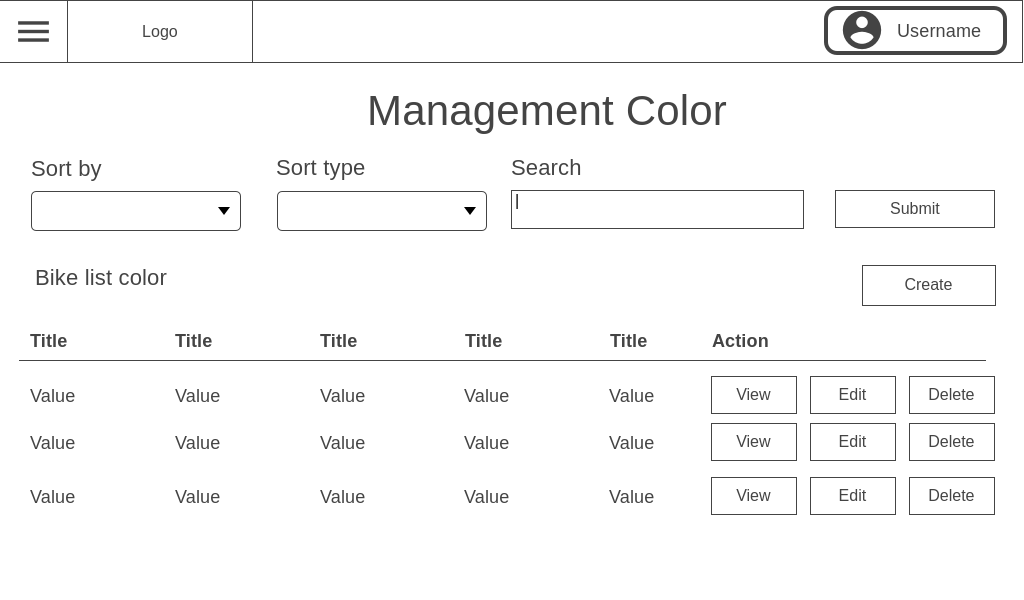
**6.1.2.4. Dashboard Page**



*Figure 13: Dashboard wireframe*

The dashboard, the initial webpage utilized by the admin, will be covered in the wireframe section. Authenticated pages do not have a Footer, unlike public pages. The header will also be distinct from the public page's header. The username will be located on the right side of the header of the authorized page, namely as follows. When the user clicks on the username, the logout button is likewise included. A button to access the side bar menu will be situated to the left. All authenticated pages will have the same structure. As we approach the dashboard page, we will observe that there are several design components to filter the time with corresponding data and graphics based on the requirements of the customer. According to customer requirements, the actual dashboard will include additional sorts of charts.

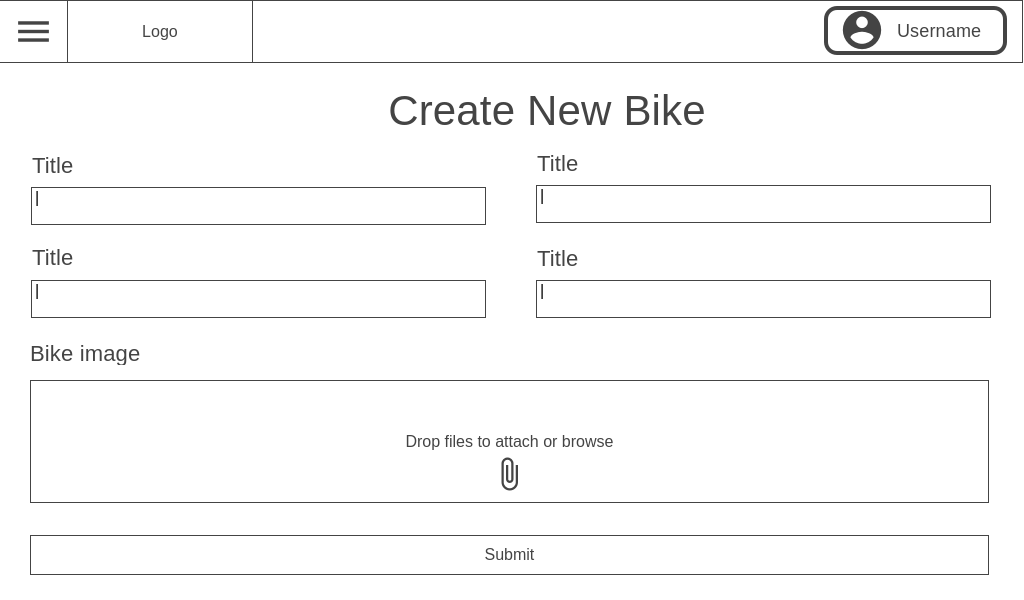
**6.1.2.5. Management List Page**



*Figure 14: Management List wireframe*

This is the interface that displays the item list and is utilized by the majority of management pages. The pages Management Color, Manufacturer, Category, Bike, Order, and Mantainence will all apply this template. Only the filter options on the table and the buttons in the Action column are different. Each page will have a unique design for these two sections based on the requirements of the customer. However, all pages will still use this structure to display the item list.

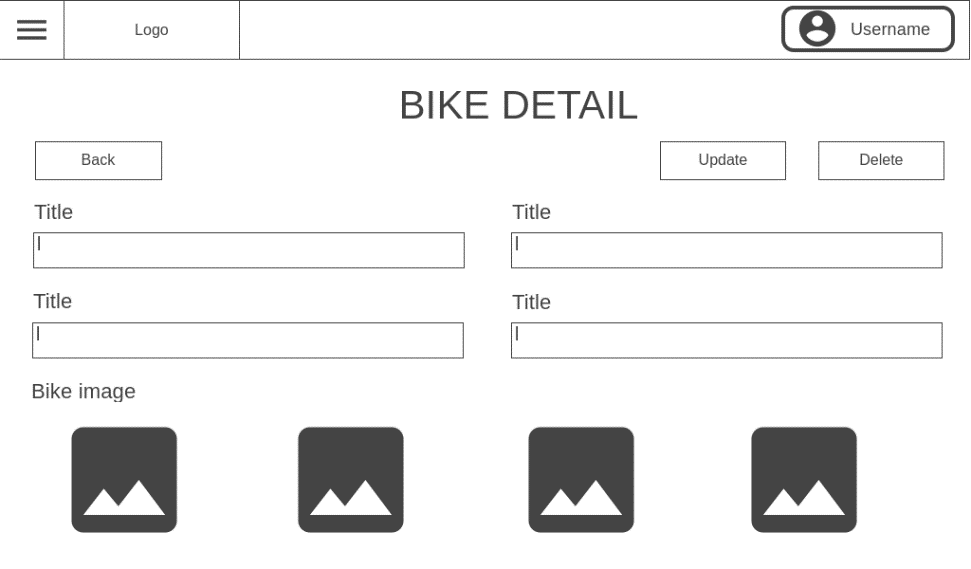
**6.1.2.6. Create Bike Page**



*Figure 15: Create Bike wireframe*

The body of the create bike page will be a form for the administrator to fill up with information about the motorcycle. The information that has to be filled in can either be manually typed in or chosen from a select box. An interface for adding motorcycle photographs can be seen below. For the bike update page, this template will be implemented.

**6.1.2.7. Bike Detail Page**



*Figure 16: Bike Detail wireframe*

This is the user interface allowing administrators to access a motorcycle's specific information. On this page, all data fields will be disabled and unavailable for user editing. There will be two Update buttons in the upper right corner, one leading to the Update bike page and the other to the vehicle deletion feature. A Back button to go back to the management bike list page will be located on the left. Finally, there will be a maximum of 4 pictures of the car below.

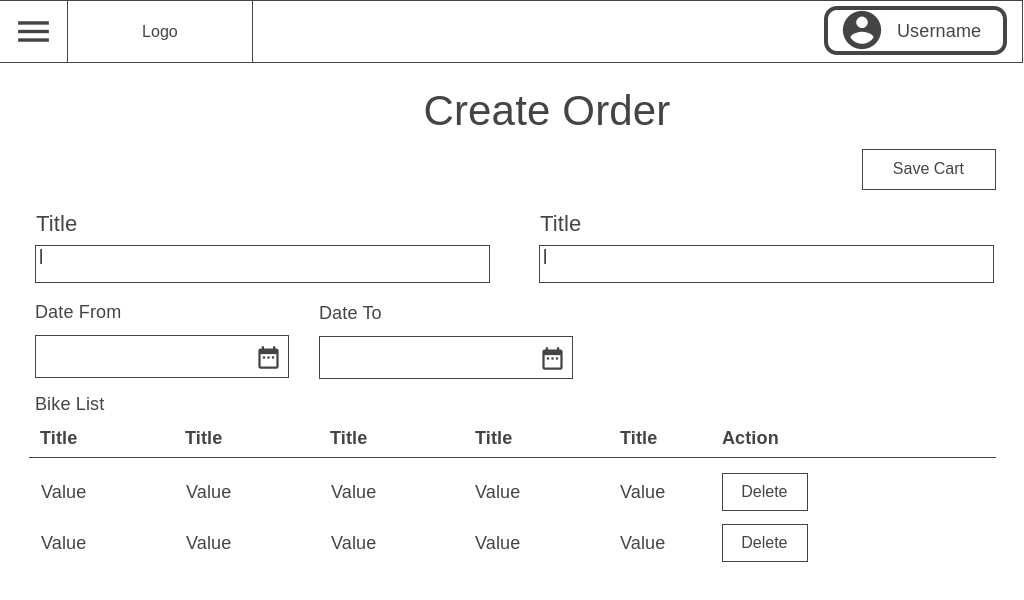
**6.1.2.8. Create Cart Page**



*Figure 17: Create Cart wireframe*

I'll then go over the website's design for creating carts. Why is there a cart? I was asked to develop this website as an e-commerce site since the client wanted to quickly and conveniently add an automobile to the order. As we can see, every motorcycle is now a separate item that can be temporarily added to the cart by clicking the Add button. There will be a cart icon and a number indicating how many cars are in the cart for the filter area on the right. This button will send us to the website where we may create an order.

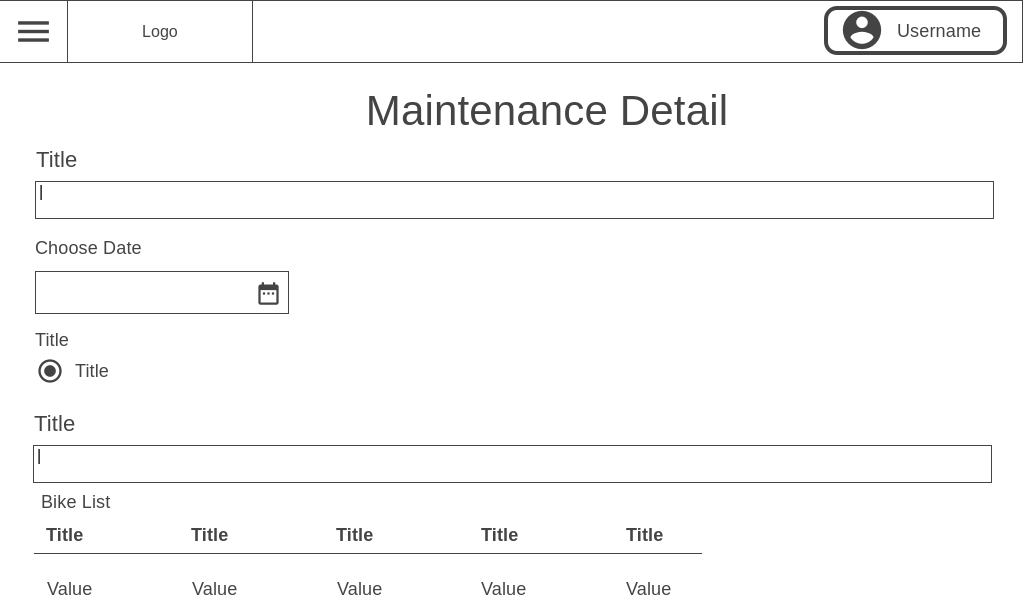
**6.1.2.9. Create Order Page**



*Figure 18: Create Order wireframe*

Following that, I'll go over the UI of the create order page. Similar to the create bike page, this website will also use a form to enter the order details. The difference here is that the list of motorcycles added to the cart is shown in the form of a table. Except for motorcycles, there will be a Save button in the upper right corner that allows users to save the entered information at any moment. This layout will be used for the order detail page. The only difference is that if Bike has a status other than PENDING, the fields on the order detail page will be blocked.

**6.1.2.9. Maintenance Detail Page**



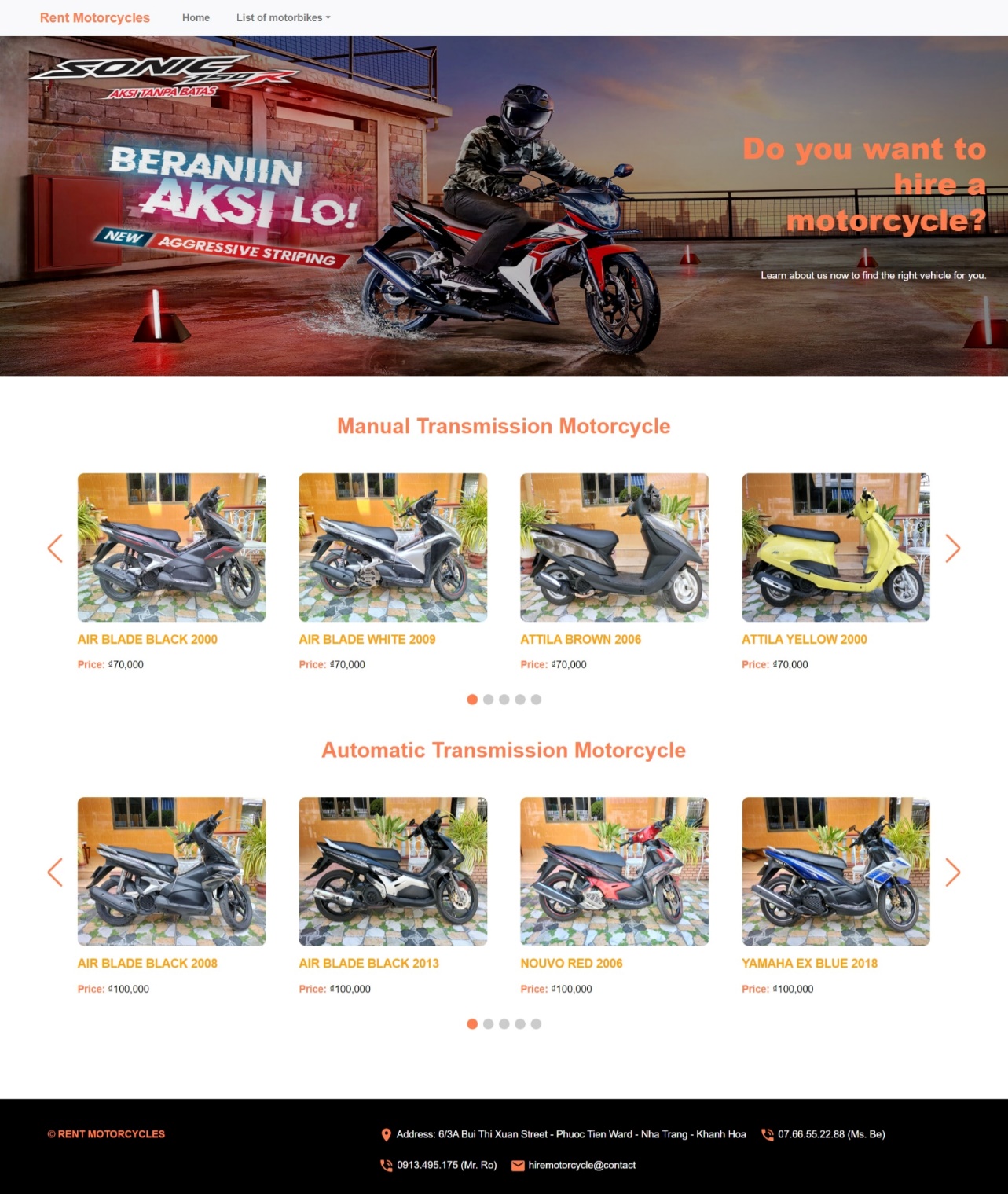
*Figure 19: Maintenance Detail wireframe*

Finally, I'll discuss the Maintenance Detail page's layout in this part. This is the user interface for creating a maintenance page just like other create pages. It also has a table that displays a list of automobiles, similar to the one on the create order page. The distinction is that this table won't have a button-filled action column. Additionally, this theme is utilized for the create maintenance page. That page will not contain a bike list table.

**6.2. Features include with screenshots**

I'm excited to announce that I've just finished the wireframe for our latest project. It's been an amazing process, and I'm excited to provide some project interface screenshots for a few critical sections.

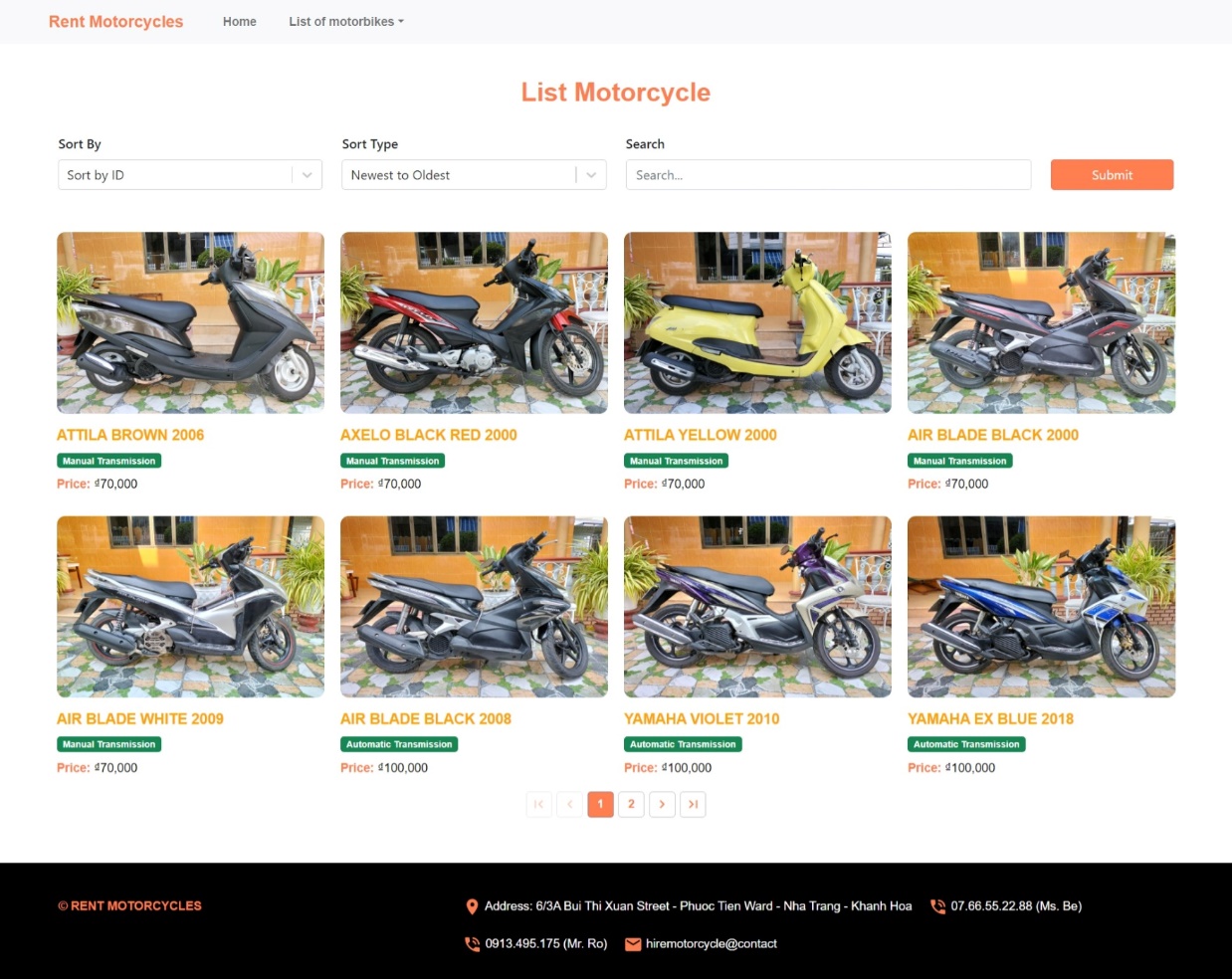
**6.2.1. Home page**



*Figure 20: Public homepage*

This is the actual interface of the Homepage page. Because this is a full-page image, we can see that each component is quite little. In actuality, a computer frame can only display half of this website. This page is made up of three parts, as I mentioned in the wireframe section. The menu is located in the header, the body comprises two lists of vehicles and banner, Manual Transmission and Automatic Transmission, and the footer contains the motorbike lessor's contact information.

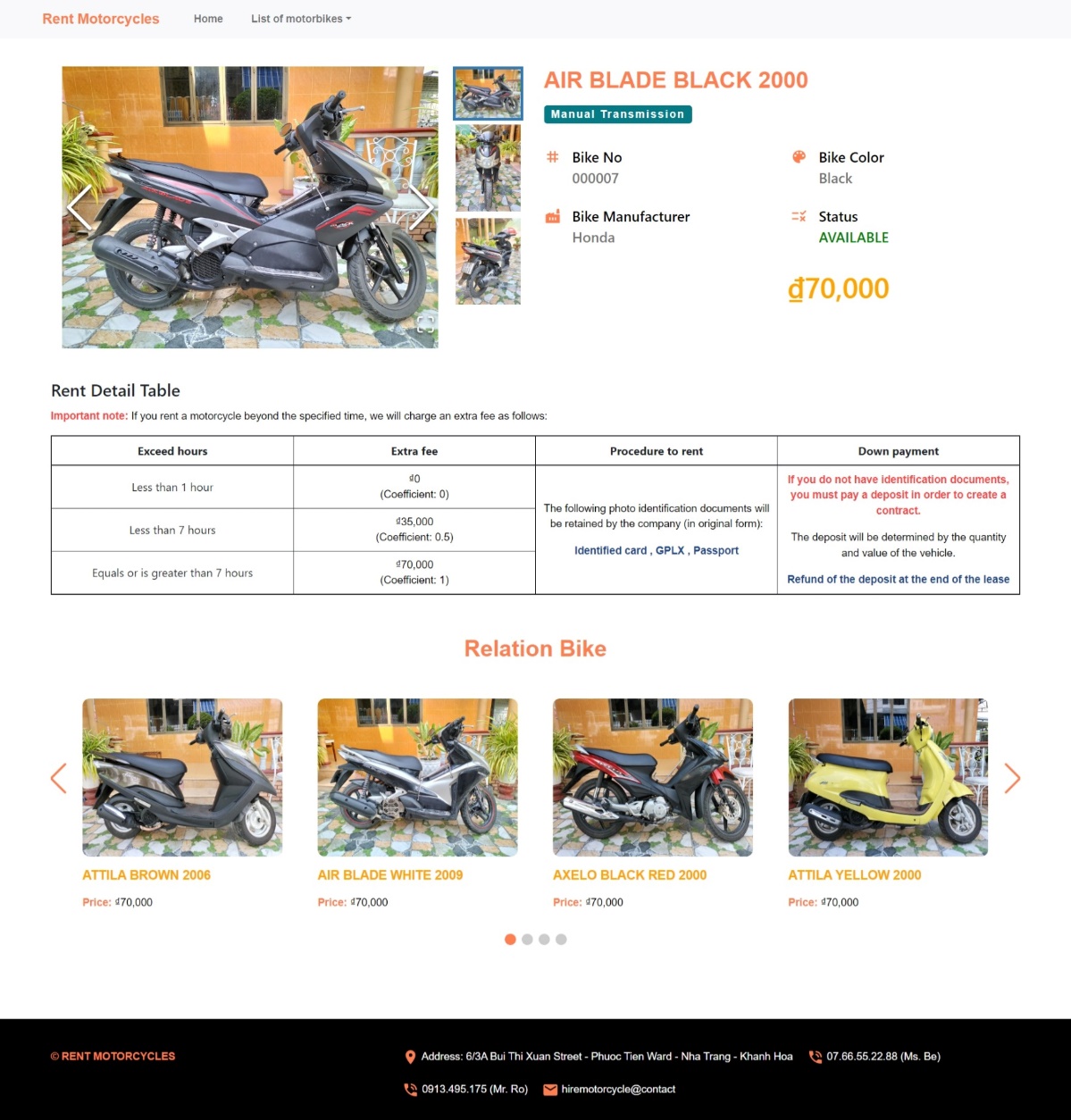
**6.2.2. Public Bike List page**



*Figure 21: Public bike list*

The following is a screenshot of the bike list page interface, which is accessible to the general public. The only difference between this page and the Homepage is the body. It includes basic filters that allow users to organize and search for vehicle information. The list of automobiles below is paginated, with 8 items per page. The UI of this page has essentially no differences from the wireframe design.

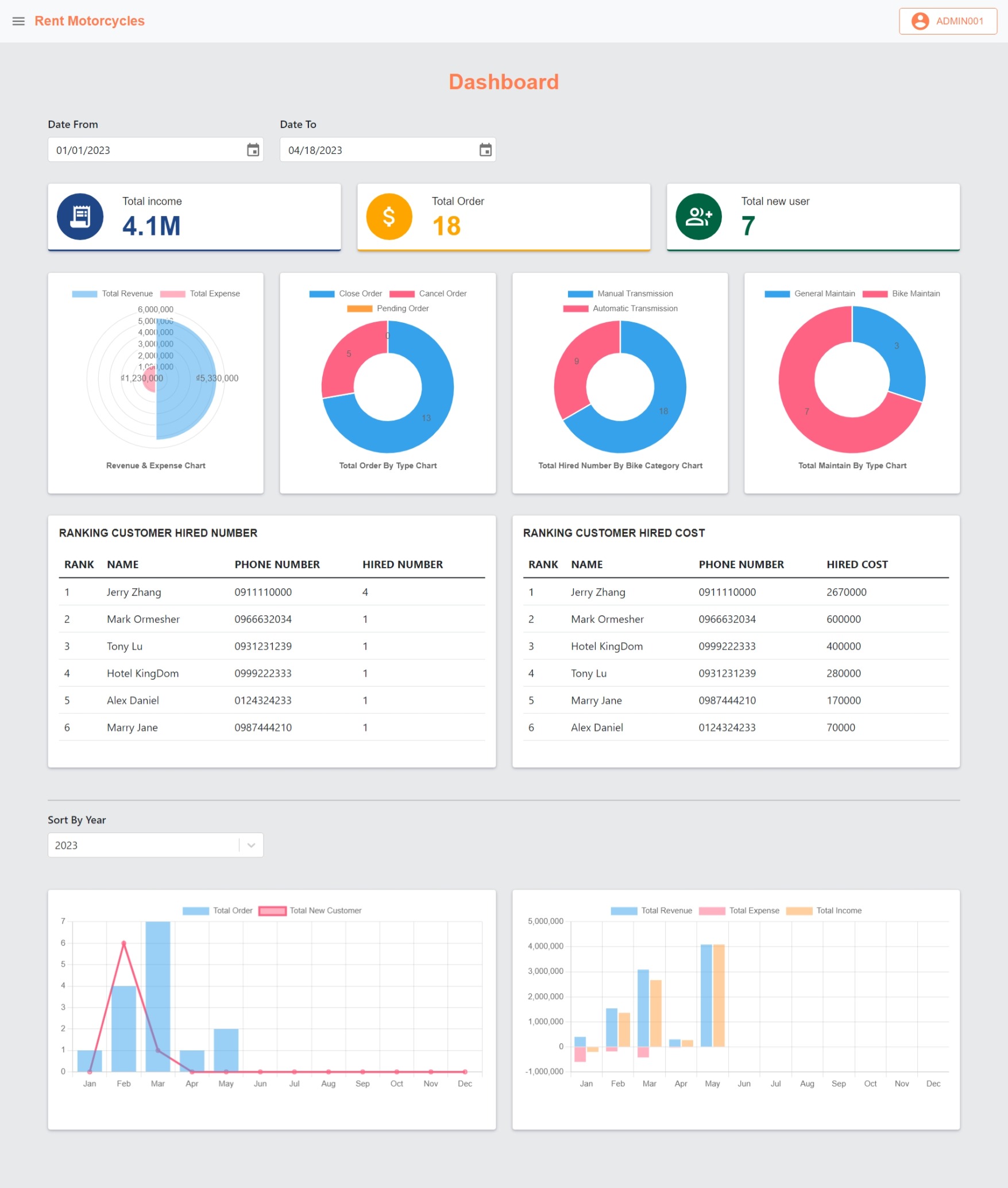
**6.2.3. Public Bike List page**



*Figure 22: Public bike detail*

Following that, I'll go over the official interface for the public bike detail page. The Body's head contains the same information as the wireframe design. This interface now includes a table with automobile rental calculations and a list of similar vehicles below. The user can view the bike image or zoom it out. The upper right corner displays the status of the motorcycle's rental as well as the rental price.

**6.2.4. Dashboard page**

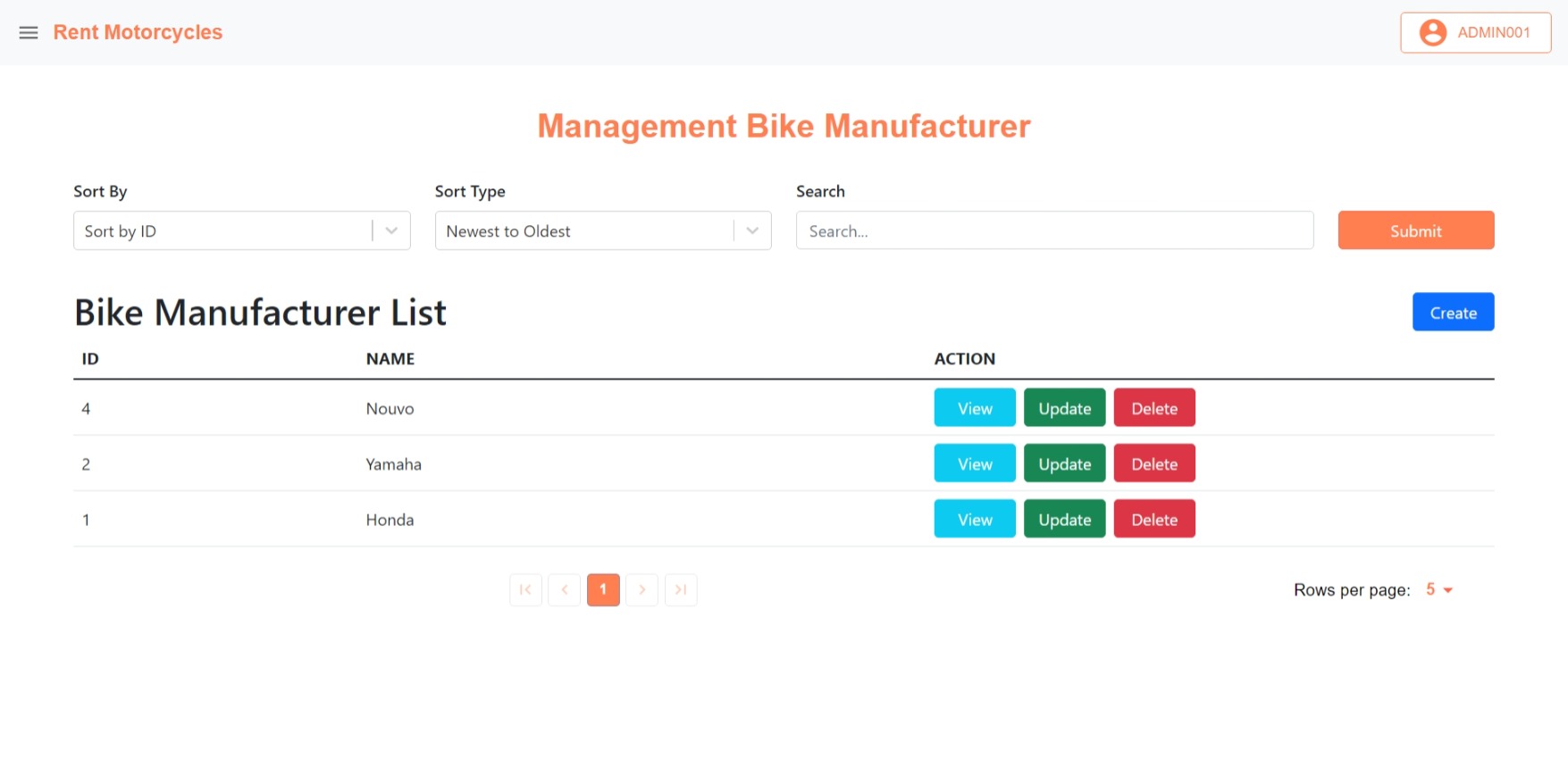


*Figure 23: Dashboard*

The following image depicts the project's Dashboard page. Because there aren't many charts, this is one of the project's liveliest webpages. We can see that there are two different chart parts here. The first is a list of components that has been filtered by "Date From" and "Date To". These parts contain three primary indications at the top of the page: "total income", "total order", and "total new user". In addition, there are pie charts and two table presentations that rank clients based on the number of times the vehicle rental service is used and the amount of car rental.

The second part of the Dashboard page contains two column charts that provide data from all 12 months filtered by year. The chart on the left shows the total number of new invoices and clients, while the chart on the right shows the income, expenses, and profit for each month.

**6.2.5. Management Bike Manufacturer page**



*Figure 24: Management Bike Manufacturer*

The next interface is for a page that displays a list of bike manufacturers. This page, as we can see, is separated into two sections: filter components at the top and table pagination below. This format will be utilized on all subsequent management pages. The distinction is that each page will contain different actions, buttons, and filter components based on the requirements of the consumer. We can set the maximum amount of items that a page can display in the lower right corner.

**6.2.6. Admin Bike Detail page**



*Figure 25: Admin bike detail*

Next, I'll show the authorized Bike Detail page of the project that the admin is observing. As we can see, this page's design is centered on the form, and the image below differs from the public Bike Detail page. If the public site emphasizes the attractive interface and attracts viewers, the authenticated page will include data for each value as well as each motorcycle pictures. Above, there will be buttons such as "BACK TO LIST" that will re-render the Bike List page, "UPDATE" that will re-render the Bike Update page, and "DELETE" that will permanently erase the motorcycle from the system.

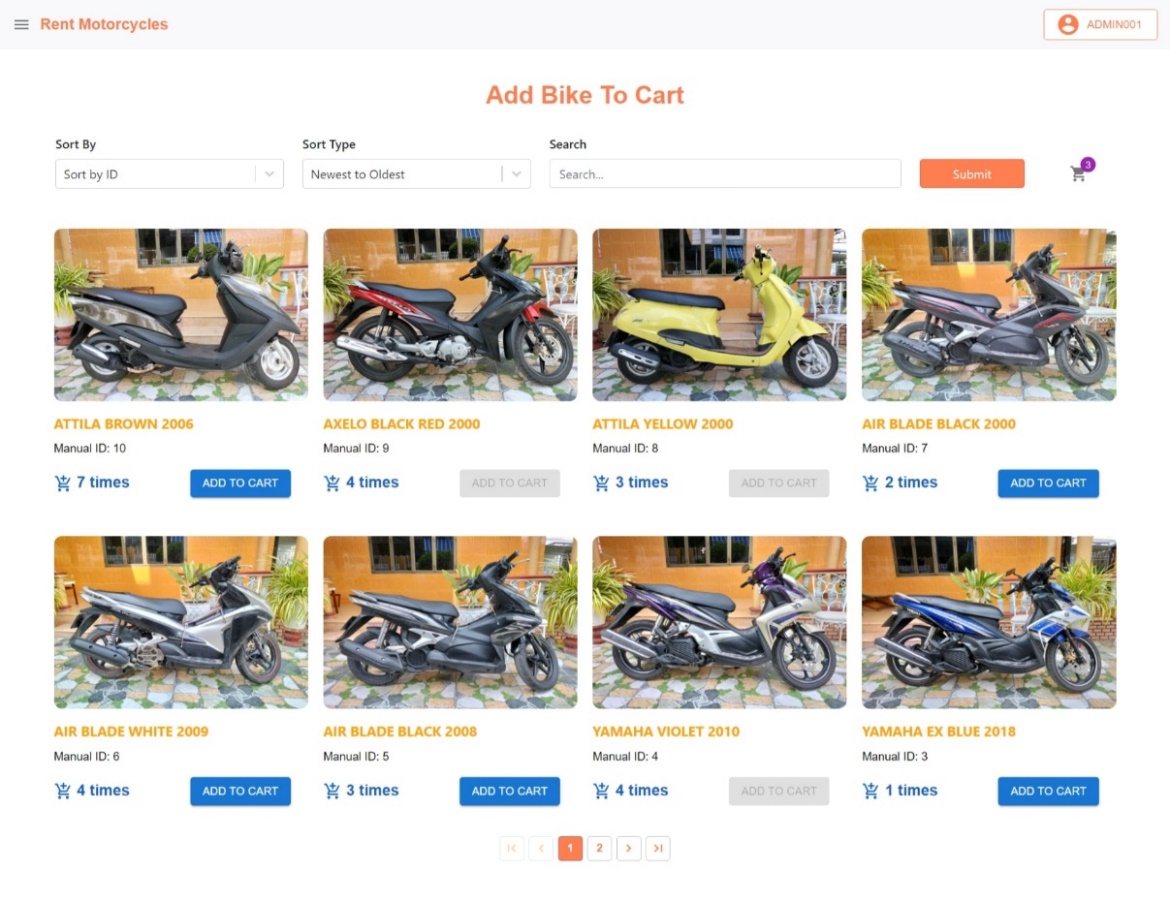
**6.2.7. Create new bike page**



*Figure 26: Create bike*

Here is a screenshot of the website for building a new motorcycle. The design is mostly similar to the wireframe style, with text boxes and choose boxes where the admin inputs the vehicle's details. A form for uploading photos from a personal computer is provided below.

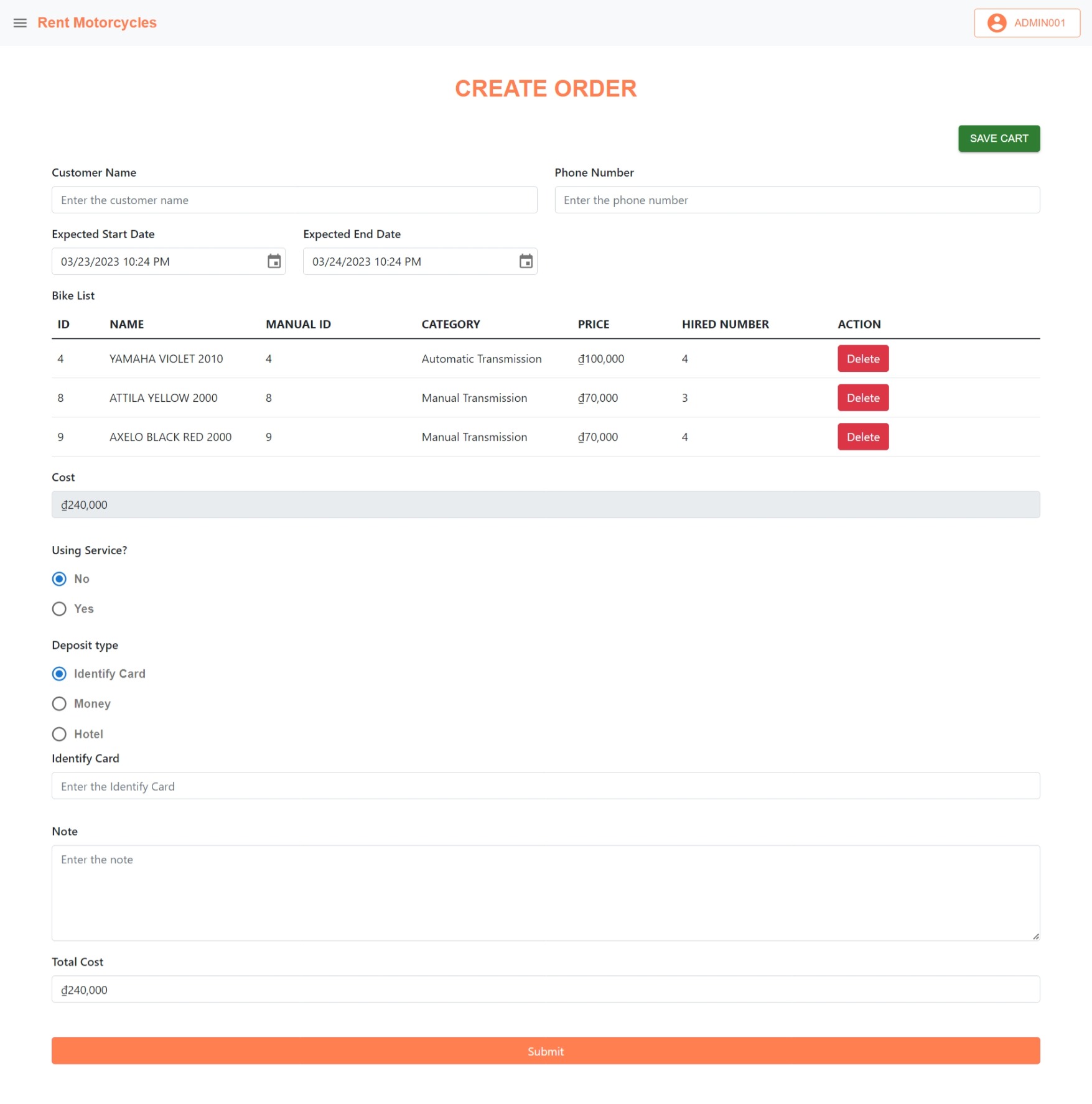
**6.2.8. Create cart page**



*Figure 27: Create cart*

This is the website interface for adding bikes to the shopping cart. Its layout was motivated by e-commerce websites. Users can see a list of vehicles here, similar to the public bike list website, but each motorbike will indicate the number of rentals as well as a "ADD TO CART" button. When a motor vehicle is successfully added to the cart, the bike add button is disabled.

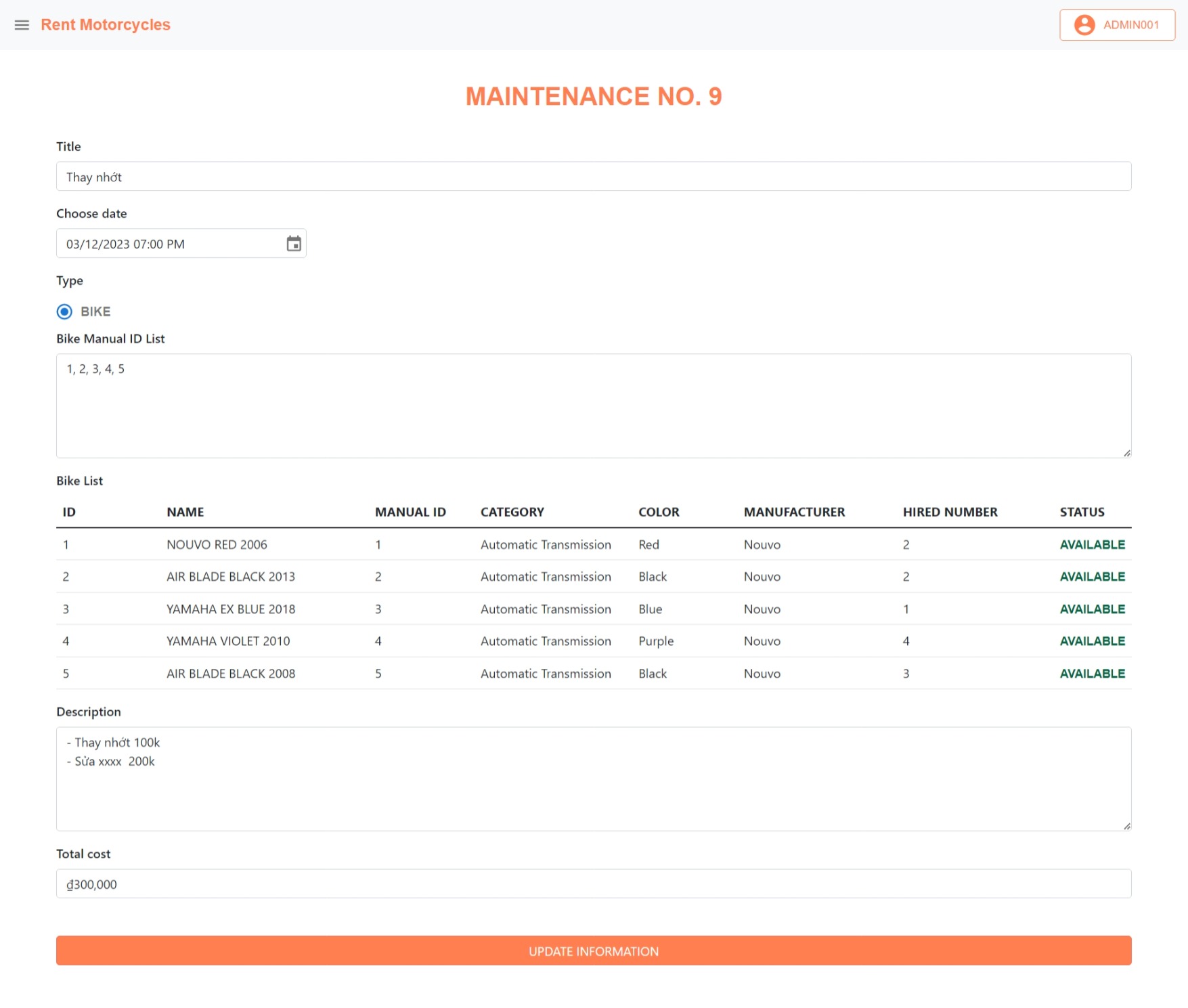
**6.2.9. Create order page**



*Figure 28: Create order*

Following that, I'll go over one of the project's more complicated forms. It is the interface for generating order. As we can see, the structure of this page is also made up of form components such text boxes, datetime pick boxes, radio buttons, and text areas. The list of vehicles being added to the order will be displayed in a table in the middle of the form. Users can delete a motorcycle from an order by clicking the "Delete" button in the ACTION column. Furthermore, if consumers don't want to create an order right immediately, they can save the current information by clicking the "SAVE CART" button in the top right corner of the screen.

**6.2.10. Maintenance Detail page**



*Figure 29: Maintenance detail*

The last interface concerns the Maintenance Detail page. This is the information display page for a Maintenance, which includes information as well as a list of vehicles associated with that Maintenance. The information can be edited and saved by the user. They can modify the motorcycle list by typing into the text area on the table show list bike. They must enter the vehicle's manual id, separated by commas, for the system to recognize the information. When users alter the value in this text section and click the "UPDATE INFORMATION" button, the table show list bike is reloaded to display the updated list of motorbikes.

**6.3. Product Implementation**

I'll provide some examples of the project's typical code in this part. I'll break it down into two sections: back-end code and front-end code.

**6.3.1. Back-end Code**

**6.3.1.1. WebSecurityConfig**

@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(  
 securedEnabled = true,  
 jsr250Enabled = true,  
 prePostEnabled = true)  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter{  
  
 @Autowired  
 UserDetailServiceManager userDetailsService;  
  
 @Autowired  
 private AuthEntryPointJwt unauthorizedHandler;  
  
 @Bean  
 public AuthTokenFilter authenticationJwtTokenFilter() {  
 return new AuthTokenFilter();  
 }  
  
 @Override  
 public void configure(AuthenticationManagerBuilder authenticationManagerBuilder) throws Exception {  
 authenticationManagerBuilder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());  
 }  
  
 @Bean  
 @Override  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.cors().and().csrf().disable()  
 .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.*STATELESS*).and()  
 .authorizeRequests()  
 .antMatchers("/authen/signin").permitAll()  
 .antMatchers("/authen/signup").permitAll()  
 .antMatchers("/public/\*\*").permitAll()  
 .anyRequest().authenticated()  
 .and().formLogin()  
 .loginPage("/login").permitAll();  
 http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);  
 }  
}

This is the code that configures authentication for Spring Boot. As we've seen, every request need authentication, with the exception of a few standard urls like "/authen/signin" for the login page and "/authen/signup" for the account creation feature, which are only accessible to IT departments. Finally, there are "/public" links to websites that renters of motorbikes visit.

**6.3.1.2. Order Controller**

@CrossOrigin(origins = "\*", maxAge = 3600)  
@RestController  
@RequestMapping("/admin/order")  
public class OrderController {  
  
 @Autowired  
 ResponseUtils responseUtils;  
  
 @Autowired  
 JwtUtils jwtUtils;  
  
 @Autowired  
 OrderService orderService;  
  
 @PostMapping("/cart/add-bike")  
 public ResponseEntity<?> cartAddBike(@RequestBody OrderRequest orderRequest, HttpServletRequest request) {  
 try {  
 String jwt = jwtUtils.*getJwtFromRequest*(request);  
 String username = jwtUtils.*getUserNameFromJwtToken*(jwt);  
 Result result = orderService.cartAddBike(username, orderRequest.getBikeId());  
 if(result.getCode() == Constant.*SUCCESS\_CODE*){  
 return responseUtils.getResponseEntity( result.getObject(), result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 else{  
 return responseUtils.getResponseEntity(null, result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 }catch (Exception e) {  
 e.printStackTrace();  
 return responseUtils.getResponseEntity(null, Constant.*SYSTEM\_ERROR\_CODE*, Constant.*SYSTEM\_ERROR*, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 @GetMapping("/cart/get")  
 public ResponseEntity<?> cartGetByUsername(HttpServletRequest request){  
 try{  
 String jwt = jwtUtils.*getJwtFromRequest*(request);  
 String username = jwtUtils.*getUserNameFromJwtToken*(jwt);  
 Result result = orderService.cartGetByUsername(username);  
 if(result.getCode() == Constant.*SUCCESS\_CODE*){  
 return responseUtils.getResponseEntity( result.getObject(), result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 else{  
 return responseUtils.getResponseEntity(null, result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 }catch(Exception e){  
 e.printStackTrace();  
 return responseUtils.getResponseEntity(null, Constant.*SYSTEM\_ERROR\_CODE*, Constant.*SYSTEM\_ERROR*, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 @GetMapping("/cart/get/bike-number")  
 public ResponseEntity<?> cartGetBikeNumber(HttpServletRequest request){  
 try{  
 String jwt = jwtUtils.*getJwtFromRequest*(request);  
 String username = jwtUtils.*getUserNameFromJwtToken*(jwt);  
 Result result = orderService.cartGetBikeNumber(username);  
 if(result.getCode() == Constant.*SUCCESS\_CODE*){  
 return responseUtils.getResponseEntity( result.getObject(), result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 else{  
 return responseUtils.getResponseEntity(null, result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 }catch(Exception e){  
 e.printStackTrace();  
 return responseUtils.getResponseEntity(null, Constant.*SYSTEM\_ERROR\_CODE*, Constant.*SYSTEM\_ERROR*, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 @PostMapping("/cart/delete-bike/orderId={orderId}&bikeId={bikeId}")  
 public ResponseEntity<?> cartDeleteBike(@PathVariable Long orderId,@PathVariable Long bikeId, HttpServletRequest request){  
 try{  
 String jwt = jwtUtils.*getJwtFromRequest*(request);  
 String username = jwtUtils.*getUserNameFromJwtToken*(jwt);  
 Result result = orderService.cartDeleteBike(orderId, bikeId, username);  
 return responseUtils.getResponseEntity(null, result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 catch(Exception e){  
 e.printStackTrace();  
 return responseUtils.getResponseEntity(null, Constant.*SYSTEM\_ERROR\_CODE*, Constant.*SYSTEM\_ERROR*, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 @PostMapping("/cart/calculate-hiring-cost")  
 public ResponseEntity<?> cartCalculateHiringCost(@RequestBody OrderRequest orderRequest) {  
 try {  
 Result result = orderService.cartCalculateHiringCost(orderRequest);  
 if(result.getCode() == Constant.*SUCCESS\_CODE*){  
 return responseUtils.getResponseEntity( result.getObject(), result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 else{  
 return responseUtils.getResponseEntity(null, result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 }catch (Exception e) {  
 e.printStackTrace();  
 return responseUtils.getResponseEntity(null, Constant.*SYSTEM\_ERROR\_CODE*, Constant.*SYSTEM\_ERROR*, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 @PostMapping("/cart/save")  
 public ResponseEntity<?> cartSave (@RequestBody OrderRequest orderRequest, HttpServletRequest request) {  
 try {  
 String jwt = jwtUtils.*getJwtFromRequest*(request);  
 String username = jwtUtils.*getUserNameFromJwtToken*(jwt);  
 Result result = orderService.cartSave(orderRequest, username);  
 return responseUtils.getResponseEntity(null, result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }catch (Exception e) {  
 e.printStackTrace();  
 return responseUtils.getResponseEntity(null, Constant.*SYSTEM\_ERROR\_CODE*, Constant.*SYSTEM\_ERROR*, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 @PostMapping("/get")  
 public ResponseEntity<?> getOrderPagination(@RequestBody PaginationOrderRequest reqBody, HttpServletRequest request){  
 try{  
 String jwt = jwtUtils.*getJwtFromRequest*(request);  
 String username = jwtUtils.*getUserNameFromJwtToken*(jwt);  
 PageDto result = orderService.getOrderPagination(reqBody, username);  
 if (result != null) {  
 return responseUtils.getResponseEntity(result, Constant.*SUCCESS\_CODE*, "Get Successfully", HttpStatus.*OK*);  
 }  
 return responseUtils.getResponseEntity(null, Constant.*SYSTEM\_ERROR\_CODE*, "Failed", HttpStatus.*OK*);  
 }  
 catch(Exception e){  
 e.printStackTrace();  
 return responseUtils.getResponseEntity(null, Constant.*SYSTEM\_ERROR\_CODE*, Constant.*SYSTEM\_ERROR*, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 @GetMapping("/get")  
 public ResponseEntity<?> getOrderById(@RequestParam Long id){  
 try{  
 Result result = orderService.getOrderById(id);  
 if(result.getCode() == Constant.*SUCCESS\_CODE*){  
 return responseUtils.getResponseEntity( result.getObject(), result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 else{  
 return responseUtils.getResponseEntity(null, result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }  
 }catch(Exception e){  
 return responseUtils.getResponseEntity(e, -1, "Login fail!", HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
  
 }  
  
 @PostMapping("/save")  
 public ResponseEntity<?> saveOrder (@RequestBody OrderRequest orderRequest, HttpServletRequest request) {  
 try {  
 String jwt = jwtUtils.*getJwtFromRequest*(request);  
 String username = jwtUtils.*getUserNameFromJwtToken*(jwt);  
 Result result = orderService.saveOrder(orderRequest, username);  
 return responseUtils.getResponseEntity(null, result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }catch (Exception e) {  
 e.printStackTrace();  
 return responseUtils.getResponseEntity(e, -1, "Login fail!", HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 @PostMapping("/cancel")  
 public ResponseEntity<?> cancelOrder (@RequestBody OrderRequest orderRequest, HttpServletRequest request) {  
 try {  
 String jwt = jwtUtils.*getJwtFromRequest*(request);  
 String username = jwtUtils.*getUserNameFromJwtToken*(jwt);  
 Result result = orderService.cancelOrder(orderRequest, username);  
 return responseUtils.getResponseEntity(null, result.getCode(), result.getMessage(), HttpStatus.*OK*);  
 }catch (Exception e) {  
 e.printStackTrace();  
 return responseUtils.getResponseEntity(e, -1, "Login fail!", HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
}

Here is an example controller's code for handling requests for the Order entity from the project. This is also the controller with the most functions in the project. We can see that only two methods, GET and POST, are utilized to call the API from the front end. GET methods are solely used to retrieve data from the database and not to modify it. The POST methods will use the Service classes to process the data. All of the functions in the controller use the try catch structure to convey the API error when a server-side problem occurs.

**6.3.1.3. Bike entity and BaseEntity**

@Entity  
@Data  
@Table(name = "bike")  
public class Bike extends BaseEntity{  
  
 @Column(name = "name", length = 100)  
 private String name;  
  
 @Column(name = "bike\_manual\_id", nullable = false)  
 private String bikeManualId;  
  
 @Column(name = "bike\_no", length = 50, nullable = false)  
 private String bikeNo;  
  
 @Column(name = "bike\_category\_id")  
 private Long bikeCategoryId;  
  
 @Column(name = "bike\_color\_id")  
 private Long bikeColorId;  
  
 @Column(name = "bike\_manufacturer\_id")  
 private Long bikeManufacturerId;  
  
 @Column(name = "status", length = 50, nullable = false)  
 private String status = "AVAILABLE";  
  
 @Column(name = "hired\_number", nullable = false)  
 private Integer hiredNumber = 0;  
  
}

@Data  
@MappedSuperclass  
public class BaseEntity implements Serializable{  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(name = "created\_user", nullable = false)  
 private String createdUser;  
  
 @Column(name = "created\_date")  
 private Date createdDate;  
  
 @Column(name = "modified\_user")  
 private String modifiedUser;  
  
 @Column(name = "modified\_date")  
 private Date modifiedDate;  
  
 @Column(name = "is\_deleted")  
 private Boolean isDeleted = Boolean.*FALSE*;  
  
 public BaseEntity() {  
 }  
  
 public BaseEntity(Long id, String createUser, Date createTime, String modifiedUser, Date modifiedTime, boolean isDeleted) {  
 this.id = id;  
 this.createdUser = createUser;  
 this.createdDate = createTime;  
 this.modifiedUser = modifiedUser;  
 this.modifiedDate = modifiedTime;  
 this.isDeleted = isDeleted;  
 }  
  
  
}

In this section, I will introduce two typical project entities. Spring Boot, as I said in Section 3.1.5 of Chapter 3, uses the ORM (Object Relational Mapping) technology to generate tables in the MySQL database from Java Objects. In Spring Boot, these Objects are Entities. In this project, I develop an abstract entity named BaseEntity that contains features common to all tables, such as create user, create date, modified user, modified date, and so on. All other Entities extend this entity and inherit all of its properties. Following that is the Bike entity, which will hold all vehicle information such as bike name, bike number, and so on.

**6.3.1.4. Maintain Bike Repository**

public interface MaintainBikeRepository extends JpaRepository<MaintainBike, Long>, JpaSpecificationExecutor<MaintainBike> {  
 Boolean existsByMaintainIdAndIsDeleted(Long maintainId, Boolean check);  
 Boolean existsByMaintainIdAndBikeId(Long maintainId, Long bikeId);  
  
 List<MaintainBike> findAllByMaintainIdAndIsDeleted(Long id, boolean check);  
 MaintainBike findAllByMaintainIdAndBikeId(Long maintainId, Long bikeId);  
  
 @Modifying  
 @Transactional  
 @Query("UPDATE MaintainBike mb SET mb.isDeleted = true WHERE mb.maintainId = :maintainId")  
 void updateIsDelete(@Param("maintainId") Long id);  
}

Next, I'll go over one of the more common interfaces, MaintainBikeRepository. This interface is used to create queries using the JpaRepository library from Spring Boot. Instead of creating a lengthy SQL query, this library makes it simple to write inquiries using words like FIND, EXISTS, and others. It also supports a variety of return values, including List, Object, and Boolean. The limitation of this approach is that it can only be used for simple queries or table queries. I advise utilizing a pure query statement using @Query if we want to build intricate queries. This annotation allows us to modify a query's parameters and return data type. For instance, the updateIsDelete method from the previous section is of type void and returns nothing. This function is used to change the isDeleted field of all Maintenance entities based on the id given.

**6.3.1.5. Bike Service**

@Service  
public class BikeService {  
  
 @Autowired  
 BikeRepository bikeRepository;  
  
 @Autowired  
 BikeCategoryRepository bikeCategoryRepository;  
  
 @Autowired  
 BikeImageRepository bikeImageRepository;  
  
 @Autowired  
 OrderRepository orderRepository;  
  
 @Autowired  
 OrderDetailRepository orderDetailRepository;  
  
 @Autowired  
 BikeSpecification bikeSpecification;  
  
 @Autowired  
 ResponseUtils responseUtils;  
  
 @Autowired  
 ModelMapper modelMapper;  
  
 @Autowired  
 HistoryService historyService;  
  
 @Autowired  
 CheckEntityExistService checkEntityExistService;  
  
 public PageDto getBikePagination(PaginationBikeRequest paginationBikeRequest) {  
 try {  
 String searchKey = paginationBikeRequest.getSearchKey();  
 Integer page = paginationBikeRequest.getPage();  
 Integer limit = paginationBikeRequest.getLimit();  
 String sortBy = paginationBikeRequest.getSortBy();  
 String sortType = paginationBikeRequest.getSortType();  
 Long categoryId = paginationBikeRequest.getCategoryId();  
 Boolean isInCart = paginationBikeRequest.getIsInCart();  
 String username = paginationBikeRequest.getUsername();  
  
 Map<String, Object> mapBike = bikeSpecification.getBikePagination(searchKey, page, limit, sortBy, sortType, categoryId, isInCart);  
 List<BikeResponse> listRes = (List<BikeResponse>) mapBike.get("data");  
 Long totalItems = (Long) mapBike.get("count");  
 Integer totalPage = responseUtils.getPageCount(totalItems, limit);  
  
 // Image handling  
 List<BikeResponse> listResult = new ArrayList<>();  
 for(BikeResponse bikeResponse : listRes){  
 List<BikeImage> listImage = bikeImageRepository.findAllByBikeIdAndIsDeletedOrderByNameAsc(bikeResponse.getId(), false);  
  
 if(!listImage.isEmpty()){  
  
 List<AttachmentResponse> listImageResponse = new ArrayList<>();  
 for(BikeImage bikeImage : listImage){  
 AttachmentResponse attachmentResponse = new AttachmentResponse();  
 attachmentResponse.setId(bikeImage.getId());  
 attachmentResponse.setFilePath(bikeImage.getPath());  
 attachmentResponse.setFileName(bikeImage.getName());  
 listImageResponse.add(attachmentResponse);  
 }  
 bikeResponse.setImageList(listImageResponse);  
 }  
 listResult.add(bikeResponse);  
 }  
  
 // Get orderId IF in CART  
 if(isInCart != null)  
 {  
 if(orderRepository.existsByCreatedUserAndStatusAndIsDeleted(username, "IN CART", false))  
 {  
 Order order = orderRepository.findByCreatedUserAndStatusAndIsDeleted(username, "IN CART", false);  
 List<OrderDetail> listOrderDetail = orderDetailRepository.findAllOrderDetailByOrderIdAndIsDeleted(order.getId(), false);  
 for(OrderDetail item : listOrderDetail)  
 {  
 for(BikeResponse bikeResponse : listResult)  
 {  
 if(bikeResponse.getId() == item.getBikeId())  
 {  
 bikeResponse.setOrderId(order.getId());  
 }  
 }  
 }  
 }  
 }  
  
 return PageDto.*builder*()  
 .content(listResult)  
 .numberOfElements(Math.*toIntExact*(totalItems))  
 .page(page)  
 .size(limit)  
 .totalPages(totalPage)  
 .totalElements(totalItems)  
 .build();  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public Result getBikeById(Long bikeId){  
 try{  
 Result result = new Result();  
  
 /\*-------------- GET BIKE --------------------\*/  
 Map<String, Object> mapBike = bikeSpecification.getBikeById(bikeId);  
 if(mapBike.size() == 0) {  
 result.setMessage("No Bike found");  
 result.setCode(Constant.*LOGIC\_ERROR\_CODE*);  
 return result;  
 }  
  
 BikeResponse bikeResponse = (BikeResponse) mapBike.get("data");  
 List<BikeImage> listImage = bikeImageRepository.findAllByBikeIdAndIsDeletedOrderByNameAsc(bikeResponse.getId(),false);  
 List<AttachmentResponse> listImageResponse = new ArrayList<>();  
 if(!listImage.isEmpty()){  
 for(BikeImage bikeImage : listImage){  
 AttachmentResponse attachmentResponse = new AttachmentResponse();  
 attachmentResponse.setId(bikeImage.getId());  
 attachmentResponse.setFilePath(bikeImage.getPath());  
 attachmentResponse.setFileName(bikeImage.getName());  
 listImageResponse.add(attachmentResponse);  
 }  
 }  
 bikeResponse.setImageList(listImageResponse);  
  
 /\*-------------- GET RELATION BIKE LIST --------------------\*/  
 PaginationBikeRequest paginationBikeRequest = new PaginationBikeRequest();  
 paginationBikeRequest.setSearchKey(null);  
 paginationBikeRequest.setLimit(7);  
 paginationBikeRequest.setPage(1);  
 paginationBikeRequest.setSortBy("hiredNumber");  
 paginationBikeRequest.setSortType("DESC");  
 paginationBikeRequest.setCategoryId(bikeResponse.getBikeCategoryId());  
 PageDto pageDto = getBikePagination(paginationBikeRequest);  
 List<BikeResponse> listBike = pageDto.getContent();  
 listBike = listBike.stream().filter(x -> x.getId() != bikeResponse.getId()).collect(Collectors.*toList*());  
 bikeResponse.setListBike(listBike);  
  
  
 result.setMessage("Get successful");  
 result.setCode(Constant.*SUCCESS\_CODE*);  
 result.setObject(bikeResponse);  
 return result;  
  
 }catch (Exception e) {  
 e.printStackTrace();  
 return new Result(Constant.*SYSTEM\_ERROR\_CODE*, "System error", null);  
 }  
 }  
  
 public Result createBike(BikeRequest bikeRequest, String username){  
 try{  
 if(bikeRepository.existsByBikeNoAndName(bikeRequest.getBikeNo(), bikeRequest.getName())){  
 return new Result(Constant.*LOGIC\_ERROR\_CODE*, "The bike number has been existed!!!");  
 }  
  
 Bike newBike = modelMapper.map(bikeRequest, Bike.class);  
 newBike.setCreatedDate(new Date());  
 newBike.setCreatedUser(username);  
 newBike.setStatus("AVAILABLE");  
 newBike.setHiredNumber(0);  
 Bike savedBike = bikeRepository.save(newBike);  
  
 List<BikeImage> saveList = new ArrayList<>();  
 for(AttachmentRequest item : bikeRequest.getFiles()){  
 BikeImage bikeImage = new BikeImage();  
 bikeImage.setBikeId(savedBike.getId());  
 bikeImage.setName(item.getFileName());  
 bikeImage.setPath(item.getFilePath());  
 bikeImage.setCreatedDate(new Date());  
 bikeImage.setCreatedUser(username);  
 saveList.add(bikeImage);  
 }  
  
 bikeImageRepository.saveAll(saveList);  
  
 HistoryObject historyObject = new HistoryObject();  
 historyObject.setUsername(username);  
 historyObject.setEntityId(savedBike.getId());  
 historyService.saveHistory(Constant.*HISTORY\_CREATE*, savedBike, historyObject);  
  
 return new Result(Constant.*SUCCESS\_CODE*, "Create new bike successfully");  
 }catch (Exception e) {  
 e.printStackTrace();  
 return new Result(Constant.*SYSTEM\_ERROR\_CODE*, "Fail");  
 }  
 }  
  
 public Result deleteBike(Long id, String username){  
 try{  
 // Check bike exist  
 if(!checkEntityExistService.isEntityExisted(Constant.*BIKE*, "id", id)){  
 return new Result(Constant.*LOGIC\_ERROR\_CODE*, "The bike id " + id + " has not been existed!!!");  
 }  
 Bike bike = bikeRepository.findBikeById(id);  
 if(bike.getIsDeleted() == true){  
 return new Result(Constant.*LOGIC\_ERROR\_CODE*, "The bike has not been existed!!!");  
 }  
  
 if(!bike.getStatus().equalsIgnoreCase("AVAILABLE")){  
 return new Result(Constant.*LOGIC\_ERROR\_CODE*, "The bike is being hired! Please check status");  
 }  
  
 // REMOVE BIKE  
 bike.setModifiedDate(new Date());  
 bike.setModifiedUser(username);  
 bike.setIsDeleted(true);  
 bikeRepository.save(bike);  
  
 // HISTORY FOR BIKE  
 HistoryObject historyObject = new HistoryObject();  
 historyObject.setUsername(username);  
 historyObject.setEntityId(bike.getId());  
 historyService.saveHistory(Constant.*HISTORY\_DELETE*, bike, historyObject);  
  
 // HISTORY FOR IMAGES  
 List<BikeImage> removedBikeImage = bikeImageRepository.findAllByBikeIdAndIsDeletedOrderByNameAsc(bike.getId(), false);  
 for(BikeImage image : removedBikeImage){  
 HistoryObject historyBikeObjectImage = new HistoryObject();  
 historyBikeObjectImage.setUsername(username);  
 historyBikeObjectImage.setEntityId(image.getId());  
 historyService.saveHistory(Constant.*HISTORY\_DELETE*, image, historyBikeObjectImage);  
 }  
  
 // REMOVE BIKE IMAGES  
 bikeImageRepository.updateIsDelete(bike.getId());  
  
 return new Result(Constant.*SUCCESS\_CODE*, "Delete bike successfully");  
 }catch (Exception e) {  
 e.printStackTrace();  
 return new Result(Constant.*SYSTEM\_ERROR\_CODE*, "Fail");  
 }  
 }  
  
 public Result updateBike(BikeRequest bikeRequest){  
 try{  
 if(!checkEntityExistService.isEntityExisted(Constant.*BIKE*, "id", bikeRequest.getId())){  
 return new Result(Constant.*LOGIC\_ERROR\_CODE*, "The bike has not been existed!!!");  
 }  
 Bike bike = bikeRepository.findBikeById(bikeRequest.getId());  
  
 //History Bike  
 HistoryObject historyBikeObject = new HistoryObject();  
 historyBikeObject.setUsername(bikeRequest.getUsername());  
 historyBikeObject.setEntityId(bike.getId());  
 historyBikeObject.getComparingMap().put("name", new ComparedObject(bike.getName(), bikeRequest.getName()));  
 historyBikeObject.getComparingMap().put("bikeManualId", new ComparedObject(bike.getBikeManualId(), bikeRequest.getBikeManualId()));  
 historyBikeObject.getComparingMap().put("bikeNo", new ComparedObject(bike.getBikeNo(), bikeRequest.getBikeNo()));  
 historyBikeObject.getComparingMap().put("bikeCategoryId", new ComparedObject(bike.getBikeCategoryId(), bikeRequest.getBikeCategoryId()));  
 historyBikeObject.getComparingMap().put("bikeColorId", new ComparedObject(bike.getBikeColorId(), bikeRequest.getBikeColorId()));  
 historyBikeObject.getComparingMap().put("bikeManufacturerId", new ComparedObject(bike.getBikeManufacturerId(), bikeRequest.getBikeManufacturerId()));  
 historyBikeObject.getComparingMap().put("status", new ComparedObject(bike.getStatus(), bikeRequest.getStatus()));  
 historyBikeObject.getComparingMap().put("hiredNumber", new ComparedObject(bike.getHiredNumber(), bikeRequest.getHiredNumber()));  
 historyService.saveHistory(Constant.*HISTORY\_UPDATE*, bike, historyBikeObject);  
  
 // Save bike  
 bike.setModifiedDate(new Date());  
 bike.setModifiedUser(bikeRequest.getUsername());  
 bike.setName(bikeRequest.getName());  
 bike.setBikeManualId(bikeRequest.getBikeManualId());  
 bike.setBikeNo(bikeRequest.getBikeNo());  
 bike.setBikeCategoryId(bikeRequest.getBikeCategoryId());  
 bike.setBikeColorId(bikeRequest.getBikeColorId());  
 bike.setBikeManufacturerId(bikeRequest.getBikeManufacturerId());  
 bike.setStatus(bikeRequest.getStatus());  
 bike.setHiredNumber(bikeRequest.getHiredNumber());  
 bikeRepository.save(bike);  
  
 // Save new image  
 List<BikeImage> saveList = new ArrayList<>();  
 for(AttachmentRequest item : bikeRequest.getFiles()){  
 BikeImage bikeImage = new BikeImage();  
 bikeImage.setBikeId(bike.getId());  
 bikeImage.setName(item.getFileName());  
 bikeImage.setPath(item.getFilePath());  
 bikeImage.setCreatedDate(new Date());  
 bikeImage.setCreatedUser(bikeRequest.getUsername());  
 saveList.add(bikeImage);  
 }  
 List<BikeImage> savedList = bikeImageRepository.saveAll(saveList);  
  
 //History Image  
 for(BikeImage image : savedList){  
 HistoryObject historyBikeObjectImage = new HistoryObject();  
 historyBikeObjectImage.setUsername(bikeRequest.getUsername());  
 historyBikeObjectImage.setEntityId(image.getId());  
 historyService.saveHistory(Constant.*HISTORY\_CREATE*, image, historyBikeObjectImage);  
 }  
  
 return new Result(Constant.*SUCCESS\_CODE*, "Update new bike successfully");  
 }catch (Exception e) {  
 e.printStackTrace();  
 return new Result(Constant.*SYSTEM\_ERROR\_CODE*, "Fail");  
 }  
 }  
  
 public Result deleteBikeImageById(Long imageId, String username){  
 try{  
 if(!checkEntityExistService.isEntityExisted(Constant.*BIKE\_IMAGE*, "id", imageId)){  
 return new Result(Constant.*LOGIC\_ERROR\_CODE*, "The bike image " + imageId +" has not been existed!!!");  
 }  
  
 BikeImage bikeImage = bikeImageRepository.findBikeImageById(imageId);  
 if(bikeImage.getIsDeleted() == true){  
 return new Result(Constant.*LOGIC\_ERROR\_CODE*, "The bike image " + imageId +" has not been existed!!!");  
 }  
 bikeImage.setModifiedDate(new Date());  
 bikeImage.setModifiedUser(username);  
 bikeImage.setIsDeleted(true);  
 bikeImageRepository.save(bikeImage);  
  
 HistoryObject historyObject = new HistoryObject();  
 historyObject.setUsername(username);  
 historyObject.setEntityId(imageId);  
 historyService.saveHistory(Constant.*HISTORY\_DELETE*, bikeImage, historyObject);  
 return new Result(Constant.*SUCCESS\_CODE*, "Delete bike image successfully");  
 }catch (Exception e) {  
 e.printStackTrace();  
 return new Result(Constant.*SYSTEM\_ERROR\_CODE*, "Fail");  
 }  
 }  
  
}

I'll display the code for the Service class of the Bike entity in this part. We'll cover every logical approach for dealing with bike-related operations like CRUD. These service classes will make calls to specification classes or repository interfaces to access the database. Every function has code to log the times these methods are invoked through historyService, with the exception of the view function.

**6.3.1.6. Bike Specification**

@Service  
public class BikeSpecification {  
  
 @PersistenceContext  
 EntityManager entityManager;  
  
 @Autowired  
 BikeCategoryRepository bikeCategoryRepository;  
  
 @Autowired  
 CheckEntityExistService checkEntityExistService;  
  
 public Map<String, Object> getBikePagination(String searchKey, Integer page, Integer limit, String sortBy, String sortType, Long categoryId, Boolean isInCart){  
 try{  
 Map<String, Object> mapFinal = new HashMap<>();  
  
 //----------------------CREATE QUERY -----------------------------//  
 CriteriaBuilder cb = entityManager.getCriteriaBuilder();  
  
 // ROOT  
 CriteriaQuery<BikeResponse> query = cb.createQuery(BikeResponse.class);  
 Root<Bike> root = query.from(Bike.class);  
 Root<BikeCategory> rootCate = query.from(BikeCategory.class);  
 Root<BikeColor> rootColor = query.from(BikeColor.class);  
 Root<BikeManufacturer> rootManufacturer = query.from(BikeManufacturer.class);  
  
 // ROOT COUNT  
 CriteriaQuery<Long> countQuery = cb.createQuery(Long.class);  
 Root<Bike> rootCount = countQuery.from(Bike.class);  
 Root<BikeCategory> rootCateCount = countQuery.from(BikeCategory.class);  
 Root<BikeColor> rootColorCount = countQuery.from(BikeColor.class);  
 Root<BikeManufacturer> rootManufacturerCount = countQuery.from(BikeManufacturer.class);  
  
  
 //---------------------- CONDITION -----------------------------//  
  
 // CONDITION  
 // EXIST BY CATEGORY  
 Boolean isCategoryExist = false;  
 if(categoryId != null && checkEntityExistService.isEntityExisted(Constant.*BIKE\_CATEGORY*, "id", categoryId)){  
 isCategoryExist = true;  
 }  
  
 // CONDITION  
 // ROOT  
 List<Predicate> predicates = new ArrayList<>();  
 predicates.add(cb.equal(root.get("bikeCategoryId"), rootCate.get("id")));  
 predicates.add(cb.equal(root.get("bikeColorId"), rootColor.get("id")));  
 predicates.add(cb.equal(root.get("bikeManufacturerId"), rootManufacturer.get("id")));  
  
 predicates.add(cb.isFalse(root.get("isDeleted")));  
 predicates.add(cb.isFalse(rootCate.get("isDeleted")));  
 predicates.add(cb.isFalse(rootColor.get("isDeleted")));  
 predicates.add(cb.isFalse(rootManufacturer.get("isDeleted")));  
  
 if(isCategoryExist){  
 predicates.add(cb.equal(rootCate.get("id"), categoryId));  
 }  
  
 if(isInCart != null)  
 {  
 predicates.add(cb.equal(root.get("status"), "AVAILABLE"));  
 }  
  
 if (!StringUtils.*isEmpty*(searchKey)) {  
 predicates.add(cb.or(  
 cb.like(cb.lower(root.get("name")) , "%" + searchKey.toLowerCase() + "%"),  
 cb.like(cb.lower(root.get("bikeNo")) , "%" + searchKey.toLowerCase() + "%"),  
 cb.like(cb.lower(rootCate.get("name")) , "%" + searchKey.toLowerCase() + "%"),  
 cb.like(cb.lower(rootColor.get("name")) , "%" + searchKey.toLowerCase() + "%"),  
 cb.like(cb.lower(rootManufacturer.get("name")) , "%" + searchKey.toLowerCase() + "%")  
 ));  
 }  
  
 // CONDITION  
 // ROOT COUNT  
 List<Predicate> predicatesCount = new ArrayList<>();  
 predicatesCount.add(cb.equal(rootCount.get("bikeCategoryId"), rootCateCount.get("id")));  
 predicatesCount.add(cb.equal(rootCount.get("bikeColorId"), rootColorCount.get("id")));  
 predicatesCount.add(cb.equal(rootCount.get("bikeManufacturerId"), rootManufacturerCount.get("id")));  
  
 predicatesCount.add(cb.isFalse(rootCount.get("isDeleted")));  
 predicatesCount.add(cb.isFalse(rootCateCount.get("isDeleted")));  
 predicatesCount.add(cb.isFalse(rootColorCount.get("isDeleted")));  
 predicatesCount.add(cb.isFalse(rootManufacturerCount.get("isDeleted")));  
  
 if(isCategoryExist){  
 predicatesCount.add(cb.equal(rootCateCount.get("id"), categoryId));  
 }  
  
 if(isInCart != null)  
 {  
 predicatesCount.add(cb.equal(rootCount.get("status"), "AVAILABLE"));  
 }  
  
 if (!StringUtils.*isEmpty*(searchKey)) {  
 predicatesCount.add(cb.or(  
 cb.like(cb.lower(rootCount.get("name")) , "%" + searchKey.toLowerCase() + "%"),  
 cb.like(cb.lower(rootCount.get("bikeNo")) , "%" + searchKey.toLowerCase() + "%"),  
 cb.like(cb.lower(rootCateCount.get("name")) , "%" + searchKey.toLowerCase() + "%"),  
 cb.like(cb.lower(rootColorCount.get("name")) , "%" + searchKey.toLowerCase() + "%"),  
 cb.like(cb.lower(rootManufacturerCount.get("name")) , "%" + searchKey.toLowerCase() + "%")  
 ));  
 }  
  
  
 //------------------------CREATE SORT-----------------------------//  
 // Sort theo Name - Cate Name - Hired Number - Price  
 if (sortType.equalsIgnoreCase("asc")) {  
 switch (sortBy) {  
 case "id":  
 query.orderBy(cb.asc(root.get("id")));  
 break;  
 case "name":  
 query.orderBy(cb.asc(root.get("name")));  
 break;  
 case "bikeManualId":  
 query.orderBy(cb.asc(root.get("bikeManualId")));  
 break;  
 case "hiredNumber":  
 query.orderBy(cb.asc(root.get("hiredNumber")));  
 break;  
 case "color":  
 query.orderBy(cb.asc(root.get("bikeColorId")));  
 break;  
 case "manufacturer":  
 query.orderBy(cb.asc(root.get("bikeManufacturerId")));  
 break;  
 }  
 } else {  
 switch (sortBy) {  
 case "id":  
 query.orderBy(cb.desc(root.get("id")));  
 break;  
 case "name":  
 query.orderBy(cb.desc(root.get("name")));  
 break;  
 case "bikeManualId":  
 query.orderBy(cb.desc(root.get("bikeManualId")));  
 break;  
 case "hiredNumber":  
 query.orderBy(cb.desc(root.get("hiredNumber")));  
 break;  
 case "color":  
 query.orderBy(cb.desc(root.get("bikeColorId")));  
 break;  
 case "manufacturer":  
 query.orderBy(cb.desc(root.get("bikeManufacturerId")));  
 break;  
 }  
 }  
  
 //----------------------END SORT-----------------------------//  
 query.multiselect(  
 root.get("id"),  
 root.get("name"),  
 root.get("bikeManualId"),  
 root.get("bikeNo"),  
 root.get("hiredNumber"),  
 rootCate.get("id"),  
 rootCate.get("name"),  
 rootCate.get("price"),  
 rootColor.get("id"),  
 rootColor.get("name"),  
 rootManufacturer.get("id"),  
 rootManufacturer.get("name"),  
 root.get("status")  
 ).where(cb.and(predicates.stream().toArray(Predicate[]::new)));  
 List<BikeResponse> listResult = entityManager.createQuery(query) != null ? entityManager.createQuery(query).  
 setFirstResult((page - 1) \* limit)  
 .setMaxResults(limit).getResultList() : new ArrayList<>();  
  
 countQuery.select(cb.count(rootCount)).where(cb.and(predicatesCount.stream().toArray(Predicate[]::new)));  
 Long count = entityManager.createQuery(countQuery).getSingleResult();  
 mapFinal.put("data", listResult);  
 mapFinal.put("count", count);  
 return mapFinal;  
 } catch (Exception e) {  
 e.printStackTrace();  
 return new HashMap<>();  
 }  
 }  
  
 public Map<String, Object> getBikeById(Long bikeId){  
 try{  
 Map<String, Object> mapFinal = new HashMap<>();  
  
 CriteriaBuilder cb = entityManager.getCriteriaBuilder();  
 CriteriaQuery<BikeResponse> query = cb.createQuery(BikeResponse.class);  
 Root<Bike> root = query.from(Bike.class);  
 Root<BikeCategory> rootCate = query.from(BikeCategory.class);  
 Root<BikeColor> rootColor = query.from(BikeColor.class);  
 Root<BikeManufacturer> rootManufacturer = query.from(BikeManufacturer.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
 predicates.add(cb.equal(root.get("bikeCategoryId"), rootCate.get("id")));  
 predicates.add(cb.equal(root.get("bikeColorId"), rootColor.get("id")));  
 predicates.add(cb.equal(root.get("bikeManufacturerId"), rootManufacturer.get("id")));  
  
 predicates.add(cb.isFalse(root.get("isDeleted")));  
 predicates.add(cb.isFalse(rootCate.get("isDeleted")));  
 predicates.add(cb.isFalse(rootColor.get("isDeleted")));  
 predicates.add(cb.isFalse(rootManufacturer.get("isDeleted")));  
  
 if(bikeId != null){  
 predicates.add(cb.equal(root.get("id"), bikeId));  
 }  
  
 query.multiselect(  
 root.get("id"),  
 root.get("name"),  
 root.get("bikeManualId"),  
 root.get("bikeNo"),  
 root.get("hiredNumber"),  
 rootCate.get("id"),  
 rootCate.get("name"),  
 rootCate.get("price"),  
 rootColor.get("id"),  
 rootColor.get("name"),  
 rootManufacturer.get("id"),  
 rootManufacturer.get("name"),  
 root.get("status"),  
 root.get("createdUser"),  
 root.get("createdDate"),  
 root.get("modifiedUser"),  
 root.get("modifiedDate")  
 ).where(cb.and(predicates.stream().toArray(Predicate[]::new)));  
  
 List<BikeResponse> result = entityManager.createQuery(query) != null ? entityManager.createQuery(query).getResultList() : new ArrayList<>();  
 if(result.size() == 0) {  
 return new HashMap<>();  
 }else{  
 mapFinal.put("data", result.get(0));  
 return mapFinal;  
 }  
 }catch (Exception e) {  
 e.printStackTrace();  
 return new HashMap<>();  
 }  
 }  
  
 public Map<String, Object> getBikeListById(List<Long> listBikeID){  
 try {  
 Map<String, Object> mapFinal = new HashMap<>();  
  
 //----------------------CREATE QUERY -----------------------------//  
 CriteriaBuilder cb = entityManager.getCriteriaBuilder();  
 // ROOT  
 CriteriaQuery<BikeResponse> query = cb.createQuery(BikeResponse.class);  
 Root<Bike> root = query.from(Bike.class);  
 Root<BikeCategory> rootCate = query.from(BikeCategory.class);  
 Root<BikeColor> rootColor = query.from(BikeColor.class);  
 Root<BikeManufacturer> rootManufacturer = query.from(BikeManufacturer.class);  
  
 // CONDITION  
 // ROOT  
 List<Predicate> predicates = new ArrayList<>();  
 predicates.add(cb.equal(root.get("bikeCategoryId"), rootCate.get("id")));  
 predicates.add(cb.equal(root.get("bikeColorId"), rootColor.get("id")));  
 predicates.add(cb.equal(root.get("bikeManufacturerId"), rootManufacturer.get("id")));  
  
 predicates.add(cb.isFalse(root.get("isDeleted")));  
 predicates.add(cb.isFalse(rootCate.get("isDeleted")));  
 predicates.add(cb.isFalse(rootColor.get("isDeleted")));  
 predicates.add(cb.isFalse(rootManufacturer.get("isDeleted")));  
  
 predicates.add(root.get("id").in(listBikeID));  
 //----------------------END SORT-----------------------------//  
 query.multiselect(  
 root.get("id"),  
 root.get("name"),  
 root.get("bikeManualId"),  
 root.get("bikeNo"),  
 root.get("hiredNumber"),  
 rootCate.get("id"),  
 rootCate.get("name"),  
 rootCate.get("price"),  
 rootColor.get("id"),  
 rootColor.get("name"),  
 rootManufacturer.get("id"),  
 rootManufacturer.get("name"),  
 root.get("status")  
 ).where(cb.and(predicates.stream().toArray(Predicate[]::new)));  
  
 List<BikeResponse> listResult = entityManager.createQuery(query) != null ?  
 entityManager.createQuery(query).getResultList() : new ArrayList<>();  
  
 mapFinal.put("data", listResult);  
 return mapFinal;  
 } catch (Exception e) {  
 e.printStackTrace();  
 return new HashMap<>();  
 }  
 }  
  
 public Map<String, Object> getBikePriceListById(List<OrderDetail> listOrderDetail){  
 try {  
 Map<String, Object> mapFinal = new HashMap<>();  
 List<Long> listBikeID = new ArrayList<>();  
 for(OrderDetail item:listOrderDetail) {  
 listBikeID.add(item.getBikeId());  
 }  
  
 //----------------------CREATE QUERY -----------------------------//  
 CriteriaBuilder cb = entityManager.getCriteriaBuilder();  
 // ROOT  
 CriteriaQuery<BikeResponse> query = cb.createQuery(BikeResponse.class);  
 Root<Bike> root = query.from(Bike.class);  
 Root<BikeCategory> rootCate = query.from(BikeCategory.class);  
  
 // CONDITION  
 // ROOT  
 List<Predicate> predicates = new ArrayList<>();  
 predicates.add(cb.equal(root.get("bikeCategoryId"), rootCate.get("id")));  
  
 predicates.add(cb.isFalse(root.get("isDeleted")));  
 predicates.add(cb.isFalse(rootCate.get("isDeleted")));  
  
 predicates.add(root.get("id").in(listBikeID));  
 //----------------------END SORT-----------------------------//  
 query.select(  
 rootCate.get("price")  
 ).where(cb.and(predicates.stream().toArray(Predicate[]::new)));  
  
 List<BikeResponse> listResult = entityManager.createQuery(query) != null ?  
 entityManager.createQuery(query).getResultList() : new ArrayList<>();  
  
 mapFinal.put("data", listResult);  
 return mapFinal;  
 } catch (Exception e) {  
 e.printStackTrace();  
 return new HashMap<>();  
 }  
 }  
   
}

Finally, I'll discuss the project's specifications. Specification, as presented in section 3.1.7, is an alternative to JpaRepository for querying data from the database. Instead of using repository acronyms in this case, we'll utilize Hibernate commands. The specification is typically used to manage complicated queries that join numerous specialty tables, like Bike. We can see that the Bike object is the central table and has relationships with many other entities. If the query is not carefully worded, it will not cover all scenarios of missing or inaccurate data in secondary tables. Using the Bike specification allows me to efficiently manage the flow of data queried from the databases. It also supports pagination when implementing the function get Bikes, as well as dynamically adding conditions to the query.

**6.3.2. Front-end Code**

**6.3.2.1. App.js**

const cookies = new Cookies();

const ProtectedRoute = ({ token, redirectPath = '/signin' }) => {

    if (!token) {

        return <Navigate to={redirectPath} replace />;

    }

    return <Outlet />;

};

function App() {

    let reduxToken = null;

    let reduxIsShowPublicNavBar = null;

    reduxToken = useSelector((state) => state.reduxAuthenticate.accessToken);

    reduxIsShowPublicNavBar = useSelector((state) => state.reduxAuthenticate.isShowPublicNavBar);

    if (reduxToken == null && cookies.get('accessToken')) {

        reduxToken = cookies.get('accessToken')

    }

    return (

        <Fragment>

            {reduxIsShowPublicNavBar && <div id="content-wrap">

                <BrowserRouter>

                    {reduxIsShowPublicNavBar && <MenuBar />}

                    {!reduxIsShowPublicNavBar && <SideBar />}

                    <Routes>

                        <Route path='/signin' exact element={<SignIn />} />

                        <Route path='/' exact element={<Home />} />

                        <Route path='/list' exact element={<List />} />

                        <Route path='/list/manual' exact element={<List category={2} />} />

                        <Route path='/list/automatic' exact element={<List category={1} />} />

                        <Route path='/bike/:id' element={<Detail />} />

                        <Route path='\*' element={<Navigate to='/404' />} />

                        <Route path='/404' exact element={<PageNotFound warn={"Website is developed"} />} />

                        <Route element={<ProtectedRoute token={reduxToken} />}>

                            <Route path='/dashboard' exact element={<Dashboard />} />

                            <Route path='/manage-bike/bike-list' exact element={<ManageBikeList />} />

                            <Route path='/manage-bike/bike-create' exact element={<ManageBikeCreate />} />

                            <Route path='/manage-bike/bike-update/:id' exact element={<ManageBikeUpdate />} />

                            <Route path='/manage-bike/bike/:id' element={<ManageBikeDetail />} />

                            <Route path='/manage-bike/category' exact element={<ManageBikeCategory />} />

                            <Route path='/manage-bike/color' exact element={<ManageBikeColor />} />

                            <Route path='/manage-bike/manufacturer' exact element={<ManageBikeManufacturer />} />

                            <Route path='/manage-order/cart-create' exact element={<CreateCart />} />

                            <Route path='/manage-order/order-list' exact element={<ManageOrderList />} />

                            <Route path='/manage-order/order-create' exact element={<CreateOrder />} />

                            <Route path='/manage-order/order/:id' element={<ManageOrderDetail />} />

                            <Route path='/manage-maintenance/maintenance-list' exact element={<ManageMaintain />} />

                            <Route path='/manage-maintenance/maintenance-create' exact element={<ManageMaintainCreate />} />

                            <Route path='/manage-maintenance/maintenance/:id' element={<ManageMaintainDetail />} />

                        </Route>

                    </Routes>

                </BrowserRouter>

            </div>}

            {!reduxIsShowPublicNavBar && <BrowserRouter>

                {reduxIsShowPublicNavBar && <MenuBar />}

                {!reduxIsShowPublicNavBar && <SideBar />}

                <Routes>

                    <Route path='/signin' exact element={<SignIn />} />

                    <Route path='/' exact element={<Home />} />

                    <Route path='/list' exact element={<List />} />

                    <Route path='/list/manual' exact element={<List category={2} />} />

                    <Route path='/list/automatic' exact element={<List category={1} />} />

                    <Route path='/bike/:id' element={<Detail />} />

                    <Route path='\*' element={<Navigate to='/404' />} />

                    <Route path='/404' exact element={<PageNotFound warn={"Website is developed"} />} />

                    <Route element={<ProtectedRoute token={reduxToken} />}>

                        <Route path='/dashboard' exact element={<Dashboard />} />

                        <Route path='/manage-bike/bike-list' exact element={<ManageBikeList />} />

                        <Route path='/manage-bike/bike-create' exact element={<ManageBikeCreate />} />

                        <Route path='/manage-bike/bike-update/:id' exact element={<ManageBikeUpdate />} />

                        <Route path='/manage-bike/bike/:id' element={<ManageBikeDetail />} />

                        <Route path='/manage-bike/category' exact element={<ManageBikeCategory />} />

                        <Route path='/manage-bike/color' exact element={<ManageBikeColor />} />

                        <Route path='/manage-bike/manufacturer' exact element={<ManageBikeManufacturer />} />

                        <Route path='/manage-order/cart-create' exact element={<CreateCart />} />

                        <Route path='/manage-order/order-list' exact element={<ManageOrderList />} />

                        <Route path='/manage-order/order-create' exact element={<CreateOrder />} />

                        <Route path='/manage-order/order/:id' element={<ManageOrderDetail />} />

                        <Route path='/manage-maintenance/maintenance-list' exact element={<ManageMaintain />} />

                        <Route path='/manage-maintenance/maintenance-create' exact element={<ManageMaintainCreate />} />

                        <Route path='/manage-maintenance/maintenance/:id' element={<ManageMaintainDetail />} />

                    </Route>

                </Routes>

            </BrowserRouter>}

            {reduxIsShowPublicNavBar && <Footer />}

        </Fragment>

    )

}

export default App;

App.js is one of the most crucial ReactJS files. It can be claimed that when the project launches and index.js is called, this component is the first to be rendered. The code in this file is mostly concerned with using the React Router to render the project's pages. On successful login, I grant authorization to visit private sites using a JWT token saved in redux. In addition, I set up the menus and footers so that they appear correctly on each page. The footer and menu bar will appear on public sites, but only the side bar menu will appear on private sites.

**6.3.2.2. ManageBikeCategory.js**

const cookies = new Cookies();

const SortBy = [

    { value: "id", label: "Sort by ID", key: "1" },

    { value: "name", label: "Sort by name", key: "2" },

    { value: "price", label: "Sort by price", key: "3" },

];

const showAlert = (setAlert, message, isSuccess) => {

    if (isSuccess) {

        setAlert({

            alertShow: true,

            alertStatus: "success",

            alertMessage: message

        })

    } else {

        setAlert({

            alertShow: true,

            alertStatus: "error",

            alertMessage: message

        })

    }

}

const handleGetDataPagination = async (

    setListData,

    setLoadingData,

    setTotalPages,

    reduxFilter,

    reduxPagination

) => {

    const body = {

        searchKey: reduxFilter.reduxSearchKey,

        page: reduxPagination.reduxPage,

        limit: reduxPagination.reduxRowsPerPage,

        sortBy: reduxFilter.reduxSortBy,

        sortType: reduxFilter.reduxSortType

    };

    await AxiosInstance.post(CategoryAPI.getPagination, body, {

        headers: { Authorization: `Bearer ${cookies.get('accessToken')}` }

    }).then((res) => {

        var listData = res.data.data.content.map((data) => {

            return {

                id: data.id,

                name: data.name,

                price: GetFormattedCurrency(data.price)

            }

        })

        setListData(listData)

        setTotalPages(res.data.data.totalPages)

        setTimeout(() => {

            setLoadingData(false)

        }, 500);

    }).catch((error) => {

        if (error && error.response) {

            console.log("Error: ", error);

        }

    });

};

const handleGetDataById = async (dataID, setLineItem) => {

    await AxiosInstance.get(CategoryAPI.getById + dataID, {

        headers: { Authorization: `Bearer ${cookies.get('accessToken')}` }

    }).then((res) => {

        if (res.data.code === 1) {

            setLineItem(res.data.data);

        }

    }).catch((error) => {

        if (error && error.response) {

            console.log("Error: ", error);

        }

    });

}

const handleCreateData = async (

    values,

    setAlert,

    setLoadingData,

    setShowCloseButton

) => {

    const body = {

        name: values.name,

        price: ParseCurrencyToNumber(values.price),

    };

    await AxiosInstance.post(CategoryAPI.create, body, {

        headers: { Authorization: `Bearer ${cookies.get('accessToken')}` }

    }).then((res) => {

        if (res.data.code === 1) {

            showAlert(setAlert, res.data.message, true);

            setLoadingData(true)

            setShowCloseButton(true);

        } else {

            showAlert(setAlert, res.data.message, false);

        }

    }).catch((error) => {

        if (error && error.response) {

            console.log("Error: ", error);

        }

        showAlert(setAlert, error, false);

    });

}

const handleUpdateData = async (

    values,

    dataID,

    setAlert,

    setLoadingData,

    setShowCloseButton

) => {

    const body = {

        name: values.name,

        price: ParseCurrencyToNumber(values.price),

    };

    await AxiosInstance.post(CategoryAPI.update + dataID, body, {

        headers: { Authorization: `Bearer ${cookies.get('accessToken')}` }

    }).then((res) => {

        if (res.data.code === 1) {

            showAlert(setAlert, res.data.message, true);

            setLoadingData(true)

            setShowCloseButton(true);

        } else {

            showAlert(setAlert, res.data.message, false);

        }

    }).catch((error) => {

        if (error && error.response) {

            console.log("Error: ", error);

        }

        showAlert(setAlert, error, false);

    });

}

const handleDeleteData = async (

    dataID,

    setAlert,

    setLoadingData,

    setShowCloseButton

) => {

    await AxiosInstance.post(CategoryAPI.delete + dataID, {}, {

        headers: { Authorization: `Bearer ${cookies.get('accessToken')}` }

    }).then((res) => {

        if (res.data.code === 1) {

            showAlert(setAlert, res.data.message, true);

            setLoadingData(true)

        } else {

            showAlert(setAlert, res.data.message, false);

        }

        setShowCloseButton(true);

    }).catch((error) => {

        if (error && error.response) {

            console.log("Error: ", error);

        }

        showAlert(setAlert, error, false);

    });

}

function ManageBikeCategory() {

    // Show Public Navigation

    const dispatch = useDispatch();

    const [loadingPage, setLoadingPage] = useState(true);

    if (loadingPage === true) {

        dispatch(reduxAuthenticateAction.updateIsShowPublicNavBar(false));

        dispatch(reduxAction.setSortType('DESC'));

        dispatch(reduxAction.setSortBy('id'));

        setLoadingPage(false);

    }

    // Table variables

    const tableTitleList = [

        { name: 'ID', width: '10%' },

        { name: 'NAME', width: '20%' },

        { name: 'PRICE', width: '20%' },

    ]

    // Formik variables

    const initialValues = {

        name: "",

        price: 0,

    };

    // Redux - Filter form

    let reduxFilter = {

        reduxSearchKey: useSelector((state) => state.redux.searchKey),

        reduxSortBy: useSelector((state) => state.redux.sortBy),

        reduxSortType: useSelector((state) => state.redux.sortType),

    }

    const reduxIsSubmitting = useSelector((state) => state.redux.isSubmitting);

    // Redux - Pagination

    const [totalPages, setTotalPages] = useState(1);

    let reduxPagination = {

        reduxPage: useSelector((state) => state.reduxPagination.page),

        reduxRowsPerPage: useSelector((state) => state.reduxPagination.rowsPerPage)

    }

    // Table useState

    const [loadingData, setLoadingData] = useState(true);

    const [listData, setListData] = useState([]);

    // Popup useState

    const [dataID, setDataID] = useState(0);

    const [lineItem, setLineItem] = useState({});

    const [showPopup, setShowPopup] = useState(false);

    const [titlePopup, setTitlePopup] = useState("");

    const [showCloseButton, setShowCloseButton] = useState(false);

    const [isDelete, setIsDelete] = useState(false);

    const [isUpdate, setIsUpdate] = useState(false);

    const [alert, setAlert] = useState({

        alertShow: false,

        alertStatus: "success",

        alertMessage: "",

    })

    // useEffect

    // Table loading - page load

    useEffect(() => {

        if (loadingData === true) {

            handleGetDataPagination(setListData, setLoadingData, setTotalPages, reduxFilter, reduxPagination);

        }

    }, [loadingData])

    // Table loading filter submit

    useEffect(() => {

        if (reduxIsSubmitting === true) {

            if (reduxPagination.reduxPage === 1) {

                handleGetDataPagination(setListData, setLoadingData, setTotalPages, reduxFilter, reduxPagination);

            } else {

                dispatch(reduxPaginationAction.updatePage(1));

            }

            dispatch(reduxAction.setIsSubmitting({ isSubmitting: false }));

        }

    }, [reduxIsSubmitting])

    // Table loading pagination - change page

    useEffect(() => {

        handleGetDataPagination(setListData, setLoadingData, setTotalPages, reduxFilter, reduxPagination);

    }, [reduxPagination.reduxPage])

    // Table loading pagination - change row per page -> call above useEffect

    useEffect(() => {

        if (reduxPagination.reduxPage === 1) {

            handleGetDataPagination(setListData, setLoadingData, setTotalPages, reduxFilter, reduxPagination);

        } else {

            dispatch(reduxPaginationAction.updatePage(1));

        }

    }, [reduxPagination.reduxRowsPerPage])

    // Trigger Get Data by ID API

    useEffect(() => {

        if (isDelete === false && dataID !== 0) {

            handleGetDataById(dataID, setLineItem);

        }

    }, [isDelete, dataID])

    // Update initialValues

    if (isUpdate === true && Object.keys(lineItem).length !== 0) {

        initialValues.name = lineItem.name;

        initialValues.price = GetFormattedCurrency(lineItem.price);

    }

    // Popup Interface

    let popupTitle;

    if (titlePopup === "Create") {

        popupTitle = <Popup showPopup={showPopup} title={"Create"} child={

            showCloseButton ?

                < Fragment >

                    <AlertMessage

                        isShow={alert.alertShow}

                        message={alert.alertMessage}

                        status={alert.alertStatus}

                    />

                    <div className="popup-button">

                        <button className="btn btn-secondary btn-cancel-view"

                            onClick={() => {

                                setShowPopup(false);

                                setShowCloseButton(false);

                                setAlert({ alertShow: false });

                                dispatch(reduxPaginationAction.updatePage(1));

                            }}>Close</button>

                    </div>

                </ Fragment>

                :

                <Fragment>

                    <AlertMessage

                        isShow={alert.alertShow}

                        message={alert.alertMessage}

                        status={alert.alertStatus}

                    />

                    <Formik

                        initialValues={initialValues}

                        onSubmit={(values) => {

                            handleCreateData(

                                values,

                                setAlert,

                                setLoadingData,

                                setShowCloseButton

                            );

                        }}>

                        {({

                            isSubmitting,

                            handleChange,

                            handleBlur,

                            handleSubmit,

                            values,

                            errors,

                            touched,

                            setFieldValue,

                        }) => (

                            <Form className="d-flex flex-column">

                                <TextFieldCustom

                                    label={"Name"}

                                    name={"name"}

                                    type={"text"}

                                    placeholder={"Enter the category name"}

                                />

                                <TextFieldCustom

                                    label={"Price"}

                                    name={"price"}

                                    type={"number"}

                                    onWheel={(e) => e.target.blur()}

                                    placeholder={"Enter the category price"}

                                    onChange={(event) => {

                                        let value = event.target.value;

                                        let decimalValue = ParseCurrencyToNumber(InputNumber(value))

                                        setFieldValue("price", GetFormattedCurrency(decimalValue));

                                    }}

                                />

                                <div className="popup-button">

                                    <button className="btn btn-secondary btn-cancel"

                                        onClick={() => {

                                            setShowPopup(false);

                                            setAlert({ alertShow: false })

                                        }}>Cancel</button>

                                    <button className="btn btn-primary btn-action" type="submit">{titlePopup}</button>

                                </div>

                            </Form>

                        )}

                    </Formik>

                </Fragment>

        } />

    } else if (titlePopup === "Update") {

        popupTitle = <Popup showPopup={showPopup} setShowPopup={setShowPopup} title={"Update"} child={

            showCloseButton ?

                < Fragment >

                    <AlertMessage

                        isShow={alert.alertShow}

                        message={alert.alertMessage}

                        status={alert.alertStatus}

                    />

                    <div className="popup-button">

                        <button className="btn btn-secondary btn-cancel-view"

                            onClick={() => {

                                setShowPopup(false);

                                setShowCloseButton(false);

                                setAlert({ alertShow: false });

                                setDataID(0);

                                setIsUpdate(false)

                            }}>Close</button>

                    </div>

                </ Fragment>

                :

                <Fragment>

                    <AlertMessage

                        isShow={alert.alertShow}

                        message={alert.alertMessage}

                        status={alert.alertStatus}

                    />

                    <Formik

                        enableReinitialize

                        initialValues={initialValues}

                        onSubmit={(values) => {

                            handleUpdateData(

                                values,

                                dataID,

                                setAlert,

                                setLoadingData,

                                setShowCloseButton

                            );

                        }}>

                        {({

                            isSubmitting,

                            handleChange,

                            handleBlur,

                            handleSubmit,

                            values,

                            errors,

                            touched,

                            setFieldValue,

                        }) => (

                            <Form className="d-flex flex-column">

                                <TextFieldCustom

                                    label={"Name"}

                                    name={"name"}

                                    type={"text"}

                                    placeholder={"Enter the category name"}

                                    disabled={true}

                                />

                                <TextFieldCustom

                                    label={"Price"}

                                    name={"price"}

                                    type={"text"}

                                    onWheel={(e) => e.target.blur()}

                                    placeholder={"Enter the category price"}

                                    onChange={(event) => {

                                        let value = event.target.value;

                                        let decimalValue = ParseCurrencyToNumber(InputNumber(value))

                                        setFieldValue("price", GetFormattedCurrency(decimalValue));

                                    }}

                                />

                                <div className="popup-button">

                                    <button className="btn btn-secondary btn-cancel"

                                        onClick={() => {

                                            setShowPopup(false);

                                            setAlert({ alertShow: false });

                                            setDataID(0); setIsUpdate(false)

                                        }}>Cancel</button>

                                    <button className="btn btn-primary btn-action" type="submit">{titlePopup}</button>

                                </div>

                            </Form>

                        )}

                    </Formik>

                </Fragment>

        } />

    } else if (titlePopup === "View") {

        popupTitle = <Popup showPopup={showPopup} setShowPopup={setShowPopup} title={"View"} child={

            <Fragment>

                <div className='popup-view-container'>

                    <div className="popup-view-body">

                        <Row>

                            <Col lg={6} xs={6}><Grid3x3Icon className='body-icon' /><label className="body-title">Category Id</label></Col>

                            <Col lg={6} xs={6}><CategoryIcon className='body-icon' /><label className="body-title">Category Name</label></Col>

                            <Col lg={6} xs={6}><p className='body-detail'>{lineItem.id}</p></Col>

                            <Col lg={6} xs={6}><p className='body-detail'>{lineItem.name}</p></Col>

                            <Col lg={6} xs={6}><DateRangeIcon className='body-icon' /><label className="body-title">Create Date</label></Col>

                            <Col lg={6} xs={6}><PersonIcon className='body-icon' /><label className="body-title">Create User</label></Col>

                            <Col lg={6} xs={6}><p className='body-detail'>{GetFormattedDate(lineItem.createdDate)}</p></Col>

                            <Col lg={6} xs={6}><p className='body-detail'>{lineItem.createdUser}</p></Col>

                            <Col lg={6} xs={6}><UpdateIcon className='body-icon' /><label className="body-title">Modified Date</label></Col>

                            <Col lg={6} xs={6}><ManageAccountsIcon className='body-icon' /><label className="body-title">Modified User</label></Col>

                            <Col lg={6} xs={6}><p className='body-detail'>{lineItem.modifiedDate === null ? "N/A" : GetFormattedDate(lineItem.modifiedDate)}</p></Col>

                            <Col lg={6} xs={6}><p className='body-detail'>{lineItem.modifiedUser === null ? "N/A" : lineItem.modifiedUser}</p></Col>

                            <Col lg={12} xs={12}><AttachMoneyIcon className='body-icon' /><label className="body-title">Price</label></Col>

                            <Col lg={12} xs={12}><p className='body-detail'>{GetFormattedCurrency(lineItem.price)}</p></Col>

                        </Row>

                    </div>

                    <div className="popup-view-footer">

                        <div className="popup-button">

                            <button className="btn btn-secondary btn-cancel-view"

                                onClick={() => {

                                    setShowPopup(false);

                                    setDataID(0)

                                }}>Cancel</button>

                        </div>

                    </div>

                </div>

            </Fragment >

        } />

    } else if (titlePopup === "Delete") {

        popupTitle = <Popup showPopup={showPopup} setShowPopup={setShowPopup} title={"Delete ID " + dataID} child={

            showCloseButton ?

                < Fragment >

                    <AlertMessage

                        isShow={alert.alertShow}

                        message={alert.alertMessage}

                        status={alert.alertStatus}

                    />

                    <div className="popup-button">

                        <button className="btn btn-secondary btn-cancel-view"

                            onClick={() => {

                                setShowPopup(false);

                                setShowCloseButton(false);

                                setAlert({ alertShow: false });

                                setDataID(0);

                                setIsDelete(false)

                            }}>Close</button>

                    </div>

                </ Fragment>

                :

                <Fragment>

                    <div className='popup-message text-center mb-3'>

                        <label>Do you really want to delete this record?</label>

                        <p>This process cannot be undone</p>

                    </div>

                    <div className="popup-button">

                        <button className="btn btn-secondary btn-cancel"

                            onClick={() => {

                                setShowPopup(false);

                                setDataID(0);

                                setIsDelete(false)

                            }}>Cancel</button>

                        <button className="btn btn-danger btn-action"

                            onClick={() => handleDeleteData(

                                dataID,

                                setAlert,

                                setLoadingData,

                                setShowCloseButton

                            )}>{titlePopup}</button>

                    </div>

                </Fragment >

        } />

    }

    // Table - Pagination

    let tablePagination;

    if (listData.length > 0) {

        tablePagination = <div className='table-pagination'>

            <TableCRUD

                tableTitleList={tableTitleList}

                listData={listData}

                setShowPopup={setShowPopup}

                setTitlePopup={setTitlePopup}

                setDataID={setDataID}

                setIsDelete={setIsDelete}

                setIsUpdate={setIsUpdate}

                isShowDeleteBtn={false}

            />

            <PaginationCustom

                totalPages={totalPages}

            />

        </div>

    } else {

        tablePagination = <div className='text-center'>

            <label style={{ fontSize: '36px' }}>No data found</label>

        </div>

    }

    return (

        !loadingData ?

            <Fragment>

                <div className='container'>

                    {popupTitle}

                    <h2 className="text-center">Management Bike Category</h2>

                    <SortBar SortBy={SortBy} />

                    <div className='table-header'>

                        <Row>

                            <Col lg={6} xs={6}><label style={{ fontSize: '36px' }}>Bike Category List</label></Col>

                        </Row>

                    </div>

                    {tablePagination}

                </div>

            </Fragment>

            :

            <Fragment>

                <PageLoad />

            </Fragment>

    )

}

export default ManageBikeCategory;

This is the source code for the bike category management page, which is a typical project website. As we can see, the structure of this source code consists of an exported main function that calls other components and contains the JSX, useState, and useEffect logic. This method is the website's interface, through which ReactJS transforms the code to standard HTML and CSS. Additionally, we can observe that this file's top contains functions that support or call the back-end API via "axios" library. It's a fantastic ReactJS library that allows developers to effectively customize and call the specified API from the back-end side.

**6.3.2.3. MenuBar component**

const MenuBar = () => {

    return (

        <Fragment>

            <Navbar key="lg" expand="lg" className="navbar-light px-3 px-lg-5" sticky="top" bg="light" role="navigation" collapseOnSelect >

                <Container fluid>

                    <Navbar.Brand href="/">Rent Motorcycles</Navbar.Brand>

                    <Navbar.Toggle aria-controls={`offcanvasNavbar-expand-lg`} />

                    <Navbar.Offcanvas

                        id={`offcanvasNavbar-expand-lg`}

                        aria-labelledby={`offcanvasNavbarLabel-expand-lg`}

                        placement="end"

                    >

                        <Offcanvas.Header closeButton>

                            <Offcanvas.Title id={`offcanvasNavbarLabel-expand-lg`}>

                                Rent Motorcycles

                            </Offcanvas.Title>

                        </Offcanvas.Header>

                        <Offcanvas.Body>

                            <Nav className="me-auto" variant="pills">

                                <Nav.Link className="nav-item mx-lg-4" href="/">Home</Nav.Link>

                                {/\* <Nav.Link className="nav-item mx-lg-4" href="/about-us">About Us</Nav.Link> \*/}

                                <NavDropdown

                                    title="List of motorbikes"

                                    id={`offcanvasNavbarDropdown-expand-lg`}

                                >

                                    <NavDropdown.Item href="/list/manual">Manual Transmission Motorcycle</NavDropdown.Item>

                                    <NavDropdown.Item href="/list/automatic">Automatic Transmission Motorcycle</NavDropdown.Item>

                                    <NavDropdown.Divider />

                                    <NavDropdown.Item href="/list">See all</NavDropdown.Item>

                                </NavDropdown>

                                {/\* <Nav.Link className="nav-item mx-lg-4" href="/contact-us">Contact Us</Nav.Link> \*/}

                            </Nav>

                        </Offcanvas.Body>

                    </Navbar.Offcanvas>

                </Container>

            </Navbar>

        </Fragment>

    )

}

export default MenuBar;

Finally, I'll discuss the project's basic component. This component is the menu bar that may be seen in the public pages' header. To create the UI, this component mostly makes use of the React Bootstrap library. This is one of the most fundamental libraries for creating interfaces with ReactJS. We observe that it provides pre-built child components so that we can personally design a menu bar in accordance with the requirements of the customer. This clearly demonstrates how the Navbar component is being used most often. All other pages can call and utilise this menu after the menu bar is finished. Because of the permission to display component, I called this component in App.js.

**6.4. Evaluation of our product (good/bad)**

In section 1.5, I provided an overview of the project's final product. However, that is only a brief overview of this website. In this section, I will go over the Motorcycle Rental Management System project in further detail. I'll list some of the criteria that are used to compare a web application to in the community. We will discuss each criterion and rank it from 0 to 10, with 0 being the worst and 10 being the greatest. The scores can be scaled as follows:

0 - 2: Very bad

3 – 4: Bad

5 - 6: Normal

7 – 8: Good

9 - 10: Excellent

* Criteria 1 – Usability: This is the criterion used to determine whether or not the website application is simple to use. Because this website does not have many complex functionalities, the majority of the functions are relatively simple to use for both customer and admin users. Customers only see the vehicle information and a simple filter on the Bike List website. It's a little more complicated for administrators; they must comprehend the process as well as the user handbook to understand these functions. Nonetheless, the project's usability is quite approachable. I believe that with just one tutorial, admin will be able to use this system. Based on the factors listed above, I determined that the first criterion was worth 9/10 points.
* Criteria 2 – Speed: This is the criterion for evaluating the speed with which the online application loads and responds to user activity. The responsiveness of the website is fairly good due to the simple functionalities. Only image-related tasks, such as adding new motorcycles, have a faster reaction time than other average functions. The reason for this is that this function requires the assistance of a third party, Firebase. When I save a new image to the system, the MySQL server simply saves the path, while the image gets saved in Firebase. As a result, Firebase will need more time to process the motorcycle image. However, this is not a major issue because Firebase's additional processing time is only a few seconds. Based on these considerations, I will assign this criterion a score of 9/10.
* Criteria 3 – Security: This is the criterion used to assess information security and user safety, as well as to prevent hacker attack tactics. Decentralization is handled in two tiers, from the front end to the back end, for the Motorcycle Rental Management System project. I'll go over some of the security aspects of the project. The first is a decentralized project that uses the JWT token method and is implemented by the Spring Security framework, which means that all APIs under the Back-end must be authorized in order to function. JWT tokens are becoming more popular since they are encrypted into a random string and saved on the front end. This token has been stored in the website's cookie by me, and we will need it to access authenticated sites. This is the router mechanism's security for entering subsites. Secondly, the database contains an encrypted version of the account's password. So, even if the IT department has access to the database, it cannot determine the password. When a login request is made, the entered password and the password in the database are encrypted and matched to allow a successful login. The following part is the user's tracking action. I created a history database to track every data modification and login actions. The IT department may monitor all information from this table. When data is updated, it also preserves the previous values. As a result, the system's security is improved. That is the system's main strength. However, I have yet to implement features such as locking the account after a number of unsuccessful logins, lost password, reset password, and so on. These are the functions that an authenticated system should have. When an administrator forgets their password, they must contact IT to have it reset with the code. It's inconvenient, but it has no impact on security. Additionally, I lack the ability to tell users when their passwords have changed after a set amount of time, nor the logic to verify that the password contains special characters, uppercase letters, lowercase letters, and numerals. I give the security factor 8/10 based on these considerations.
* Criteria 4 – Compatibility: This criterion is used to assess how well the web application runs on all different devices and browsers. The project is currently designed to be responsive and to display effectively on tiny screen electronic devices such as smartphones and tablets. I utilized React Bootstrap to provide UI resizing based on screen size. As a result, I rate this compatibility factor 9/10.
* Criteria 5 – Design: This is the evaluation criterion for the interface as well as the system's intuitiveness to determine whether or not it is user-friendly. For this project, I appreciate the style and feel of the public pages because they were designed to entice new clients. However, the interface of the admin's management pages is not very unique or professional. Forms and components are all built using the most fundamental ReactJS libraries. I usually use the original components and only make minor changes. As a result, I give the admin interface a low rating because it has several limitations. However, when viewed overall, the interface is pretty harmonious and not overly vibrant. This criterion gets a 6/10 from me.
* Criteria 6 – Content: This is the criterion for assessing the website's content in terms of relevancy and usefulness. In general, all project information is available for users to use. However, it is limited in terms of language and grammar. Because this website was designed to be used by Vietnamese people, the sentences that have been translated into English are not very coherent. The terminology is often extremely rudimentary, yet it nonetheless helps users understand the meaning of the functions. I only rate this criterion a 6/10.
* Criteria 7 – Performance: This is the criterion used to assess how well the website operates under high traffic conditions. This criterion is currently difficult to evaluate because a large portion of traffic comes from public pages accessible by anonymous people. Because there are only functions that display information, practically every function works well. The number of users on authenticated sites is quite small. However, if the number of administrators exceeds 100, the system may experience slowness as well as logic errors in the access flow. Currently, the Motorcycle Rental Management System project lacks code to optimize multi-threaded access. As a result, I will assign this criterion a score of 6/10.
* Criteria 8 – Maintenance and Support: This criterion evaluates how well the system is maintained and updated in the future for new functions, as well as how responsive the IT team responds to user complaints. The first is about maintenance; all of the logic in the code, from front-end to back-end, has now been streamlined and readily reused by me. We will see this more clearly at the back end because each function is coded separately, and the code may be reused efficiently. However, some front-end source components, such as tables, are not fully optimized. The next section is about support. Because I will be working solely on this project, I will also be working in the IT department. If the project is implemented, only I will assist the customer in maintaining and updating the system. Because I am alone, I believe the help capability will be limited if a large number of requests come in. Based on the foregoing, I will award this criterion 7/10 points.

We went through several criteria to evaluate the Motorcycle Rental Management System project's products. As can be observed, the most typical point range is between 6 and 9 points. Thus, even if the product did not receive a perfect rating, it is still incredibly well-liked. It still serves the demands of simple users. This is an excellent result for a job accomplished by a single person.

**Chapter 7: Conclusions**

**7.1. What you have learned in this project?**

In this section, I will discuss some of the lessons I gained while implementing the Motorcycle Rental Management System project.

* Learn new techniques: This project taught me a lot about two of the most popular frameworks at the time, Spring Boot and ReactJS. For each framework, I was able to grasp all of the fundamentals and apply them to any project. In addition, I learnt a lot of other information, such as the libraries that support these two frameworks. I also learned how to use helpful tools like as Visual Studio Code, IntelliJ, and Postman, among others. Overall, this project has greatly expanded my understanding about information technology.
* Coding Skills: Coding skills are one of the most significant aspects for programmers. Working on this project by myself forced me to develop from front-end to back-end, which improved my coding abilities. There will be distinct conventions and code forms for different programming languages. By coding the entire project, I was able to understand the architecture of the source code for each framework, how to name variables, how to optimize a function, and so on.
* Problem-solving: This is perhaps one of the most essential skills I have developed during this assignment. When I receive a request from a customer, I must consider how to execute the request as well as convert all of their needs into code. In addition, some functions are quite tough to implement, thus I must brainstorm to find a solution. As a programmer, I frequently run into bugs. There are some bugs that are difficult to deal with, and I have to deal with them all the time. This initiative has offered an excellent environment for me to improve my problem-solving abilities.
* Project Management: In addition to coding and problem solving, I learned about project management. As I stated in Section 1, because I am working on the project alone, I must understand the whole software development process. To operate my own project, I had to study a lot from internet sources about typical ideas like planning and organization. Currently, I have sufficient project management knowledge. I am always convinced that I can take on any genuine project and swiftly adjust to it.
* Documentation: As I work on this project, I am frequently required to create documentation for each phase. I had to finish the project proposal in the early periods. Following the planning step, I had to collect and document customer information in order to develop backlogs and user stories. In the design and coding process, I had to record each function and how far the interface was developed, as well as how the code and Api for each function worked. In addition, I have to keep track of both the bugs and the solutions. The report writing portion is the most unique towards the end of this assignment. The report-writing phase is the most distinctive towards the end of this project. Needless to say, this is where I spend the most of my documentation time. My document writing abilities have greatly increased since then.

**7.2. What is the result of this project?**

We're almost done with the report. In this section, I will give the project's results for the Motorcycle Rental Management System:

* The final product is the project's first result. Because everything is focused on the product, this is the most significant outcome of the project. We developed a motorbike rental management system that runs on the website as part of this project. We created several public pages that everyone can access. This website is accessible by PC, laptop, and mobile device. This product can also be put to use by individuals or businesses that provide this motorbike rental service. The most crucial aspect of the product is education. The instructor will utilize this assignment to assess me regarding the educational standards of Greenwich Vietnam.
* The project's second result is the final report and proposal. These are the two most crucial documents for summarizing the entire project, as well as the report that will be used to evaluate my achievements on the project. The proposal is the summary and ideation form of this project that has been accepted by the teacher, whilst the final report is the document that contains all of the project information according to the Greenwich Vietnam University pattern. This report is separated into seven chapters that correlate to knowledge and planning, as well as actual images and source code.
* Following that is a presentation. In this session, I will present my project to the instructor and other participants. Based on my demonstration, the teachers will assess me and assign me grades in accordance with the university's training program standards.
* The finished product is a personal portfolio. Working on the Motorcycle Rental Management System project has given me confidence that I have learnt and am familiar with knowledge that can be used to real-world tasks. This project will help me improve my portfolio and advance my profession. It is the biggest and most passionate make an effort in my portfolio. I can interview with enthusiasm and confidence for jobs that utilize two well-known frameworks for this project. Although this result is unimportant in the viewpoint of the University of Greenwich, it is the accomplishment in that effort that I am proudest of.

**7.3. Further development of this project**

In this final section, I will discuss the potential development of the Motorcycle Rental Management System project.

* Improved design and user experience on the web sites. As I mentioned in the preceding sections, the UI of the project is not very professional due to time constraints and the fact that I am the only one that does it. To make this interface more aesthetically pleasing in the eyes of users, I believe it needs to be greatly enhanced.
* Expansion of the system to include more types of motorcycles: At the moment, the initiative exclusively rents two types of motorcycles: manual and automatic transmission motorcycles. I hope that in the future, the system can be expanded to rent more innovative vehicles such as electric motorcycles, cars, and so on.
* Add a function to calculate monthly and yearly motorcycle rentals, as well as a variety of other formulas: At the moment, I'm working on a database that allows for flexible computation and formula editing. However, the system is simply saving and applying a simple consumer formula. If a customer rents a motorcycle for an extended period of time, I believe the calculation will change. It's great that the system can manage so many different payment formulas.
* Add a feature that allows administrators to change the formula used to calculate motorcycle rentals on the website. The formulas are kept and directly used in the code and database. When there are more formulas, I believe it is beneficial to establish a website where admins can change the formulas. This will greatly assist administration, since they will no longer need to contact the IT department to update formulas.
* Develop mobile applications. This is one of the later bootable projects for admin functions. Those who want to hire motorcycles only need to look at the material on the website. However, motorbike rental service providers want an application that allows them to check reports on the phone as well as add and delete information on mobile.
* Add online payment capabilities via banks or third-party services. Currently, no money has been paid for this project. When the order is successfully established, we can add an online payment function for the consumer. As a result, there will be an additional status in the order that has been paid or has not yet been paid. This will be useful in providing customers with immediate payment. However, this function is completely unnecessary. Because most tourists rent through the hotel, they will mostly pay for the hotel. This function is intended only for guests who wish to rent motorcycles directly from the motorcycle rental service.
* Include GPS positioning for each motorcycle. This is a fantastic function, but it is fraught with danger. The development of this capability will assist the automobile lessor in monitoring the motorcycle in order to reduce the chance of the motorcycle being stolen or carried away by robbers. However, it breaches customer information. Because we will be monitoring all of the people's activities, they will feel uneasy knowing that they are being tracked by GPS positioning. Therefore, when carrying out this task, we must ensure that the project does not violate the law and make every effort to be as open as possible.
* Live chat support should be implemented. One of the most useful elements is the development of a virtual worker to assist users while they use the project's functionality. AI will present and answer queries on hiring vehicles for customers that hire motorcycles. The AI will provide instructions and suggestions for each function, as well as clarify the system's terms, to the administrator. It is an interesting function, but it necessitates time, money, and staff investment. I hope to be able to build an AI that supports live chat in the future.

All of the functions that the Motorcycle Rental Management System project may enhance are listed above. I hope that this project will be expanded with these new functionalities in order to diversify it.

REFERENCES

**PROJECT PROPOSAL**

**APPENDIX 1**