End of Studies Project

Mohamed Dhia Ben Amar

September 21, 2024

Dedication

This work is dedicated to

My family, whose unwavering support and countless sacrifices have been crucial to my success. I hope this work reflects the fruits of your sacrifices and makes you proud, as that is very important to me.

My dear friends, who have consistently believed in me, offering their support through difficult times. I hope for a future for you that is filled with joy and accomplishments. Your belief in me is invaluable.

To all those who have supported me, my gratitude is immense for your encouragement; and to those who doubted me, thank you for pushing me to unleash my full potential against all odds. This is incredibly meaningful to me.

And finally, **to all those who contributed to making this project possible,** I dedicate this work to you.

Mohamed Dhia Ben Amar

Acknowledgments

I express my heartfelt gratitude to all the individuals and organizations that have contributed significantly to the success of my internship.

Special thanks go to Mr. Sami Kammoun, my supervisor at Converty, for his warm welcome and invaluable mentorship throughout my internship. His dedication and expertise have been crucial to my professional growth and achievements.

I am also profoundly grateful to Ms. Mariem Bouzouita, my academic supervisor at ESPRIT, for her insightful guidance, support, and patience during my internship. Her advice and oversight have greatly shaped my approach and understanding of our field.

Additionally, I extend my thanks to Converty, the host organization, for providing a supportive and dynamic environment that has been instrumental in my learning experience and professional development.

Preface

This internship report chronicles my experience at Converty, an e-Commerce platform, during my third year of engineering studies at ESPRIT (École Supérieure Privée d'Ingénierie et de Technologies). The six-month internship provided me with valuable insight into the field of software engineering and its significant impact on the company's clients.

Converty is a dynamic e-commerce startup based in Jardins Menzah 2, Tunisia, that is transforming the online business landscape in the region. It offers a unique blend of CRM and CMS solutions, enabling businesses to effectively manage and grow their online presence.

As a CMS, Converty simplifies online store creation, customization, product management, and integration with analytics platforms such as Meta, Google, and Microsoft. As a CRM, Converty has strategic partnerships with leading Tunisian delivery companies and fulfillment centers, an extensive order management system, and real-time analytics for performance monitoring.

This report provides a comprehensive analysis of my internship experience at Converty, detailing the challenges faced, the solutions implemented, and the valuable lessons learned. It underscores the importance of software engineering principles in optimizing company processes and driving innovation in the ever-evolving e-commerce industry.

Contents

List of Tables

List of Figures

List of Abbreviations

CRM	Customer Relationship Management
CMS	Content Management System
SDLC	Software Development Life Cycle
COD	Cash on Delivery
UI	User Interface
API	Application Programming Interface
SEO	Search Engine Optimization
ODM	Object Data Modeling
CRUD	Create, Read, Update, Delete

Chapter 1 General Introduction

During the past decade, the e-commerce industry in the MENA region has experienced significant growth. However, the development and adoption of online payment services have not kept pace with this expansion, leading to a substantial gap between demand and availability. As a result, cash-on-delivery (COD) has become the predominant payment method, fueling the sector's growth.

E-Commerce platforms like Shopify are well known for their dominance in online retail, especially in the drop shipping sector. Despite its flexibility, Shopify's extensive feature set can cause performance issues and a steep learning curve for new users. The high number of applications required for various functionalities, each with its own subscription fee, presents a barrier to small businesses. Moreover, the platform's level of abstraction makes extensive customization challenging.

To address these challenges, Converty was created with a focus on enhancing user efficiency and customization. Converty offers a comprehensive CRM and CMS platform designed to streamline the creation of high-converting online stores and manage all aspects of a business from a single platform. This approach eliminates the need for coding, spreadsheets, and additional applications, simplifying the management process.

During my six-month internship at Converty, I worked on resolving critical issues within the CRM and CMS dashboards, addressing performance-related concerns, and completing the mobile app experience. My tasks included enhancing existing features, implementing new functionalities, and improving the overall user experience on the dashboard and the mobile app.

This report provides an in-depth analysis of my internship experience, detailing the challenges encountered, the solutions implemented, and the lessons learned. It is structured as follows:

The first chapter covers the scope of the internship, including an overview of COD and its implications, the specific problems addressed, and the objectives of the internship with actionable items.

The second chapter explores the project setup, including the engineering decisions made, the project environment, and details of the infrastructure and DevOps practices employed.

Subsequent chapters describe the enhancements and implementations performed for the CRM and CMS dashboards and the mobile app, detailing the requirements, design, implementation, and validation processes for each.

The report concludes with a reflection on the internship experience, including insights gained, challenges faced, and recommendations for future improvements.

Chapter 2 Scope Statement

Contents

2.1 Host Company

2.1.1 Converty

Converty is an advanced platform for building and managing high-converting online stores. Converty provides powerful tools and advanced features that enable business owners and merchants to create effective online stores without the need for programming skills.

The platform offers a variety of services including online store management, customer relationship management (CRM), marketing automation, and reporting tools. Converty's mission is to enhance the e-commerce experience for SMEs by delivering an integrated solution that simplifies operations, reduces costs, and boosts efficiency.

2.1.2 Services

Converty offers a comprehensive range of services to help businesses succeed in the e-commerce industry. These services include:

- Creating and customizing online shops: Converty provides a user-friendly platform that allows business owners to easily create and customize their online stores without the need for programming skills.
- Integration with analytics tools: Converty seamlessly integrates with various analytics tools, providing valuable insights into customer behavior, sales performance, and marketing effectiveness.
- Integration with delivery companies: Converty offers integration with popular delivery companies, streamlining the order fulfillment process and ensuring efficient shipping and delivery.
- Order management: Converty provides robust order management features, allowing businesses to efficiently process and track orders, manage inventory, and handle returns and refunds.

- Product management: Converty's platform enables businesses to easily manage their product catalog, including adding new products, updating product information, and organizing products into categories.
- Customer relationship management (CRM): Converty's CRM features help businesses build and maintain strong relationships with their customers, including managing customer profiles, tracking customer interactions, and implementing targeted marketing campaigns.
- Marketing automation: Converty offers powerful marketing automation tools, allowing businesses to automate various marketing activities such as email campaigns, social media promotions, and personalized recommendations.

2.2 Study Of The Existing

Shopify is a popular e-Commerce platform that allows individuals and businesses to create online stores effortlessly. Among its many advantages are the performant templates that ensure fast loading times and a seamless shopping experience for customers.

In addition to high-performing templates, Shopify offers a variety of sales-boosting features. These include abandoned cart recovery, targeted email campaigns, and advanced analytics. These tools help store owners optimize their online sales and increase revenue.

Furthermore, Shopify provides a comprehensive mobile app that allows store owners to manage their business on the go. The app includes features like order management, product updates, and sales tracking, making it convenient to maintain an online store anytime, anywhere.

2.3 Problem Statement

Despite Converty's notable progress in the e-commerce industry, the product currently faces several critical challenges that must be addressed to compete effectively with Shopify.

- **Performance Issues:** One of the four templates provided by Converty has performance problems that negatively affect the overall user experience.
- **Missing Key Features:** The platform lacks essential features such as upselling, which are critical to maximizing sales and improving customer satisfaction.
- **Incomplete Mobile App Experience:** The mobile app experience is incomplete, limiting accessibility and convenience for users who prefer to manage their online stores on mobile devices.

Addressing these challenges is crucial for Converty to fully leverage its potential and offer a robust, competitive solution in the e-commerce market.

2.4 Proposed Solution

After a long discussion with the company, the following fixes were decided to be made:

- **Performance Optimization:** The first template will be recoded using React to enhance performance. This will involve refactoring the codebase to improve efficiency and responsiveness, leading to a significantly better user experience.
- **Upsell Feature Integration:** An upsell feature will be added to the platform, incorporating algorithms and interfaces that recommend complementary products to customers. This feature aims to increase the average order value and boost customer satisfaction.
- **Mobile App Enhancements:** Key mobile features will be incorporated to complete the mobile app experience, ensuring a seamless and intuitive user interface for managing online stores on mobile devices.

These solutions address the critical issues facing Converty, improving its performance, functionality, and overall user experience in the competitive e-commerce market.

2.5 About the Project

2.5.1 Goals

The primary goals of this project are to improve the performance of the Converty platform, introduce key features to improve functionality, and complete the mobile app experience to ensure a seamless user interface across all devices.

2.5.2 Objectives

To achieve these goals, the following objectives were set:

- Recode the first template using React to enhance performance.
- Integrate an upsell feature to boost average order value and customer satisfaction.
- Enhance the mobile app with essential features to provide a comprehensive and user-friendly experience.

2.5.3 Deliverables

- **Source Code:** The complete source code, including all necessary files for building and running the project. The code should be well organized, adhere to the coding standards, and include comments for clarity.
- **Documentation:** Comprehensive documentation covering the project architecture, major modules, functions, and classes. It should also include setup instructions, usage examples, and relevant information for users and developers.
- **Presentation:** A presentation outlining the project's objectives, design, implementation, key features, challenges, and solutions. It should include visual aids such as diagrams, charts, and screenshots.

2.5.4 Management Plan

Software Development Life Cycle (SDLC)

The SDLC employed for this project involved iterative development and continuous feedback to meet all project requirements effectively.

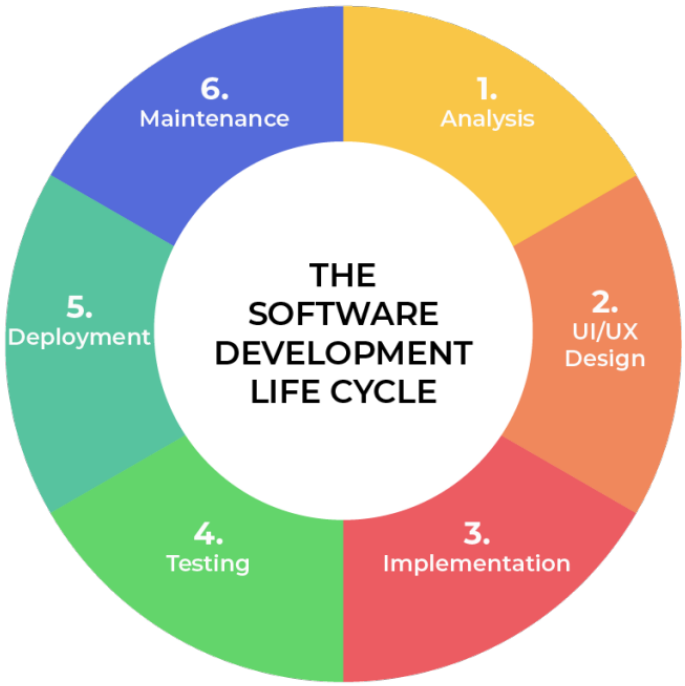

Figure 2.1: SDLC Phases

Comparison of SDLC Models

Model	Advantages	Disadvantages	Suitability
Waterfall Model	Simple and easy to understand and use	Inflexible, difficult to make changes once the process is underway	Suitable for small projects with well-defined requirements
Agile Model	Flexible and adaptive to changes, promotes collaboration and customer feedback	Can be difficult to predict effort required and can lead to scope creep	Suitable for projects with dynamic requirements
Scrum Model	Iterative approach promotes continuous improvement, increases team accountability	Requires experienced team members, can be difficult to implement for complex projects	Suitable for complex projects requiring frequent changes
Kanban Model	Visual workflow management, promotes continuous delivery	Lack of time frames can lead to lack of urgency	Suitable for projects needing continuous delivery and improvement

Table 2.1: Comparison of SDLC Models

Agile Methodology

The Agile methodology emphasizes flexibility, collaboration, customer feedback, and rapid releases. It supports adaptive planning and continuous improvement, making it ideal for dynamic and complex projects. Agile encourages iterative development cycles (sprints) that allow teams to adapt quickly to changing requirements and deliver functional software incrementally.

Scrum Framework

The Scrum framework, a subset of Agile, manages complex product development through iterative cycles known as sprints, which typically last 2-4 weeks. Key roles in Scrum include the Product Owner (responsible for project vision and backlog prioritization), the Scrum Master (facilitates the process and resolves issues), and the Development Team (completes the work). Scrum ceremonies such as sprint planning, daily stand-ups, sprint reviews, and retrospectives ensure ongoing progress and improvement.

Scrum Tools

To support the Scrum framework, the following tools were utilized:

- **GitHub:** For version control and collaborative code management, allowing team members to work

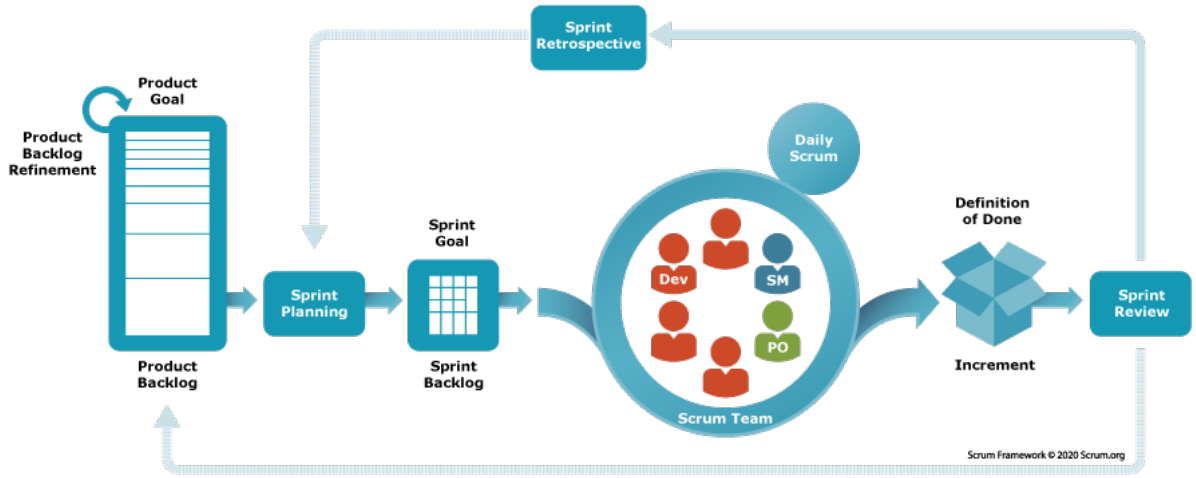

Figure 2.2: Scrum Framework

on code simultaneously, track changes, and manage issues.

- **Google Meet:** For virtual meetings including daily stand-ups, sprint planning, reviews, and retrospectives, ensuring remote team members can collaborate effectively.
- **Notion:** An all-in-one workspace for project management, documentation, and collaboration, helping organize tasks, track progress, and maintain a shared knowledge base.

2.5.5 Timeline

Scrum Team

- Product Owner: Mr. Ayoub Nejem (CEO)
- Scrum Master: Mr. Sami Kammoun (Co-Founder and CTO)

Product Backlog

Sprint	Theme	ID	User Story	Priority
1	Shop Visiting	1.1	As a shop visitor, I want to access the shop landing page	High
		1.2	As a shop visitor, I want to load additional products	High

Sprint	Theme	ID	User Story	Priority
		1.3	As a shop visitor, I want to follow social media links	Medium
		1.4	As a shop visitor, I want to reach out to the support team	Medium
		1.5	As a shop visitor, I want to search for specific products	High
		1.6	As a shop visitor, I want to complete the checkout form	High
		1.7	As a shop visitor, I want to add items to the cart	High
		1.8	As a shop visitor, I want to purchase items	High
		1.9	As a shop visitor, I want to accept or decline offers	Medium
2	Upsell Management	2.1	As a shop admin, I want to display all upsells and cross-sells	High
		2.2	As a shop admin, I want to perform search operations	High
		2.3	As a shop admin, I want to edit upsell and cross-sell details	High
		2.4	As a shop admin, I want to delete upsell and cross-sell entries	Medium
		2.5	As a shop admin, I want to view previews	Medium
3	Order Management	3.1	As a shop admin, I want to display all orders	High
		3.2	As a shop admin, I want to view order details	High
		3.3	As a shop admin, I want to add or edit orders	High
		3.4	As a shop admin, I want to filter orders by status, product, or delivery company	Medium

Sprint	Theme	ID	User Story	Priority
		3.5	As a shop admin, I want to perform search operations	High
4	Financial Management	4.1	As a shop admin, I want to enter various cost parameters	High
		4.2	As a shop admin, I want to calculate key financial metrics	High
		4.3	As a shop admin, I want to view all budgets	High
		4.4	As a shop admin, I want to select specific budgets	Medium
		4.5	As a shop admin, I want to edit budget information	Medium
		4.6	As a shop admin, I want to view total balance, revenue, and expenses	High
5	Statistical Analysis	5.1	As a shop admin, I want to view different types of statistics	High
		5.2	As a shop admin, I want to filter statistical data	Medium
6	Notification Center	6.1	As a shop admin, I want to customize notification sounds	Medium
		6.2	As a shop admin, I want to receive real-time notifications	High

Table 2.2: Product Backlog

Gantt Chart

A Gantt chart visually represents the project schedule, showing the start and end dates of various project elements, and helps track the project's timeline and progress.

2.6 Conclusion

In summary, this chapter outlined the scope of the project, detailing the host company, identified problems, and proposed solutions. The project aimed to optimize Converty's platform performance, add crucial features, and enhance the mobile app experience. The detailed management plan and timeline

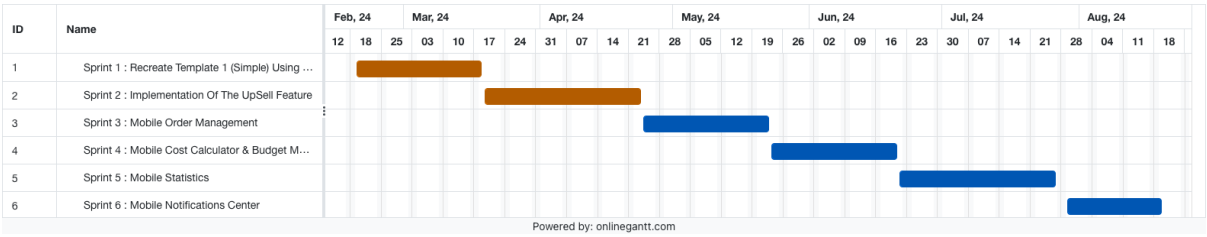

Figure 2.3: Gantt Chart

were presented to ensure effective project execution. By addressing the identified issues, the project aims to significantly improve Converty's functionality and user experience.

Chapter 3 Analysis and Specification of Requirements

Contents

3.1 Introduction

This section aims to define and categorize the essential requirements of the project, distinguishing between functional requirements, which describe specific behaviors and capabilities of the system, and non-functional requirements, which focus on performance, security, accessibility criteria, and other overall qualities.

3.2 Functional Requirements

The functional requirements of our solution include the features that it must incorporate to meet the project's requirements and objectives.

The system must offer **shop visitors** the possibility to:

- View Shop
  - View the shop's products.
  - View the shop's categories.
  - Search for products.
  - View the product's details.
  - Add products to the cart.
  - View the cart.
  - Accept or decline upsells.

The system must offer **shop owners** the possibility to:

- Manage upsells
  - Add upsells.
  - Edit upsells.
  - Delete upsells.
  - View upsells preview.
  - Search for upsells.
- Manage Orders
  - View orders.

  - Search for orders.
  - View order details.
  - Change order status.
  - Filter orders.
- Manage Budgets
  - View budget list.
  - Select budget.
  - Edit budget information.
  - View total balance, expenses, and incomes.
- Manage Costs
  - Input various costs.
  - Calculate total costs.
- Manage Notifications
  - View notifications.
  - Edit notifications sound.
- View Statistics
  - View statistics.
  - Filter statistics.

3.3 Non-functional Requirements

The non-functional requirements of the system are as follows:

- Performance: The system should be able to handle a large number of users and transactions without significant delays.
- Security: The system should protect sensitive data, such as customer information and financial records, from unauthorized access or theft.
- Scalability: The system should be able to grow and adapt to changing business needs, such as increased traffic or new features.
- Reliability: The system should be available and operational at all times, with minimal downtime or disruptions.

- Usability: The system should be easy to use and navigate, with clear instructions and intuitive interfaces.

3.4 Analysis of Requirements

3.4.1 Identification of System Actors

The system involves the following actors:

Actor	**Description**
Shop Owner	The owner of the shop who can customize the template, manage orders, manage upsells, view statistics, customize notifications, manage budget and costs.
Shop Visitor	A visitor who visits the shop built on top of the template.
Superadmin	A special user who takes over temporarily to solve issues related to upsells.

Table 3.1: System Actors

3.4.2 Global Use Case Diagram

The global use case diagram represents the interactions between the system and its actors. It provides an overview of the system's functionality and the actors involved.

- **Actors:** Represents a role played by an external entity (human user, hardware device, or another system) that interacts directly with the system under study.
- **Use Case:** Represents a set of action sequences performed by the system that produce an observable result of interest to a particular actor.
- **Relations between actors:** The only relationship between actors is the generalization relationship. When a child actor inherits from a parent actor, it inherits all the associations of the parent.
- **Relations between use cases:** We can distinguish two types of relationships between use cases:
  - **Association:** Represents a relationship between two use cases. It is represented by a line connecting the two use cases.
  - **Include:** Represents a relationship between two use cases where one use case includes the functionality of another use case. It is represented by a dashed line with an arrow pointing to the included use case.

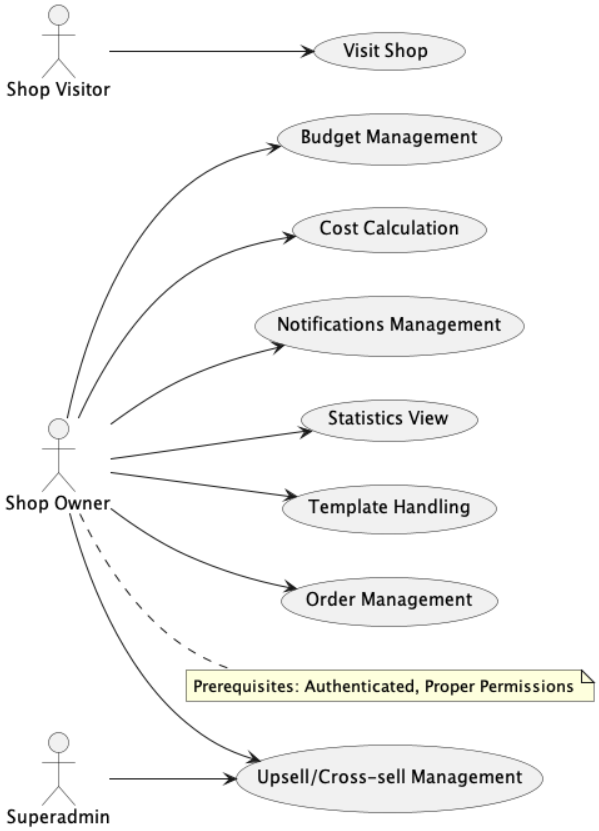

Figure 3.1: Global Use Case Diagram

3.4.3 Refinement of Use Cases

It will be necessary to refine the already developed use case diagram to obtain more details about the features offered by the system and the constraints associated with them. This will allow us to better understand the system's behavior and interactions with its actors.

Use Case 1: View Shop

Title	Visit Shop
Summary	This use case describes the actions performed by a shop visitor to view the shop's products and categories, search for products, view product details, add products to the cart, view the cart, and accept or decline upsells.
Actors	Shop Visitor
Preconditions	The shop visitor has accessed the shop's website.
Postconditions	The shop visitor has viewed the desired products, added products to the cart, and accepted or declined upsells.

Nominal Scenario	1. The shop visitor opens the shop's website. 2. The shop visitor views the shop's products. 3. The shop visitor views the shop's categories. 4. The shop visitor searches for products. 5. The shop visitor selects a product to view its details. 6. The shop visitor adds the product to the cart. 7. The shop visitor views the cart. 8. The shop visitor accepts or declines upsells.
Alternative Scenarios	• The shop visitor does not find the desired products or categories. • The shop visitor does not find any search results. • The shop visitor encounters an error while adding a product to the cart. • The shop visitor encounters an error while viewing the cart.

Use Case 2: Manage Upsells

Title	Manage Upsells
Summary	This use case describes the actions performed by a shop owner to add, edit, delete, view, and search for upsells.
Actors	Shop Owner
Preconditions	• The shop owner has logged into the system. • The shop owner has accessed the upsells management section. • The shop owner has the appropriate permissions to manage upsells.

Postconditions	The shop owner has managed the upsells according to the desired actions.
Nominal Scenario	1. The shop owner logs into the system. 2. The shop owner adds an upsell. 3. The shop owner edits an upsell. 4. The shop owner deletes an upsell. 5. The shop owner views the upsells preview. 6. The shop owner searches for upsells. 7. The shop owner logs out of the system.
Alternative Scenarios	• The shop owner encounters an error while adding an upsell. • The shop owner encounters an error while editing an upsell. • The shop owner encounters an error while deleting an upsell. • The shop owner does not find any search results.

Use Case 3: Manage Orders

Title	Manage Orders
Summary	This use case describes the actions performed by a shop owner to view orders, search for orders, view order details, change order status, and filter orders.
Actors	Shop Owner
Preconditions	• The shop owner has logged into the application. • The shop owner has accessed the orders management tab. • The shop owner has the appropriate permissions to manage orders.
Postconditions	The shop owner has managed the orders according to the desired actions.

Nominal Scenario	1. The shop owner logs into the system. 2. The shop owner views the orders. 3. The shop owner searches for orders. 4. The shop owner selects an order to view its details. 5. The shop owner changes the order status. 6. The shop owner filters the orders. 7. The shop owner logs out of the system.
Alternative Scenarios	• The shop owner does not find any orders. • The shop owner does not find any search results. • The shop owner encounters an error while viewing order details. • The shop owner encounters an error while changing the order status.

Use Case 4: Manage Budgets

Title	Manage Budgets
Summary	This use case describes the actions performed by a shop owner to view the budget list, select a budget, edit budget information, and view the total balance, expenses, and incomes.
Actors	Shop Owner
Preconditions	• The shop owner has logged into the application. • The shop owner has accessed the budgets management section. • The shop owner has the appropriate permissions to manage budgets.
Postconditions	The shop owner has managed the budgets according to the desired actions.

Nominal Scenario	1. The shop owner logs into the system. 2. The shop owner views the budget list. 3. The shop owner selects a budget. 4. The shop owner edits the budget information. 5. The shop owner views the total balance, expenses, and incomes. 6. The shop owner logs out of the system.
Alternative Scenarios	• The shop owner does not find any budgets. • The shop owner encounters an error while selecting a budget. • The shop owner encounters an error while editing the budget information.

Use Case 5: Manage Costs

Title	Manage Costs
Summary	This use case describes the actions performed by a shop owner to input various costs and calculate the total costs.
Actors	Shop Owner
Preconditions	• The shop owner has logged into the application. • The shop owner has accessed the costs management section. • The shop owner has the appropriate permissions to manage costs.
Postconditions	The shop owner has inputted various costs and calculated the total costs.

Nominal Scenario	1. The shop owner logs into the system. 2. The shop owner inputs various costs. 3. The shop owner calculates the total costs. 4. The shop owner logs out of the system.
Alternative Scenarios	• The shop owner encounters an error while inputting costs. • The shop owner encounters an error while calculating the total costs.

Use Case 6: Manage Notifications

Title	Manage Notifications
Summary	This use case describes the actions performed by a shop owner to view notifications and edit the notifications sound.
Actors	Shop Owner
Preconditions	• The shop owner has logged into the application. • The shop owner has accessed the notifications management section.
Postconditions	The shop owner has viewed notifications and edited the notifications sound.
Nominal Scenario	1. The shop owner logs into the system. 2. The shop owner views notifications. 3. The shop owner edits the notifications sound. 4. The shop owner logs out of the system.

Alternative Scenarios	• The shop owner does not find any notifications. • The shop owner encounters an error while editing the notifications sound.

Use Case 7: View Statistics

Title	View Statistics
Summary	This use case describes the actions performed by a shop owner to view statistics and filter statistics.
Actors	Shop Owner
Preconditions	• The shop owner has logged into the application. • The shop owner has accessed the statistics section.
Postconditions	The shop owner has viewed statistics and filtered statistics.
Nominal Scenario	1. The shop owner logs into the system. 2. The shop owner views statistics. 3. The shop owner filters statistics. 4. The shop owner logs out of the system.
Alternative Scenarios	• The shop owner does not find any statistics. • The shop owner encounters an error while filtering statistics.

3.5 Conclusion

This chapter has provided an analysis and specification of the requirements for the project. It has identified the functional and non-functional requirements of the system, as well as the actors involved in

the system. The global use case diagram has been presented, along with the refinement of use cases to provide more detailed information about the system's features and constraints.

Chapter 4 Project Architecture & Choice Of Technologies

Contents

4.1 Introduction

This chapter provides an overview of the project architecture and the choice of technologies used in the development process. It aims to explain the rationale behind the selection of specific architectural patterns, frameworks, and tools. By understanding the project architecture and the technologies employed, readers will gain insights into the design principles and the technical foundations of the system. Additionally, this chapter highlights the importance of making informed decisions when it comes to selecting technologies that align with the project requirements and objectives.

4.2 Project Architecture - Modular Monolith

4.2.1 Backend & Services

The backend of Converty, which follows a Modular Monolith architecture, is implemented using Node.js and Express.js. It encompasses the authentication system, simple CRUD operations, and the business logic. On the other hand, the microservices, developed in Go (Golang), handle specific tasks such as sending notifications, cron jobs, serverless functions, and other background tasks. This architectural approach ensures a flexible and scalable system capable of efficiently handling diverse workloads.

Node.js - Express.js

Node.js is a popular runtime environment that allows developers to run JavaScript code outside the browser. It is built on the V8 JavaScript engine and provides a non-blocking, event-driven architecture that is ideal for building scalable network applications. Express.js, a minimal and flexible Node.js web application framework, is used to build the backend of Converty. It simplifies the process of building APIs and web applications by providing a robust set of features and middleware. Express.js is known for its simplicity, performance, and extensibility, making it an excellent choice for building the backend of the application.

Figure 4.1: Node.js and Express.js

The benefits of using Node.js and Express.js for the backend of Converty include:

- **Performance:** Node.js is known for its high performance due to its non-blocking, event-driven architecture. This allows the application to handle a large number of concurrent connections efficiently.
- **Scalability:** Node.js is designed to be scalable, making it suitable for applications that need to handle a large number of users and requests.
- **Simplicity:** Express.js provides a simple and intuitive API that makes it easy to build APIs and web applications.
- **Extensibility:** Express.js is highly extensible, allowing developers to add middleware and plugins to enhance the functionality of the application.
- **Community Support:** Node.js and Express.js have a large and active community that provides support, documentation, and a wide range of libraries and plugins.
- **Flexibility:** Node.js and Express.js are flexible and can be used to build a wide range of applications, from simple APIs to complex web applications.
- **Compatibility:** Node.js and Express.js are compatible with a wide range of databases, libraries, and tools, making it easy to integrate them with other technologies.

Go (Golang)

Go (Golang) is a statically typed, compiled programming language designed for building simple, reliable, and efficient software. It is known for its performance, simplicity, and ease of use, making it an excellent choice for building microservices. Go is widely used in the industry for building high-performance applications, cloud services, and distributed systems. The microservices in Converty are developed

in Go to handle specific tasks such as sending notifications, cron jobs, serverless functions, and other background tasks.

Figure 4.2: Go (Golang)

The benefits of using Go for building microservices in Converty include:

- **Performance:** Go is known for its high performance and efficiency, making it suitable for building high-performance applications.
- **Concurrency:** Go provides built-in support for concurrency, making it easy to write concurrent programs that can efficiently utilize multiple cores.
- **Simplicity:** Go has a simple and clean syntax that is easy to learn and use, making it ideal for building microservices.
- **Reliability:** Go is designed to be reliable and robust, with built-in error handling and garbage collection.
- **Scalability:** Go is designed to be scalable, making it suitable for building microservices that need to handle a large number of requests.
- **Community Support:** Go has a large and active community that provides support, documentation, and a wide range of libraries and tools.
- **Cross-Platform:** Go is cross-platform and can be compiled to run on different operating systems, making it easy to deploy microservices on different platforms.

RabbitMQ

RabbitMQ is a message broker that is used to implement asynchronous communication between microservices in Converty. It provides a reliable and scalable messaging system that allows microservices to communicate with each other in a decoupled and asynchronous manner. RabbitMQ supports multiple messaging protocols, including AMQP, MQTT, and STOMP, making it suitable for building distributed systems.

Figure 4.3: RabbitMQ

The benefits of using RabbitMQ for asynchronous communication between microservices in Converty include:

- **Reliability:** RabbitMQ provides reliable message delivery and ensures that messages are not lost even in the event of failures.
- **Scalability:** RabbitMQ is designed to be scalable and can handle a large number of messages and connections.
- **Decoupling:** RabbitMQ allows microservices to communicate with each other in a decoupled manner, reducing dependencies between services.
- **Asynchronous Communication:** RabbitMQ supports asynchronous communication, allowing microservices to communicate with each other without blocking.
- **Multiple Protocols:** RabbitMQ supports multiple messaging protocols, making it suitable for building distributed systems that use different messaging protocols.
- **Monitoring:** RabbitMQ provides monitoring and management tools that allow developers to monitor the performance and health of the messaging system.
- **Community Support:** RabbitMQ has a large and active community that provides support, documentation, and a wide range of plugins and tools.

4.2.2 Frontend - Web Dashboard

React + Vite + TypeScript

The frontend of Converty, which is a web dashboard, is implemented using React, Vite, and TypeScript. React is a popular JavaScript library for building user interfaces, while Vite is a modern build tool that provides fast and efficient development experience. TypeScript is a statically typed superset of JavaScript that adds type checking and other features to the language. The combination of React, Vite, and TypeScript provides a powerful and flexible frontend development environment that is ideal for building modern web applications.

The four templates in Converty are also coded using React, Vite, and TypeScript. These templates provide the result online store that the shop owner configured. By using React, Vite, and TypeScript, the templates are able to leverage the benefits of these technologies, including component-based development, fast development and hot module replacement, and type safety. This ensures a consistent and efficient development process for creating and maintaining the shop owners' online shop.

Figure 4.4: React + Vite + TypeScript

The benefits of using React, Vite, and TypeScript for the frontend of Converty include:

- **Component-Based Development:** React allows developers to build reusable and composable components that can be easily shared and reused across the application.
- **Fast Development:** Vite provides fast development and hot module replacement, allowing developers to see changes in real-time without reloading the page.
- **Type Safety:** TypeScript adds type checking and other features to JavaScript, making it easier to catch errors and write more reliable code.
- **Performance:** React and Vite are known for their performance and efficiency, making them suitable for building fast and responsive web applications.
- **Community Support:** React, Vite, and TypeScript have large and active communities that provide support, documentation, and a wide range of libraries and tools.
- **Flexibility:** React, Vite, and TypeScript are flexible and can be used to build a wide range of web applications, from simple websites to complex web applications.
- **Scalability:** React, Vite, and TypeScript are designed to be scalable, making them suitable for building web applications that need to handle a large number of users and requests.

Redux

Redux is a predictable state container for JavaScript applications that helps manage the state of the application in a consistent and predictable way. It provides a single source of truth for the state of the application and allows changes to the state to be made in a predictable and controlled manner. Redux is

used in the frontend of Converty to manage the state of the application, including user authentication, data fetching, and UI state.

Figure 4.5: Redux

The benefits of using Redux for state management in the frontend of Converty include:

- **Predictability:** Redux provides a predictable state container that helps manage the state of the application in a consistent and predictable way.
- **Single Source of Truth:** Redux provides a single source of truth for the state of the application, making it easier to manage and update the state.
- **Debugging:** Redux provides tools for debugging and monitoring the state of the application, making it easier to identify and fix issues.
- **Scalability:** Redux is designed to be scalable, making it suitable for building large and complex web applications.
- **Community Support:** Redux has a large and active community that provides support, documentation, and a wide range of plugins and tools.
- **Middleware:** Redux provides middleware that allows developers to add custom logic and side effects to the state management process.
- **Performance:** Redux is known for its performance and efficiency, making it suitable for building fast and responsive web applications.

4.2.3 Frontend - Mobile App

React Native

The mobile app of Converty is implemented using React Native, a popular framework for building cross-platform mobile applications. React Native allows developers to build mobile apps using JavaScript and React, providing a fast and efficient development experience. By using React Native, Converty is able to build a mobile app that runs on both iOS and Android devices, reducing development time and cost.

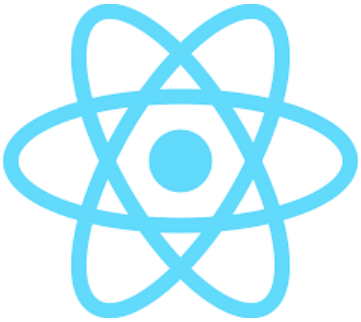

Figure 4.6: React Native

The benefits of using React Native for building the mobile app of Converty include:

- **Cross-Platform:** React Native allows developers to build mobile apps that run on both iOS and Android devices, reducing development time and cost.
- **Efficiency:** React Native provides a fast and efficient development experience, allowing developers to build mobile apps quickly and easily.
- **Performance:** React Native is known for its performance and efficiency, making it suitable for building fast and responsive mobile apps.
- **Community Support:** React Native has a large and active community that provides support, documentation, and a wide range of libraries and tools.
- **Hot Reloading:** React Native provides hot reloading, allowing developers to see changes in real-time without reloading the app.
- **Native Components:** React Native provides access to native components and APIs, allowing developers to build mobile apps with a native look and feel.
- **Code Reusability:** React Native allows developers to reuse code across different platforms, reducing duplication and improving maintainability.

Expo

Expo is a set of tools and services that help developers build React Native applications more easily. It provides a fast and efficient development environment, allowing developers to build, test, and deploy mobile apps quickly. Expo provides a wide range of features and APIs, including push notifications, camera access, and location services, making it easy to build feature-rich mobile apps.

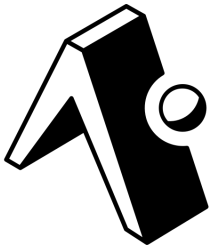

Figure 4.7: Expo

The benefits of using Expo for building the mobile app of Converty include:

- **Fast Development:** Expo provides a fast and efficient development environment, allowing developers to build mobile apps quickly and easily.
- **Feature-Rich:** Expo provides a wide range of features and APIs, including push notifications, camera access, and location services, making it easy to build feature-rich mobile apps.
- **Community Support:** Expo has a large and active community that provides support, documentation, and a wide range of libraries and tools.
- **Cross-Platform:** Expo allows developers to build mobile apps that run on both iOS and Android devices, reducing development time and cost.
- **Hot Reloading:** Expo provides hot reloading, allowing developers to see changes in real-time without reloading the app.
- **Ease of Use:** Expo is easy to use and provides a simple and intuitive API that makes it easy to build mobile apps.
- **Performance:** Expo is known for its performance and efficiency, making it suitable for building fast and responsive mobile apps.

Zustand

Zustand is a simple and fast state management library for React applications. It provides a way to manage the state of the application in a simple and efficient way, reducing the complexity of managing state in React components. Zustand is used in the mobile app of Converty to manage the state of the application, including user authentication, data fetching, and UI state.

Figure 4.8: Zustand

The benefits of using Zustand for state management in the mobile app of Converty include:

- **Simplicity:** Zustand provides a simple and intuitive API that makes it easy to manage the state of the application.
- **Performance:** Zustand is known for its performance and efficiency, making it suitable for building fast and responsive mobile apps.
- **Predictability:** Zustand provides a predictable state container that helps manage the state of the application in a consistent and predictable way.
- **Community Support:** Zustand has a large and active community that provides support, documentation, and a wide range of plugins and tools.
- **Flexibility:** Zustand is flexible and can be used to manage the state of the application in a wide range of scenarios.
- **Scalability:** Zustand is designed to be scalable, making it suitable for building large and complex mobile apps.
- **Performance:** Zustand is known for its performance and efficiency, making it suitable for building fast and responsive mobile apps.

4.2.4 Database - MongoDB

MongoDB is a popular NoSQL database that is used in Converty to store data such as user information, product information, and configuration data. MongoDB is a document-oriented database that provides a flexible and scalable data storage solution. It is known for its performance, scalability, and ease of use, making it an excellent choice for building modern web applications.

Figure 4.9: MongoDB

The benefits of using MongoDB for storing data in Converty include:

- **Flexibility:** MongoDB is a document-oriented database that provides a flexible data storage solution, allowing developers to store data in a schema-less format.
- **Scalability:** MongoDB is designed to be scalable, making it suitable for building web applications that need to handle a large amount of data.
- **Performance:** MongoDB is known for its performance and efficiency, making it suitable for building fast and responsive web applications.
- **Ease of Use:** MongoDB is easy to use and provides a simple and intuitive API that makes it easy to interact with the database.
- **Community Support:** MongoDB has a large and active community that provides support, documentation, and a wide range of tools and libraries.
- **Scalability:** MongoDB is designed to be scalable, making it suitable for building web applications that need to handle a large amount of data.
- **Compatibility:** MongoDB is compatible with a wide range of programming languages, frameworks, and tools, making it easy to integrate with other technologies.

Data Modeling - Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB that provides a simple and efficient way to model data in MongoDB. It provides a schema-based solution for modeling data, allowing developers to define the structure of the data and enforce validation rules. Mongoose is used in Converty to model data in MongoDB, including user information, product information, and configuration data.

The benefits of using Mongoose for data modeling in Converty include:

- **Schema-Based:** Mongoose provides a schema-based solution for modeling data in MongoDB, allowing developers to define the structure of the data and enforce validation rules.

- **Validation:** Mongoose provides built-in validation features that allow developers to enforce validation rules on the data.
- **Relationships:** Mongoose provides support for defining relationships between different types of data, making it easy to model complex data structures.
- **Middleware:** Mongoose provides middleware that allows developers to add custom logic and side effects to the data modeling process.
- **Community Support:** Mongoose has a large and active community that provides support, documentation, and a wide range of plugins and tools.
- **Performance:** Mongoose is known for its performance and efficiency, making it suitable for modeling data in MongoDB.
- **Scalability:** Mongoose is designed to be scalable, making it suitable for modeling large and complex data structures.

Caching - Redis

Redis is an in-memory data store that is used in Converty for caching data such as user sessions, product information, and configuration data. Redis provides a fast and efficient way to store and retrieve data in memory, making it ideal for caching frequently accessed data. It is known for its performance, scalability, and reliability, making it an excellent choice for building modern web applications.

Figure 4.10: Redis

The benefits of using Redis for caching data in Converty include:

- **Performance:** Redis is an in-memory data store that provides fast and efficient data storage and retrieval.
- **Scalability:** Redis is designed to be scalable, making it suitable for caching large amounts of data.
- **Reliability:** Redis is known for its reliability and fault tolerance, making it suitable for building high-availability applications.

- **Ease of Use:** Redis is easy to use and provides a simple and intuitive API that makes it easy to interact with the data store.
- **Community Support:** Redis has a large and active community that provides support, documentation, and a wide range of tools and libraries.
- **Performance:** Redis is known for its performance and efficiency, making it suitable for caching data in modern web applications.
- **Scalability:** Redis is designed to be scalable, making it suitable for caching large amounts of data in memory.

4.3 Conclusion

In conclusion, the project architecture of Converty is designed to be modular, scalable, and efficient. By using a combination of Node.js, Express.js, Go, React, Vite, TypeScript, Redux, React Native, Expo, MongoDB, Mongoose, RabbitMQ, and Redis, Converty is able to build a flexible and feature-rich system that meets the requirements of modern web applications. The choice of technologies is based on their performance, scalability, reliability, ease of use, and community support. By selecting the right technologies and architectural patterns, Converty is able to build a robust and reliable system that can handle diverse workloads and provide a seamless user experience.

Chapter 5 Realization - Web Features

Contents

5.1 Introduction

In this chapter, we will be discussing the two web-related sprints. We will provide a detailed explanation of the conception part, including a class diagram and a sequence diagram. Additionally, we will present the sprint backlog and include relevant screenshots of these sprints. Furthermore, we will showcase some code snippets to illustrate the implementation process.

5.2 Sprint 1 - Recreating the Simple Template

5.2.1 Sprint Backlog

In this section, we present the Sprint Backlog, detailing the tasks planned for developing shop visiting features. It is a concise list of user stories and specific actions that will guide us through the sprint.

Sprint	User Story	Task	ID
1	1.1	Implement the shop landing page	1.0.1
1	1.2	Add functionality to load additional products	1.0.2
1	1.3	Include social media links in the shop interface	1.0.3
1	1.4	Implement a support team contact section	1.0.4
1	1.5	Develop a search feature for specific products	1.0.5
1	1.6	Create a checkout form for shop visitors	1.0.6
1	1.7	Enable the ability to add items to the cart	1.0.7
1	1.8	Implement the purchase functionality	1.0.8
1	1.9	Add the option for shop visitors to accept or decline offers	1.0.9

Table 5.1: Sprint 1 - Recreating the Simple Template

5.2.2 Design & Implementation

In this section, we will explore the design and implementation of the shop visiting process. We will begin by examining the backend design and implementation, followed by an analysis of the frontend design.

Database Schema

The database schema for the shop visiting process is shown in Figure 5.1. It consists of the following tables:

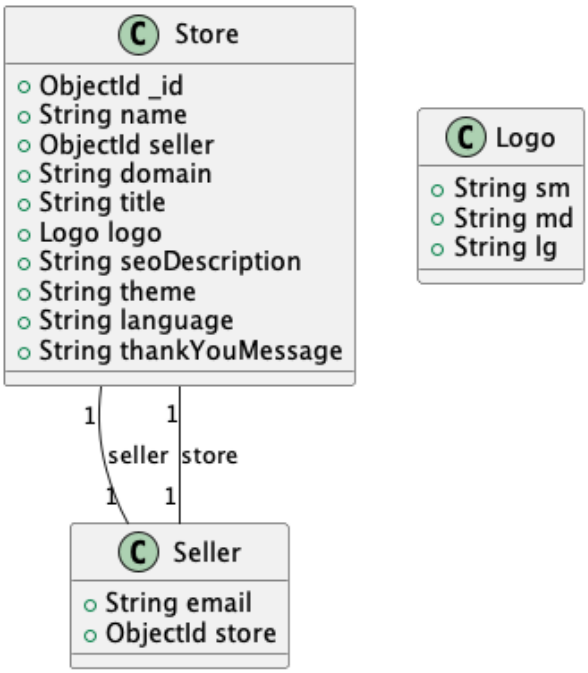

Figure 5.1: Database Schema for Sprint 1

The class diagram illustrates the relationships between three main classes: `Store`, `Seller`, and `Logo`.

- **Store Class:** Represents a shop with attributes such as an ID (`_id`), name, associated seller, domain, title, logo, SEO description, theme, language, and a thank-you message. The `Logo` is a composite object with different sizes (small, medium, large).
- **Seller Class:** Represents the seller associated with a store, containing attributes like email and a reference to the `Store`.
- **Logo Class:** Contains attributes for the different sizes of logos (`sm`, `md`, `lg`).
- **Relationships:**
  - Each `Store` is associated with a `Seller` through a one-to-one relationship.
  - Conversely, each `Seller` is linked back to a `Store`, maintaining the one-to-one relationship.

Sequence Diagram

The sequence diagram in Figure 5.2 illustrates the interactions between the different components of the shop visiting process. The sequence of events is as follows:

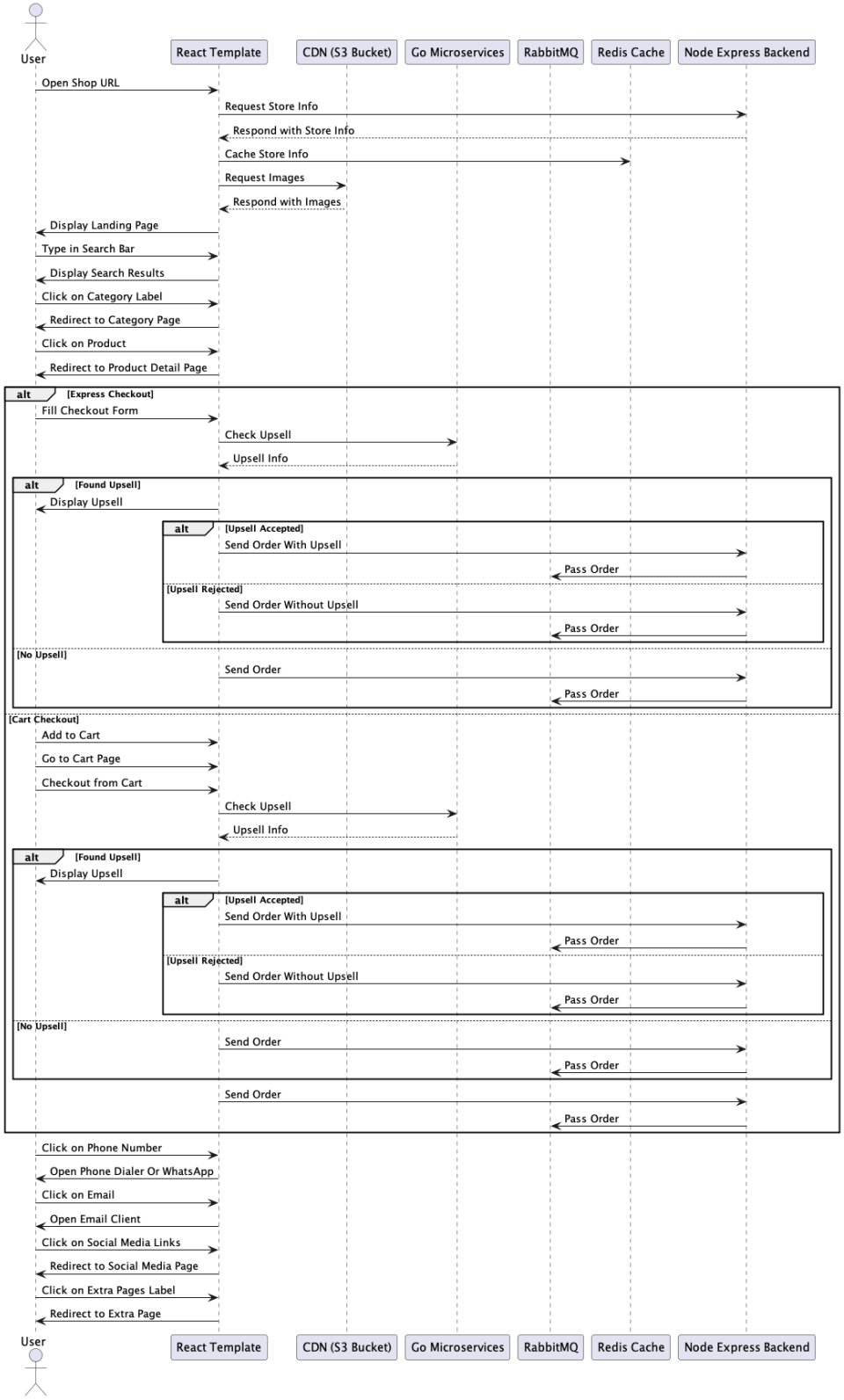

Figure 5.2: Sequence Diagram for Sprint 1

User Journey

The user journey for the shop visiting process is as follows:

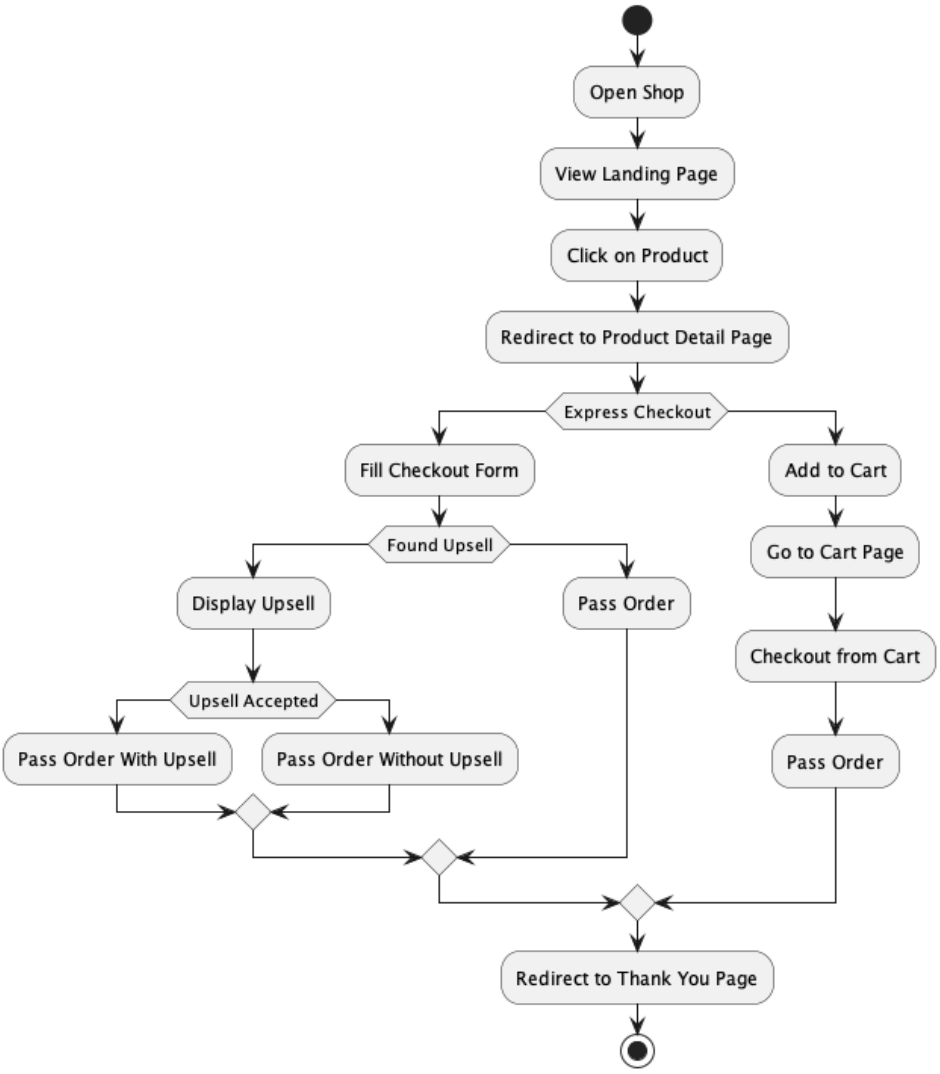

Figure 5.3: User Journey for Sprint 1

This activity diagram outlines the steps a shop visitor takes to buy a product:

- The user opens the shop's website and views the landing page.
- The user clicks on a product, which redirects them to the product detail page.
- If the shop is configured for express checkout, the user fills out the checkout form. If an upsell is found, it is displayed to the user. The user can either accept or reject the upsell. Based on their choice, the order is passed with or without the upsell.
- If the shop is not configured for express checkout, the user adds the product to the cart, goes to the cart page, and proceeds to checkout from the cart. The order is then passed.
- Finally, the user is redirected to the thank you page, completing the purchase process.

Screenshots - UI Design

- **Shop Landing Page:** The shop landing page is the first page a user sees when visiting a shop. It contains a list of products, a search bar, and social media links. Figures 5.4 and 5.5 shows the shop landing page.

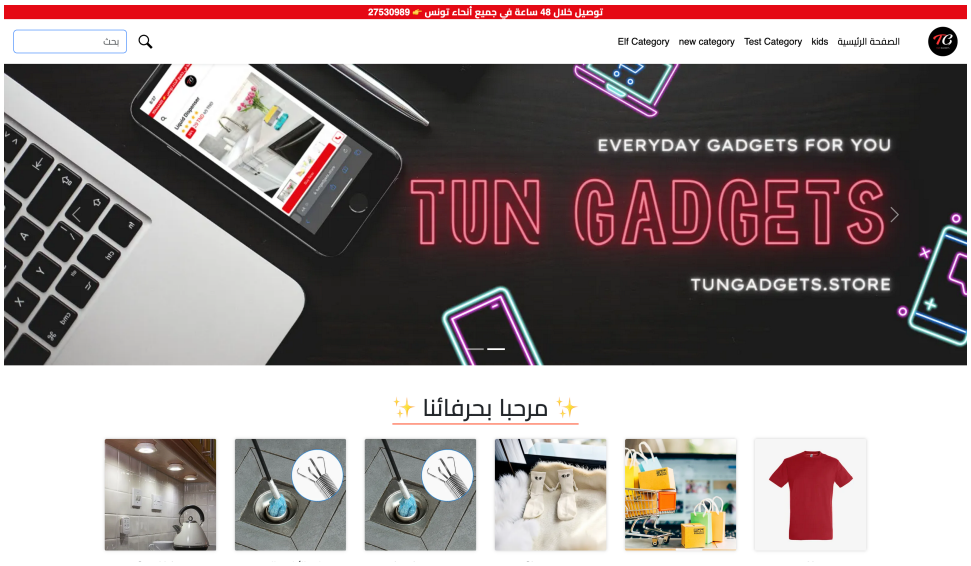

Figure 5.4: Shop Landing Page 1

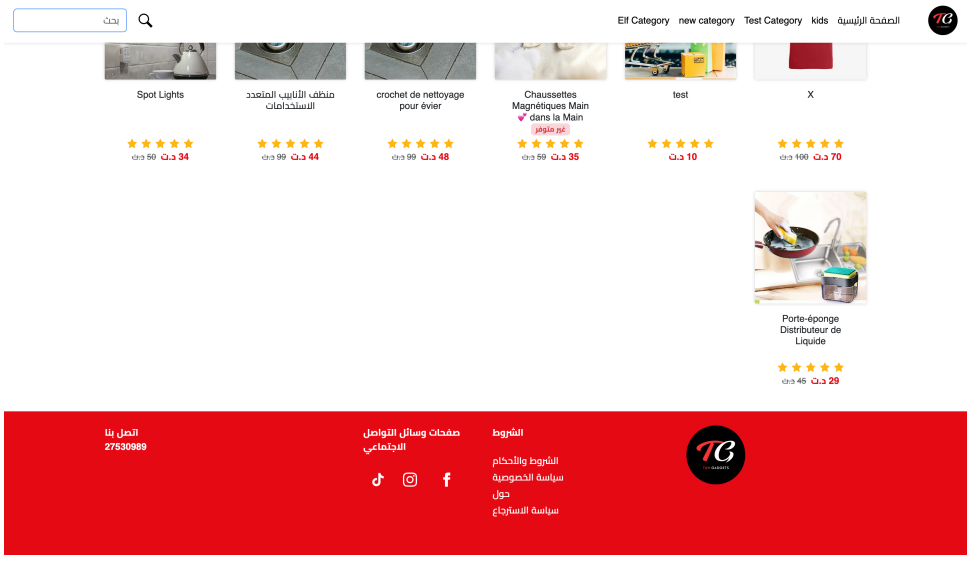

Figure 5.5: Shop Landing Page 2

- **Category Page:** The category page displays a list of products in a specific category. Figure 5.6 shows the category page.

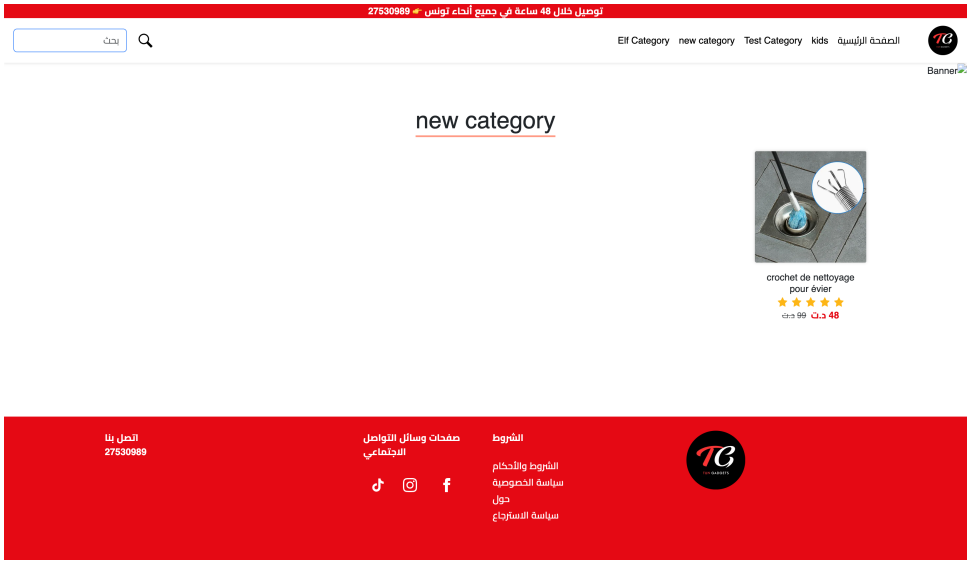

Figure 5.6: Category Page

- **Search Page:** The search page displays a list of products based on the user's search query. Figure 5.7 shows the search page.

Figure 5.7: Search Page

- **Extra Page (Ex: Refund & Return Policy):** The extra page displays additional information about the shop, such as the refund and return policy. Figure 5.8 shows the extra page.

Figure 5.8: Extra Page

- **Product Detail Page:** The product detail page displays information about a specific product alonng with a checkout form and other clients opinions about it. Figures 5.9, 5.10 and 5.11 shows the product detail page.

Figure 5.9: Product Detail Page 1

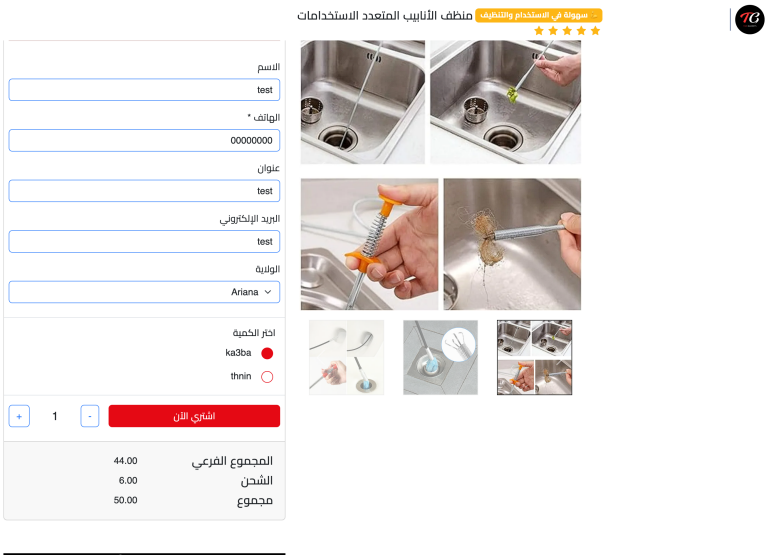

Figure 5.10: Product Detail Page 2

Figure 5.11: Product Detail Page 3

- **Thank You Page:** The thank you page is displayed after a successful purchase. Figure 5.12 shows the thank you page and figure 5.13 shows the upsell popup.

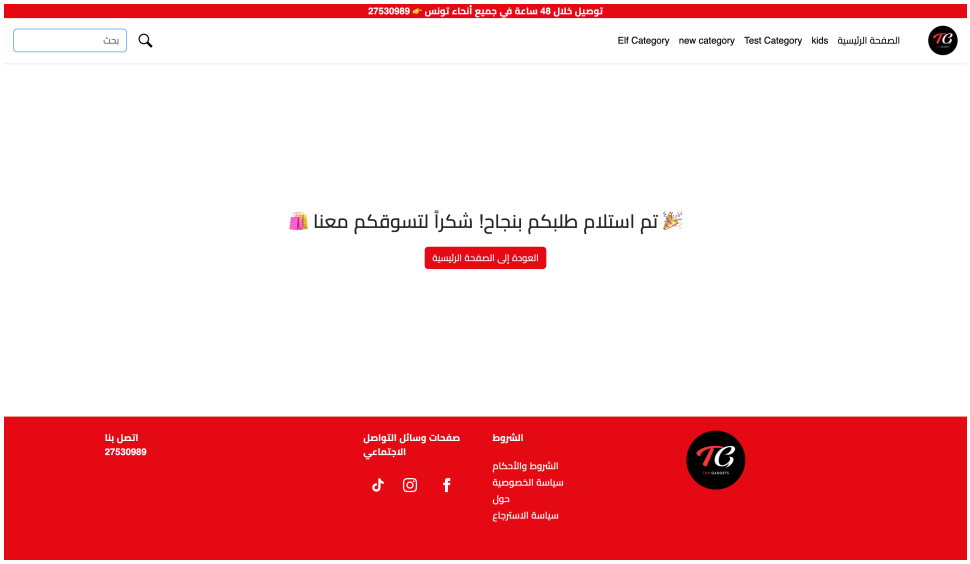

Figure 5.12: Thank You Page

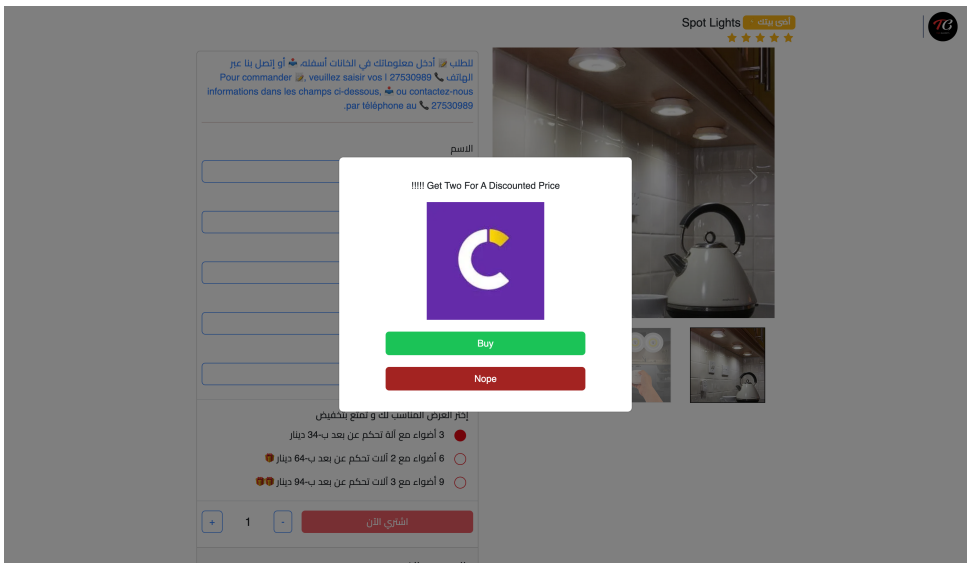

Figure 5.13: Upsell Popup

Summary

In this sprint, we successfully recreated the simple template for the shop visiting process. We designed and implemented the frontend components, sequence diagram, and user journey. We also created the various needed pages.

5.3 Sprint 2 - Implementing the Upsell/Cross-sell Feature

5.3.1 Sprint Backlog

In this section, we present the Sprint Backlog, detailing the tasks planned for developing the upsell/cross-sell feature. It is a concise list of user stories and specific actions that will guide us through the sprint.

Sprint	User Story	Task	ID
2	2.1	Backend: Implement Read API endpoints for upsells/cross-sells	2.0.1
2	2.2	Frontend: Design upsell/cross-sell management UI for Read operations	2.0.2
2	2.3	Backend: Implement API endpoints for searching upsells/cross-sells by name.	2.0.3
2	2.4	Implement upsell/cross-sell search functionality by name.	2.0.4
2	2.5	Backend: Implement API endpoints for editing upsells/cross-sells	2.0.5
2	2.6	Frontend: Design upsell/cross-sell management UI for Edit operations	2.0.6
2	2.7	Backend: Implement API endpoints for deleting upsells/cross-sells	2.0.7
2	2.8	Frontend: Design upsell/cross-sell management UI for Delete operations	2.0.8
2	2.9	Frontend: Design upsell/cross-sell management UI for Preview operations	2.0.9

Table 5.2: Sprint 2 - Implementing the Upsell/Cross-sell Feature

5.3.2 Design & Implementation

In this section, we will explore the design and implementation of the upsell/cross-sell feature. We will begin by examining the backend design and implementation, followed by an analysis of the frontend design.

Database Schema

The database schema for the upsell/cross-sell feature is shown in Figure 5.14. It consists of the following tables:

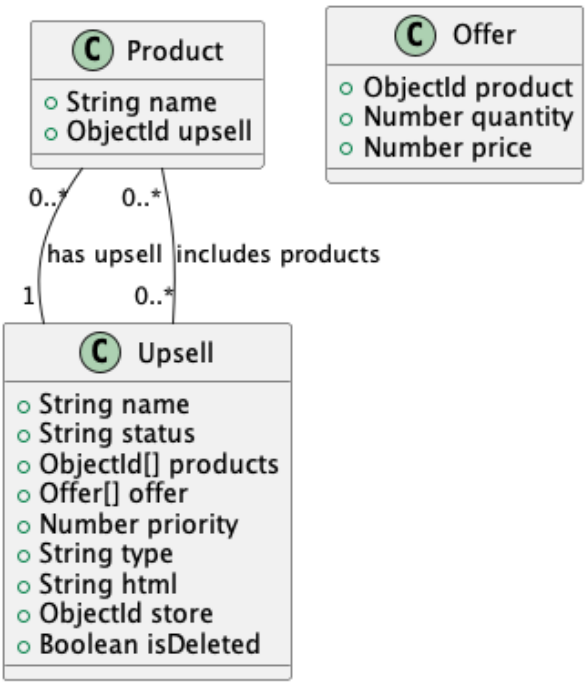

Figure 5.14: Database Schema for Sprint 2

The class diagram illustrates the relationships between two main classes: `Upsell` and `Product`.

- **Upsell Class:** Represents an upsell with attributes such as an ID (`_id`), status, offer, priority, type, HTML, store, and isDeleted. The `Offer` is a composite object with different attributes like product, quantity, and price.
- **Product Class:** Represents a product with attributes like an ID (`_id`), name, description, price, and a reference to the `Upsell`.
- **Offer Class:** Contains attributes for the product, quantity, and price.
- **Relationships:**
  - Each `Upsell` can include multiple `Products` through a one-to-many relationship.
  - Conversely, each `Product` can have an associated `Upsell`, maintaining the one-to-one relationship.

Sequence Diagram

- **Search Operation:** The sequence diagram in Figure 5.15 illustrates the interactions between the different components of the search operation. The sequence of events is as follows:

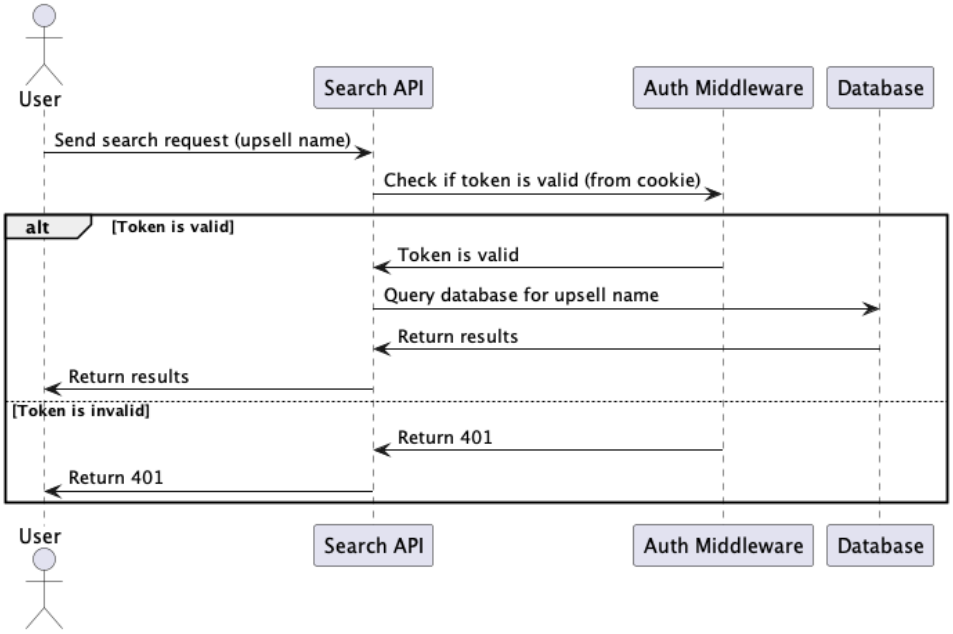

Figure 5.15: Sequence Diagram for Sprint 2 - Search Operation

- **Create Operation:** The sequence diagram in Figure 5.16 illustrates the interactions between the different components of the create operation. The sequence of events is as follows:

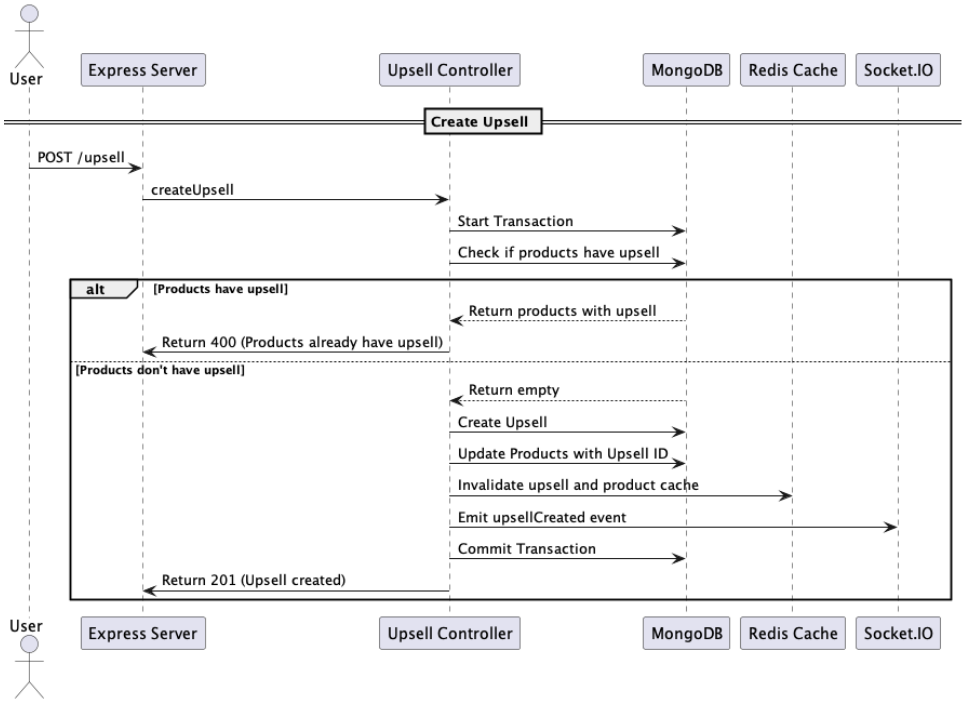

Figure 5.16: Sequence Diagram for Sprint 2 - Create Operation

- **Read Operation:** The sequence diagram in Figure 5.17 illustrates the interactions between the different components of the read all operation. The sequence of events is as follows:

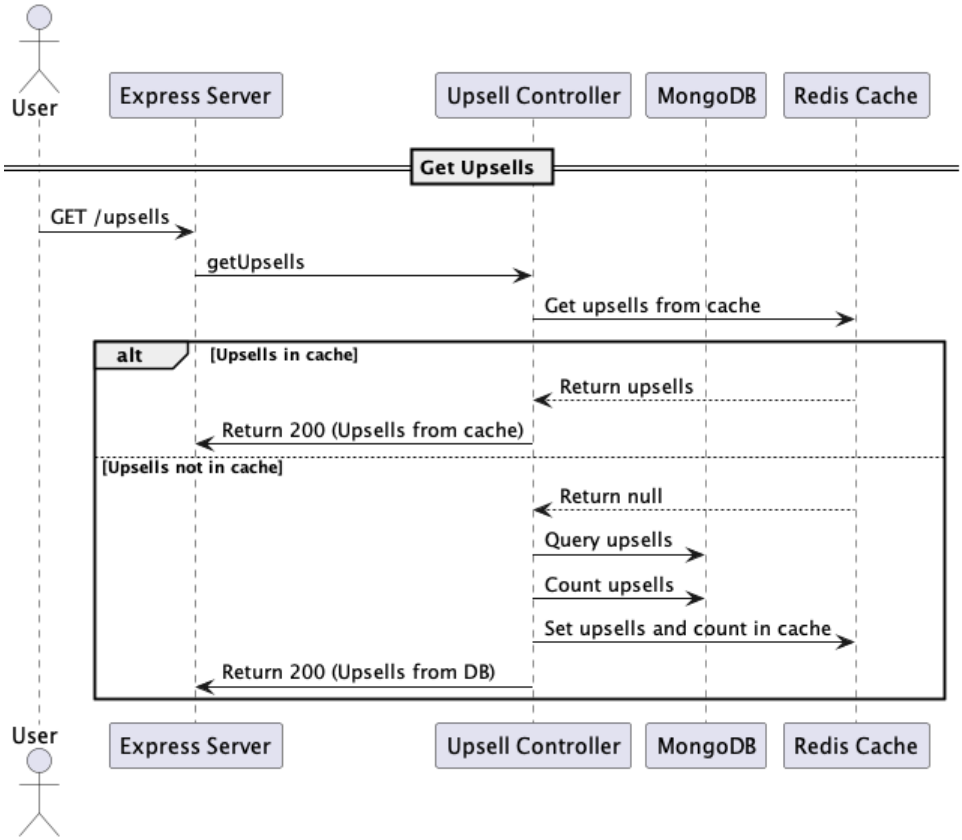

Figure 5.17: Sequence Diagram for Sprint 2 - Read Operation

- **Update Operation:** The sequence diagram in Figure 5.18 illustrates the interactions between the different components of the update operation. The sequence of events is as follows:

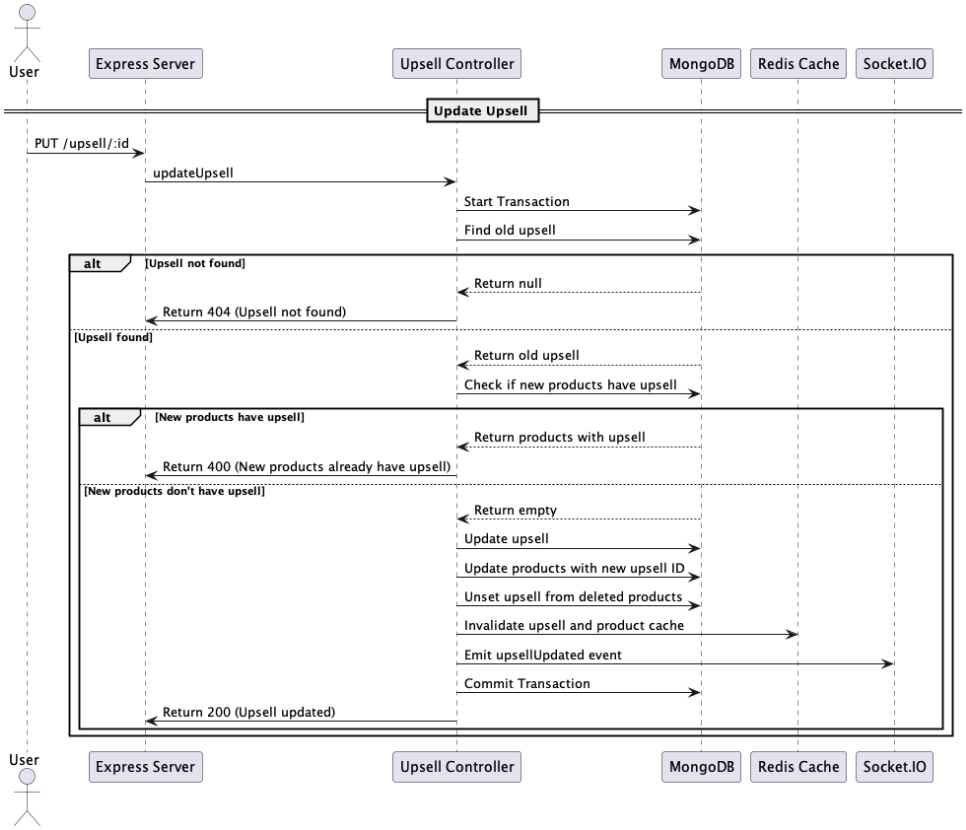

Figure 5.18: Sequence Diagram for Sprint 2 - Update Operation

- **Delete Operation:** The sequence diagram in Figure 5.19 illustrates the interactions between the different components of the delete operation. The sequence of events is as follows:

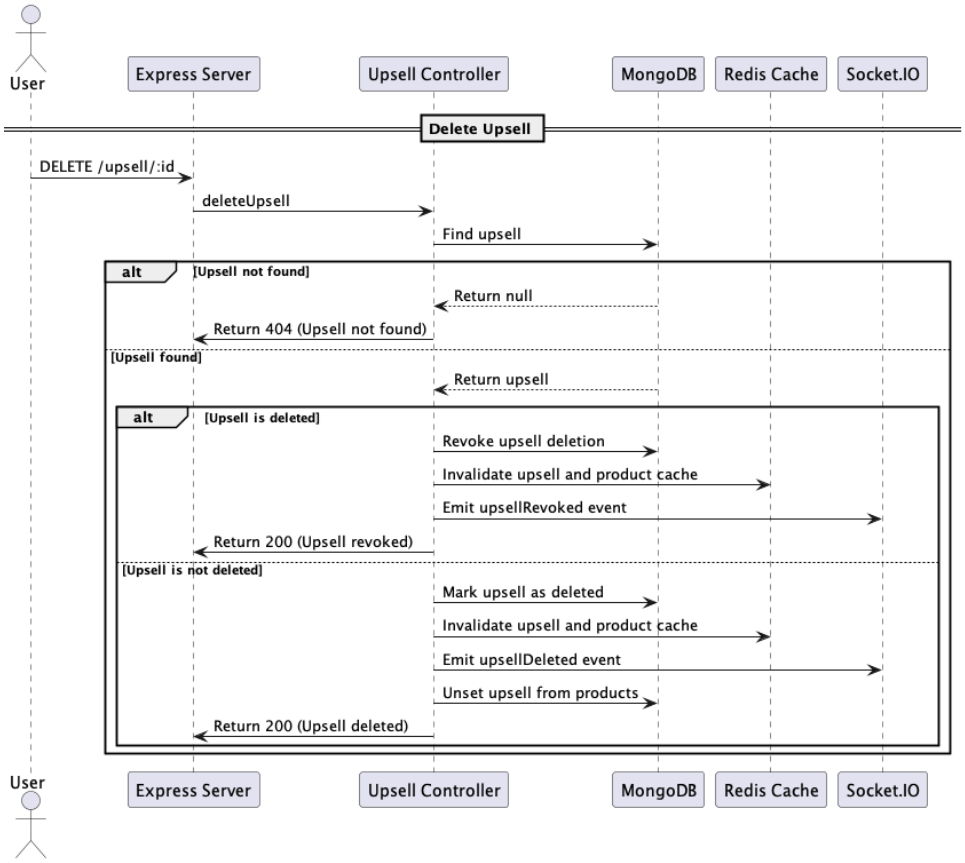

Figure 5.19: Sequence Diagram for Sprint 2 - Delete Operation

- **Read One Operation:** The sequence diagram in Figure 5.20 illustrates the interactions between the different components of the read one operation. The sequence of events is as follows:

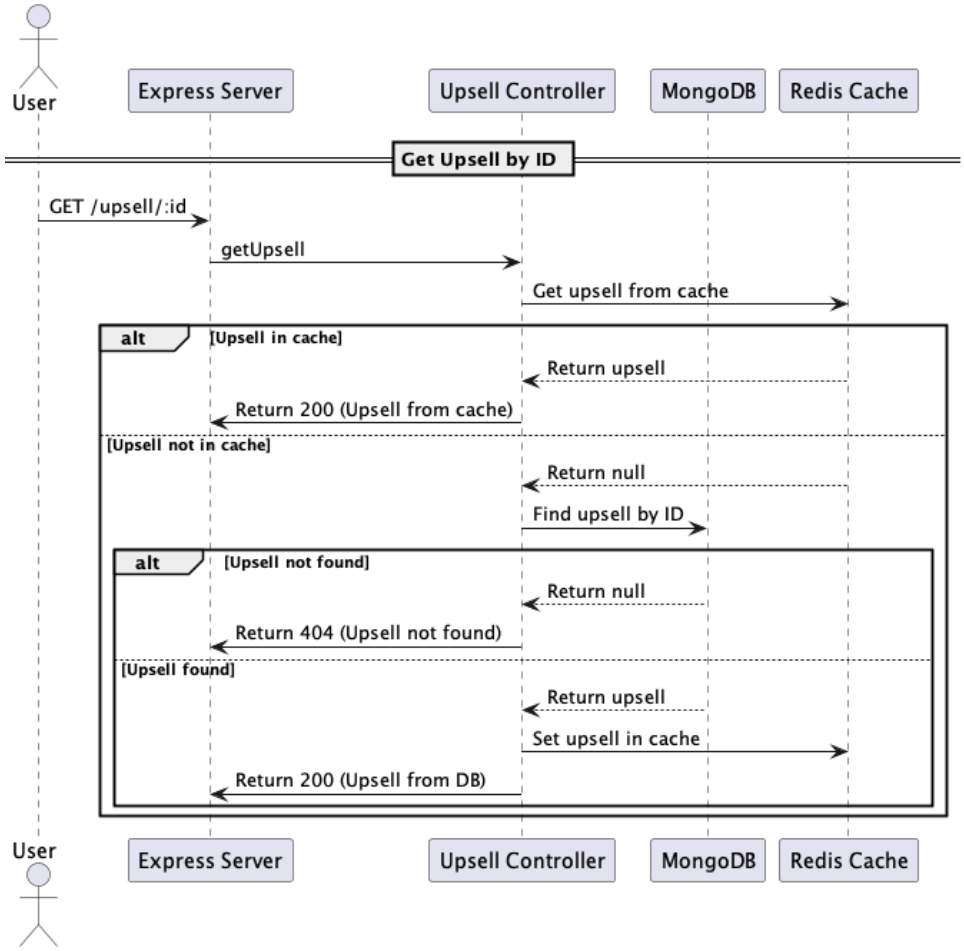

Figure 5.20: Sequence Diagram for Sprint 2 - Read One Operation

Caching Layer

The caching layer is an essential component of the upsell/cross-sell feature. It helps improve the performance of the application by reducing the number of database queries. The caching layer stores the results of database queries in memory, allowing the application to retrieve data quickly without querying the database each time. The caching layer is implemented using Redis, an open-source, in-memory data structure store.

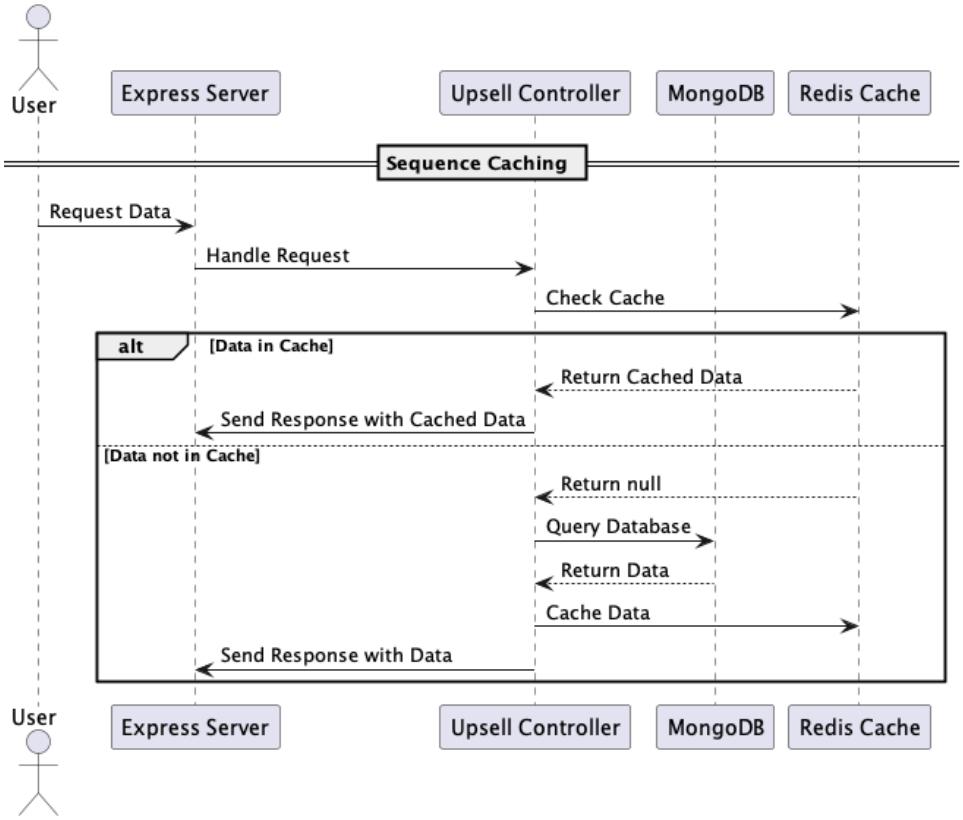

Figure 5.21: Sequence Diagram for Sprint 2 - Caching Layer

Benefits of using a caching layer include:

- **Improved Performance:** By storing data in memory, the application can retrieve it quickly without querying the database each time.
- **Reduced Database Load:** The caching layer reduces the number of database queries, which helps reduce the load on the database server.
- **Scalability:** The caching layer can be scaled horizontally by adding more cache servers to handle increased load.

And the following is a code snippet of the caching layer implementation:

Listing 5.1: Redis Caching Layer Implementation

```
let data = await redisClient.get(key);


if (data) {
    console.log('Data fetched from Redis');
    return JSON.parse(data);
}
// Data not found in Redis, query MongoDB
const collection = db.collection('mycollection');
data = await collection.findOne({ key });
if (data) {
    console.log('Data fetched from MongoDB');
    redisClient.setex(key, 3600, JSON.stringify(data));
    return data;
}
```

User Journey

The user journey for the upsell/cross-sell creation feature is as follows:

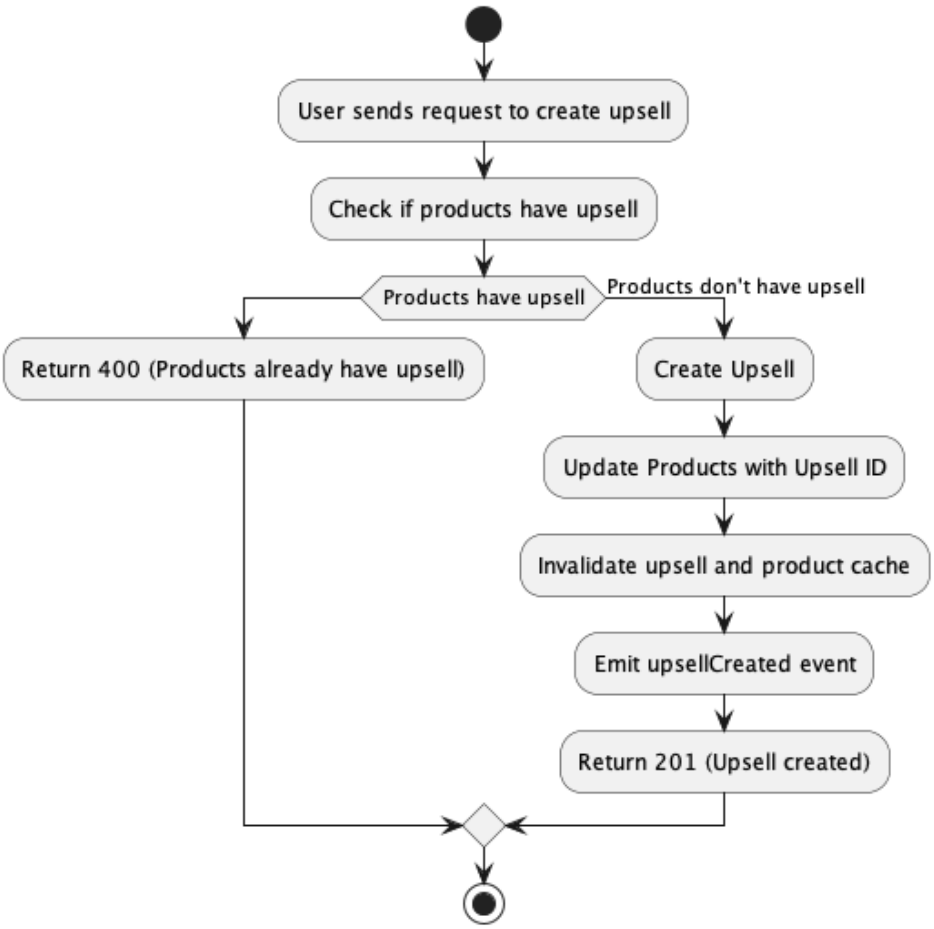

Figure 5.22: User Journey for Sprint 2

Screenshots - UI Design

- **Table View Dashboard :** The table view dashboard displays a list of upsells/cross-sells in a tabular format. Figure 5.23 shows the table view dashboard.

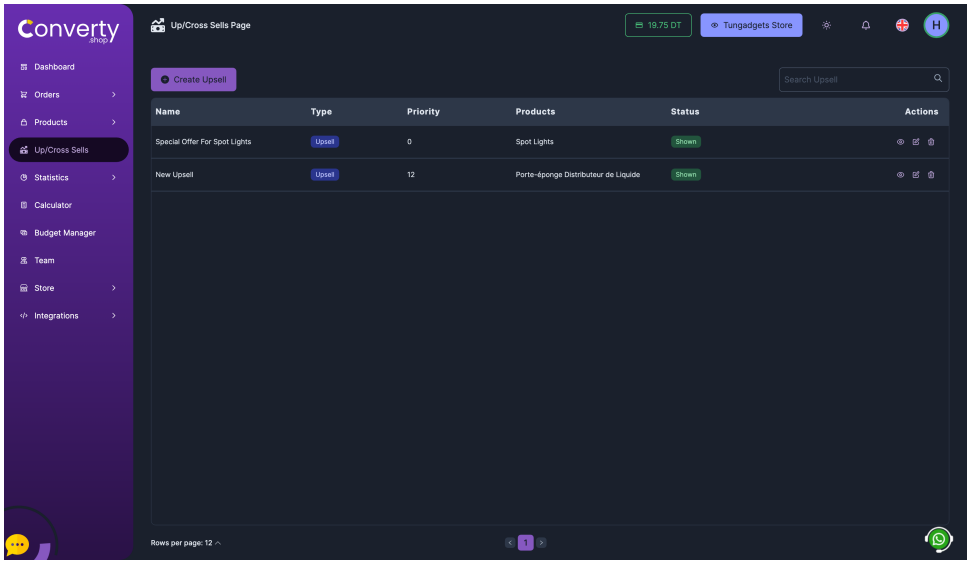

Figure 5.23: Table View Dashboard

- **Upsell/Cross-sell Preview:** The upsell/cross-sell preview displays a preview of the upsell/cross-sell. Figure 5.24 shows the upsell/cross-sell preview.

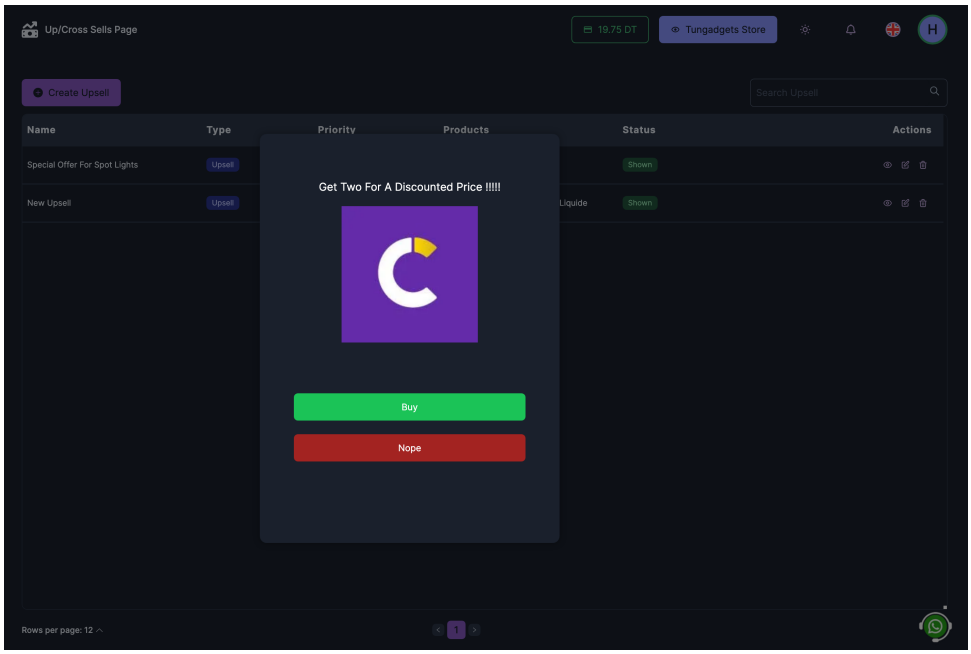

Figure 5.24: Upsell/Cross-sell Preview

- **Edit/Add Upsell/Cross-sell:** The edit/add upsell/cross-sell page allows users to create or edit an upsell/cross-sell. Figures 5.25 and 5.26 shows the edit/add upsell/cross-sell page.

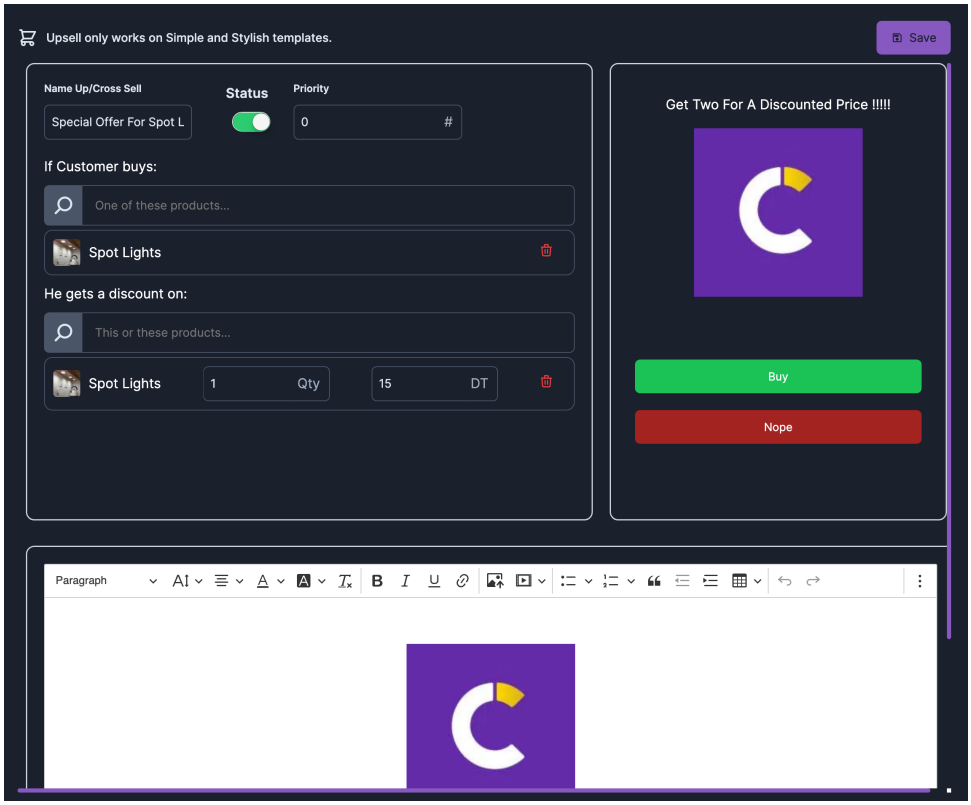

Figure 5.25: Edit/Add Upsell/Cross-sell 1

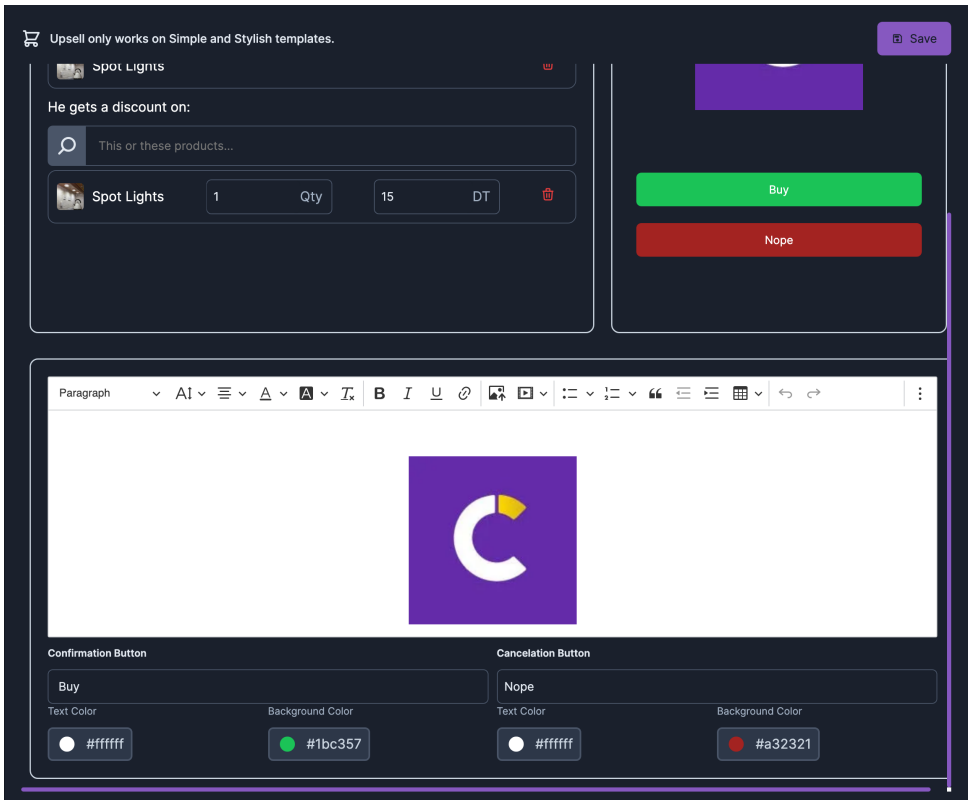

Figure 5.26: Edit/Add Upsell/Cross-sell 2

Summary

In this sprint, we successfully implemented the upsell/cross-sell feature. We designed and implemented the frontend and backend components, sequence diagrams, and user journey. We also created the various needed pages.

5.4 Conclusion

In this chapter, we discussed the two web-related sprints. We provided a detailed explanation of the conception part, including a class diagram and a sequence diagram. Additionally, we presented the sprint backlog and included relevant screenshots of these sprints. Furthermore, we showcased some code snippets to illustrate the implementation process.

Chapter 6 Realization - Mobile Features

Contents

6.1 Introduction

In this chapter, we will be discussing the four mobile-related sprints. We will provide a detailed explanation of the conception part, including a class diagram and a sequence diagram. Additionally, we will present the sprint backlog and include relevant screenshots of these sprints. Furthermore, we will showcase some code snippets to illustrate the implementation process.

6.2 Sprint 3: Mobile - Order Management

6.2.1 Sprint Backlog

In this section, we present the Sprint Backlog, detailing the tasks planned for developing order management features. It is a concise list of user stories and specific actions that will guide us through the sprint.

Sprint	User Story	Task	ID
3	3.1	Frontend: Create Order Management Screen To View All Orders	3.0.1
3	3.2	Frontend: Create Order Management Screen To View Order Details	3.0.2
3	3.3	Frontend: Create Order Management Screen To Update Or Create Orders	3.0.3
3	3.4	Frontend: Add Filter Chips To Filter Orders	3.0.4
3	3.5	Frontend: Add Search Bar To Search Orders	3.0.5

Table 6.1: Sprint 3 - Mobile Order Management

6.2.2 Design & Implementation

In this section, we will explore the design and implementation of the mobile ordeer management feature. We will begin by examining the backend design and implementation, followed by an analysis of the frontend design.

Database Schema

The database schema for the order management feature is shown in Figure 6.1. It consists of the following tables:

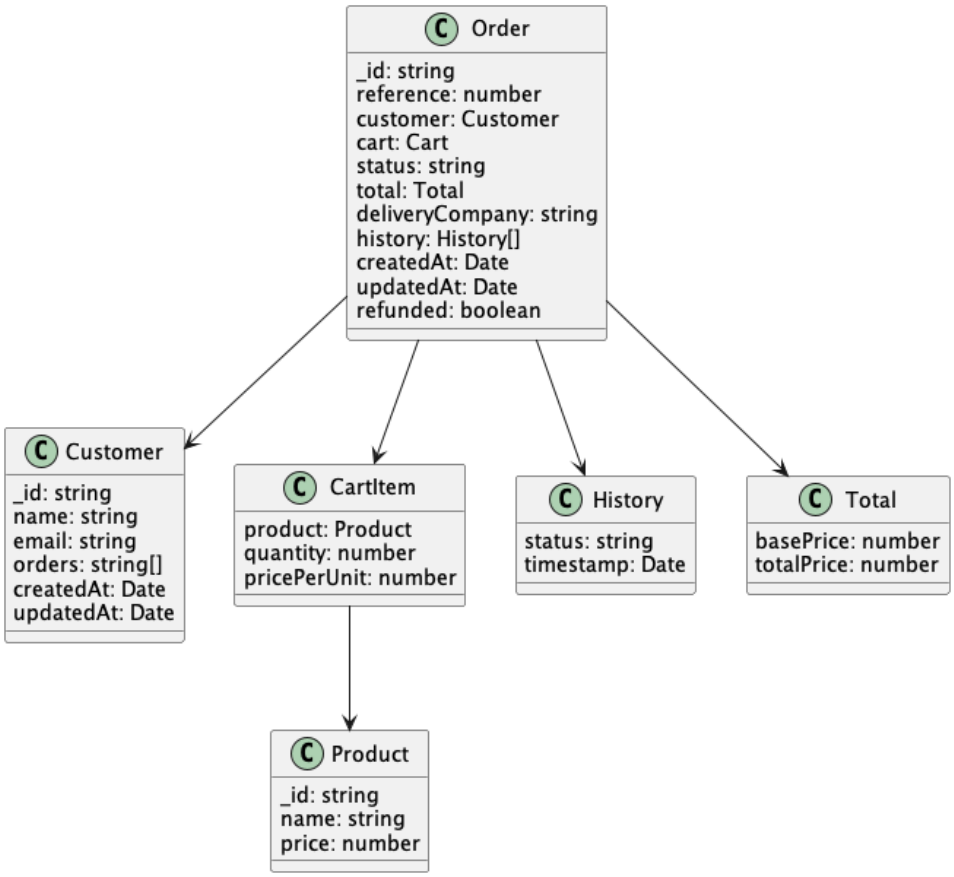

Figure 6.1: Database Schema for Sprint 3

The class diagram illustrates the relationships between several key classes: `Order`, `Customer`, `CartItem`, `Product`, `History`, and `Total`.

- **Order Class:** Represents an order with attributes such as an ID (`_id`), reference number, customer, cart, status, total, delivery company, history, creation date (`createdAt`), update date (`updatedAt`), and refunded status (`refunded`). The `Customer` is an object representing the customer associated with the order, and the `Cart` is a composite object containing `CartItem` instances.
- **Customer Class:** Represents a customer with attributes like an ID (`_id`), name, email, list of order IDs (`orders`), creation date (`createdAt`), and update date (`updatedAt`).
- **CartItem Class:** Represents an item in the cart with attributes for the product, quantity, and price per unit. The `Product` is an object representing the product associated with the cart item.
- **Product Class:** Represents a product with attributes such as an ID (`_id`), name, and price.
- **History Class:** Represents the history of an order with attributes for status and timestamp (`timestamp`).

- **Total Class:** Represents the pricing details with attributes for the base price (`basePrice`) and total price (`totalPrice`).
- **Relationships:**
  - Each `Order` is associated with one `Customer`, represented by a one-to-one relationship.
  - Each `Order` can include multiple `CartItem` instances, representing a one-to-many relationship between `Order` and `CartItem`.
  - Each `Order` can have multiple `History` entries, representing a one-to-many relationship.
  - Each `Order` is associated with one `Total`, representing a one-to-one relationship.
  - Each `CartItem` is associated with one `Product`, representing a one-to-one relationship.

Sequence Diagram

The sequence diagram in Figure 6.2 illustrates the interactions between the frontend and backend components along with keey librairies when exploring the order management feature.

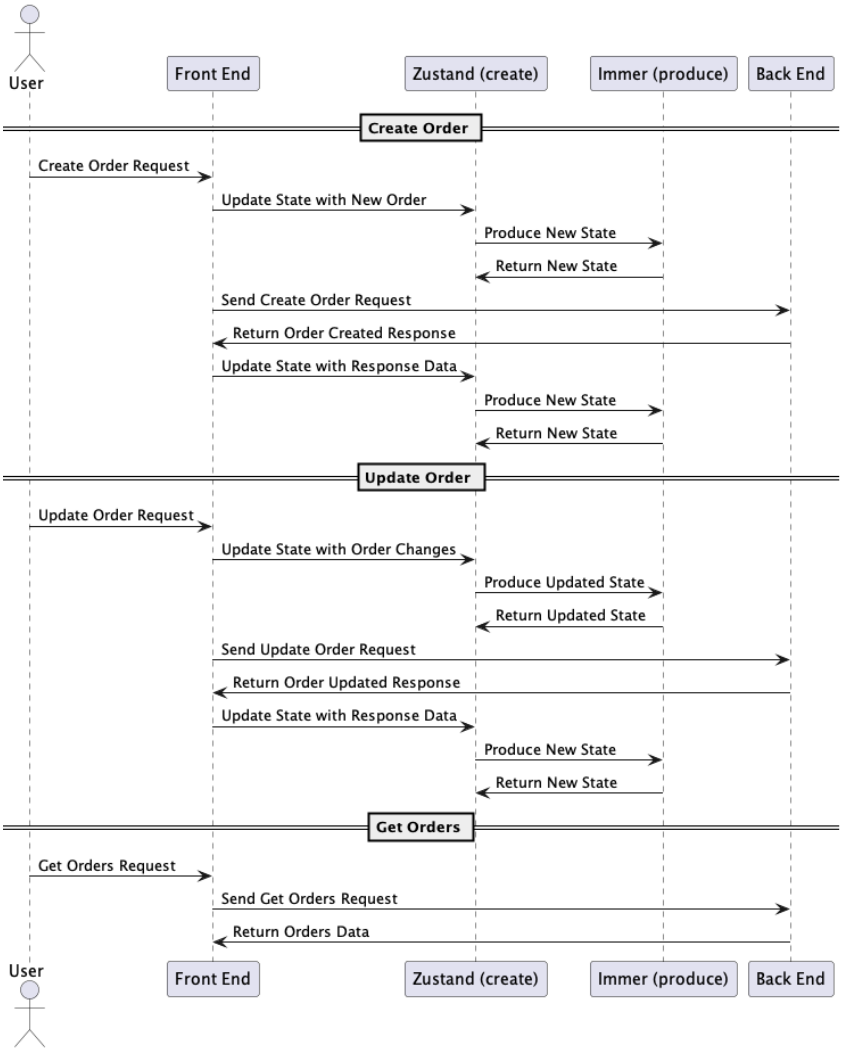

Figure 6.2: Sequence Diagram for Sprint 3

Zustand - Immer

- **Zustand:** Zustand is a small, fast, and scalable state management library for React applications. It provides a simple way to create global state containers that can be shared across components. Zustand is lightweight and efficient, making it ideal for managing state in mobile apps. It supports features like selectors, middleware, and devtools, making it a versatile choice for state management in React applications.
- **Immer:** Immer is a library that simplifies immutable state updates in JavaScript applications. It allows developers to write code that looks like it mutates state directly, while ensuring immutability behind the scenes. Immer provides a convenient way to update state in a more intuitive and readable manner, making it easier to manage complex state in mobile apps.
- **Zustand + Immer:** Together, Zustand and Immer provide a powerful combination for managing state in React applications. Zustand simplifies the creation of global state containers, while Immer

enhances the immutability of state updates. By using Zustand with Immer, developers can create efficient, scalable, and maintainable state management solutions for mobile apps.

Listing 6.1: Zustand + Immer Example

```
import create from 'zustand';
import produce from 'immer';

const useCounterStore = create((set) => ({
  counter: 0,
  increment: () => set(produce((state) => {
    state.counter += 1;
  })),
  decrement: () => set(produce((state) => {
    state.counter -= 1;
  })),
}));

export default useCounterStore;
```

User Journey

The user journey for the order management feature is as follows:

Figure 6.3: User Journey for Sprint 3

Screenshots

The screenshots in Figure 6.4 showcase the order management feature in the mobile app. The first screenshot displays the order management screen, which allows users to view all orders. The second screenshot shows the order details screen, which provides detailed information about a specific order. The third screenshot illustrates the order creation screen, where users can update or create orders.

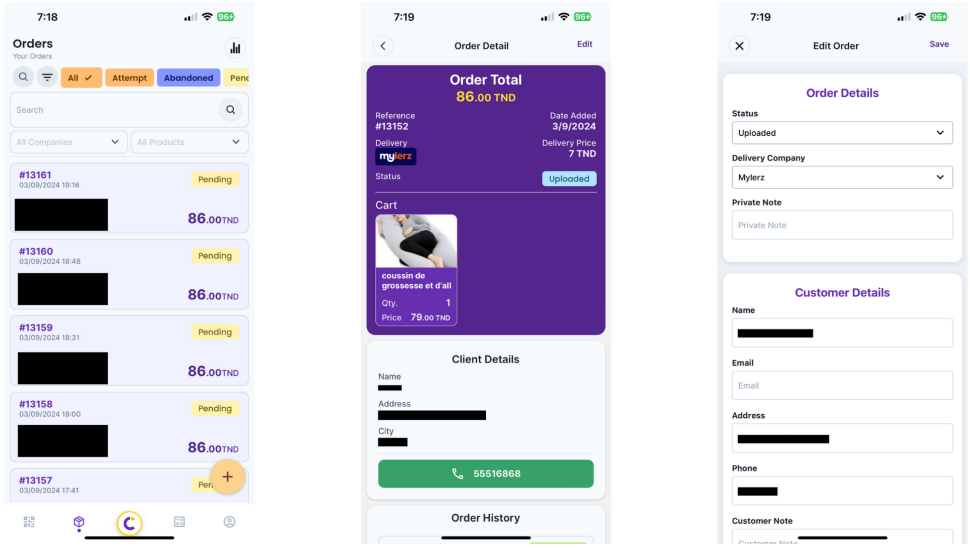

Figure 6.4: Screenshots for Sprint 3

Summary

In this section, we discussed the design and implementation of the order management feature in the mobile app. We explored the database schema, class diagram, sequence diagram, and user journey for this feature. Additionally, we provided code snippets and screenshots to illustrate the implementation process.

6.3 Sprint 4: Mobile - Budget & Costs Management

6.3.1 Sprint Backlog

In this section, we present the Sprint Backlog, detailing the tasks planned for developing budget and costs management features. It is a concise list of user stories and specific actions that will guide us through the sprint.

Sprint	User Story	Task	ID
4	4.1	Frontend: Create Screen to Enter Various Cost Parameters	4.1.1
4	4.2	Frontend: Implement Calculation of Key Financial Metrics	4.2.1
4	4.3	Frontend: Create Screen to View All Budgets	4.3.1
4	4.4	Frontend: Implement Selection of Specific Budgets	4.4.1
4	4.5	Frontend: Implement Editing of Budget Information	4.5.1
4	4.6	Frontend: Display Total Balance, Revenue, and Expenses	4.6.1

Table 6.2: Sprint 4 - Mobile Budget & Costs Management

6.3.2 Design & Implementation

In this section, we will explore the design and implementation of the mobile budget and costs management feature. We will begin by examining the backend design and implementation, followed by an analysis of the frontend design.

Database Schema

The database schema for the budget and costs management feature is shown in Figure 6.5. It consists of the following tables:

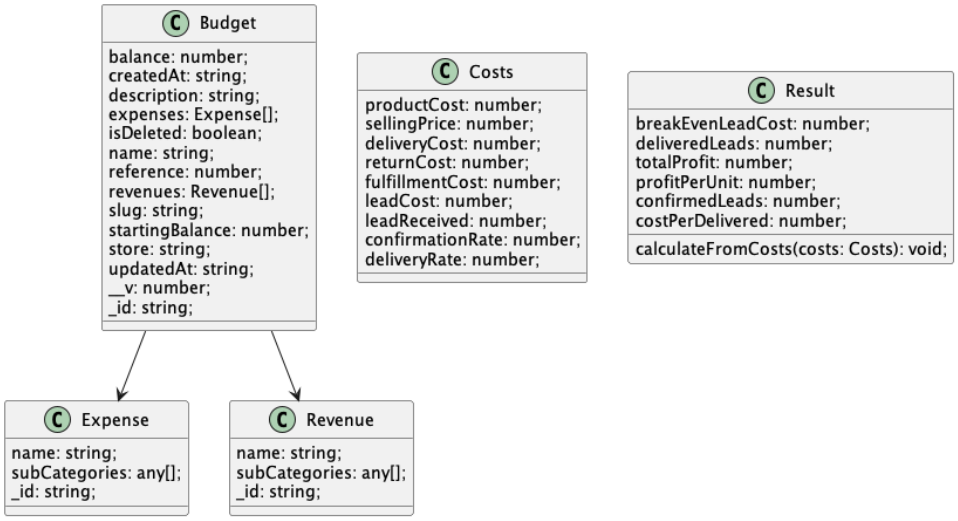

Figure 6.5: Database Schema for Sprint 4

The class diagram depicts relationships between four main classes: `Budget`, `Expense`, `Revenue`, and `Result`.

- **Budget Class:** Represents a budget with attributes like balance, creation and update dates, description, a list of expenses and revenues, starting balance, store, and a unique identifier (`_id`).
- **Expense Class** and **Revenue Class:** Both represent financial items linked to the budget, with attributes such as name, subcategories, and a unique identifier (`_id`).
- **Costs Class:** Captures various cost-related metrics like product cost, selling price, and delivery cost.
- **Result Class:** Calculates financial results based on costs, including break-even lead cost, total profit, and cost per delivered lead. It has a method `calculateFromCosts()` that processes data from the `Costs` class.
- **Relationships:**

- `Budget` has one-to-many relationships with both `Expense` and `Revenue`, indicating that a budget can include multiple expenses and revenues.

Sequence Diagram

The sequence diagram in Figure 6.6 illustrates the interactions between the frontend and backend components when exploring the budget and costs management feature.

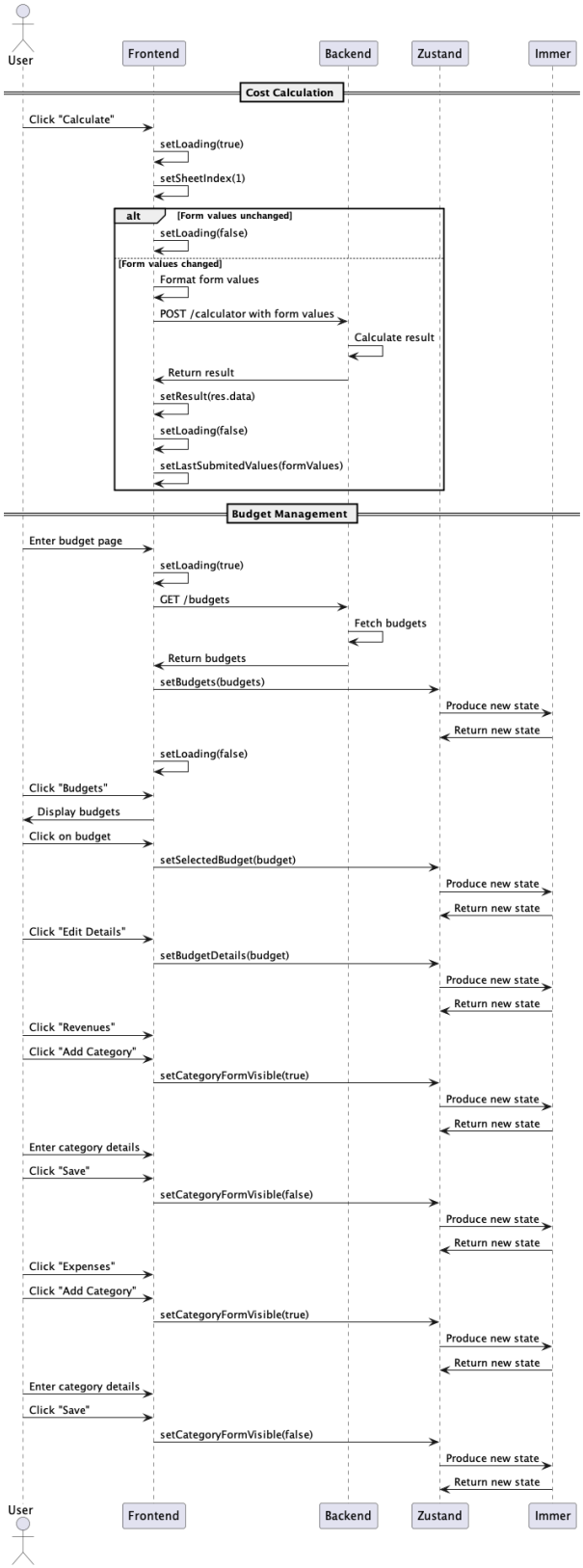

Figure 6.6: Sequence Diagram for Sprint 4

User Journey

The user journey for the budget and costs management feature is as follows:

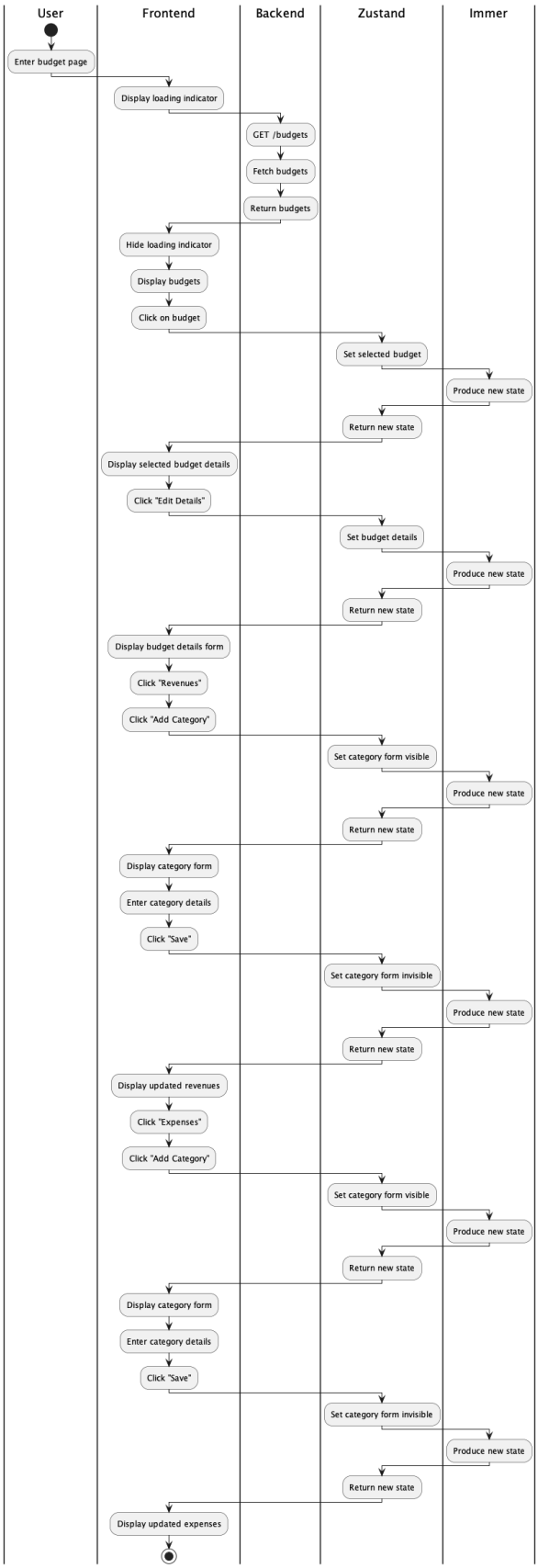

Figure 6.7: User Journey for Sprint 4

Screenshots

The screenshots in Figures 6.8, 6.9, and 6.10 showcase the budget and costs management features in the mobile app.

- **Figure 1:** The first figure includes three screenshots:
  - The tools page where users can select either the budget management tool or the cost calculator tool.
  - Screenshots displaying the various costs that can be input.
- **Figure 2:** The second figure includes:
  - The various outputs calculated from the costs.
  - The budget home page.
  - The popup showing various budgets to select from.
- **Figure 3:** The third figure includes:
  - The popup where users can modify the selected budget.
  - The popup where users can add a category to the revenues tab.
  - The expenses tab.

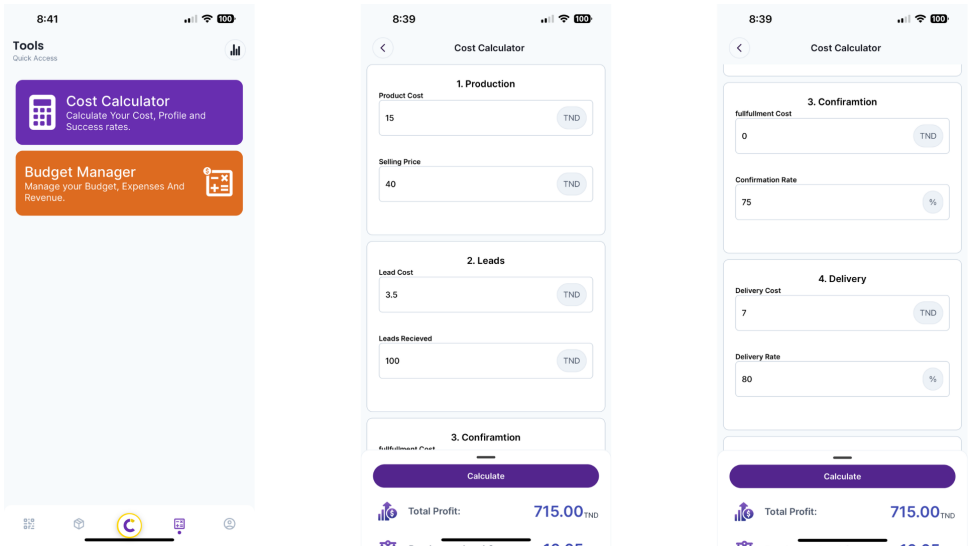

Figure 6.8: Screenshots for Figure 1

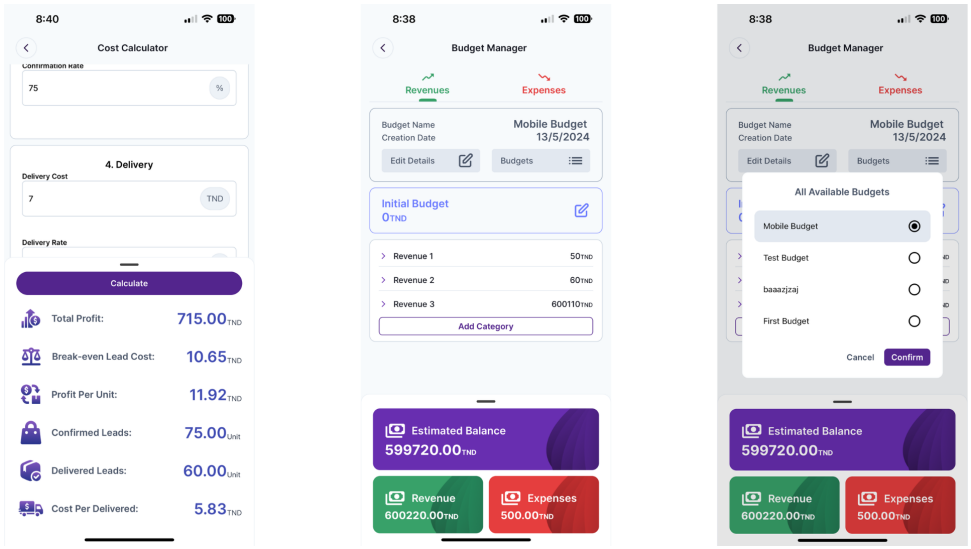

Figure 6.9: Screenshots for Figure 2

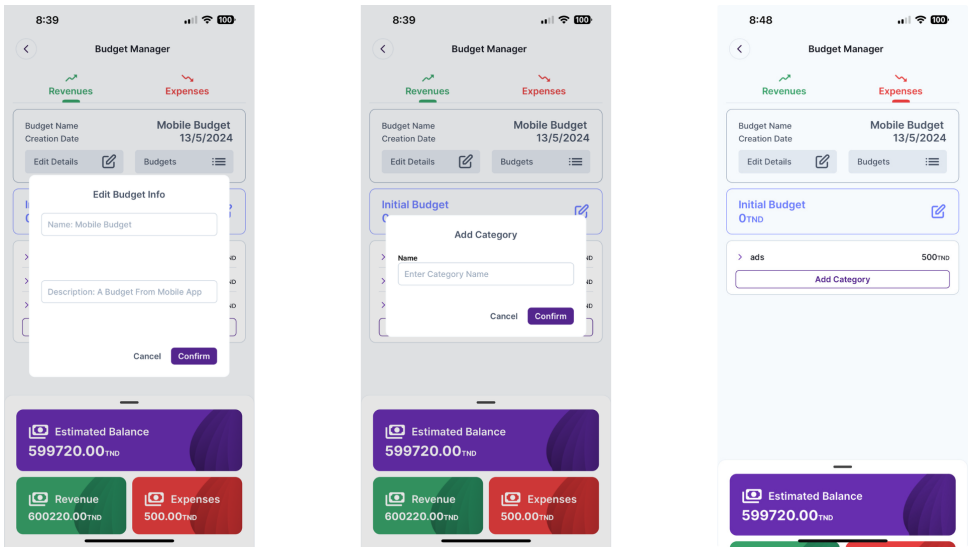

Figure 6.10: Screenshots for Figure 3

Summary

In this section, we discussed the design and implementation of the budget and costs management feature in the mobile app. We explored the database schema, class diagram, sequence diagram, and user journey for this feature. Additionally, we provided code snippets and screenshots to illustrate the implementation process.

6.4 Sprint 5: Mobile - Statistics Management

6.4.1 Sprint Backlog

In this section, we present the Sprint Backlog, detailing the tasks planned for developing statistics management features. It is a concise list of user stories and specific actions that will guide us through the sprint.

Sprint	User Story	Task	ID
5	5.1	Frontend: Create Screen to Display Different Types of Statistics	5.1.1
5	5.2	Frontend: Implement Filtering Options for Statistical Data	5.2.1

Table 6.3: Sprint 5 - Statistical Analysis

6.4.2 Design & Implementation

In this section, we will explore the design and implementation of the mobile statistics management feature. We will begin by examining the backend design and implementation, followed by an analysis of the frontend design.

Database Schema

The database schema for the statistics management feature is shown in Figure 6.11. It consists of the following tables:

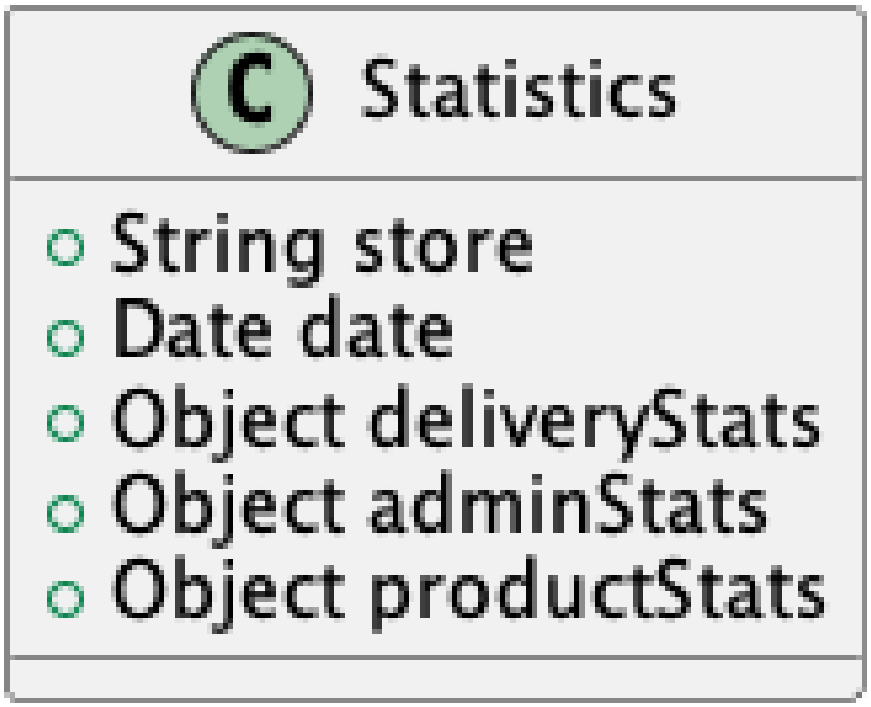

Figure 6.11: Database Schema for Sprint 5

The `Statistics` class is designed to capture various statistical data within an application. It includes the following attributes:

- `store` **(String):** Represents the store for which the statistics are being collected.
- `date` **(Date):** Captures the date for the statistical data, indicating when the statistics were recorded.
- `deliveryStats` **(Object):** Contains statistical data related to deliveries, including metrics such as delivery times, success rates, etc.
- `adminStats` **(Object):** Holds statistical data pertinent to administrative functions, such as user activity, system performance, and other admin-related metrics.
- `productStats` **(Object):** Stores statistical data for products, including sales figures, product performance, and other relevant metrics.

Sequence Diagram

The sequence diagram in Figure 6.12 illustrates the interactions between the frontend and backend components when exploring the statistics management feature.

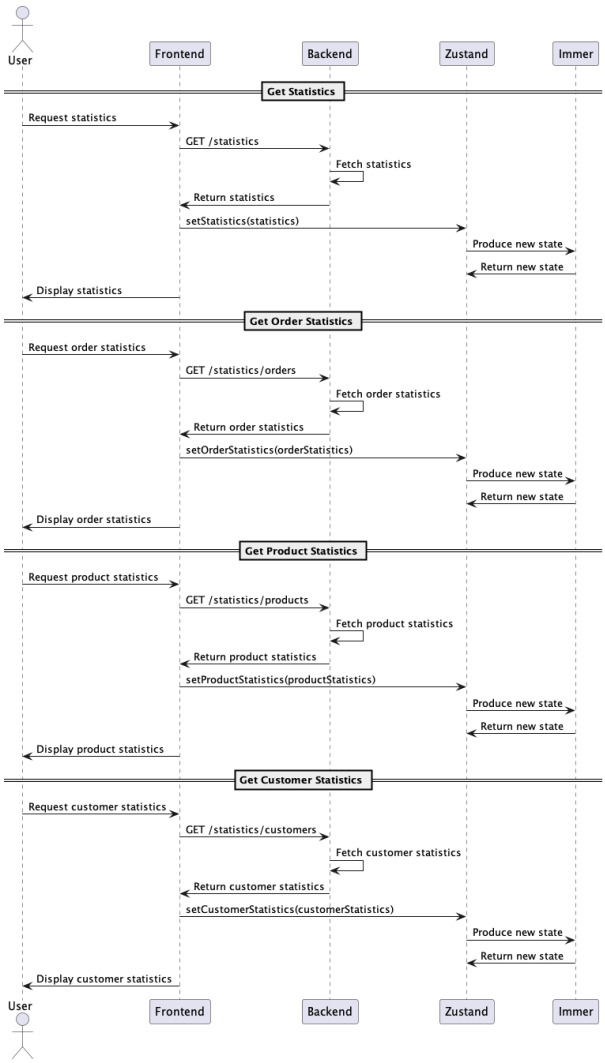

Figure 6.12: Sequence Diagram for Sprint 5

User Journey

The user journey for the statistics management feature is as follows:

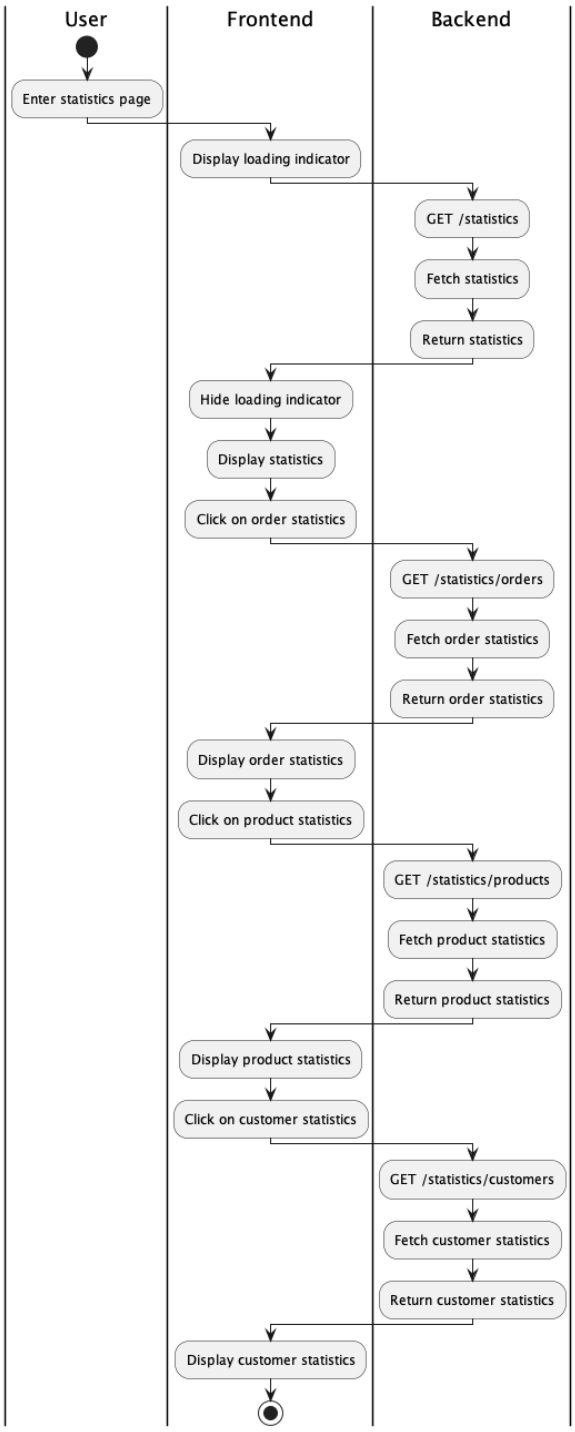

Figure 6.13: User Journey for Sprint 5

Screenshots

The screenshots in Figure 6.14 showcase the statistics management feature in the mobile app. The first screenshot displays the delivery stats tab, which allows users to view delivery-related statistical data. The second screenshot shows the products stats tab, providing insights into product performance. The third screenshot presents the team stats tab, offering statistics related to team activities.

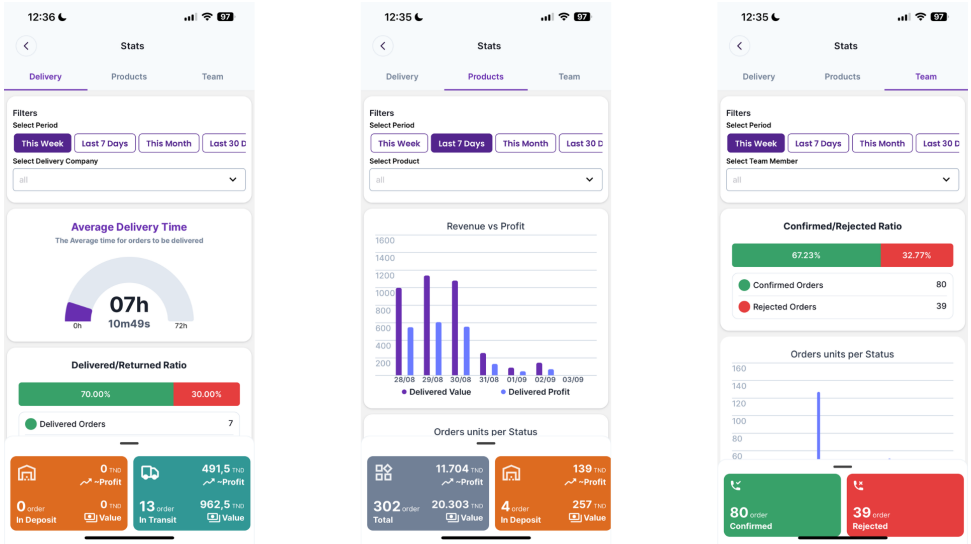

Figure 6.14: Screenshots for Sprint 5

Summary

In this section, we discussed the design and implementation of the statistics management feature in the mobile app. We explored the database schema, class diagram, sequence diagram, and user journey for this feature. Additionally, we provided code snippets and screenshots to illustrate the implementation process.

6.5 Sprint 6: Mobile - Notifications Center

6.5.1 Sprint Backlog

In this section, we present the Sprint Backlog, detailing the tasks planned for developing the notifications center feature. It is a concise list of user stories and specific actions that will guide us through the sprint.

Sprint	User Story	Task	ID
6	6.1	Frontend: Create Screen to Customise Notifications Sound	6.1.1
6	6.2	Frontend: Implement Socket Integration	6.2.1

Table 6.4: Sprint 6 - Notifications Center

6.5.2 Design & Implementation

In this section, we will explore the design and implementation of the notifications center feature, focusing on the integration of Socket.IO from the backend (Node.js with Express) to the frontend (React Native).

Socket.IO Integration

Socket.IO is a library that enables real-time, bidirectional, and event-based communication between the server and the client. It consists of two parts: a server-side library for Node.js and a client-side library for the browser or mobile applications.

- **Backend (Node.js with Express):** The backend setup involves installing the Socket.IO library and configuring it to work with an Express server. The following code snippet demonstrates the setup:

Listing 6.2: Backend Socket.IO Setup

```
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = socketIo(server);

io.on('connection', (socket) => {
    console.log('New client connected');

    socket.on('disconnect', () => {
        console.log('Client disconnected');
    });

    socket.on('notification', (data) => {
        io.emit('notification', data);
    });
});

server.listen(4000, () => {
    console.log('Listening on port 4000');
});
```

- **Frontend (React Native):** The frontend setup involves installing the Socket.IO client library and configuring it to connect to the backend server. The following code snippet demonstrates the

setup:

Listing 6.3: Frontend Socket.IO Setup

```
import React, { useEffect } from 'react';
import { View, Text } from 'react-native';
import io from 'socket.io-client';

const socket = io('http://localhost:4000');

const NotificationsScreen = () => {
    useEffect(() => {
        socket.on('notification', (data) => {
            console.log('Notification received:', data);
        });

        return () => {
            socket.off('notification');
        };
    }, []);

    return (
        <View>
            <Text>Notifications Center</Text>
        </View>
    );
};

export default NotificationsScreen;
```

User Journey

The user journey for the notifications center feature is as follows:

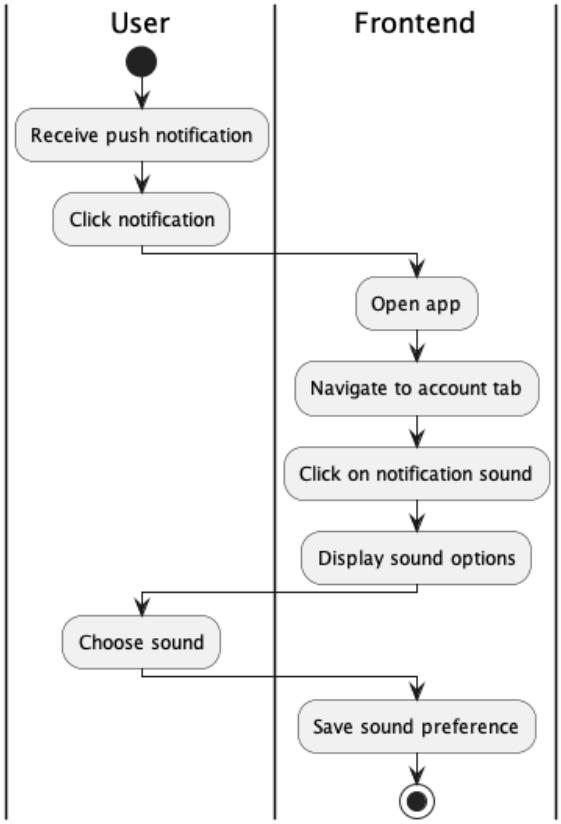

Figure 6.15: User Journey for Sprint 6

Screenshots

The screenshots in Figure 6.16 showcase the notifications center feature in the mobile app. The first screenshot displays a live push notification, which allows users to see a created order. The second screenshot shows the customization options for notification sounds.

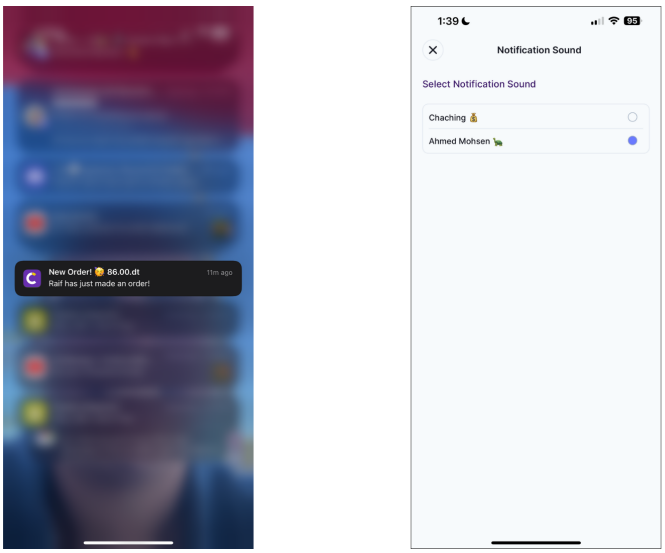

Figure 6.16: Screenshots for Sprint 6

Summary

In this section, we discussed the design and implementation of the notifications center feature in the mobile app. We explored the integration of Socket.IO from the backend to the frontend, the user journey, and provided screenshots to illustrate the implementation process.

6.6 Conclusion

In this chapter, we discussed the four mobile-related sprints, focusing on order management, budget and costs management, statistics management, and the notifications center. We explored the design and implementation of each feature, including database schemas, class diagrams, sequence diagrams, user journeys, and screenshots. Additionally, we provided code snippets to illustrate the integration of Socket.IO for real-time communication between the server and the client. Overall, these sprints helped enhance the mobile app's functionality and user experience, providing users with valuable tools for managing orders, budgets, costs, statistics, and notifications.

Chapter 7 Perspectives and Conclusion

In this internship, we had the opportunity to work on a real-world project, which provided us with invaluable experience in the software development process and team collaboration. We learned how to effectively use various technologies and tools to build a scalable and performant system. Despite facing numerous challenges throughout the internship, we overcame them through teamwork and continuous learning. We are proud of our accomplishments and are excited to see the project go live.

Our journey began with a deep dive into understanding the project requirements and familiarizing ourselves with the technologies we would be using. We then moved on to designing the system architecture and implementing the core features.

We started by laying the foundation in **Sprint 1**, where we recreated a simple template to establish a solid starting point for the project.

In **Sprint 2**, we enhanced the system's marketing capabilities by implementing upsell and cross-sell functionalities, providing additional value to the platform's users.

Sprint 3 focused on the mobile aspect, where we developed comprehensive order management features. These features included displaying, adding, editing, filtering, and searching orders, ensuring that users could efficiently manage their orders within the mobile app.

Moving forward, **Sprint 4** was dedicated to implementing budget and cost management features in the mobile app, allowing users to effectively manage their finances within the platform.

In **Sprint 5**, we introduced a statistics management feature, enabling users to view and filter various types of statistical data. This added analytical capabilities to the platform, helping users make informed decisions.

Finally, in **Sprint 6**, we significantly improved user engagement by implementing real-time notifications using Socket.IO. This feature ensured that users were always updated with the latest information, enhancing the overall user experience.

Throughout these sprints, we adhered to a modular monolith architecture, ensuring that each com-

ponent of the system was well-defined and maintainable. We utilized a variety of technologies, including Node.js with Express for the backend, React for the web dashboard, React Native for the mobile app, and MongoDB for the database.

This internship has been a rewarding experience, allowing us to apply our skills in a practical setting and gain new insights into software development. We look forward to the project's future success and its positive impact on users.

Future Work

Looking ahead, there are several areas where the project can be expanded and improved:

Advanced Features

We plan to implement additional features such as advanced analytics, AI-driven recommendations, and enhanced SEO tools. These features will provide users with deeper insights, personalized experiences, and better visibility in search engines.

Localization Expansion

To cater to a global audience, we aim to add support for more languages and adapt the platform for different regional markets. This will involve translating the user interface, adjusting content to fit cultural contexts, and ensuring compliance with local regulations.