

End of Studies Project

Mohamed Dhia Ben Amar

September 30, 2024

Dedication

I dedicate this work to my family, my parents, my sister, my friends and my teachers who have always supported me and encouraged me to pursue my dreams. I am grateful for their love, patience and understanding. I would not have been able to achieve this without them. I dedicate this work to them with all my heart.

For this project has been a long journey, I would like to thank my family for their support and encouragement. I would also like to thank my friends for their help and advice. Finally, I would like to thank my teachers for their guidance and support.

I am also very grateful for the lessons i have learned during this project. I am grateful for the opportunity to learn and grow as a person. I am grateful for the opportunity to work with such a great team. I am grateful for the opportunity to work on such an interesting project.

Mohamed Dhia Ben Amar

Acknowledgments

My everlasting words of gratitude go to every member who helped bring this project to life and made this internship a memorable experience.

A heartfelt thank you to Mr. Sami Kammoun, my supervisor at Converty, for his guidance, support, and encouragement throughout this project. I am deeply grateful for his patience and understanding. This achievement would not have been possible without him. I dedicate this work to him with all my heart.

I would also like to express my sincere appreciation to Mrs. Mariem Bouzouita, my academic supervisor at ESPRIT, for her invaluable guidance and support. Her help and advice were instrumental in the completion of this project. I dedicate this work to her with all my heart.

Lastly, but certainly not least, I extend my gratitude to Converty, the hosting company, for giving me the opportunity to work on this project. I am thankful for the chance to learn and grow as a person and to collaborate with such a great team.

Preface

This report documents my six-month internship at Converty, an innovative e-commerce platform, undertaken during my third year of engineering studies at ESPRIT (École Supérieure Privée d'Ingénierie et de Technologies). Throughout this internship, I gained substantial experience in software engineering and observed firsthand its pivotal role in enhancing client satisfaction and business efficiency.

Converty, located in Jardins Menzah 2, Tunisia, is a forward-thinking startup revolutionizing the regional online business sector. The platform uniquely integrates Customer Relationship Management (CRM) and Content Management System (CMS) functionalities, empowering businesses to efficiently manage and expand their digital presence.

As a CMS, Converty streamlines the processes of online store creation, customization, and product management, while also facilitating integration with major analytics platforms like Meta, Google, and Microsoft. On the CRM front, Converty boasts strategic alliances with top Tunisian delivery services and fulfillment centers, a robust order management system, and real-time performance analytics.

This report offers an in-depth account of my internship at Converty, highlighting the obstacles encountered, the strategies employed to overcome them, and the key insights gained. It emphasizes the critical role of software engineering in refining company operations and fostering innovation within the dynamic e-commerce landscape.

Contents

1 Project Overview	3
1.1 Introduction	4
1.2 Anchor Organisation	4
1.2.1 Converty	4
1.2.2 Services	4
1.3 Study Of The Existing	5
1.4 Problem Statement	6
1.5 Proposed Solution	6
1.6 About the Project	7
1.6.1 Goals	7
1.6.2 Objectives	7
1.6.3 Deliverables	8
1.6.4 Management Plan	8
1.6.5 Project Timeline	10
1.7 Conclusion	13
2 Analysis and Specification of Requirements	14
2.1 Introduction	15
2.2 Functional Requirements	15
2.3 Non-functional Requirements	16
2.4 Analysis of Requirements	17
2.4.1 Identification of System Actors	17
2.4.2 Global Use Case Diagram	17
2.4.3 Refinement of Use Cases	18
2.5 Conclusion	25

3 Project Architecture & Choice Of Technologies	26
3.1 Introduction	27
3.2 Project Architecture - Modular Monolith	27
3.2.1 Backend & Services	27
3.2.2 Frontend - Web Dashboard	31
3.2.3 Frontend - Mobile App	33
3.2.4 Database - MongoDB	36
3.3 Explaining The Main Architecture	38
3.4 Conclusion	39
4 Realization - Web Features	40
4.1 Introduction	41
4.2 Sprint 1 - Recreating the Simple Template	41
4.2.1 Sprint Backlog	41
4.2.2 Design & Implementation	41
4.3 Sprint 2 - Implementing the Upsell/Cross-sell Feature	51
4.3.1 Sprint Backlog	51
4.3.2 Design & Implementation	51
4.4 Conclusion	63
5 Realization - Mobile Features	64
5.1 Introduction	65
5.2 Sprint 3: Mobile - Order Management	65
5.2.1 Sprint Backlog	65
5.2.2 Design & Implementation	65
5.3 Sprint 4: Mobile - Budget & Costs Management	70
5.3.1 Sprint Backlog	70
5.3.2 Design & Implementation	71
5.4 Sprint 5: Mobile - Statistics Management	77
5.4.1 Sprint Backlog	77
5.4.2 Design & Implementation	77
5.5 Sprint 6: Mobile - Notifications Center	81
5.5.1 Sprint Backlog	81
5.5.2 Design & Implementation	81
5.6 Conclusion	85
6 Perspectives and Conclusion	86

List of Tables

1.1	Comparison of SDLC Models	9
1.2	Project Timeline	11
1.3	Product Backlog	13
2.1	System Actors	17
4.1	Sprint 1 - Recreating the Simple Template	41
4.2	Sprint 2 - Implementing the Upsell/Cross-sell Feature	51
5.1	Sprint 3 - Mobile Order Management	65
5.2	Sprint 4 - Mobile Budget & Costs Management	70
5.3	Sprint 5 - Statistical Analysis	77
5.4	Sprint 6 - Notifications Center	81

List of Figures

1.1	Converty Logo	4
1.2	SDLC Phases	8
1.3	Scrum Framework	10
1.4	Gantt Chart	13
2.1	Global Use Case Diagram	18
3.1	Main Architecture	27
3.2	Node.js and Express.js	28
3.3	Go (Golang)	29
3.4	RabbitMQ	30
3.5	React + Vite + TypeScript	31
3.6	Redux	32
3.7	React Native	33
3.8	Expo	34
3.9	Zustand	35
3.10	MongoDB	36
3.11	Redis	38
4.1	Database Schema for Sprint 1	42
4.2	Sequence Diagram for Sprint 1	44
4.3	User Journey for Sprint 1	45
4.4	Shop Landing Page 1	46
4.5	Shop Landing Page 2	46
4.6	Category Page	47
4.7	Search Page	47
4.8	Extra Page	48
4.9	Product Detail Page 1	48

LIST OF FIGURES

4.10 Product Detail Page 2	49
4.11 Product Detail Page 3	49
4.12 Thank You Page	50
4.13 Upsell Popup	50
4.14 Database Schema for Sprint 2	51
4.15 Sequence Diagram for Sprint 2 - Search Operation	52
4.16 Sequence Diagram for Sprint 2 - Create Operation	53
4.17 Sequence Diagram for Sprint 2 - Read Operation	54
4.18 Sequence Diagram for Sprint 2 - Update Operation	55
4.19 Sequence Diagram for Sprint 2 - Delete Operation	56
4.20 Sequence Diagram for Sprint 2 - Read One Operation	57
4.21 Sequence Diagram for Sprint 2 - Caching Layer	58
4.22 User Journey for Sprint 2	60
4.23 Table View Dashboard	61
4.24 Upsell/Cross-sell Preview	61
4.25 Edit/Add Upsell/Cross-sell 1	62
4.26 Edit/Add Upsell/Cross-sell 2	62
 5.1 Database Schema for Sprint 3	66
5.2 Sequence Diagram for Sprint 3	67
5.3 User Journey for Sprint 3	69
5.4 Screenshots for Sprint 3	70
5.5 Database Schema for Sprint 4	71
5.6 Sequence Diagram for Sprint 4	73
5.7 User Journey for Sprint 4	74
5.8 Screenshots for Figure 1	75
5.9 Screenshots for Figure 2	76
5.10 Screenshots for Figure 3	76
5.11 Database Schema for Sprint 5	78
5.12 Sequence Diagram for Sprint 5	79
5.13 User Journey for Sprint 5	80
5.14 Screenshots for Sprint 5	81
5.15 User Journey for Sprint 6	84
5.16 Screenshots for Sprint 6	84

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

General Introduction

Over 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, "Project Overview," covers the scope of the internship, the specific problems addressed, and the objectives of the internship with actionable items. It includes sections on the host company, Converty, its services, a study of the existing solutions, the problem statement, the proposed solution, and details about the project such as goals, objectives, deliverables, management plan, timeline, and a conclusion.

The second chapter, "Analysis and Specification of Requirements," includes an introduction, functional and non-functional requirements, analysis of requirements, identification of system actors, a global

General Introduction

use case diagram, and a refinement of use cases.

The third chapter, "Project Architecture & Choice Of Technologies," describes the architecture that the project will follow and the technologies that will be used.

The fourth chapter, "Realization - Web Features," details the web features developed during the internship.

The fifth chapter, "Realization - Mobile Features," details the mobile features developed during the internship.

The report concludes with a general conclusion and the perspectives of the project.

Chapter 1 Project Overview

Summary

1.1	Introduction	4
1.2	Anchor Organisation	4
1.3	Study Of The Existing	5
1.4	Problem Statement	6
1.5	Proposed Solution	6
1.6	About the Project	7
1.7	Conclusion	13

1.1 Introduction

This chapter provides an overview of the project scope, including the host company, identified problems, proposed solutions, and project objectives. It also discusses the existing e-commerce platform, Shopify, and highlights the areas where Converty can improve to better compete in the market.

1.2 Anchor Organisation

This section introduces the startup, Converty, and its services.

1.2.1 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 [16].



Figure 1.1: Converty Logo

Converty provides a comprehensive suite of services designed to support online store management, customer relationship management (CRM), marketing automation, and detailed reporting. The core mission of Converty is to revolutionize the e-commerce landscape for small and medium-sized enterprises (SMEs) by offering an all-in-one solution that streamlines business processes, reduces operational costs, and enhances overall efficiency.

1.2.2 Services

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

1. Project Overview

- 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.

1.3 Study Of The Existing

Shopify is a complete commerce platform that lets anyone start, manage, and grow a business. You can use Shopify to build an online store, manage sales, market to customers, and accept payments in digital and physical locations. Shopify's reputation as a commerce leader comes from listening to the experiences of millions of business owners. By supporting solopreneurs and enterprise brands, Shopify has built the features and products that power today's businesses and will shape the future of commerce [1].

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.

1. Project Overview

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.

While Shopify is a well-established platform with a wide range of features, Converty aims to address some of the limitations and challenges faced by Shopify users. Converty focuses on providing a more streamlined and customizable experience, eliminating the need for multiple applications and reducing the complexity associated with extensive feature sets. By integrating essential tools directly into the platform, Converty offers a more efficient and user-friendly solution for managing online stores. This approach not only simplifies the user experience but also enhances performance and scalability, making it a strong competitor in the e-commerce market.

This comparison highlights the areas where Converty can improve to better compete with Shopify.

1.4 Problem Statement

Despite Converty's notable progress in the e-commerce industry, the product, which consists of a web dashboard and a mobile application, currently faces several critical challenges that must be addressed to compete effectively with Shopify. These challenges include:

- **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. By enhancing performance, adding key features, and completing the mobile app experience, Converty can provide a more comprehensive and user-friendly platform that meets the needs of online store owners and merchants.

1.5 Proposed Solution

To fix the challenges faced by Converty identified in the previous section, the following solutions are proposed:

- **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

1. Project Overview

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 are expected to significantly enhance Converty's platform, making it more competitive in the e-commerce market. By improving performance, adding key features, and completing the mobile app experience, Converty aims to provide a comprehensive and user-friendly solution for online store owners and merchants. These enhancements will help Converty attract new users, retain existing customers, and establish itself as a leading platform in the e-commerce industry.

1.6 About the Project

This section provides a comprehensive overview of the project, including its goals, objectives, deliverables, management plan, and timeline. It also discusses the Software Development Life Cycle (SDLC) model used for the project.

1.6.1 Goals

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

1.6.2 Objectives

To achieve these goals, the following objectives were set:

- Recode the first template using a new technology **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.

1.6.3 Deliverables

The project deliverables include the following components:

- **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.

1.6.4 Management Plan

The management plan for this project follows the Agile SDLC model, which emphasizes iterative development, collaboration, and flexibility. The project is divided into several sprints, each focusing on specific tasks and deliverables. To begin, let's look into the Software Development Life Cycle (SDLC) models.

1.6.4.1 Software Development Life Cycle (SDLC)

The SDLC employed for this project involved iterative development and continuous feedback to meet all project requirements effectively. Figure 1.2 illustrates the SDLC phases used in this project [2].

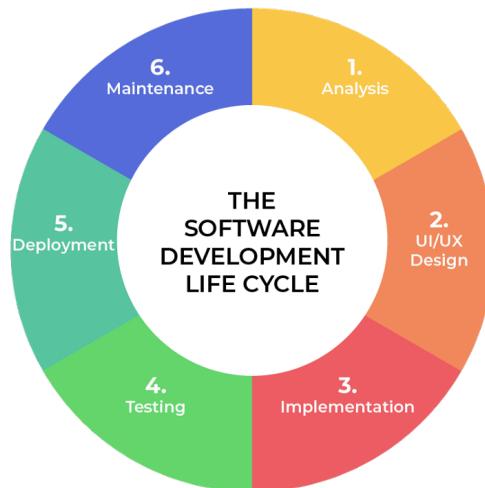


Figure 1.2: SDLC Phases

1.6.4.2 Comparison of SDLC Models

To determine the most suitable SDLC model for this project, a comparison of different models was conducted. Table 1.1 provides a comparison of the Waterfall, Agile, Scrum, and Kanban models, highlighting their advantages, disadvantages, and suitability.

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 1.1: Comparison of SDLC Models

1.6.4.3 Agile Methodology

After comparing various SDLC models, the Agile methodology was chosen for this project due to its flexibility, collaboration, customer feedback, and rapid releases. Agile 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.

1.6.4.4 Scrum Framework

Within the Agile methodology, the Scrum framework was selected to manage the project's development process. Scrum, 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

1. Project Overview

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. Figure 1.3 illustrates the Scrum framework used in this project [3].

SCRUM FRAMEWORK

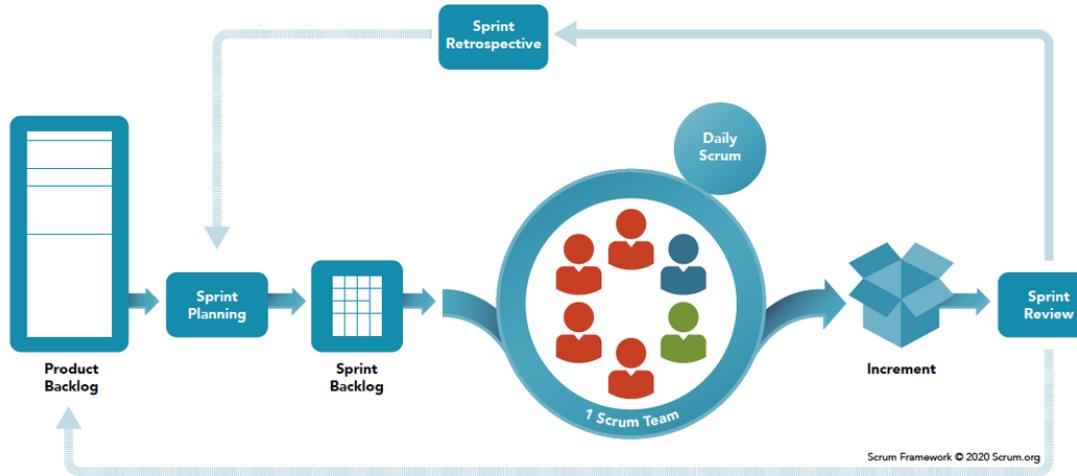


Figure 1.3: Scrum Framework

1.6.4.5 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 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.

1.6.5 Project Timeline

The project timeline is outlined in Table 1.2, which details the duration and key milestones for each sprint.

1. Project Overview

Sprint	Duration	Key Milestones
Sprint 1	4 weeks	Recreate the simple template
Sprint 2	4 weeks	Implement the upsell feature
Sprint 3	4 weeks	Enhance the mobile app with order management
Sprint 4	4 weeks	Add budget and costs management to the mobile app
Sprint 5	4 weeks	Integrate statistics management into the mobile app
Sprint 6	4 weeks	Develop the notifications center for the mobile app

Table 1.2: Project Timeline

1.6.5.1 Scrum Team

The Scrum team for this project includes the following key roles:

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

1.6.5.2 Product Backlog

The product backlog contains all the tasks and user stories that need to be completed during the project.

Table 1.3 outlines the product backlog, including the sprint, theme, user story ID, user story description, and priority.

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
		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. Project Overview

Sprint	Theme	ID	User Story	Priority
		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 pre-views	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
		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

1. Project Overview

Sprint	Theme	ID	User Story	Priority
		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 1.3: Product Backlog

1.6.5.3 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. Figure 1.4 illustrates the Gantt chart for this project [4].

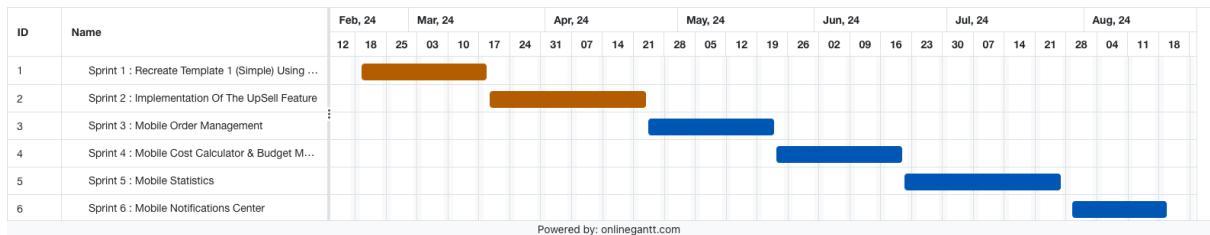


Figure 1.4: Gantt Chart

1.7 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 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 2 Analysis and Specification of Requirements

Summary

2.1	Introduction	15
2.2	Functional Requirements	15
2.3	Non-functional Requirements	16
2.4	Analysis of Requirements	17
2.5	Conclusion	25

2.1 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.

2.2 Functional Requirements

The functional requirements of our solution include the features that it must incorporate to meet the project's requirements and objectives. These requirements are essential for ensuring that the system performs the necessary tasks effectively.

The system must offer its users 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.
- 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.

Having outlined the functional requirements, it is also crucial to consider the non-functional requirements that ensure the system's overall performance and quality.

2.3 Non-functional Requirements

To ensure the system's overall performance and quality, the following non-functional requirements must be met:

- 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.

2.4 Analysis of Requirements

This section provides an in-depth analysis of the system's requirements, identifying the key actors and their interactions with the system.

2.4.1 Identification of System Actors

The system involves the following actors, as detailed in Table 2.1:

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 2.1: System Actors

2.4.2 Global Use Case Diagram

The global use case diagram represents the interactions between the system and its actors, as shown in Figure 2.1.

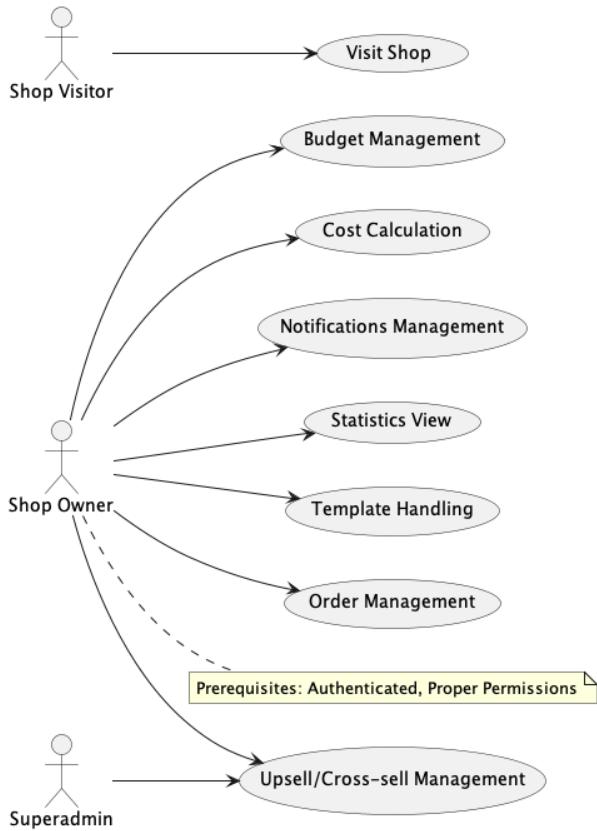


Figure 2.1: Global Use Case Diagram

2.4.3 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.

2.4.3.1 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	<ol style="list-style-type: none"> 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	<ul style="list-style-type: none"> • 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.

2.4.3.2 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	<ul style="list-style-type: none"> • 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	<ol style="list-style-type: none"> 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	<ul style="list-style-type: none"> • 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.

2.4.3.3 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	<ul style="list-style-type: none"> • 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	<ol style="list-style-type: none"> 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	<ul style="list-style-type: none"> • 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.

2.4.3.4 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	<ul style="list-style-type: none"> • 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	<ol style="list-style-type: none"> 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	<ul style="list-style-type: none"> • 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.

2.4.3.5 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	<ul style="list-style-type: none"> • 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	<ol style="list-style-type: none"> 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	<ul style="list-style-type: none"> • The shop owner encounters an error while inputting costs. • The shop owner encounters an error while calculating the total costs.

2.4.3.6 Use Case 6: Generate Financial Reports

Title	Generate Financial Reports
Summary	This use case describes the actions performed by a shop owner to generate financial reports based on the inputted costs and revenues.
Actors	Shop Owner
Preconditions	<ul style="list-style-type: none"> • The shop owner has logged into the application. • The shop owner has inputted all relevant costs and revenues. • The shop owner has accessed the financial reports section.
Postconditions	The shop owner has generated and reviewed the financial reports.

Nominal Scenario	<ol style="list-style-type: none"> 1. The shop owner logs into the system. 2. The shop owner navigates to the financial reports section. 3. The shop owner selects the type of report to generate. 4. The shop owner generates the financial report. 5. The shop owner reviews the financial report. 6. The shop owner logs out of the system.
Alternative Scenarios	<ul style="list-style-type: none"> • The shop owner encounters an error while generating the financial report. • The shop owner finds discrepancies in the financial report and needs to re-input data.

2.4.3.7 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	<ul style="list-style-type: none"> • 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	<ol style="list-style-type: none">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	<ul style="list-style-type: none">• The shop owner does not find any statistics.• The shop owner encounters an error while filtering statistics.

2.5 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. By thoroughly understanding these requirements and use cases, we can ensure that the system will meet the needs of its users and provide a robust, efficient, and user-friendly solution.

Chapter 3 Project Architecture & Choice Of Technologies

Summary

3.1	Introduction	27
3.2	Project Architecture - Modular Monolith	27
3.3	Explaining The Main Architecture	38
3.4	Conclusion	39

3.1 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. Additionally, this chapter highlights the importance of making informed decisions when it comes to selecting technologies that align with the project requirements and objectives.

3.2 Project Architecture - Modular Monolith

The project architecture of Converty follows a Modular Monolith approach, which combines the benefits of a monolithic architecture with modular design principles. This section provides an in-depth look at the backend and services, frontend web dashboard, frontend mobile app, and the database used in the project. The main architecture of Converty is shown in Figure 3.1.

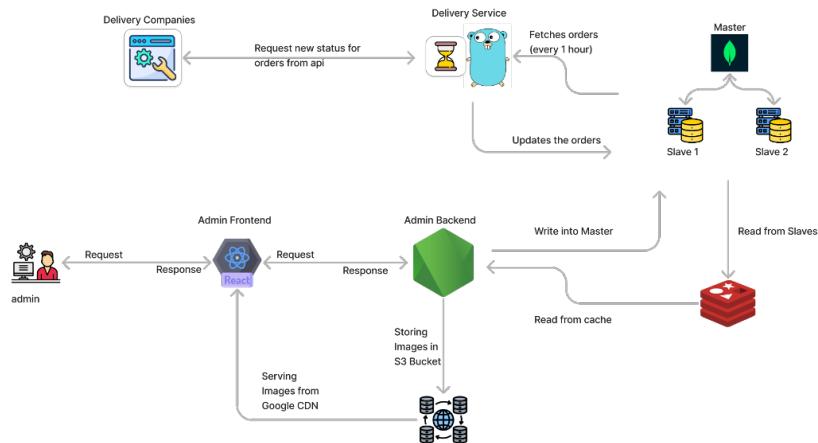


Figure 3.1: Main Architecture

3.2.1 Backend & Services

The backend of Converty 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.

3.2.1.1 Node.js - Express.js

Node.js is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools, and scripts [5]. 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 is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications [6]. 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 3.2: 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.

3.2.1.2 Go (Golang)

Go, also known as Golang, is an open-source, compiled, and statically typed programming language designed by Google. It is built to be simple, high-performing, readable, and efficient [7]. Go is known for its performance, simplicity, and ease of use, making it an excellent choice for building microservices. It 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 3.3: 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.

3.2.1.3 RabbitMQ

RabbitMQ is a message-queueing software also known as a message broker or queue manager. Simply put, it is software where queues are defined, to which applications connect in order to transfer messages [8]. We utilized RabbitMQ to handle asynchronous tasks and background processing. It supports both message queuing and publish/subscribe messaging patterns, ensuring reliable message delivery, fault tolerance, and scalability. This makes RabbitMQ an excellent choice for decoupling system components and managing distributed systems. In our implementation, RabbitMQ orchestrates tasks, events, and notifications, ensuring efficient and dependable communication between various services.



Figure 3.4: 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.

3.2.2 Frontend - Web Dashboard

3.2.2.1 React + Vite + TypeScript

Combining Vite with TypeScript and React allows developers to harness the power of fast builds, hot module replacement, and a highly efficient development workflow [9]. The three out of the four templates in Converty are also coded using React, Vite, and TypeScript. These templates provide the resulting 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, hot module replacement, and type safety. This ensures a consistent and efficient development process for creating and maintaining the shop owners' online shop.



Figure 3.5: 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.

3.2.2.2 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 3.6: 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.

3.2.3 Frontend - Mobile App

3.2.3.1 React Native

React Native is an open-source UI software framework created by Facebook Inc. (now Meta Platforms). It is used to develop applications for Android, Android TV, iOS, macOS, tvOS, Web, Windows, and UWP by enabling developers to use the React framework along with native platform capabilities [10]. It is used to develop Android and iOS applications at Facebook, Microsoft, and Shopify. It is also being used to develop virtual reality applications at Oculus. 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.

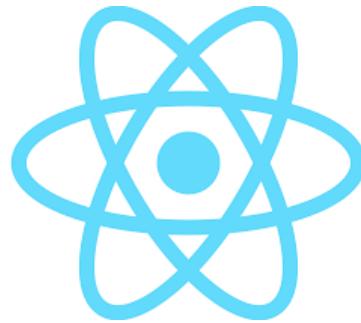


Figure 3.7: React Native

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.

3.2.3.2 Expo

Expo is an open-source platform for making universal native apps that run on Android, iOS, and the web. It includes a universal runtime and libraries that let you build native apps by writing React and JavaScript [11]. 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.

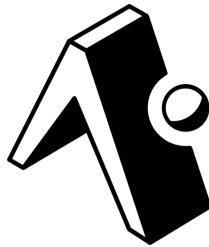


Figure 3.8: Expo

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.

3.2.3.3 Zustand

Zustand is a small, fast, and scalable barebones state management solution. Zustand has a comfy API based on hooks. It isn't boilerplate or opinionated, but has enough convention to be explicit and flux-like [12]. 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 3.9: 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.

3.2.4 Database - MongoDB

MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need [13]. 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 3.10: 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.
- **Compatibility:** MongoDB is compatible with a wide range of programming languages, frameworks, and tools, making it easy to integrate with other technologies.

3.2.4.1 Data Modeling - Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the

3. Project Architecture & Choice Of Technologies

representation of those objects in MongoDB [14]. Mongoose 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.

3.2.4.2 Caching - Redis

Redis is the world's fastest in-memory database. It provides cloud and on-prem solutions for caching, vector search, and NoSQL databases that seamlessly fit into any tech stack—making it simple for digital customers to build, scale, and deploy the fast apps our world runs on [15]. Redis 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.

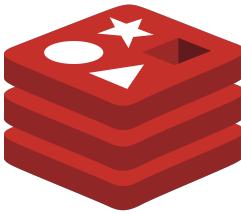


Figure 3.11: Redis

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.

3.3 Explaining The Main Architecture

After understanding the components that are present in the Figure 3.1 of the main architecture, it is crucial to have an understanding of how these components interact with each other.

The system ensures efficient communication between components for order management and admin control. Delivery companies send requests to the Go-based Delivery Service, which updates order statuses in MongoDB. The Master database handles writes, while slaves handle reads, with Redis providing cached data for faster access. The React Admin Frontend communicates with the Node.js Admin Backend, which interacts with the database and manages image storage via S3, delivering content through a Google CDN for optimized performance. This structure supports scalability and real-time data processing.

3.4 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 4 Realization - Web

Features

Summary

4.1	Introduction	41
4.2	Sprint 1 - Recreating the Simple Template	41
4.3	Sprint 2 - Implementing the Upsell/Cross-sell Feature	51
4.4	Conclusion	63

4.1 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.

4.2 Sprint 1 - Recreating the Simple Template

4.2.1 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 4.1: Sprint 1 - Recreating the Simple Template

4.2.2 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.

4.2.2.1 Database Schema

The database schema for the shop visiting process is shown in Figure 4.1. It consists of the following tables:

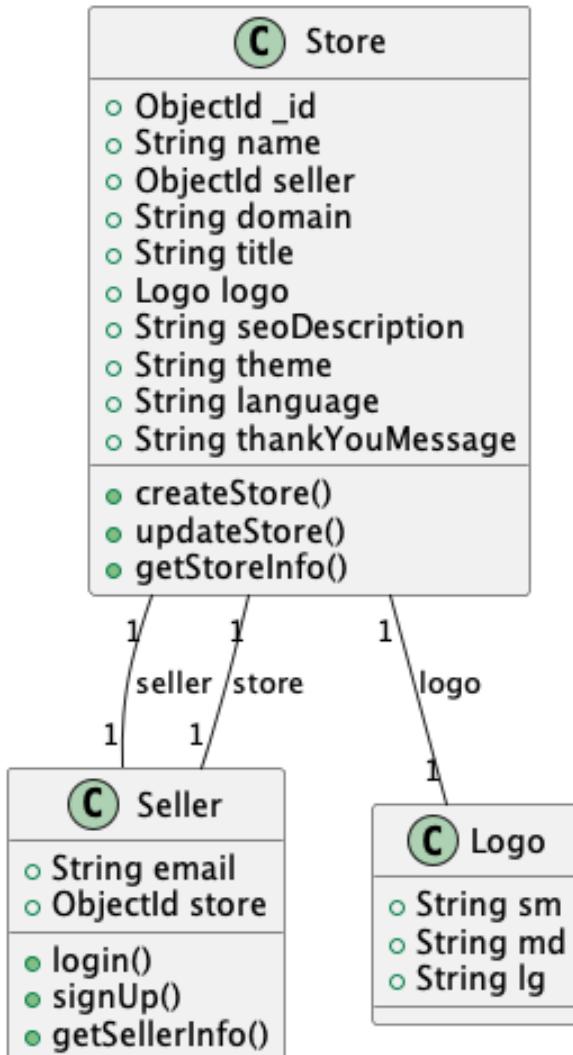


Figure 4.1: Database Schema for Sprint 1

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). The class includes methods for creating, updating, and retrieving store information (`createStore`, `updateStore`, `getStoreInfo`).
- **Seller Class:** Represents the seller associated with a store, containing attributes like email and a reference to the **Store**. The class includes methods for logging in, signing up, and retrieving seller information (`login`, `signUp`, `getSellerInfo`).
- **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.
- Each **Store** is associated with a **Logo** through a one-to-one relationship.

4.2.2.2 Sequence Diagram

The sequence diagram in Figure 4.2 illustrates the interactions between the different components of the shop visiting process. The sequence of events is as follows:

4. Realization - Web Features

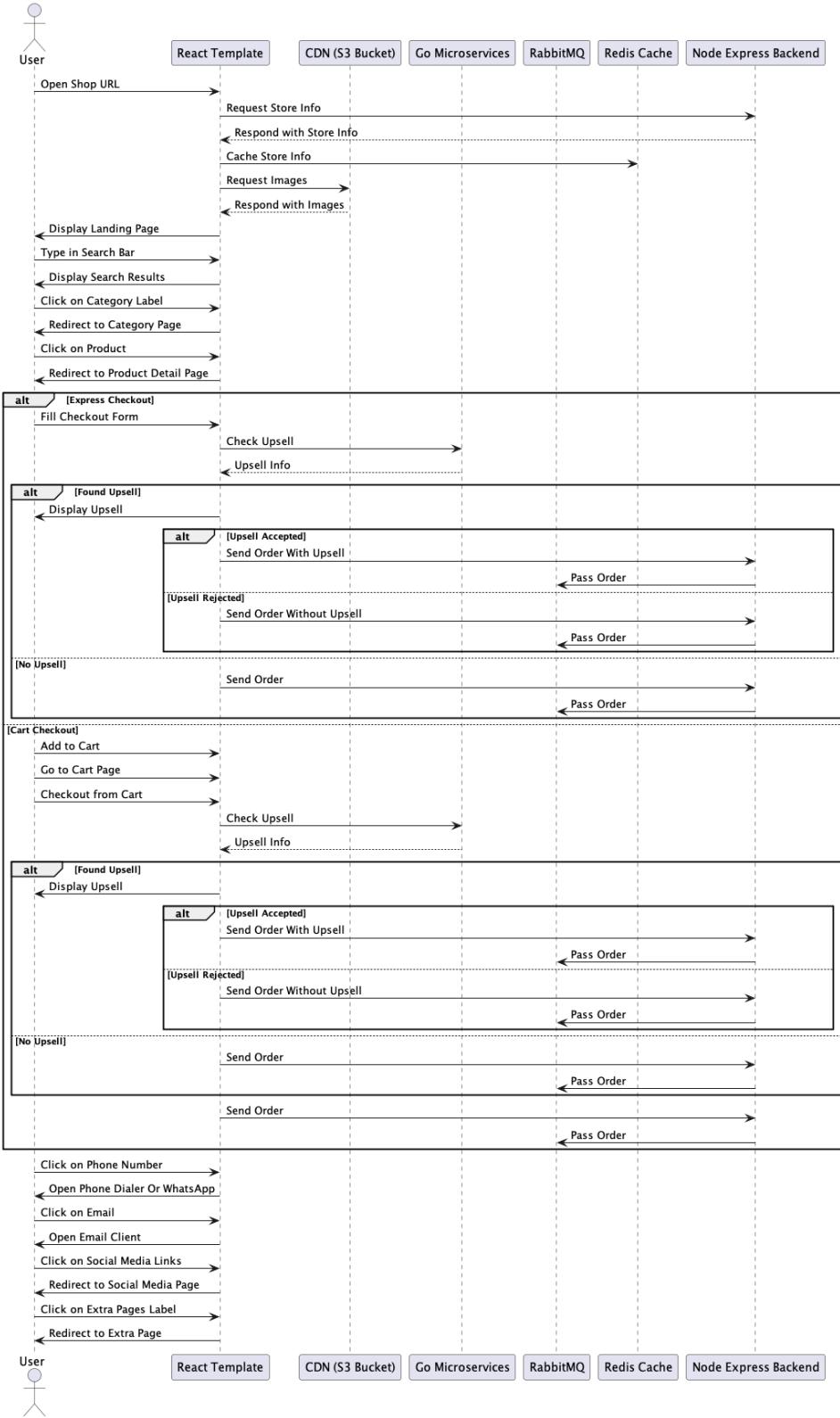


Figure 4.2: Sequence Diagram for Sprint 1

4.2.2.3 User Journey

The user journey for the shop visiting process is presented in Figure 4.3.

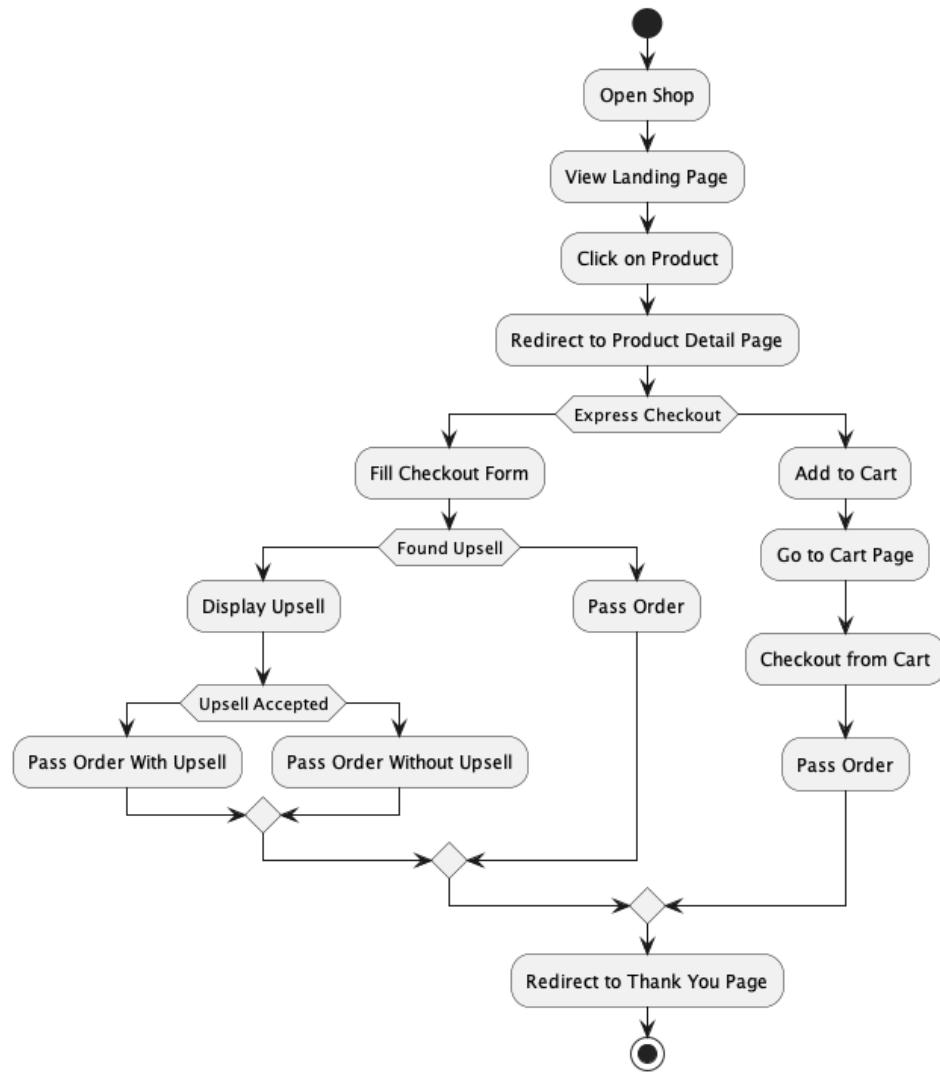


Figure 4.3: User Journey for Sprint 1

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.

4.2.2.4 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 4.4 and 4.5 show the shop landing page.

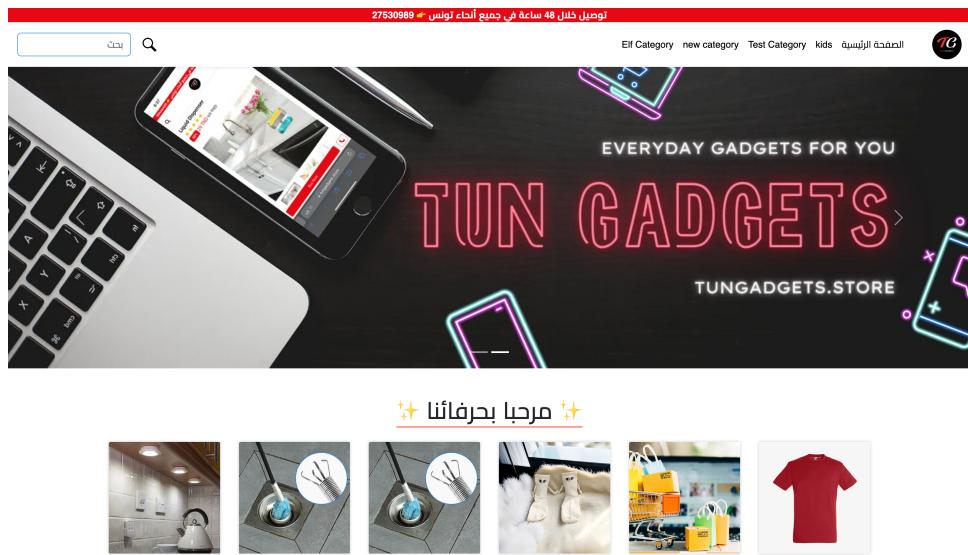


Figure 4.4: Shop Landing Page 1

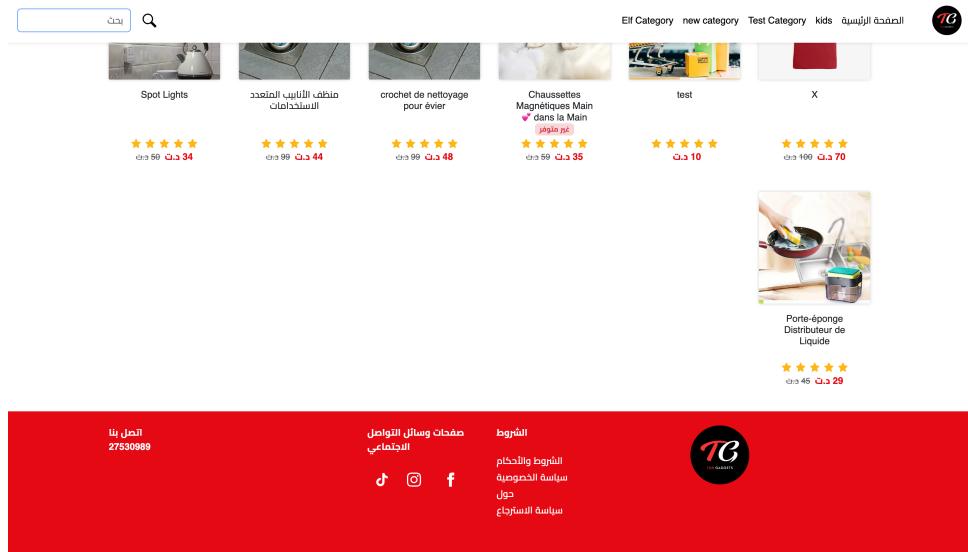


Figure 4.5: Shop Landing Page 2

- **Category Page:** The category page displays a list of products in a specific category. Figure 4.6 shows the category page.

4. Realization - Web Features

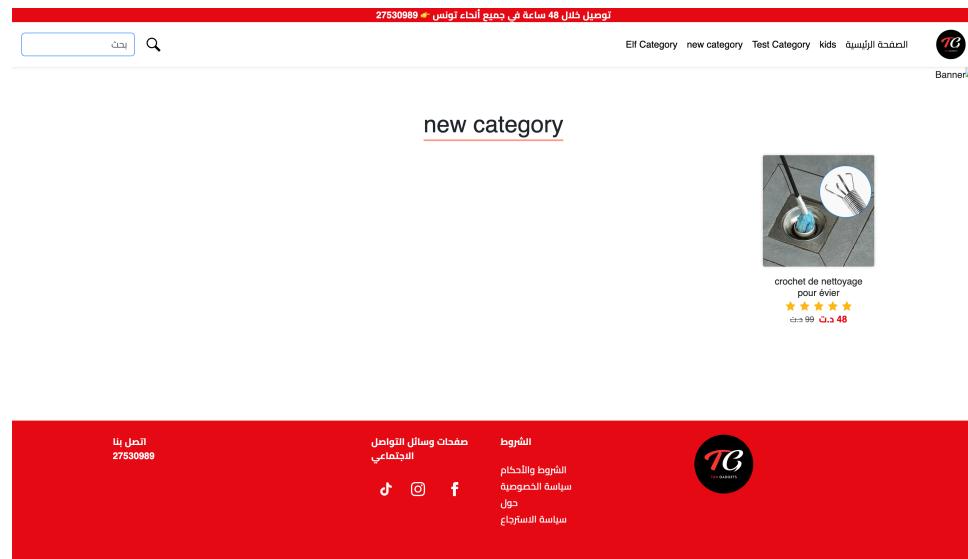


Figure 4.6: Category Page

- **Search Page:** The search page displays a list of products based on the user's search query. Figure 4.7 shows the search page.



Figure 4.7: 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 4.8 shows the extra page.

4. Realization - Web Features



Figure 4.8: Extra Page

- Product Detail Page:** The product detail page displays information about a specific product along with a checkout form and other clients' opinions about it. Figures 4.9, 4.10, and 4.11 show the product detail page.



Figure 4.9: Product Detail Page 1

4. Realization - Web Features

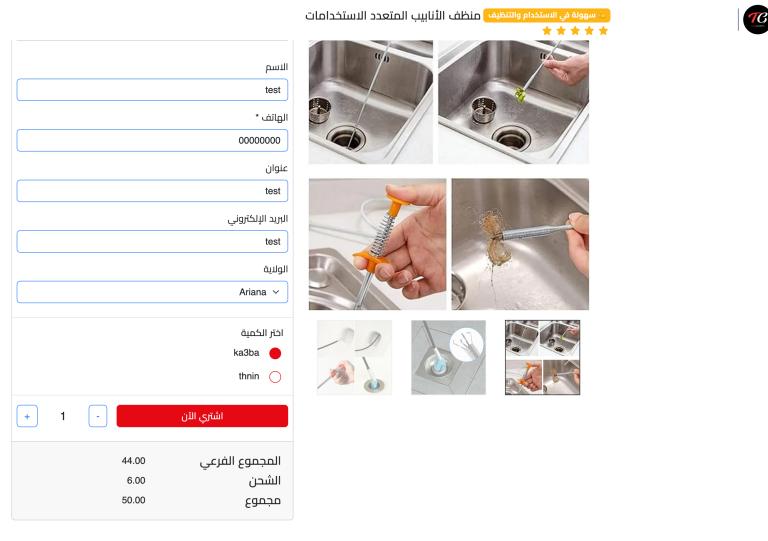


Figure 4.10: Product Detail Page 2



Figure 4.11: Product Detail Page 3

- Thank You Page:** The thank you page is displayed after a successful purchase. Figure 4.12 shows the thank you page and figure 4.13 shows the upsell popup.

4. Realization - Web Features

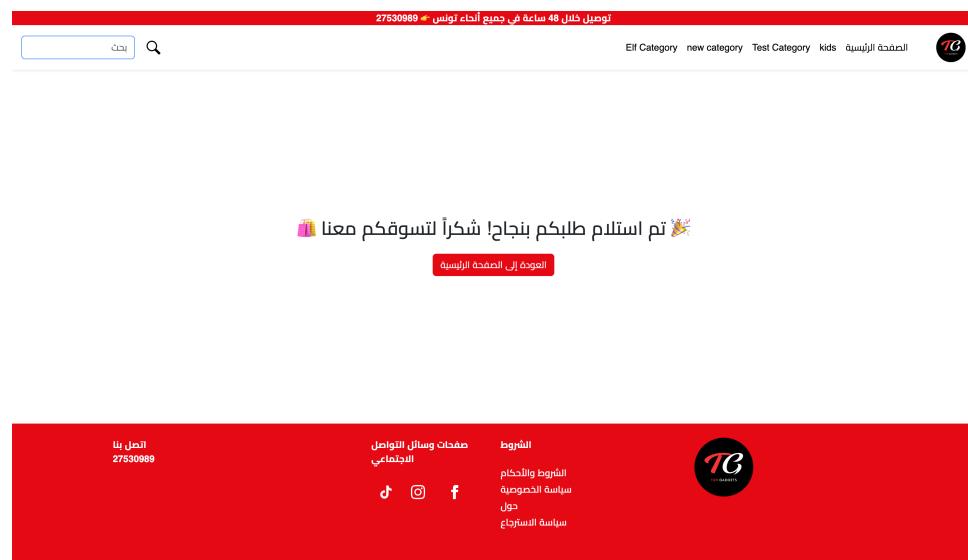


Figure 4.12: Thank You Page

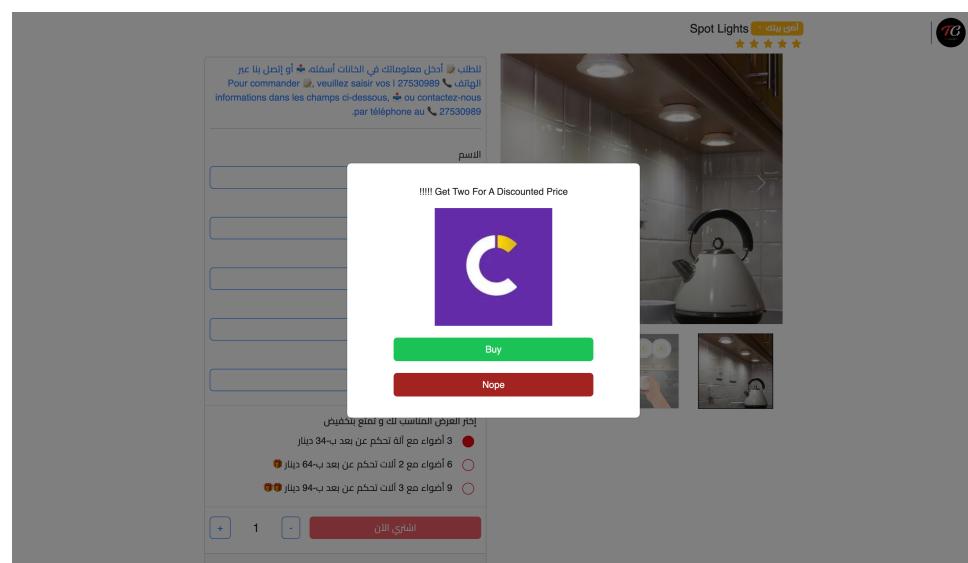


Figure 4.13: Upsell Popup

4.2.2.5 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.

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

4.3.1 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 4.2: Sprint 2 - Implementing the Upsell/Cross-sell Feature

4.3.2 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.

4.3.2.1 Database Schema

The database schema for the upsell/cross-sell feature is shown in Figure 4.14. It consists of the following tables:

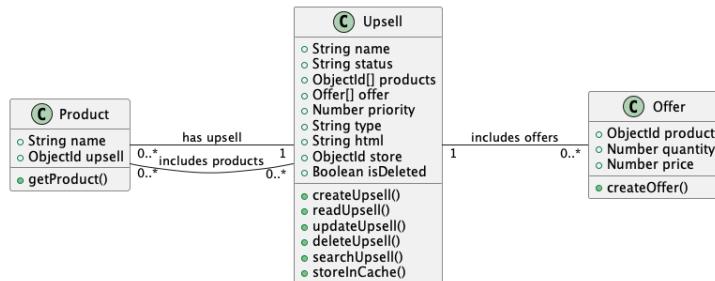


Figure 4.14: Database Schema for Sprint 2

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.

4.3.2.2 Sequence Diagram

- **Search Operation:** The sequence diagram in Figure 4.15 illustrates the interactions between the different components of the search operation. The sequence of events is as follows:

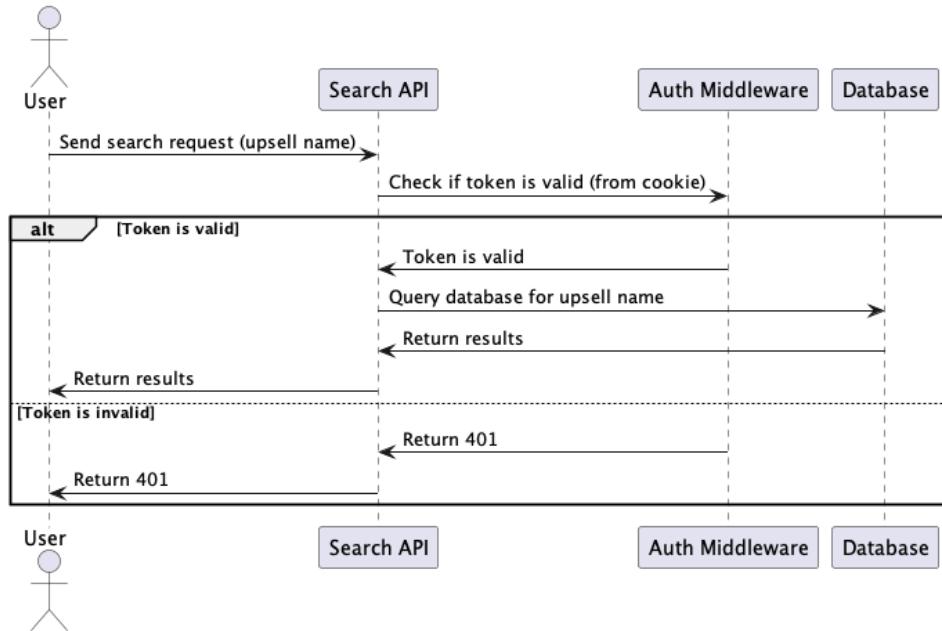


Figure 4.15: Sequence Diagram for Sprint 2 - Search Operation

- **Create Operation:** The sequence diagram in Figure 4.16 illustrates the interactions between the different components of the create operation. The sequence of events is as follows:

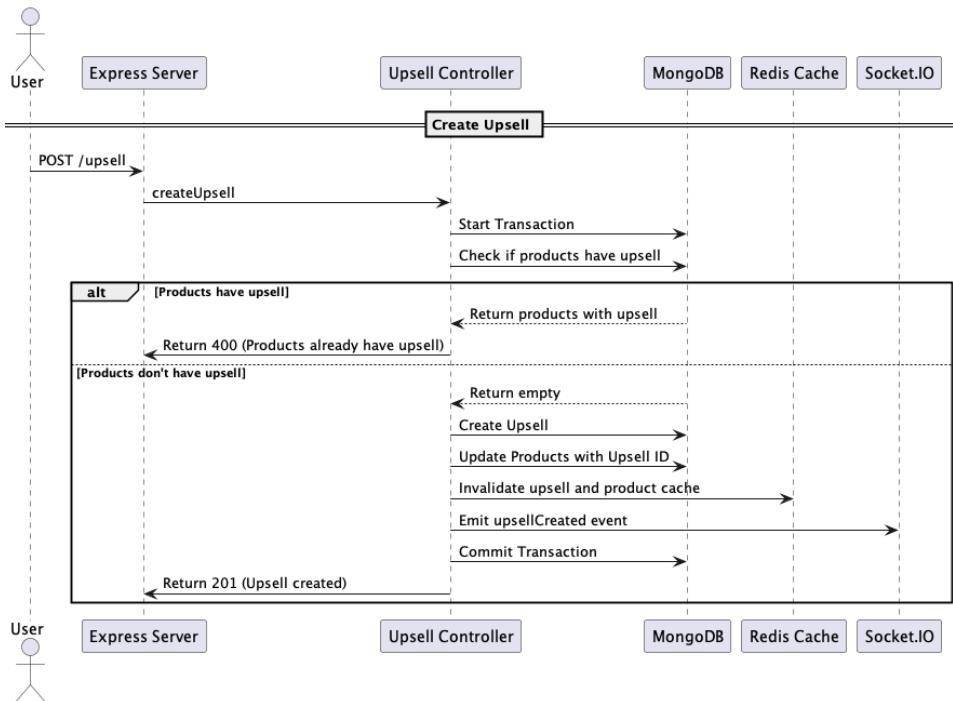


Figure 4.16: Sequence Diagram for Sprint 2 - Create Operation

- **Read Operation:** The sequence diagram in Figure 4.17 illustrates the interactions between the different components of the read all operation. The sequence of events is as follows:

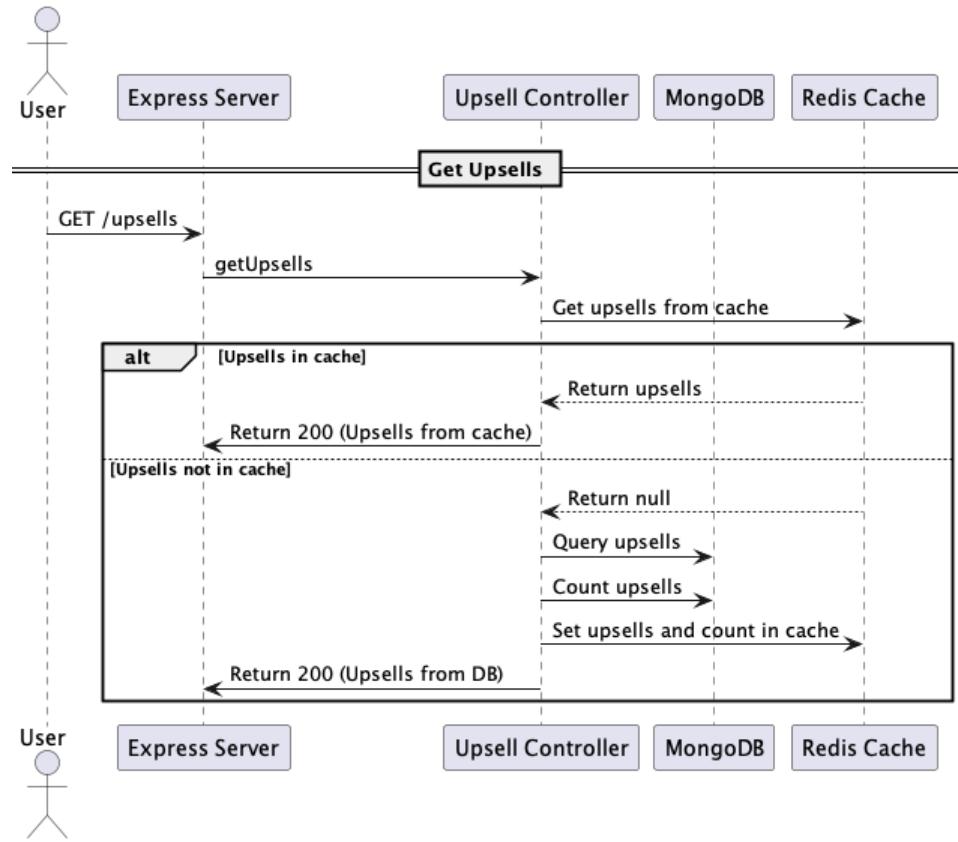


Figure 4.17: Sequence Diagram for Sprint 2 - Read Operation

- **Update Operation:** The sequence diagram in Figure 4.18 illustrates the interactions between the different components of the update operation. The sequence of events is as follows:

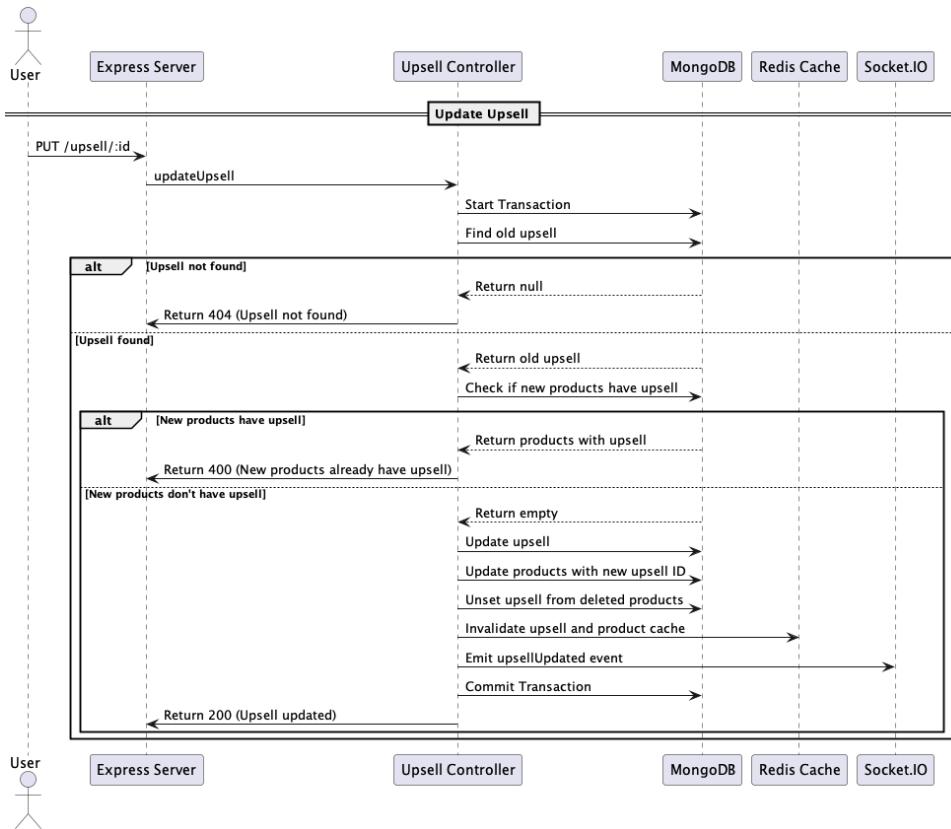


Figure 4.18: Sequence Diagram for Sprint 2 - Update Operation

- **Delete Operation:** The sequence diagram in Figure 4.19 illustrates the interactions between the different components of the delete operation. The sequence of events is as follows:

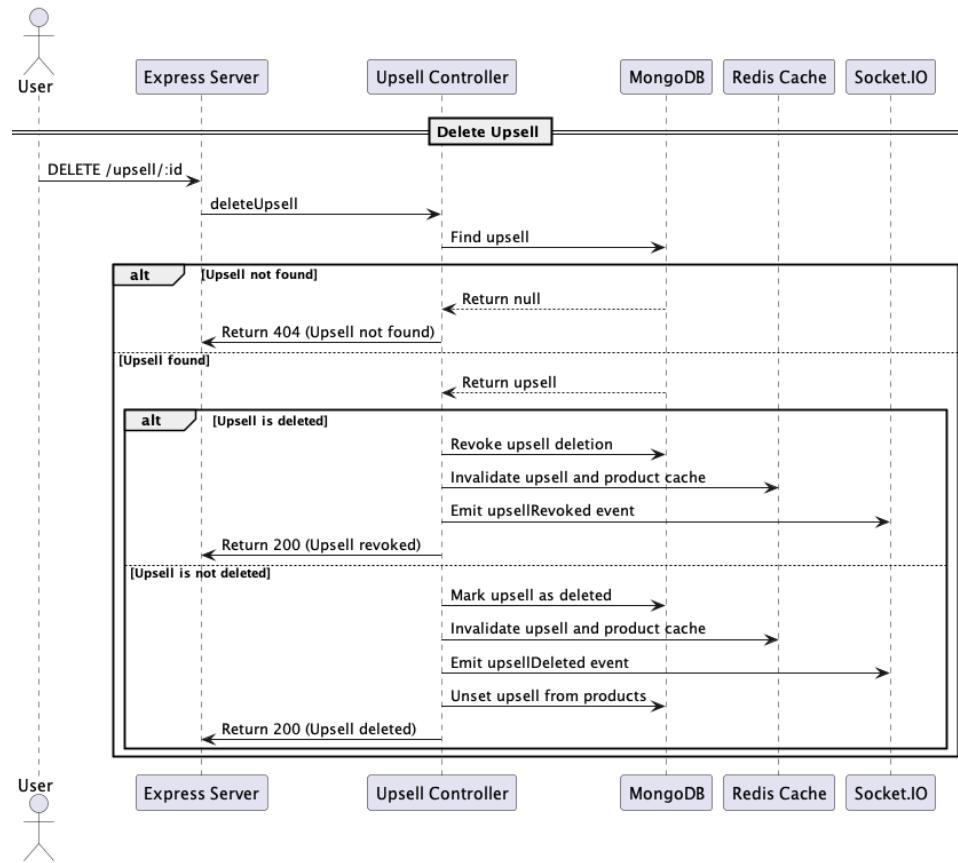


Figure 4.19: Sequence Diagram for Sprint 2 - Delete Operation

- **Read One Operation:** The sequence diagram in Figure 4.20 illustrates the interactions between the different components of the read one operation. The sequence of events is as follows:

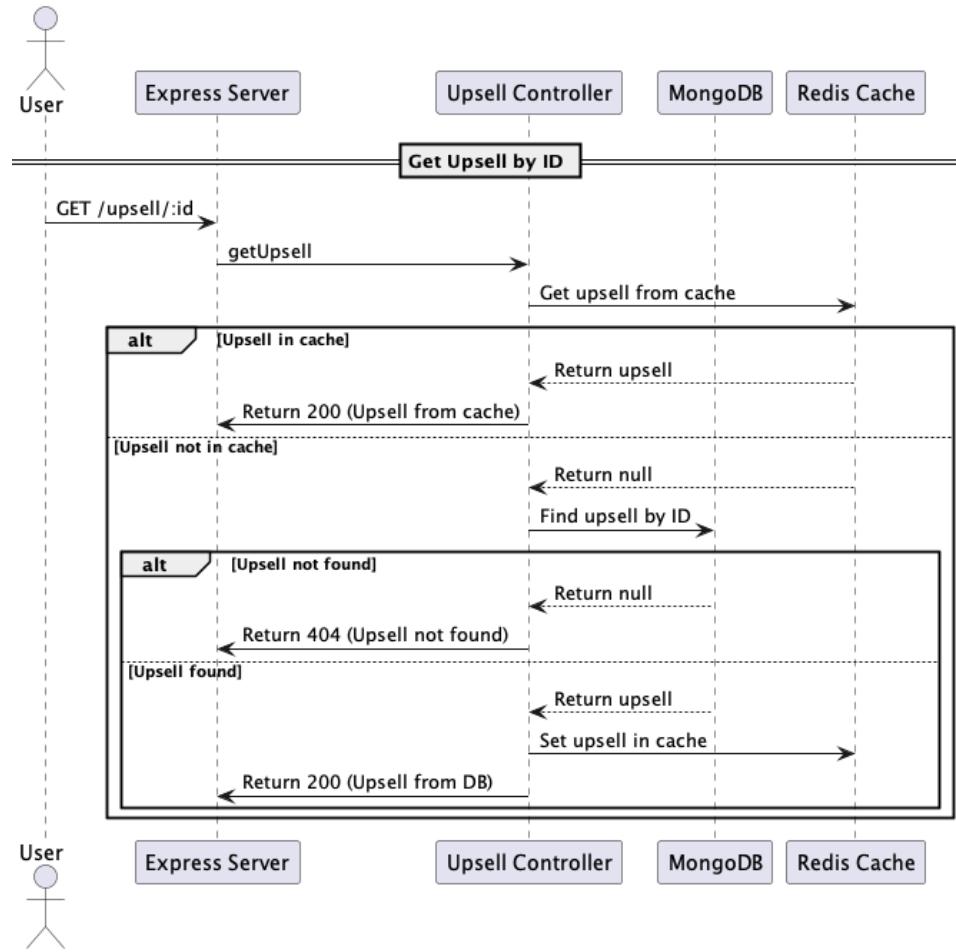


Figure 4.20: Sequence Diagram for Sprint 2 - Read One Operation

4.3.2.3 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.

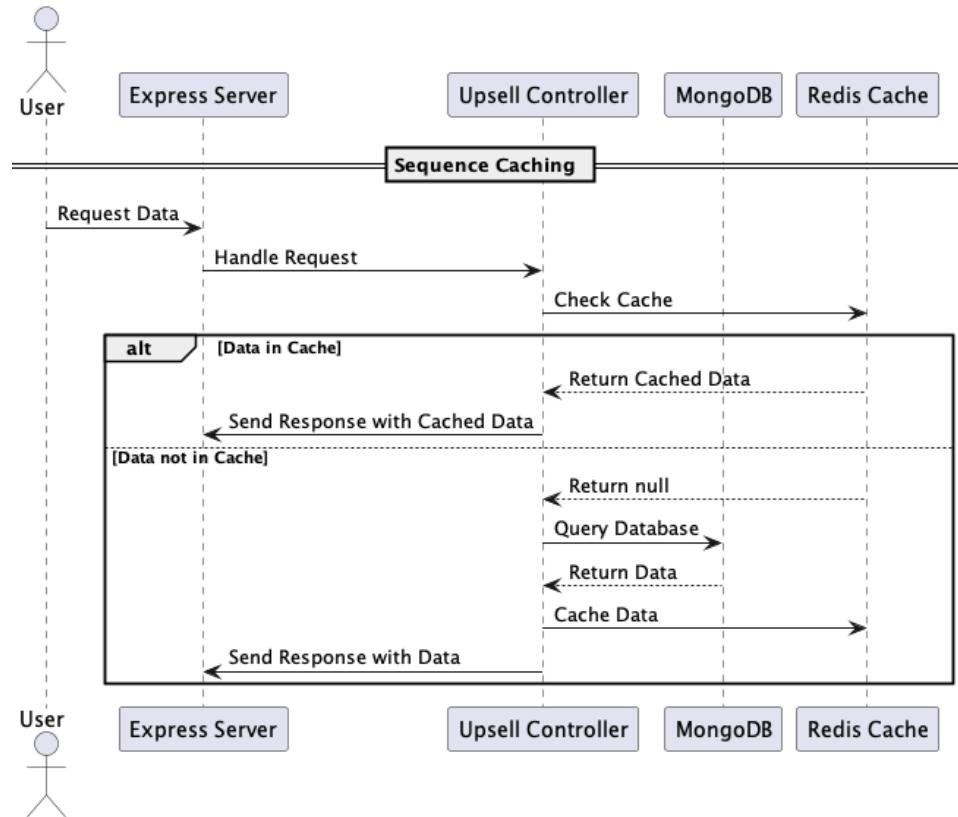


Figure 4.21: Sequence Diagram for Sprint 2 - Caching Layer

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 4.1: 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;
}
```

4.3.2.4 User Journey

The user journey for the upsell/cross-sell feature is presented in Figure 4.22.

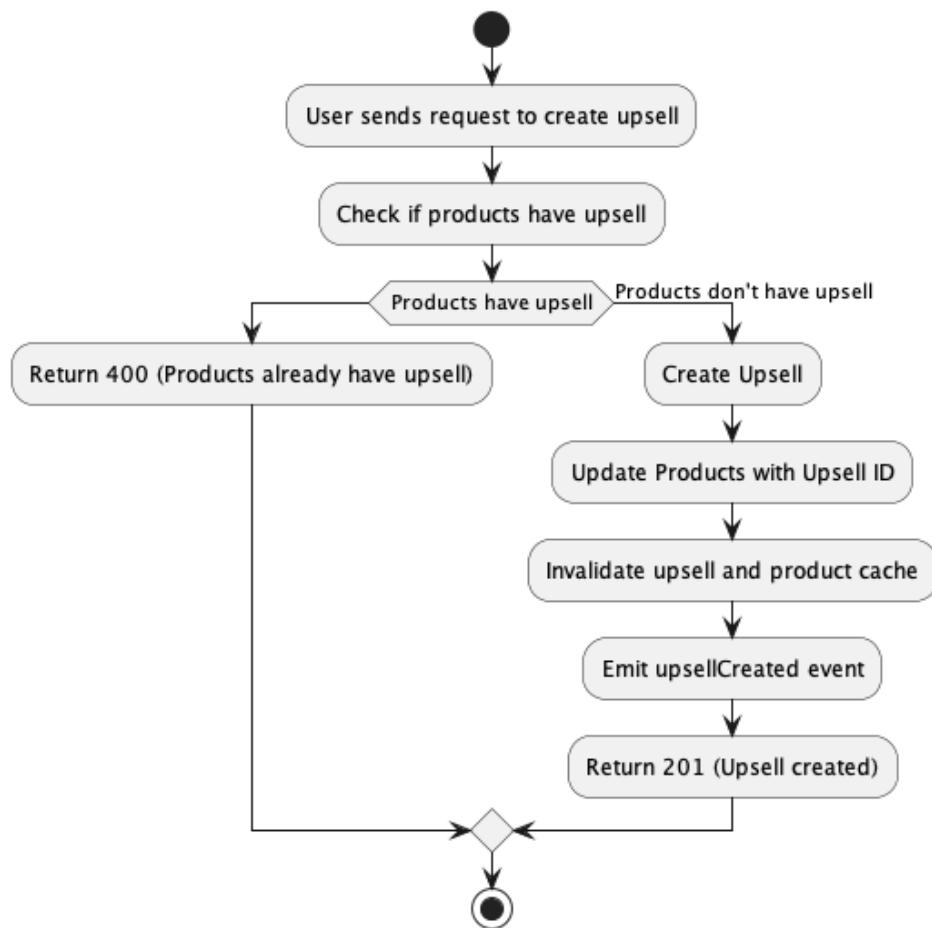


Figure 4.22: User Journey for Sprint 2

4.3.2.5 Screenshots - UI Design

- **Table View Dashboard:** The table view dashboard displays a list of upsells/cross-sells in a tabular format. Figure 4.23 shows the table view dashboard.

4. Realization - Web Features

The screenshot shows a dark-themed web application dashboard for managing upsells. On the left is a sidebar with various menu items: Dashboard, Orders, Products, Up/Cross Sells (which is selected and highlighted in blue), Statistics, Calculator, Budget Manager, Team, Store, and Integrations. The main area is titled "Up/Cross Sells Page". It features a table with columns: Name, Type, Priority, Products, Status, and Actions. There are two rows in the table:

Name	Type	Priority	Products	Status	Actions
Special Offer For Spot Lights	Upsell	0	Spot Lights	Shown	
New Upsell	Upsell	12	Porte-savon Distributeur de Liquide	Shown	

At the bottom of the table, there are buttons for "Rows per page: 12" and navigation arrows. The top right corner shows a user profile with a name, a language switcher (English), and a help icon.

Figure 4.23: Table View Dashboard

- **Upsell/Cross-sell Preview:** The upsell/cross-sell preview displays a preview of the upsell/cross-sell. Figure 4.24 shows the upsell/cross-sell preview.

This screenshot shows the "Up/Cross Sells Page" with a modal window open for a specific upsell entry. The modal has the following details:

Name: Special Offer For Spot Lights
Type: Upsell
Priority: 0
Products: Spot Lights
Status: Shown

The modal also contains a promotional message: "Get Two For A Discounted Price!!!!" followed by a large purple button with a white "C" logo. Below the button are two buttons: "Buy" (green) and "Nope" (red).

At the bottom of the modal, there are buttons for "Rows per page: 12" and navigation arrows. The top right corner shows a user profile with a name, a language switcher (English), and a help icon.

Figure 4.24: 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 4.25 and 4.26 show the edit/add upsell/cross-sell page.

4. Realization - Web Features

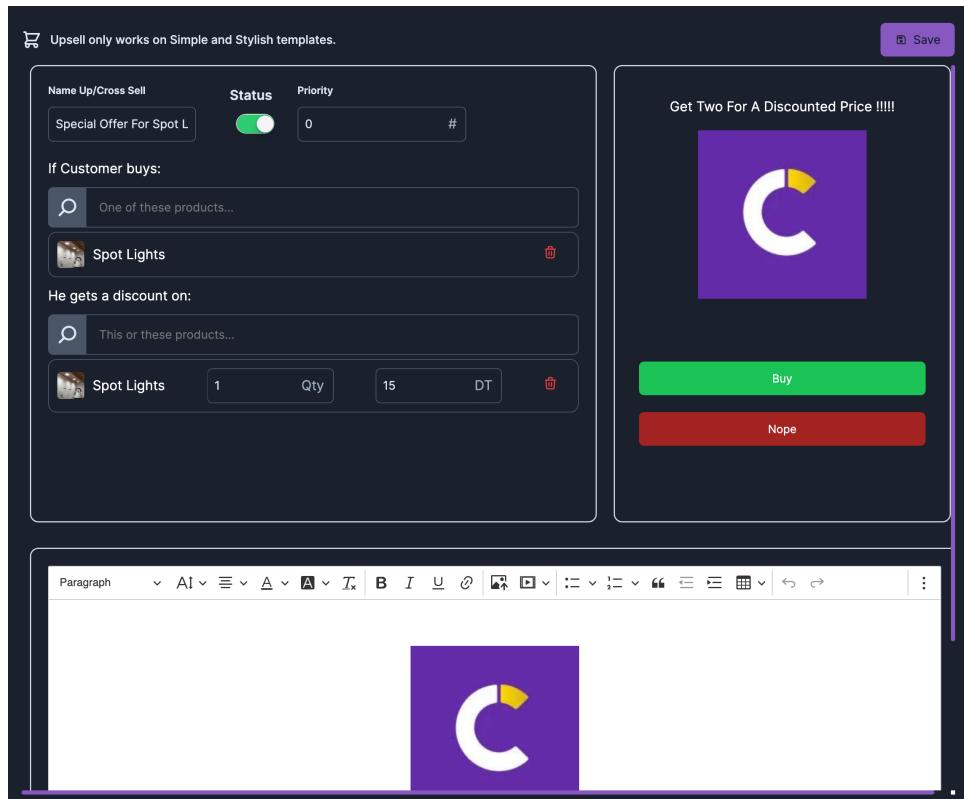


Figure 4.25: Edit/Add Upsell/Cross-sell 1

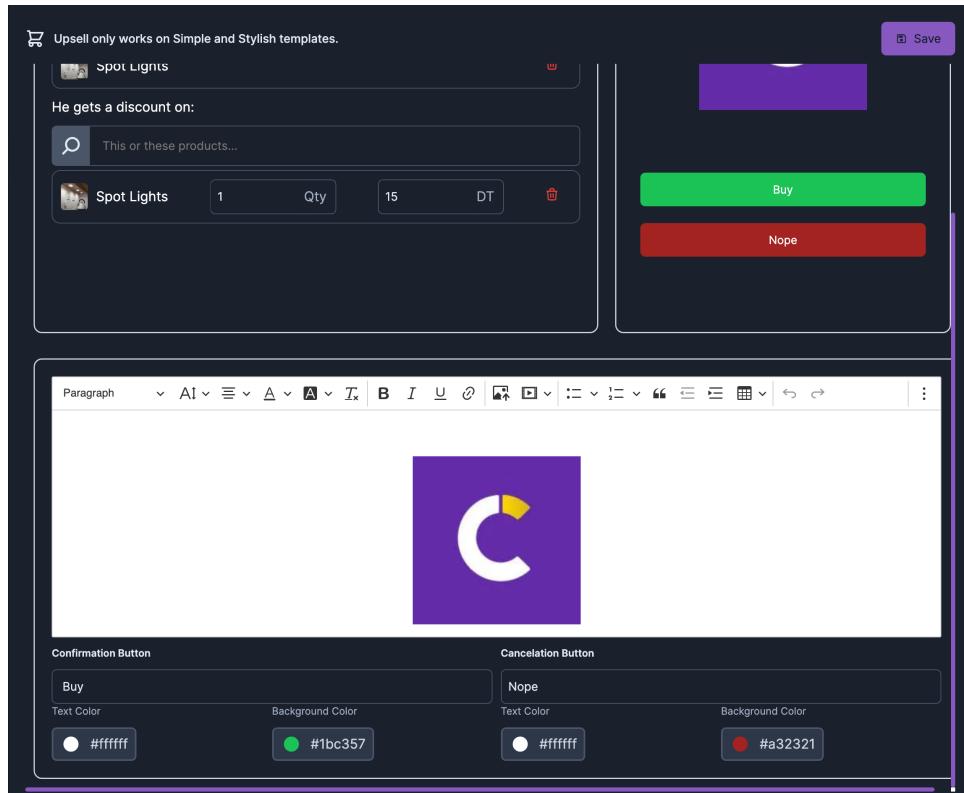


Figure 4.26: Edit/Add Upsell/Cross-sell 2

4.3.2.6 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, including the table view dashboard, upsell/cross-sell preview, and edit/add upsell/cross-sell pages. The implementation was validated through rigorous testing to ensure functionality and performance.

4.4 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.

The first sprint focused on recreating the simple template, which involved designing and implementing a new template to enhance performance and user experience. The second sprint concentrated on implementing the upsell/cross-sell feature, which aimed to increase the average order value by recommending complementary products to customers.

Both sprints were executed using the Agile methodology, ensuring iterative development and continuous feedback. The successful completion of these sprints has significantly improved the functionality and user experience of the Converty platform, making it more competitive in the e-commerce market.

Future work will involve further enhancements and optimizations based on user feedback and evolving market trends. The next steps include expanding the feature set, improving scalability, and ensuring seamless integration with other systems and platforms.

Chapter 5 Realization - Mobile

Features

Summary

5.1	Introduction	65
5.2	Sprint 3: Mobile - Order Management	65
5.3	Sprint 4: Mobile - Budget & Costs Management	70
5.4	Sprint 5: Mobile - Statistics Management	77
5.5	Sprint 6: Mobile - Notifications Center	81
5.6	Conclusion	85

5.1 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.

5.2 Sprint 3: Mobile - Order Management

5.2.1 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 5.1: Sprint 3 - Mobile Order Management

5.2.2 Design & Implementation

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

5.2.2.1 Database Schema

The database schema for the order management feature is shown in Figure 5.1. It consists of the following tables:

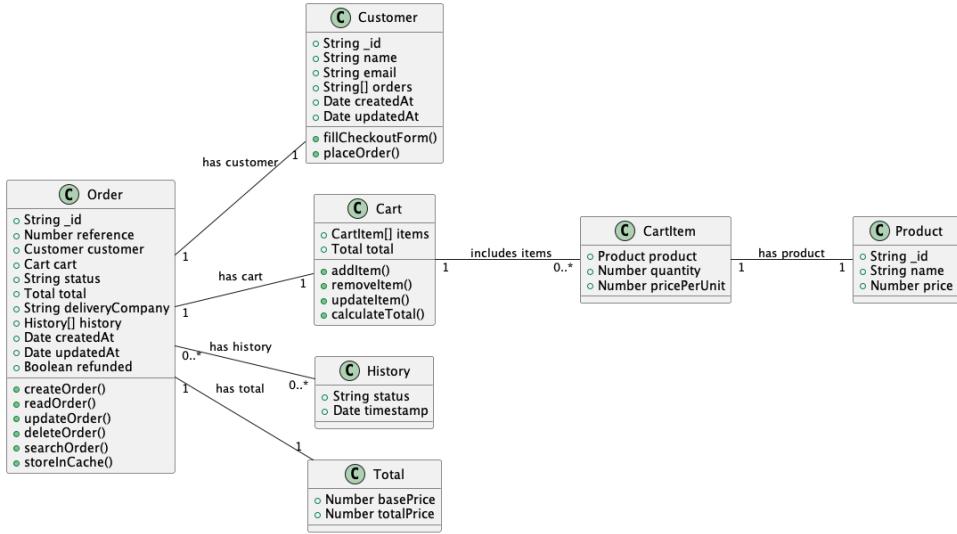


Figure 5.1: Database Schema for Sprint 3

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` is associated with one `Cart`, represented by a one-to-one relationship.
 - Each `Cart` can include multiple `CartItem` instances, representing a one-to-many relationship.

- Each `CartItem` is associated with one `Product`, representing a one-to-one relationship.
- Each `Order` is associated with one `Total`, represented by a one-to-one relationship.
- Each `Order` can have multiple `History` entries, representing a one-to-many relationship.

5.2.2.2 Sequence Diagram

The sequence diagram in Figure 5.2 illustrates the interactions between the frontend and backend components along with key libraries when exploring the order management feature.

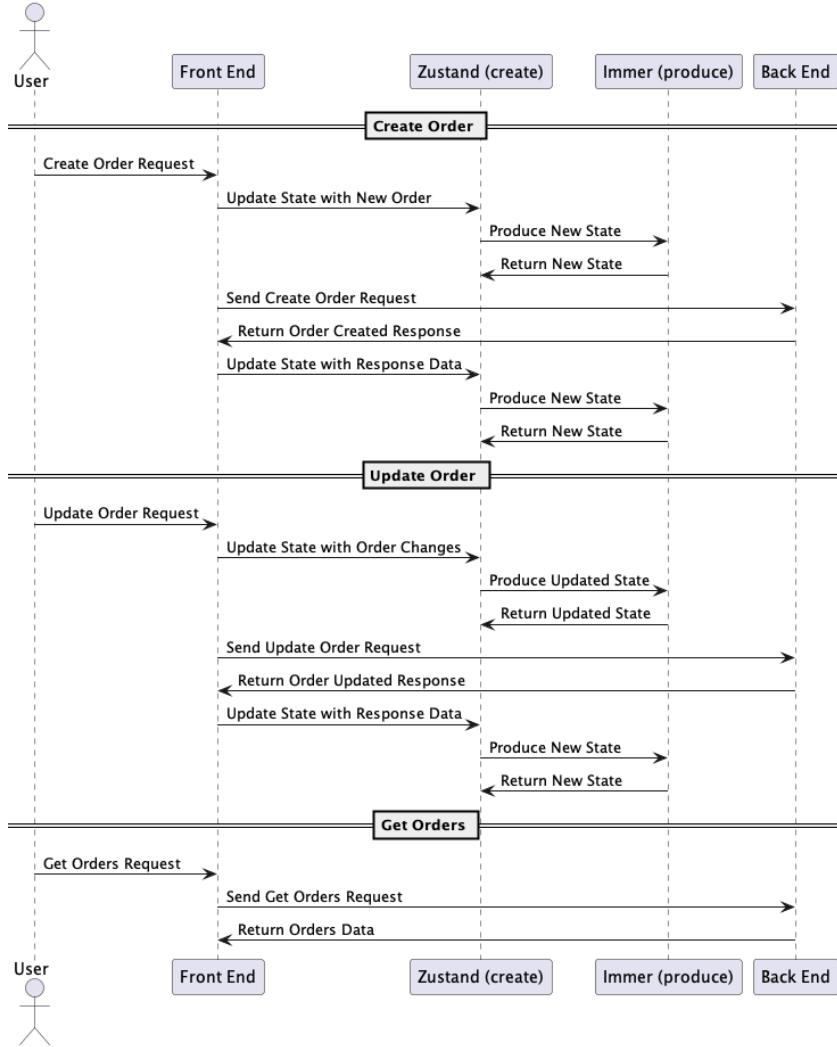


Figure 5.2: Sequence Diagram for Sprint 3

5.2.2.3 Zustand - Immer

- **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 5.1: 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;
```

5.2.2.4 User Journey

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

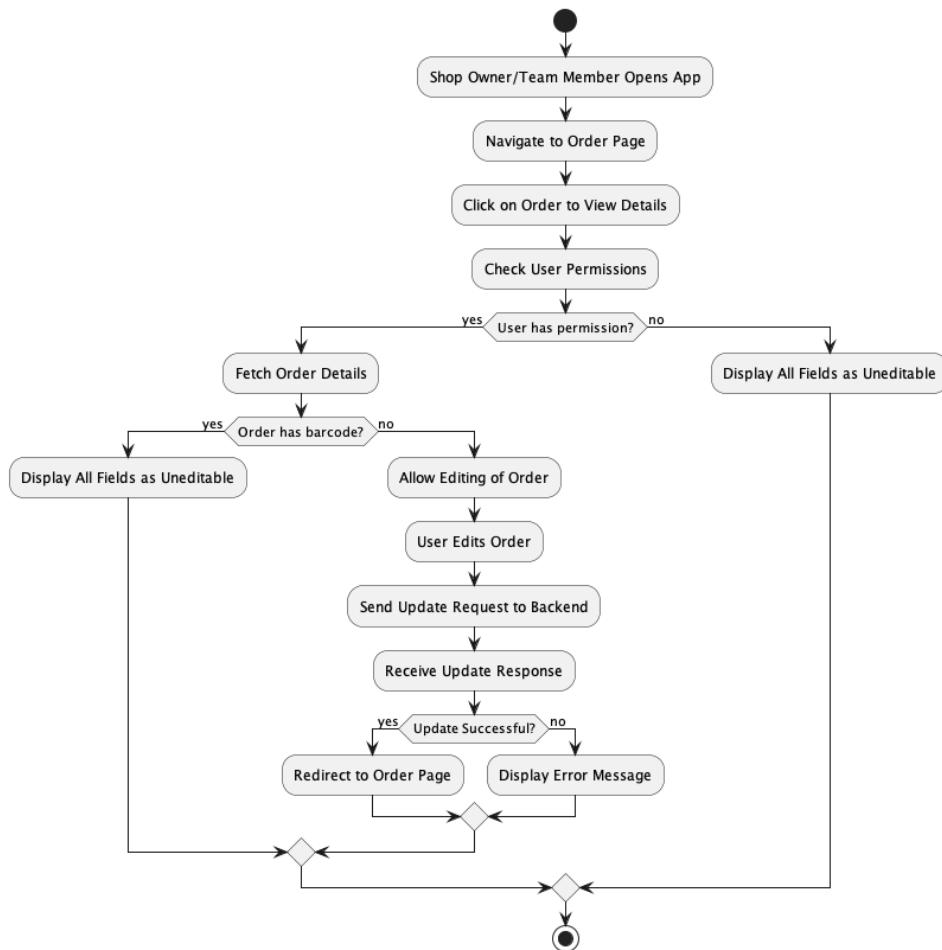


Figure 5.3: User Journey for Sprint 3

5.2.2.5 Screenshots

The screenshots in Figure 5.4 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.

5. Realization - Mobile Features

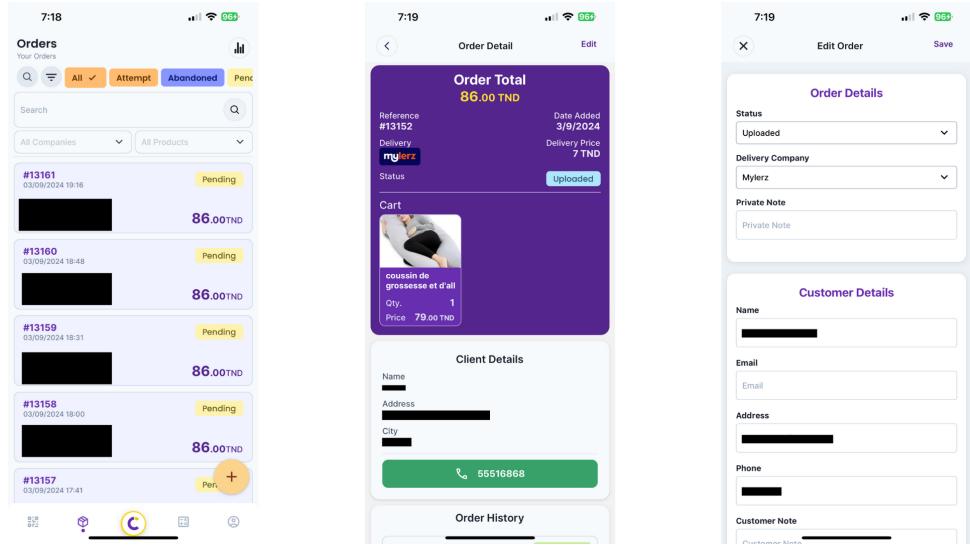


Figure 5.4: Screenshots for Sprint 3

5.2.2.6 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.

5.3 Sprint 4: Mobile - Budget & Costs Management

5.3.1 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 5.2: Sprint 4 - Mobile Budget & Costs Management

5.3.2 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.

5.3.2.1 Database Schema

The database schema for the budget and costs management feature is shown in Figure 5.5. It consists of the following tables:

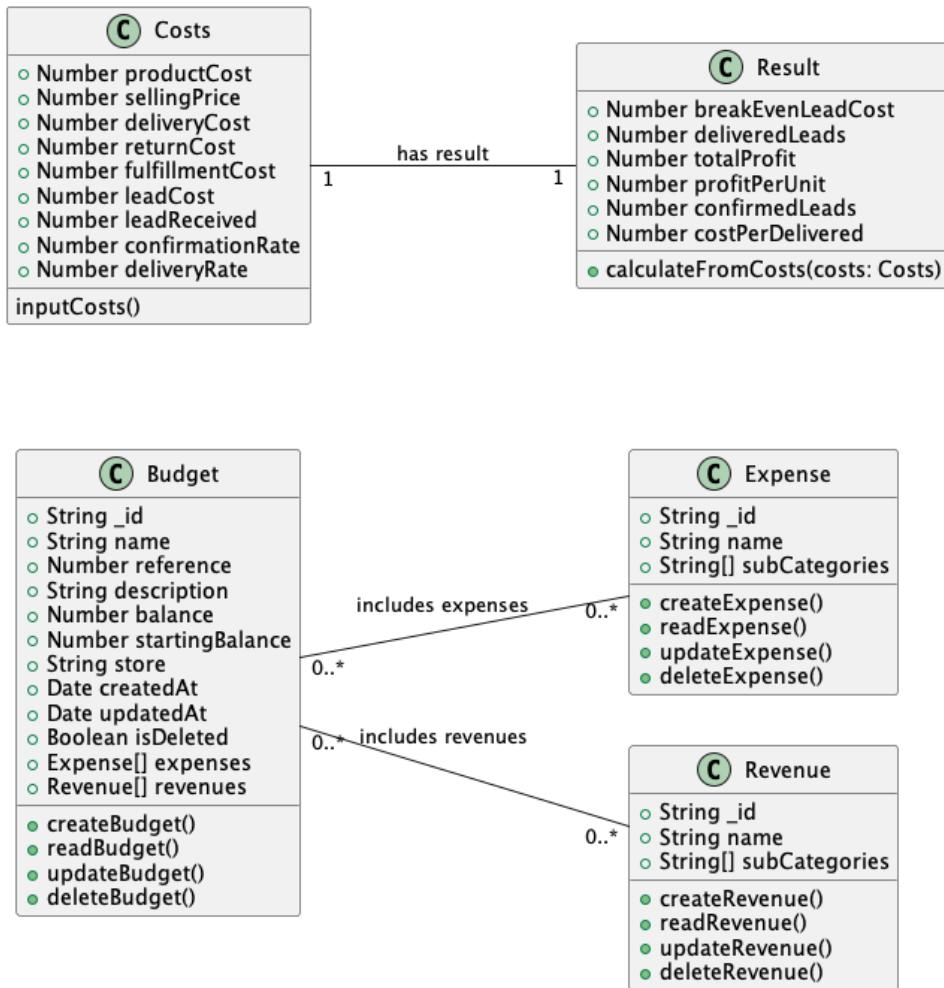


Figure 5.5: Database Schema for Sprint 4

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.
 - `Costs` has a one-to-one relationship with `Result`, indicating that costs are used to calculate a single result.

5.3.2.2 Sequence Diagram

The sequence diagram in Figure 5.6 illustrates the interactions between the frontend and backend components when exploring the budget and costs management feature.

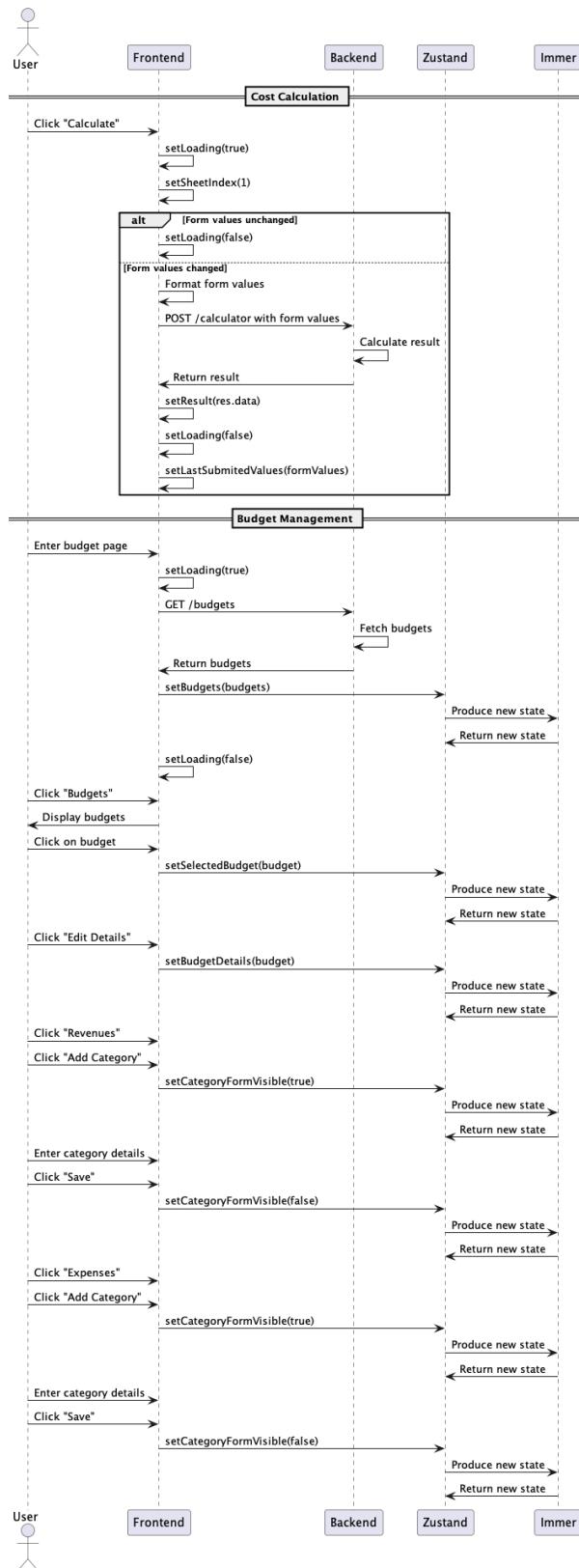


Figure 5.6: Sequence Diagram for Sprint 4

5.3.2.3 User Journey

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

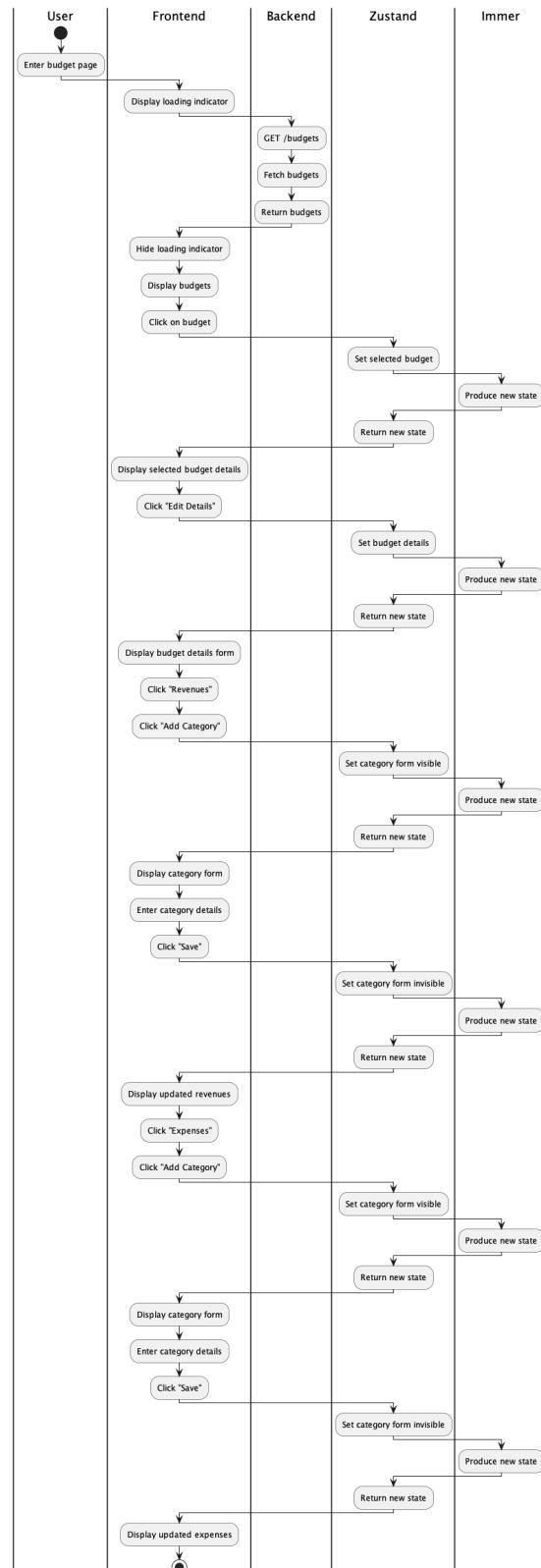


Figure 5.7: User Journey for Sprint 4

5.3.2.4 Screenshots

The screenshots in Figures 5.8, 5.9, and 5.10 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.

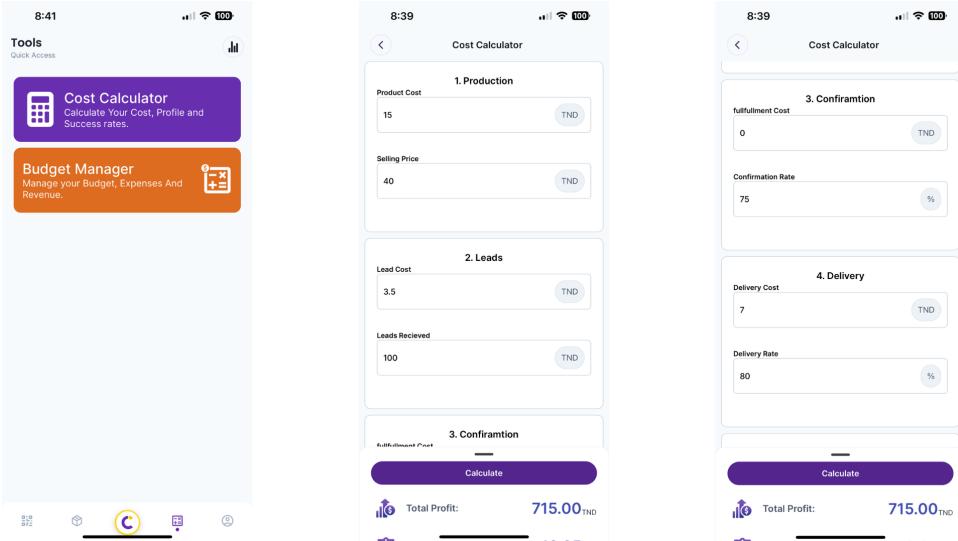


Figure 5.8: Screenshots for Figure 1

5. Realization - Mobile Features

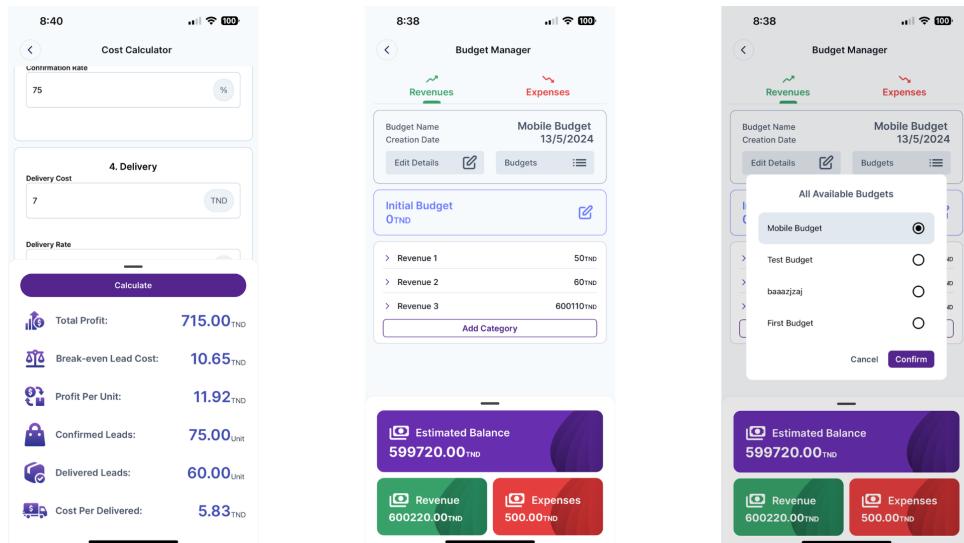


Figure 5.9: Screenshots for Figure 2

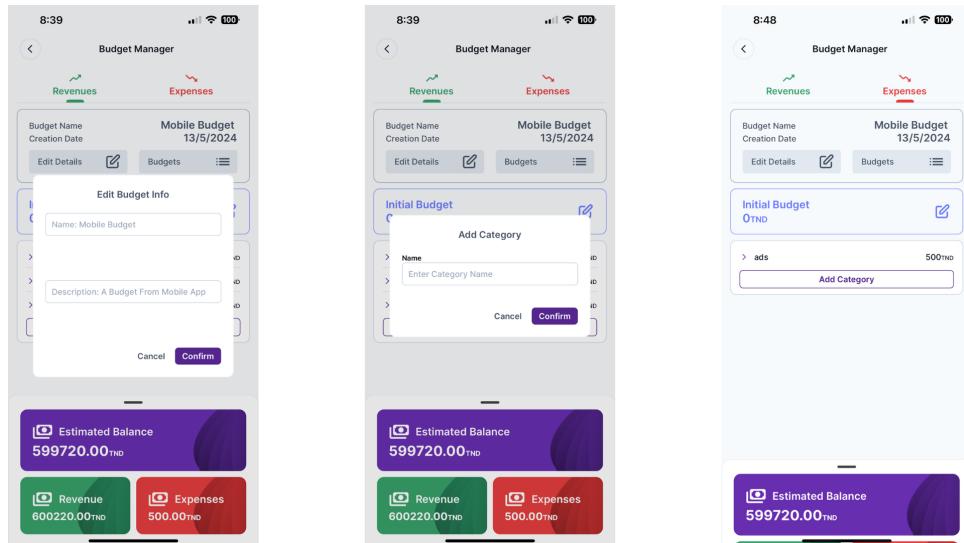


Figure 5.10: Screenshots for Figure 3

5.3.2.5 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.

5.4 Sprint 5: Mobile - Statistics Management

5.4.1 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 5.3: Sprint 5 - Statistical Analysis

5.4.2 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.

5.4.2.1 Database Schema

The database schema for the statistics management feature is shown in Figure 5.11. It consists of the following tables:

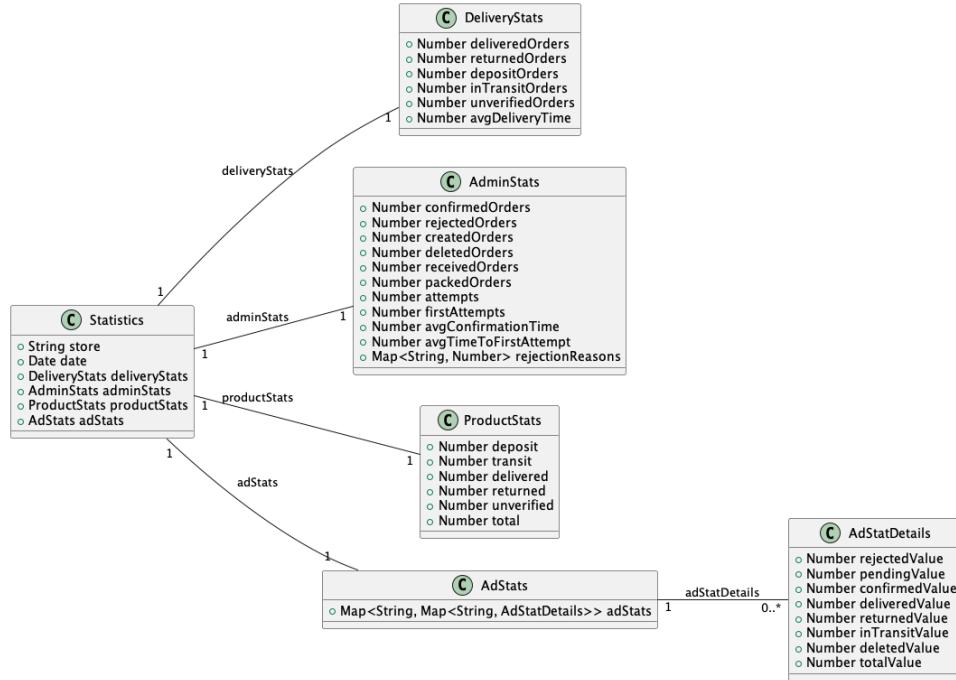


Figure 5.11: Database Schema for Sprint 5

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.
- **adStats (Object)**: Contains advertising statistics, including various metrics related to ad performance.

5.4.2.2 Sequence Diagram

The sequence diagram in Figure 5.12 illustrates the interactions between the frontend and backend components when exploring the statistics management feature.

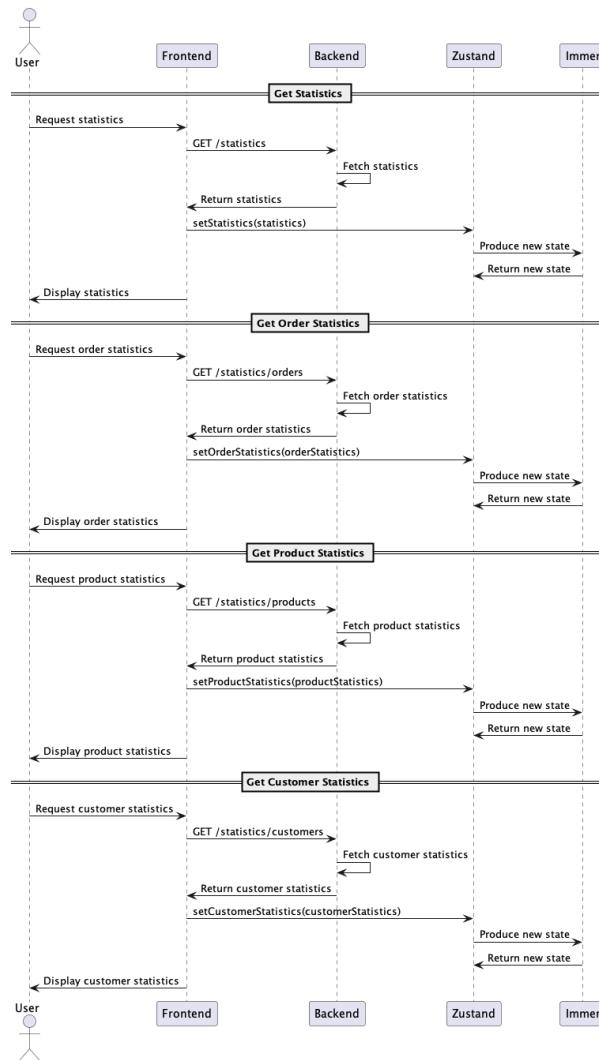


Figure 5.12: Sequence Diagram for Sprint 5

5.4.2.3 User Journey

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

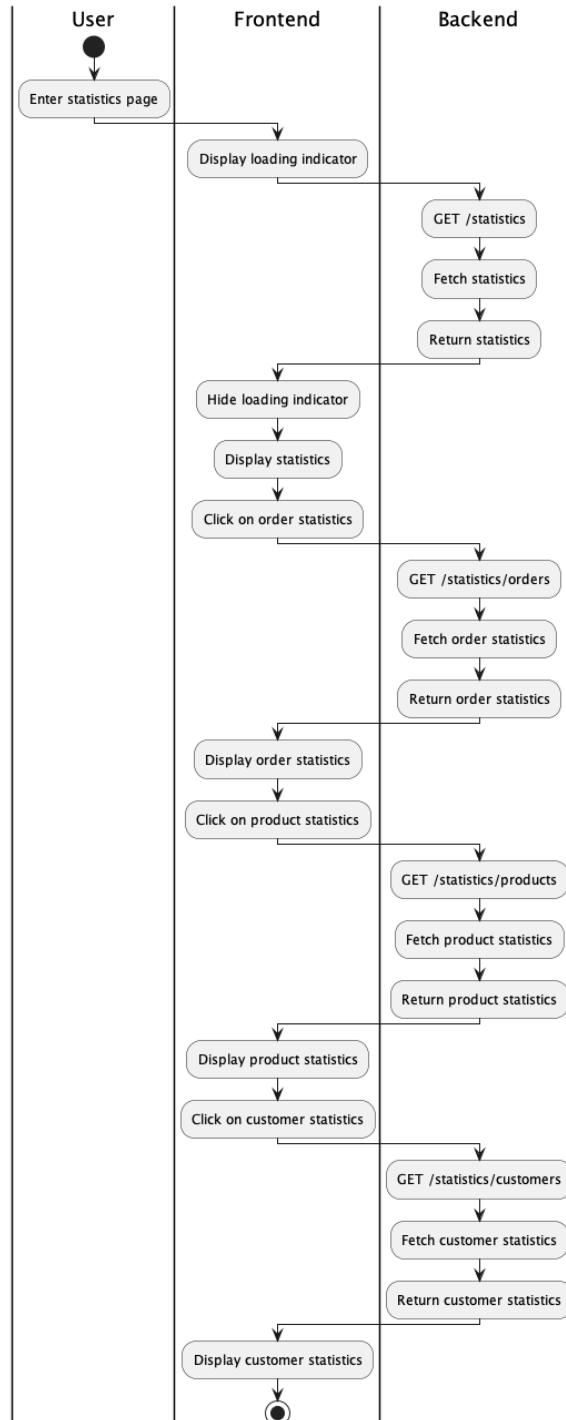


Figure 5.13: User Journey for Sprint 5

5.4.2.4 Screenshots

The screenshots in Figure 5.14 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.

5. Realization - Mobile Features

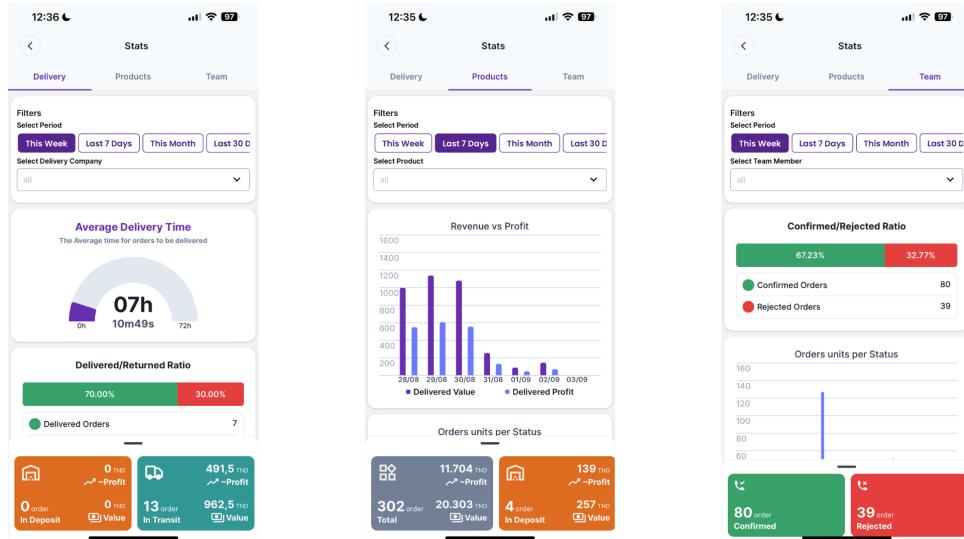


Figure 5.14: Screenshots for Sprint 5

5.4.2.5 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.

5.5 Sprint 6: Mobile - Notifications Center

5.5.1 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 5.4: Sprint 6 - Notifications Center

5.5.2 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).

5.5.2.1 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 5.2: 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 5.3: 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;
```

5.5.2.2 User Journey

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

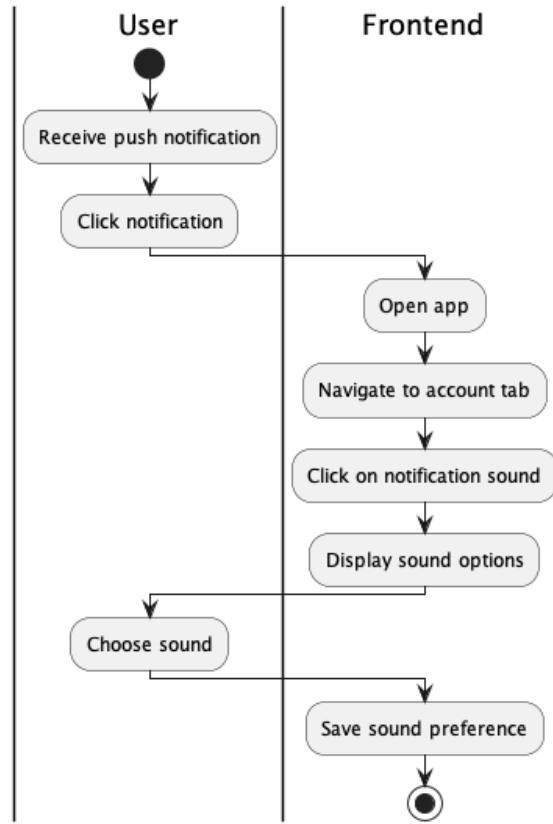


Figure 5.15: User Journey for Sprint 6

5.5.2.3 Screenshots

The screenshots in Figure 5.16 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.

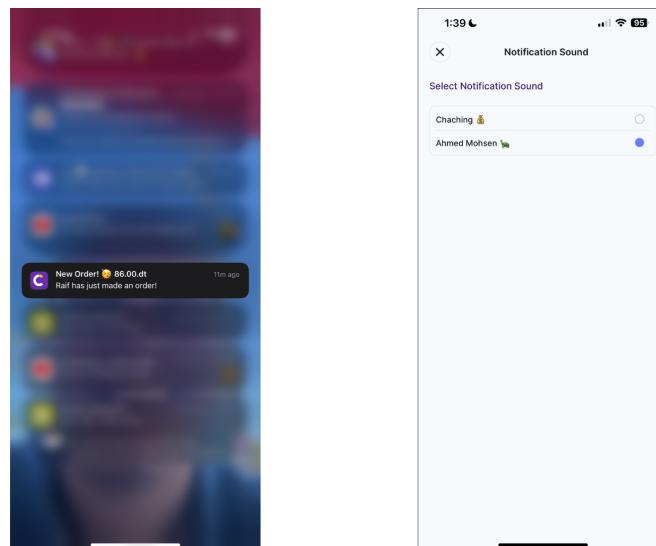


Figure 5.16: Screenshots for Sprint 6

5.5.2.4 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.

5.6 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 6 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.

Bibliography

- [1] Shopify. *What is Shopify?*. Available at: <https://www.shopify.com/blog/what-is-shopify>
- [2] LTS Group. *SDLC - Software Development Life Cycle*. Available at: <https://ltsgroup.tech/blog/sdlc-software-development-life-cycle/>
- [3] Aymen Chaqrourf. *Comment l'utilisation d'une méthodologie de gestion des projets Agile peut améliorer la performance de votre équipe*. Available at: <https://www.linkedin.com/pulse/comment-lutilisation-dune-m%C3%A9thodologie-de-gestion-des-aymen-chaqrourf/>
- [4] Online Gantt. *Online Gantt Chart Tool*. Available at: <https://www.onlinegantt.com/#/gantt>
- [5] Node.js. *Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts*. Available at: <https://nodejs.org/en>
- [6] Express.js. *Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications*. Available at: <https://expressjs.com>
- [7] FreeCodeCamp. *What is Go Programming Language?*. Available at: <https://www.freecodecamp.org/news/what-is-go-programming-language/#heading-whats-the-name-go-or-golang>
- [8] CloudAMQP. *RabbitMQ for Beginners - What is RabbitMQ?*. Available at: <https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html#:~:text=RabbitMQ%20is%20a%20message%20queueing,%20any%20kind%20of%20information.>
- [9] Dev.to. *Best Practices for Using TypeScript in React with Vite*. Available at: <https://dev.to/oppaaaaii/best-practices-for-using-typescript-in-react-with-vite-1dhf>
- [10] Wikipedia. *React Native*. Available at: https://en.wikipedia.org/wiki/React_Native
- [11] Expo. *Expo GitHub Repository*. Available at: <https://github.com/expo/expo/blob/main/README.md>

BIBLIOGRAPHY

- [12] Zustand. *Getting Started - Introduction*. Available at: <https://zustand.docs.pmnd.rs/getting-started/introduction>
- [13] MongoDB. *What is MongoDB?*. Available at: <https://www.mongodb.com/company/what-is-mongodb>
- [14] FreeCodeCamp. *Introduction to Mongoose for MongoDB*. Available at: <https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/>
- [15] Redis. *About Redis*. Available at: <https://redis.io/about/>
- [16] Converty. *About Converty*. Available at: <https://www.converty.com/about>