MinarMarket

SPROJ Report

Abdul Ahad Bin Ali 25100016
Hasan Malik 25100211
Muhammad Umer Jamil 25100017

Saad Ilyas 25100181
Aniqa Aqeel 25100257

Advisor: Dr. Waqar Ahmed
School of Science and Engineering
Lahore University of Management Sciences
Submission Date

Acknowledgement and Dedication

Certificate

I certify that the senior project titled "**MinarMarket**" was completed under my supervision by the following students:

_____

_____

_____

and the project deliverables meet the requirements of the program.

---------- Date:

Advisor (Signature)

---------- Date:

Co-advisor (if any)

Table of Contents

List of Figures

1. Introduction

a. Introduction

This project reimagines the traditional e-commerce model by introducing a platform that fosters a more collaborative relationship between buyers and sellers. In conventional marketplaces, sellers list products while buyers browse to make purchases, which can limit options for buyers with specific needs. Our solution allows buyers to post unique requests, encouraging sellers to respond with tailored offerings that meet these demands. This approach transforms the marketplace into a more interactive ecosystem, reducing the gap between supply and demand and ensuring that buyers find products closely aligned with their preferences.

The platform's primary goal is to enhance the traditional e-commerce experience by giving buyers the freedom to list products they seek while providing sellers visibility into these requests. This two-way interaction enables sellers to make targeted offers or negotiate terms, creating a more responsive, transparent, and efficient marketplace. By featuring an intuitive user interface, the platform simplifies buyer-seller communication, allowing buyers to track offers, compare sellers, and make well-informed decisions based on personalized options. Meanwhile, sellers receive real-time notifications of buyer requests that match their inventory, facilitating quick responses to meet demand.

Designed with scalability and flexibility in mind, the platform targets a diverse audience, including individual consumers, small businesses, and larger enterprises. Individual buyers can request specific items, and businesses can source bulk or niche orders. This model serves niche markets where product availability is often limited, giving sellers access to a highly motivated customer base. As the platform grows, future features like AI-driven product matching will streamline offer-making, while integration with payment gateways, shipment tracking, and review systems will enhance the overall user experience.

Domain of the Application:
This project falls within the **e-commerce and digital marketplace** domain. It focuses on innovating buyer-seller interactions and providing a flexible, demand-driven online shopping experience.

Target Users of the Application:
The primary users include **individual consumers** looking for specific products, **small businesses** seeking customized or bulk orders, and **larger enterprises** interested in sourcing specialized inventory. Both buyers who want personalized shopping experiences and sellers aiming to target customer-specific needs are the core audiences for the platform.

What Has Been Produced:
As the final product, a **web application** has been developed. The platform provides users with a responsive and intuitive interface accessible through desktop browsers, facilitating seamless buyer and seller interactions.

By empowering buyers and streamlining the seller's role in meeting demand, this marketplace aims to set a new standard in digital commerce. It bridges gaps between buyer needs and seller offerings, encouraging higher transaction success rates and fostering stronger buyer-seller relationships. This project ultimately seeks to redefine the e-commerce experience, making it more interactive, efficient, and buyer-driven, catering to the demands of modern consumers seeking personalization and convenience.

b. Objective and Scope

Objectives:

The objective of this project is to develop a buyer-centric e-commerce web application where buyers can post specific product requests and sellers can respond with customized offers. The platform aims to create a more interactive, personalized, and efficient marketplace experience by improving communication between buyers and sellers.

Reason for Development:

This application was developed to address the limitations of traditional marketplaces where buyers are restricted to existing product listings. By empowering buyers to define their needs, the platform offers a more flexible and dynamic shopping experience.

Impact on Business Operations:

The platform enhances business operations by helping sellers better align their inventory with real-time demand, improving buyer satisfaction through personalization, and fostering stronger, trust-based relationships in the marketplace.

c. Development Methodology

The development of the project followed a **modular and iterative methodology**. The system was divided into distinct modules such as buyer requests, seller offers, and notifications, which were developed. An **iterative approach** was adopted, where features were built incrementally

with continuous testing and refinement after each stage, ensuring flexibility to incorporate feedback and improve functionality throughout the development cycle.

The frontend was developed using **Next.js**, enabling server-side rendering and optimized performance for a seamless user experience. The backend was built with **Node.js**, supporting efficient server-side operations and API handling. For the database, **MongoDB** was used, providing a scalable and flexible solution to manage user and product data.

By combining a modular-iterative process with a modern technology stack, the project achieved better maintainability, scalability, and a smooth development workflow suited for a dynamic e-commerce platform.

d. Contributions

This project introduces a **buyer-driven e-commerce platform**, allowing buyers to post specific product requests and receive tailored offers from sellers. Unlike traditional marketplaces focused on seller listings, this approach increases personalization, speeds up transactions, and improves buyer satisfaction.

Innovative features include real-time seller notifications, simplified buyer-seller communication, and enhanced negotiation capabilities. By directly connecting supply to demand, the platform offers a more interactive, efficient, and user-centric alternative to existing e-commerce models.

2. System Requirements

Brief introduction of this chapter in a paragraph highlighting the content

a. System Actors

Actor Name	Description
Seller (Services)	This user is one of the two types of sellers on this platform. This user will sell his/her services at a per-hour rate. This user can also consider a better offer, either on the lower or higher ends.
Seller (Products)	This is the second type of seller. This user will sell something tangible (i.e., a product) for a fixed price. This user can also consider some flexibility on the price.
Customer (Services)	This user is one of the two types of customers on this platform. In addition to buying the services from the seller, this user can also list the services needed independently. This type of user will provide a range of their budget, i.e., a low and a high-end. The sellers can approach these customers directly with a better offer.

Customer (Goods)	This is the second type of customer. This type of user can either buy from the listings posted by sellers or list for specifications and a budget for the product they are looking for. This type of user can provide a budget range or a fixed amount. Sellers can approach these customers if they wish to sell at the customer's price.
System Administrator	The System Administrator will maintain the system, which includes managing user accounts and ensuring smooth operational running of the website. Moreover, the system administrator will also work as a moderator, overlooking all the listings being posted and will verify the authenticity of the listing. Look out for any glitches or unexpected errors, and update the website.

b. Functional Requirements

Functional Requirements	
Sr#	**Functional Requirement**

1	As a new user (seller or buyer), I want to sign up for an account by providing my email, password, and basic information so that I can access personalized features of the travel planner.
2	As a registered user (seller or buyer), I want to sign in to my account using my email and password so that I can access my saved preferences and booking history.
3	As a user (seller or buyer) I want to reset my password in case I want to change it or forget it.
4	As a buyer, I want to search for the products by typing the keywords in the search bar.
6	As a buyer, I want to list the product that I need with the requirements such as price of the product.
7	As a buyer, I want to send the price offer to the seller who is selling the product whose price can be negotiated.
8	As a buyer, I want to chat with sellers to know more about the product.
9	As a business buyer, I want to give a rating to share my experience about the product and the seller.

10	As a seller, I want to list the product with some description and the price of that product.
11	As a seller, I want to list the intangible product or a service on the platform.
12	As a seller, I want to decide if the price of the product is negotiable or not.
13	As a seller, I want to send an offer to the buyer who has listed a product which they want.
14	As a seller, I want to chat with the buyer to know more about their requirements or to put an offer to them on which conditions can be met.
15	As a seller, I want to give a rating to share my experience with the buyer.
16	As a system administrator, I want to monitor system performance and uptime so that I can ensure a smooth user experience and address any issues promptly.
17	As a system administrator, I want to update or change the design of the website.
18	As a system administrator, I want to monitor the ads and decide whether to approve the ad to be up on the website or to decline the ad.

c. Non-functional Requirements

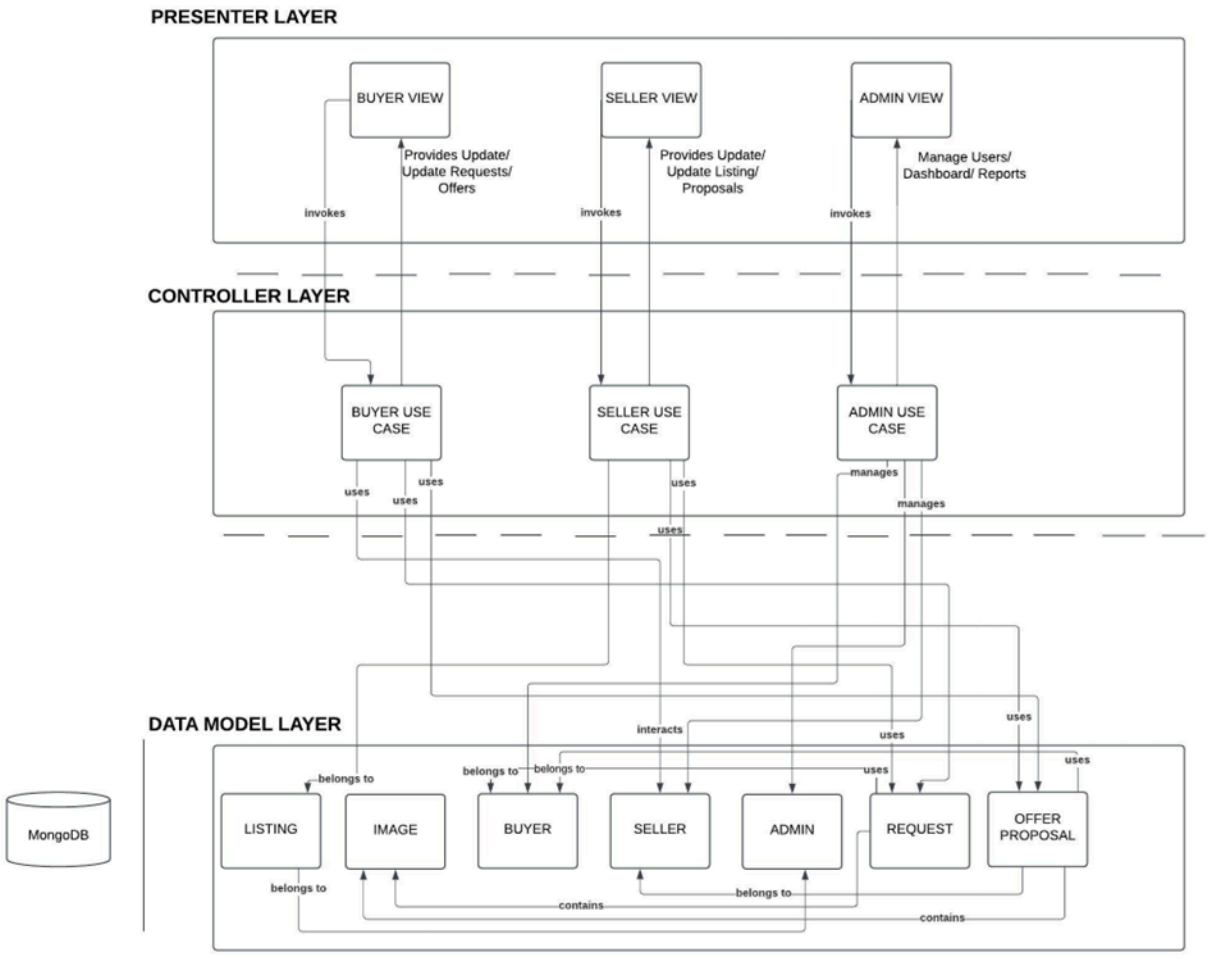
Sr#	Non- functional Requirements
1	The system should not utilize more than 1 GB of memory at any time during its execution.
2	The system should not fail more than 3 times every 24 hours. In case of a failure, the system should restore to normal operations within 5 minutes of a failure.
3	The response time for search queries should be less than 2 seconds for 95% of the requests.
4	The user interface should be responsive and provide a consistent experience across different devices, including desktops, tablets, and smartphones.
5	The system should be accessible 24/7 with an uptime of 99.9% over any given month.
6	The system should process all buyer-seller communication and transactions with a latency of no more than 500 milliseconds to ensure smooth real-time interactions.
7	The system should provide user authentication and authorization mechanisms to ensure that users can only access their own data and features.

8	The system should be compatible with the latest versions of major web browsers, including Chrome, Firefox, Safari, and Edge.
9	The system should be designed to handle up to 1 million records in the database without significant performance degradation.

3. System Architecture

Brief introduction of this chapter in a paragraph highlighting the content

a. Architecture Diagram



b. Architecture Description

Layered architecture divides an application into distinct layers (e.g., presentation, business logic, data access), each with a specific responsibility. This clear separation makes it easier to identify,

isolate, and fix issues in a particular layer without affecting the others. For example, changes to the UI layer won't impact the data access logic, simplifying code maintenance.

By encapsulating logic in well-defined layers, components from one layer can be reused across multiple parts of the application or even in different projects. For instance, a business logic layer can serve different frontends like web and mobile apps, reducing duplication and speeding up development.

The modular nature of layered architecture allows new features to be added with minimal disruption. Developers can extend the functionality of one layer, such as adding new APIs to the business logic layer, without needing to overhaul other layers, ensuring smooth and scalable growth.

Each layer in the architecture handles a single concern or responsibility, ensuring that the logic is well-organized and focused. This makes the codebase easier to understand and reduces the risk of unintended side effects, as changes to one layer don't ripple into others unnecessarily.

c. Justification of the Architecture

The layered architecture was selected for its clear separation of concerns, dividing the application into presentation, business logic, and data access layers. This structure enhances maintainability by allowing developers to update or debug individual layers without impacting others. It also supports scalability, enabling each layer to be optimized or scaled independently as the user base grows.

Pros:

- **Maintainability:** Isolated layers simplify debugging and future enhancements.
- **Scalability:** Each layer can be scaled independently to handle increased load.
- **Reusability:** Business logic can be reused across different interfaces, such as web and mobile.
- **Testability:** Layers can be tested independently, ensuring robust application performance.

Cons:

- **Complexity:** Strict separation may introduce overhead for simple features.
- **Performance:** Data passing through multiple layers can slightly impact response times.

This architecture aligns well with our non-functional requirements:

- **Scalability:** Supports growth in user base and feature set.
- **Maintainability:** Facilitates easier updates and bug fixes.
- **Security:** Sensitive operations are confined to specific layers, enhancing security.

Overall, the layered architecture provides a robust foundation for a dynamic, scalable, and maintainable e-commerce platform.

d. Tools and Technologies

1 Front-end Development

1.1 React

1.1.1 Version: 18.2.0

1.1.2 Description: A JavaScript library for building user interfaces, allowing the creation of reusable UI components.

1.2 Next.js

1.2.1 Version: 14.1.4

1.2.2 Description: A React framework that enables server-side rendering and static site generation for better performance and SEO.

1.3 Tailwind CSS

1.3.1 Version: 3.4.3

1.3.2 Description: A utility-first CSS framework that allows for rapid styling of components with a customizable design system.

2 Back-end Development

2.1 MongoDB

2.1.1 Version: 8.0

2.1.2 Description: A NoSQL database that stores data in flexible, JSON-like documents, ideal for handling diverse data types and scaling.

2.2 Node.js

2.2.1 Version: 21.0.0

2.2.2 Description: A JavaScript runtime built on Chrome's V8 JavaScript engine that enables server-side scripting and builds scalable network applications.

2.3 Express.js

2.3.1 Version: 5.0.0

2.3.2 Description: A web application framework for Node.js that simplifies the creation of APIs and server-side applications.

3 API Development and Testing

3.1 Postman

3.1.1 Version: 11.0.0

3.1.2 Description: A collaboration platform for API development that allows for testing, monitoring, and documenting APIs efficiently.

4 UI/UX Design

4.1 Figma

4.1.1 Version: Latest (cloud-based)

4.1.2 Description: A collaborative interface design tool that enables designers to create, prototype, and share user interfaces.

5 Development Environment

5.1 Visual Studio Code

5.1.1 Version: 1.93.1

5.1.2 Description: A powerful code editor that supports various programming languages, extensions, and tools for debugging and development.

6 Additional Tools

6.1 Git

6.1.1 Version: 2.47

6.1.2 Description: A version control system to manage code changes and collaborate with team members efficiently.

6.2 Jest

6.2.1 Version: 29.7.0

6.2.2 Description: A JavaScript testing framework used for unit and integration testing, especially suited for React applications.

7 Deployment

7.1 Git:

7.1.1 Version: 2.47

7.1.2 Description: A version control system to manage code changes and collaborate with team members efficiently.

7.2 NPM (Node Package Manager):

7.2.1 Version: 10.1.0

7.2.2 Description: Manages packages for Node.js applications.

7.3 Supervisord:

7.3.1 Version: 4.2.4.

7.3.2 Description: Monitors and manages application processes.

7.4 Symbolic Links:

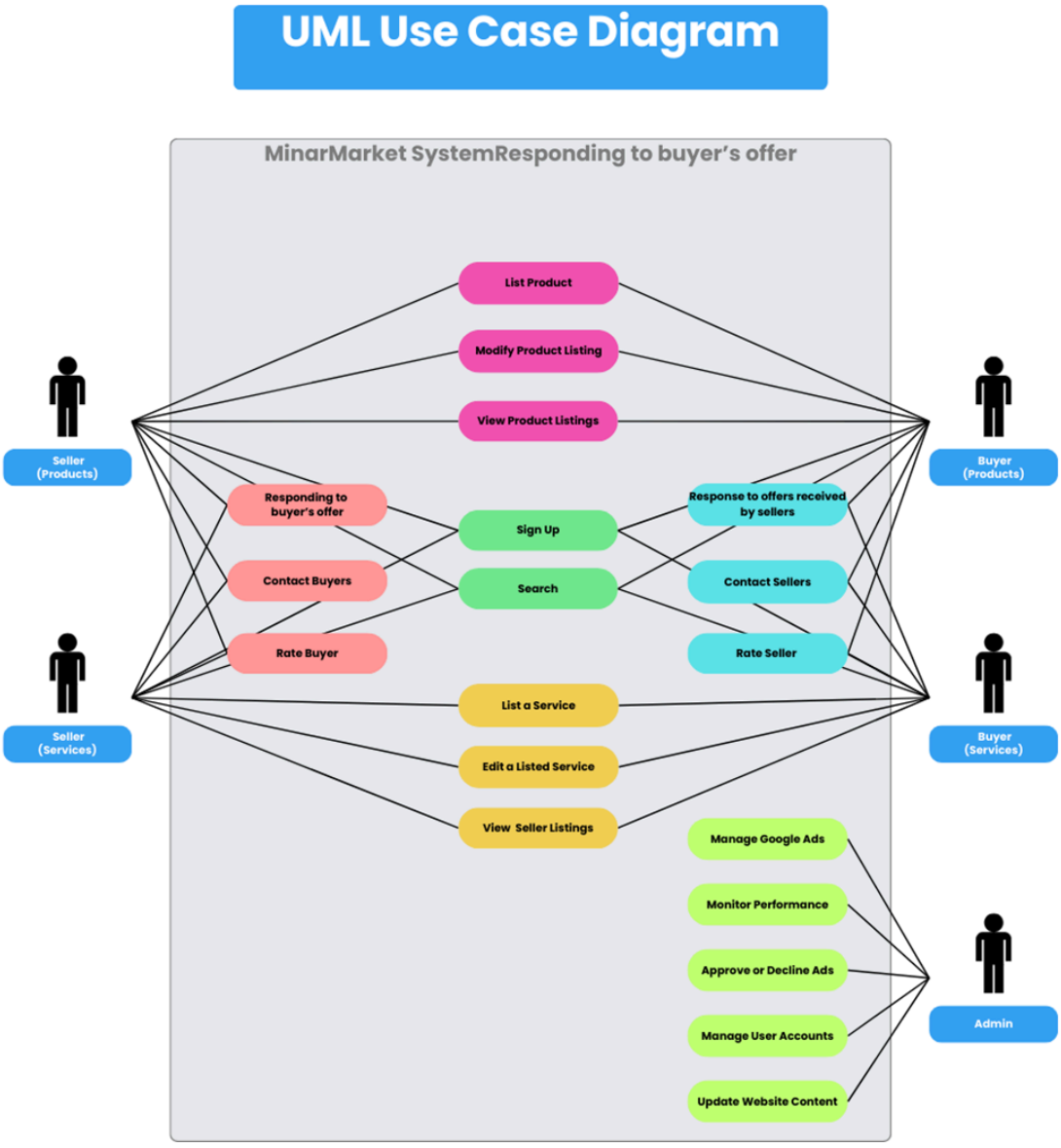
7.4.1 Description: Links files and directories for easy access.

4. Requirements Specifications

Brief introduction of this chapter in a paragraph highlighting the content

a. Use Cases

Draw use case diagrams of your system using standard UML notation. Moreover, give description of 10 core use cases of your system



1 Buyers can list the products they want with the requirement details.

Identifier	UC-002
Purpose	Buyers can list a desired product on the platform by providing specific requirement details, such as budget, condition of the product, and other relevant preferences.
Pre-conditions	The user is logged into the platform. The platform supports buyer requests for listing desired products.
Post-conditions	The buyer's product request is successfully listed on the platform. Sellers are able to view the buyer's request and respond with product offers that match the buyer's requirements.

Step #	**Typical Course of Action**
1	The buyer navigates to the "Request a Product" or "Post a Requirement" section.
2	The system displays a form for the buyer to fill out the product requirements.
3	The buyer specifies the product type or model they are looking for.
4	The buyer provides a budget range for the desired product.
5	The buyer specifies the preferred condition of the product (e.g., new, used, refurbished).
6	The buyer can provide additional details, such as preferred brand, size, color, or any special requirements.
7	The system may prompt the buyer to upload reference images or examples of the desired product.

8	The buyer reviews all the entered details and confirms the listing.
9	The system processes the request and posts it on the platform, making it visible to sellers.
10	The use case ends when the buyer's request is successfully listed.
Step #	**Alternate Courses of Action**
1	In step 5, if the buyer is unsure about the budget, they can leave the field open, and the system will notify sellers to provide offers within a wide range.
2	In step 7, if the buyer does not have specific additional details, they can leave the optional fields empty and proceed with a general request.
3	In step 9, if the buyer is not ready to submit the request, they can save it as a draft and return to complete it later.
Step #	**Exception Paths**
1	In step 10, if the system encounters an error while posting the request, the buyer is notified and can retry submitting the request.
2	If the platform is unable to match sellers with the buyer's specific requirements, the system may suggest alternative sellers or similar products available on the platform

2 Buyers can respond to the offers received by sellers

Identifier	UC-003
Purpose	Buyers can view and respond to offers from sellers based on their product requests.
Pre-conditions	The buyer has listed a product request on the platform. Sellers have submitted offers in response to the buyer's request.
Post-conditions	The buyer successfully responds to the seller's offers (e.g., accepting, rejecting, or negotiating). The status of the offer is updated accordingly in the system.
Step #	**Typical Course of Action**
1	The buyer navigates to the "My Requests" or "Offers Received" section.
2	The buyer selects the relevant product request for which they have received seller offers.
3	The system displays a list of offers from various sellers, including details such as price, product condition, and seller ratings.

4	The buyer clicks on a specific offer to view detailed information about the product and seller's terms.
5	The buyer reviews the offer, considering factors like price, product details, and seller reputation.
6	The buyer can respond by selecting one of the following options: ● Accept the offer and proceed with the purchase. ● Reject the offer and provide feedback or a reason for rejection. ● Initiate a negotiation by suggesting alternative terms (e.g., lower price or different product conditions).
7	If the buyer accepts the offer, they proceed to the payment and shipping steps as per the platform's standard process.
8	If the buyer negotiates, the system sends the counteroffer to the seller, and the process continues until both parties reach an agreement or the buyer/seller declines further negotiation.
9	If the buyer rejects the offer, the system updates the status and notifies the seller.
10	The buyer can repeat steps 4 to 9 for other offers if multiple sellers have responded.
11	The use case ends when the buyer has responded to all offers or completed the transaction with a seller.
Step #	**Alternate Courses of Action**
1	In step 6, the buyer can choose to mark an offer for later review if they are not ready to respond immediately.

2	In step 8, if the seller rejects the counteroffer, the buyer can choose to make another counteroffer or reject the offer entirely.
Step #	**Exception Paths**
1	In step 7, if the system encounters an issue during the acceptance or payment process, an error message is displayed, and the buyer can retry or contact support.
2	If the seller fails to respond to a negotiation in step 8 within a specified time frame, the system notifies the buyer and suggests reviewing other offers.

3 Negotiate Terms of Service with Buyer

Identifier	UC-009
Purpose	The seller negotiates the terms of the service (price, delivery time, etc.) with the buyer through the platform.
Pre-conditions	The seller is logged into their account. The buyer has received a proposal from the seller or has requested a negotiation. Both parties are able to communicate through the platform.
Post-conditions	A mutually agreed-upon proposal is submitted, or the negotiation is terminated by either party.

Step #	Typical Course of Action
1	The buyer reviews the seller's proposal and sends a counter-offer or requests changes.
2	The seller receives a notification of the buyer's response.
3	The seller views the buyer's requested changes (e.g., new price, delivery time).
4	The seller clicks on the "Negotiate" button.
5	The system prompts the seller to input revised terms (price, service scope, time frame).
6	The seller revises the proposal and submits the new terms.
7	The system sends the updated proposal to the buyer.
8	The buyer receives the updated proposal and reviews it.
9	The negotiation continues until both parties reach an agreement or either party cancels the negotiation.
10	Once agreed, the system finalizes the proposal and updates the service request status.
11	The use case ends.
Step #	**Alternate Courses of Action**

1	In step 3, the seller may reject the buyer's counter-offer without proposing new terms, terminating the negotiation.
Step #	**Exception Paths**
1	In step 9, if either party cancels the negotiation at any point, the system notifies the other party, and the negotiation is terminated.

4 Monitor System Performance

Identifier	UC-013
Purpose	The system administrator ensures the smooth operation of the marketplace by monitoring the system's performance.
Pre-conditions	The admin is logged into the admin panel.
Post-conditions	System performance metrics including business KPIs and Platform KPIs are recorded, and any performance issues are flagged for resolution.
Step #	**Typical Course of Action**
1	The admin logs into the admin panel.

2	The admin navigates to the system monitoring section.
3	The admin reviews the current system performance metrics (e.g., uptime, response time). The admin also views platform and business KPIs.
4	If issues are identified, the admin flags them for resolution or initiates troubleshooting steps.
5	The admin logs out after monitoring is completed.
Step #	**Alternate Courses of Action**
1	In step 3, the admin can also choose to generate a detailed report of system performance metrics instead of only reviewing them.
Step #	**Exception Paths**
1	In step 4, if no issues are detected, the admin can simply mark the performance check as complete and skip the troubleshooting steps.

5 Resolve User Complaints

Identifier	UC-014
Purpose	The admin handles user complaints or reported issues, ensuring any problems are resolved in a timely manner.

Pre-conditions	The admin is logged into the admin panel and there are user complaints to address.
Post-conditions	Complaints are resolved, and users are notified of the resolution.

Step #	**Typical Course of Action**
1	The admin logs into the admin panel.
2	The admin navigates to the "User Complaints" section.
3	The admin reviews the list of complaints and prioritizes them based on urgency.
4	The admin investigates each issue and takes appropriate action (e.g., contacting the user or resolving technical issues).
5	The admin updates the complaint status to "resolved" and notifies the user.

Step #	**Alternate Courses of Action**
1	In step 3, the admin can forward certain complaints to another department if they fall outside the admin's area of responsibility.

Step #	**Exception Paths**

1	In step 4, if the complaint cannot be resolved immediately, the admin informs the user of the delay and provides an estimated resolution time.

6 Sending an Ad offer to the buyer

Identifier	UC-022
Purpose	Allow the seller to send advertisements for their products to potential buyers, responding to the buyers' specific interests.
Pre-conditions	The seller has products available for sale on the platform. The seller has access to potential buyers' interests or requests on the platform.
Post-conditions	The potential buyer receives the ad and can review it. The ad proposal is recorded in the system for the buyer to accept or reject.

Step #	**Typical Course of Action**
1	The seller logs into the platform and navigates to the "Create Ads" section.
2	The seller reviews the list of products available for advertisement.
3	The system displays product details, including specifications, price, and images.

4	The seller selects a product to advertise and clicks the "Send Ad" button.
5	The system prompts the seller to input ad details, including a brief description, target audience, and any promotional offers.
6	The seller submits the ad, which is sent directly to the potential buyer's homepage.
7	The system confirms that the ad has been sent, and the seller can view it in their "Sent Ads" section.
8	The use case ends when the ad is successfully sent to the potential buyers.
Step #	**Alternate Courses of Action**
1	The seller can modify or withdraw the ad if the buyer hasn't responded yet.
2	The seller may send multiple ads for different products to the same buyer if more than one product matches the buyer's interests.
Step #	**Exception Paths**
1	If the system detects that the ad content does not comply with platform guidelines, a warning is shown to the seller.
2	If the ad fails to send due to a system error, the seller is notified, and the system prompts them to retry.

7 Reviewing Proposals (Accept/Deny)

Identifier	UC-026
Purpose	Allow buyers to evaluate proposals submitted by sellers in response to their service requests and make informed decisions by accepting or rejecting proposals.
Pre-conditions	The buyer has posted one or more service requests. Sellers have submitted proposals in response to these service requests. The buyer is authenticated and authorized to review proposals.
Post-conditions	The buyer has either accepted or rejected each reviewed proposal. Accepted proposals may lead to further actions such as contract signing or payment processing. Rejected proposals are marked accordingly, and sellers are notified of the decision.
Step #	**Typical Course of Action**
1.	The buyer accesses their "My Service Requests" dashboard to view all active service listings.
2.	The buyer selects a specific service request to view all received proposals.

3.	The buyer reviews each proposal's details, including seller information, pricing, timelines, and any attached documents or portfolios.
4	For each proposal, the buyer assesses its suitability based on their requirements and criteria.
5	The buyer decides to either accept or reject the proposal by clicking the corresponding "Accept" or "Reject" button.
6	If accepted, the system confirms the acceptance and may prompt the buyer to proceed with next steps such as contract formalization.
7	If rejected, the system updates the proposal status and notifies the seller of the rejection.
8	The buyer continues reviewing other proposals until all desired decisions (acceptances or rejections) are made.
Step #	**Alternate Courses of Action**
1	**Step 3:** The buyer can sort proposals based on criteria like price, rating, or submission date to facilitate easier review.
2	**Step 4:** The buyer may choose to request additional information or clarifications from the seller before making a decision.
3	**Step 5:** The buyer can temporarily save their decision on a proposal and revisit it later for final action.
Step #	**Exception Paths**

1	**Step 3:** If proposal details are incomplete or missing critical information, the system prompts the buyer to request more details from the seller.
2	**Step 5:** If the system encounters an error while processing the accept/reject action, an error message is displayed, and the buyer is prompted to retry.
3	**Step 7:** If notifying the seller fails due to a system error, the system logs the issue and retries the notification process automatically.

8 Manage Google Ads and Analyze Ad Performance

Identifier	UC-018
Purpose	The system admin manages the integration of Google ads on the platform, sets ad parameters, and monitors ad performance through statistics.
Pre-conditions	The admin is logged into the admin panel and has access to Google Ads integration features
Post-conditions	Google ads are updated, and performance statistics are recorded for further analysis.
Step #	**Typical Course of Action**
1	The admin logs into the admin panel.

2	The admin navigates to the "Google Ads Management" section.
3	The admin sets or updates ad parameters (e.g., ad type, target audience, budget).
4	The admin activates the ads and monitors real-time performance statistics (e.g., clicks, impressions).
5	The admin generates reports to analyze ad effectiveness and optimize performance.
Step #	**Alternate Courses of Action**
1	In step 3, the admin can choose to schedule ads to run during specific periods instead of setting them live immediately.
Step #	**Exception Paths**
1	In step 4, if the Google Ads API encounters an error or fails to sync, an error message is displayed, and the admin must resolve the issue before proceeding with ad updates.

9 Update Website Content

Identifier	UC-017
Purpose	The system administrator updates the design or content of the website to ensure it stays relevant and functional.

Pre-conditions	The admin is logged into the admin panel and there are new design or content updates to be applied including logo,colors etc.
Post-conditions	The website content is updated, and users can view the changes.

Step #	**Typical Course of Action**
1	The admin logs into the admin panel.
2	The admin navigates to the "Content Management" section.
3	The admin reviews the sections to be updated (e.g., homepage, terms and conditions).
4	The admin uploads or edits the content as needed.
5	The updates are reflected live on the website after confirmation.
Step #	**Alternate Courses of Action**
1	In step 3, the admin can schedule the content update to be published at a future date rather than immediately.

Step #	Exception Paths
1	In step 4, if the content update violates the formatting rules, the system displays an error message, and the update cannot proceed until corrected.

1.1.1 Manage User Accounts

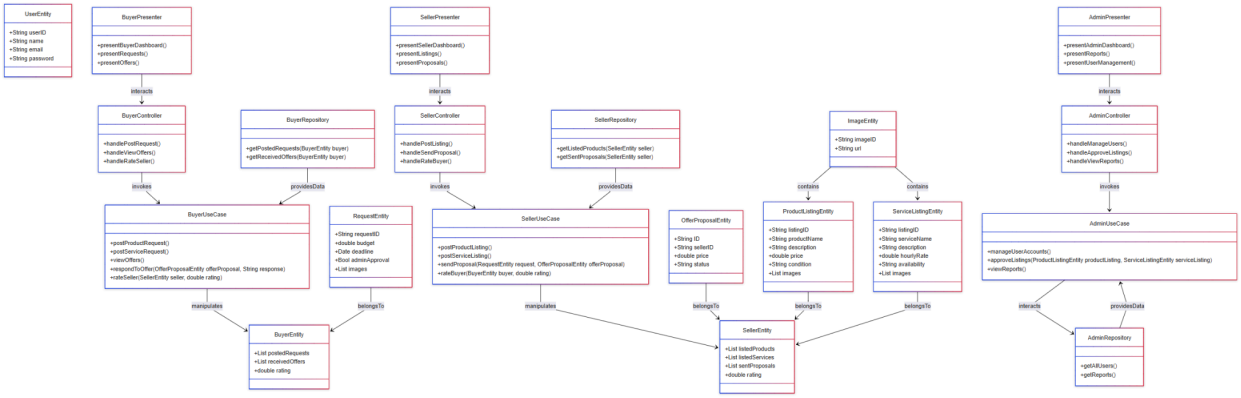
Identifier	UC-0116
Purpose	The system administrator manages user accounts, including account approval, deactivation, and reactivation.
Pre-conditions	The admin is logged into the admin panel and there are pending user actions (e.g., account creation or deactivation requests).
Post-conditions	User accounts are updated based on admin actions (e.g., activated or deactivated).

Step #	Typical Course of Action
1	The admin logs into the admin panel.
2	The admin navigates to the "User Management" section.

3	The admin reviews the list of user accounts, including pending verifications or deactivation requests.
4	The admin takes the appropriate action (e.g., verify new account, deactivate existing account).
5	The system reflects the changes to the user accounts.
Step #	**Alternate Courses of Action**
1	In step 3, the admin can choose to add user notes for further reference, instead of immediately processing requests.
Step #	**Exception Paths**
1	In step 4, if the user has violated the terms and conditions, the admin can temporarily suspend the account instead of directly deactivating or approving it.

b. Class Diagram

Link: Class DGR



Description

1 User Class

Purpose: User Class represents the base class for all users in the system, including buyers and sellers. Contains general attributes and functions related to user authentication and interaction with the platform.

Attributes:

userID: Unique identifier for each user.

name: The name of the user.

email: The email address of the user.

password: The user's password.

Operations:

login(): Allows the user to log into the platform.

logout(): Logs the user out of the platform.

2 Buyer Class

Purpose: Buyer Class represents users who post requests for products and services. Buyers interact with sellers by viewing offers and proposals, and can update their listings or rate sellers after transactions.

Attributes:

postedProductRequests: A list of product requests posted by the buyer.

postedServiceRequests: A list of service requests posted by the buyer.

receivedOffers: A list of offers received from sellers.

rating: The buyer's average rating based on reviews by sellers.

Operations:

postProductRequest(): Allows the buyer to post a product request.

postServiceRequest(): Allows the buyer to post a service request.

viewOffers(): Allows the buyer to view offers from sellers.

respondToOffer(Offer offer, String response): Allows the buyer to respond to offers by accepting, rejecting, or negotiating.

updateListing(): Allows the buyer to update their existing requests or listings.

rateSeller(Seller seller, double rating): Allows the buyer to rate a seller after a transaction.

3 Seller Class

Purpose: Seller Class represents users who list products or services for sale. Sellers can post new listings, send proposals to buyers, negotiate offers, and rate buyers after transactions.

Attributes:

listedProducts: A list of product listings posted by the seller.

listedServices: A list of service listings posted by the seller.

sentProposals: A list of proposals sent to buyers in response to their requests.

rating: The seller's average rating based on reviews by buyers.

Operations:

postProductListing(): Allows the seller to post a new product listing.

postServiceListing(): Allows the seller to post a new service listing.

sendProposal(ServiceRequest request, Proposal proposal): Allows the seller to send proposals in response to buyer requests.

negotiate(Proposal proposal, String terms): Allows the seller to negotiate terms of a proposal with the buyer.

updateListingDetails(Listing listing): Allows the seller to update details of an existing listing.

rateBuyer(Buyer buyer, double rating): Allows the seller to rate a buyer after a transaction.

4 ProductRequest Class

Purpose: ProductRequest Class represents a request posted by a buyer for a specific product. Contains details such as the product name, budget, and condition.

Attributes:

requestID: Unique identifier for the product request.

productName: Name of the requested product.

description: A description of the product the buyer is looking for.

budget: The budget range for the product.

condition: Desired condition of the product (e.g., new, used, refurbished).

deadline: Deadline for receiving offers on the request.

Operations:

addAdditionalDetails(): Allows the buyer to add extra information to the product request.

5 ServiceRequest Class

Purpose: ServiceRequest Class represents a request posted by a buyer for a service. Contains details such as the service name, budget, and specific requirements.

Attributes:

requestID: Unique identifier for the service request.

serviceName: Name of the requested service.

description: A description of the service the buyer is looking for.

budget: The budget range for the service.

requirements: Specific requirements for the requested service.

deadline: Deadline for receiving offers on the service request.

Operations:

uploadReferenceFiles(): Allows the buyer to upload reference documents or files related to the service.

viewProposals(): Allows the buyer to view proposals received from sellers.

6 Offer Class

Purpose: Offer Class represents an offer made by a seller in response to a product or service request from a buyer. Contains details about the offer, including price and status.

Attributes:

offerID: Unique identifier for the offer.

productOrServiceID: Identifier of the product or service related to the offer.

sellerID: The ID of the seller who made the offer.

price: The proposed price for the product or service.

status: The current status of the offer (e.g., pending, accepted, declined, negotiated).

Operations:

sendOffer(): Allows the seller to send the offer to the buyer.

acceptOffer(): Allows the buyer to accept the offer.

declineOffer(): Allows the buyer to decline the offer.

negotiateOffer(String newTerms): Allows the buyer to negotiate the offer terms.

7 Proposal Class

Purpose: Offer Class represents a formal proposal submitted by a seller in response to a buyer's service or product request. Includes the price and conditions.

Attributes:

proposalID: Unique identifier for the proposal.

requestID: The ID of the service or product request the proposal is responding to.

sellerID: The ID of the seller who submitted the proposal.

price: The price proposed by the seller.

conditions: Any specific conditions or terms related to the proposal.

status: The status of the proposal (e.g., pending, accepted, declined, negotiated).

Operations:

submitProposal(): Allows the seller to submit a proposal to the buyer.

updateProposal(): Allows the seller to update an existing proposal.

viewBuyerResponse(): Allows the seller to view the buyer's response to the proposal.

8 ProductListing Class

Purpose: ProductListing Class represents a product listing posted by a seller, containing details like price, description, and associated images.

Attributes:

listingID: Unique identifier for the product listing.

productName: Name of the product listed for sale.

description: Description of the product.

price: Price of the product.

condition: Condition of the product (e.g., new, used, refurbished).

category: Category under which the product is listed.

images: A list of images associated with the product.

Operations:

updateListingDetails(): Allows the seller to update the details of the product listing.

deleteListing(): Allows the seller to remove the product listing.

9 ServiceListing Class

Purpose: ServiceListing Class represents a service listing posted by a seller, including details about the service offered and its availability.

Attributes:

listingID: Unique identifier for the service listing.

serviceName: Name of the service listed for sale.

description: Description of the service.

hourlyRate: The hourly rate charged for the service.

availability: The availability of the service.

Operations:

uploadServiceDetails(): Allows the seller to upload additional details about the service.

updateServiceDetails(): Allows the seller to update the service listing.

removeServiceListing(): Allows the seller to remove the service listing from the platform.

10 Admin Class

Purpose: Admin Class represents platform administrators who manage the platform, oversee user accounts, approve or decline ads, monitor system performance, and resolve user complaints.

Attributes:

adminID: Unique identifier for the admin.

name: The name of the admin.

Operations:

viewReports(): Allows the admin to view platform reports.

manageUserAccounts(): Allows the admin to manage user accounts (activation, suspension, etc.).

monitorSystemPerformance(): Allows the admin to monitor platform performance.

resolveUserComplaints(): Allows the admin to resolve user complaints.

approveOrDeclineAds(): Allows the admin to approve or decline ads posted on the platform.

manageWebsiteContent(): Allows the admin to update the content of the website.

analyzeGoogleAds(): Allows the admin to manage and analyze Google Ads performance.

11 Image Class

Purpose: Image Class handles the media associated with product listings, allowing sellers to upload and delete images of their products.

Attributes:

imageID: Unique identifier for the image.

url: URL of the image file.

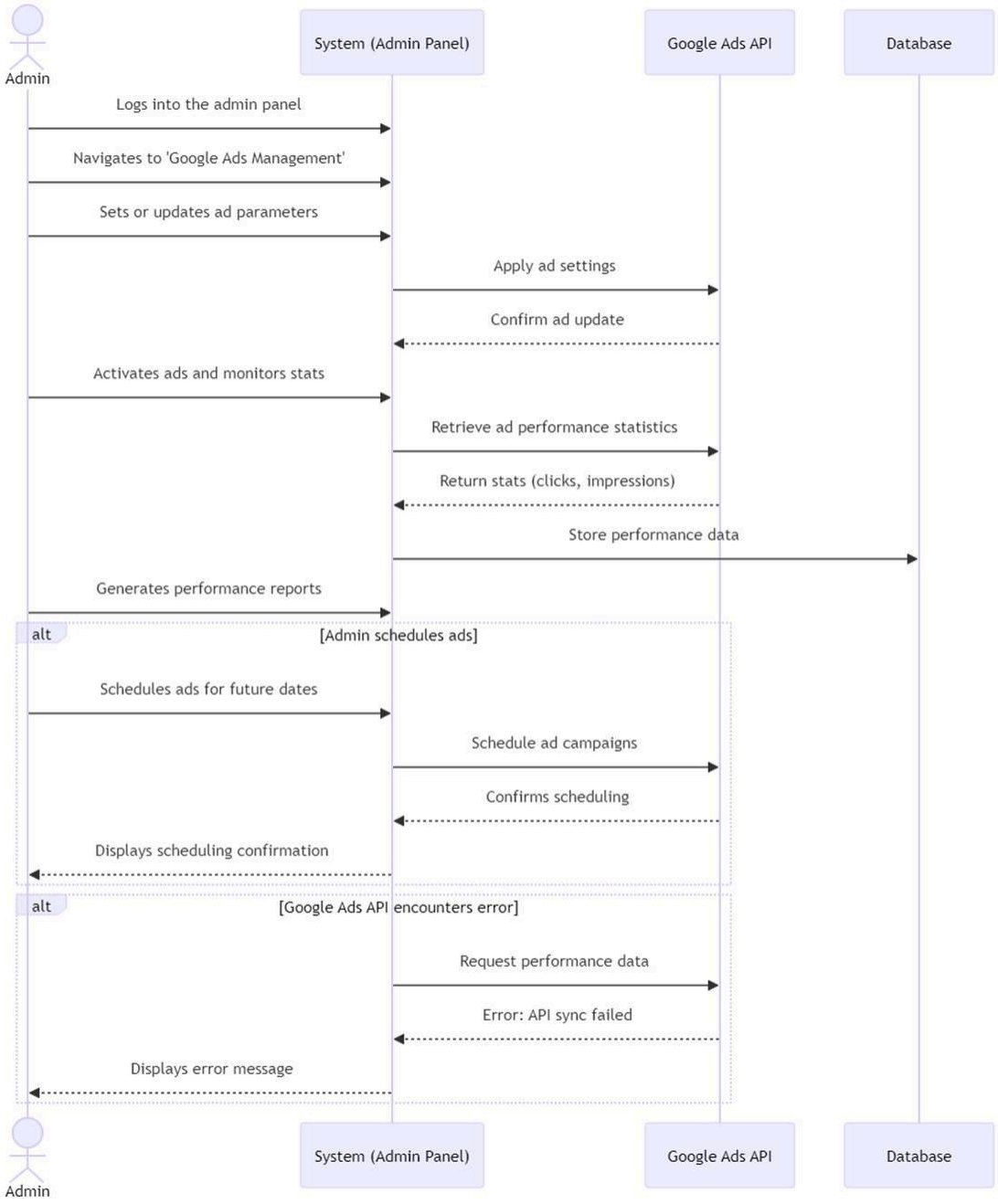
Operations:

uploadImage(): Allows the seller to upload images of their products.

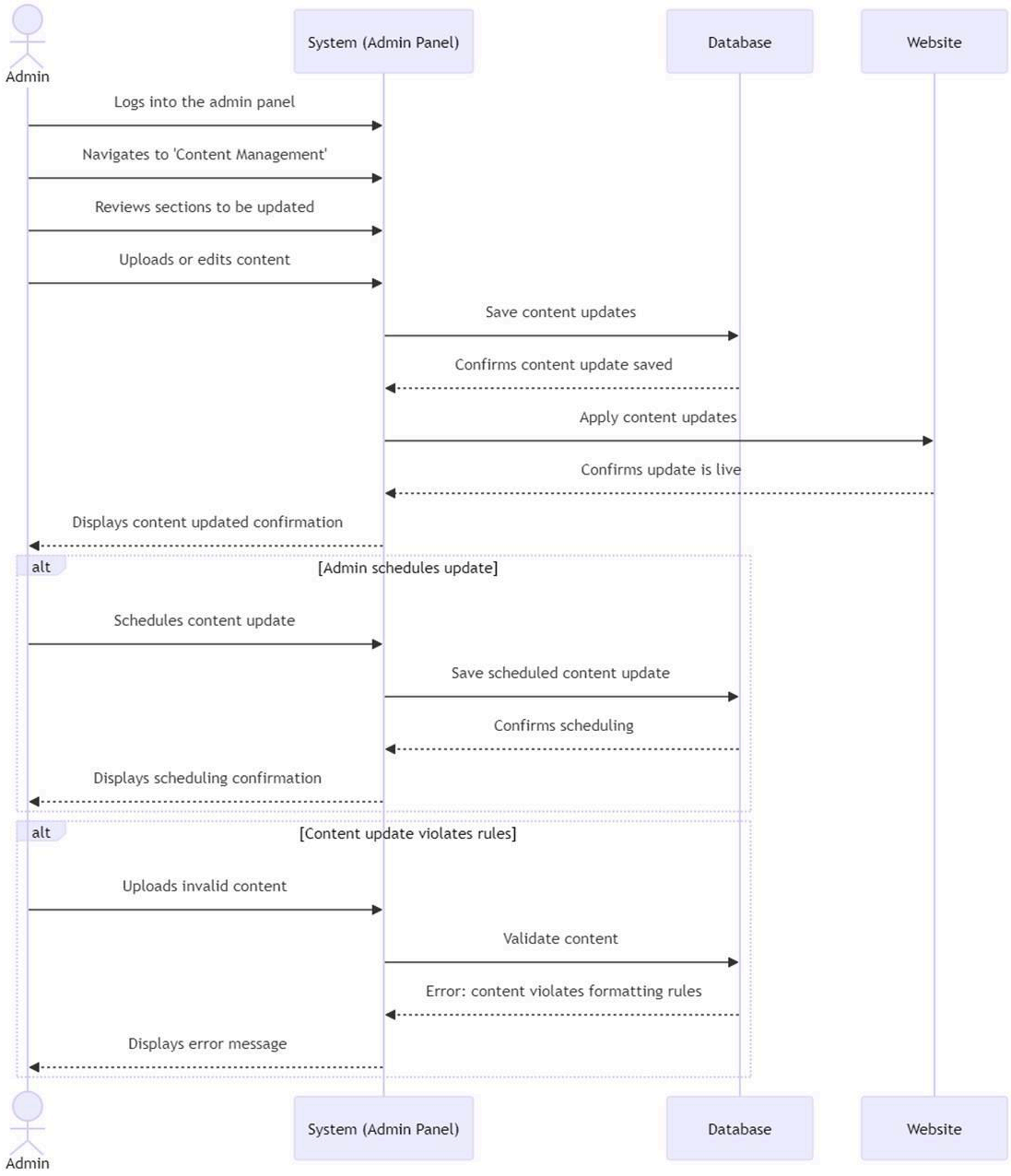
deleteImage(): Allows the seller to delete images from the listing.

c. Sequence Diagrams

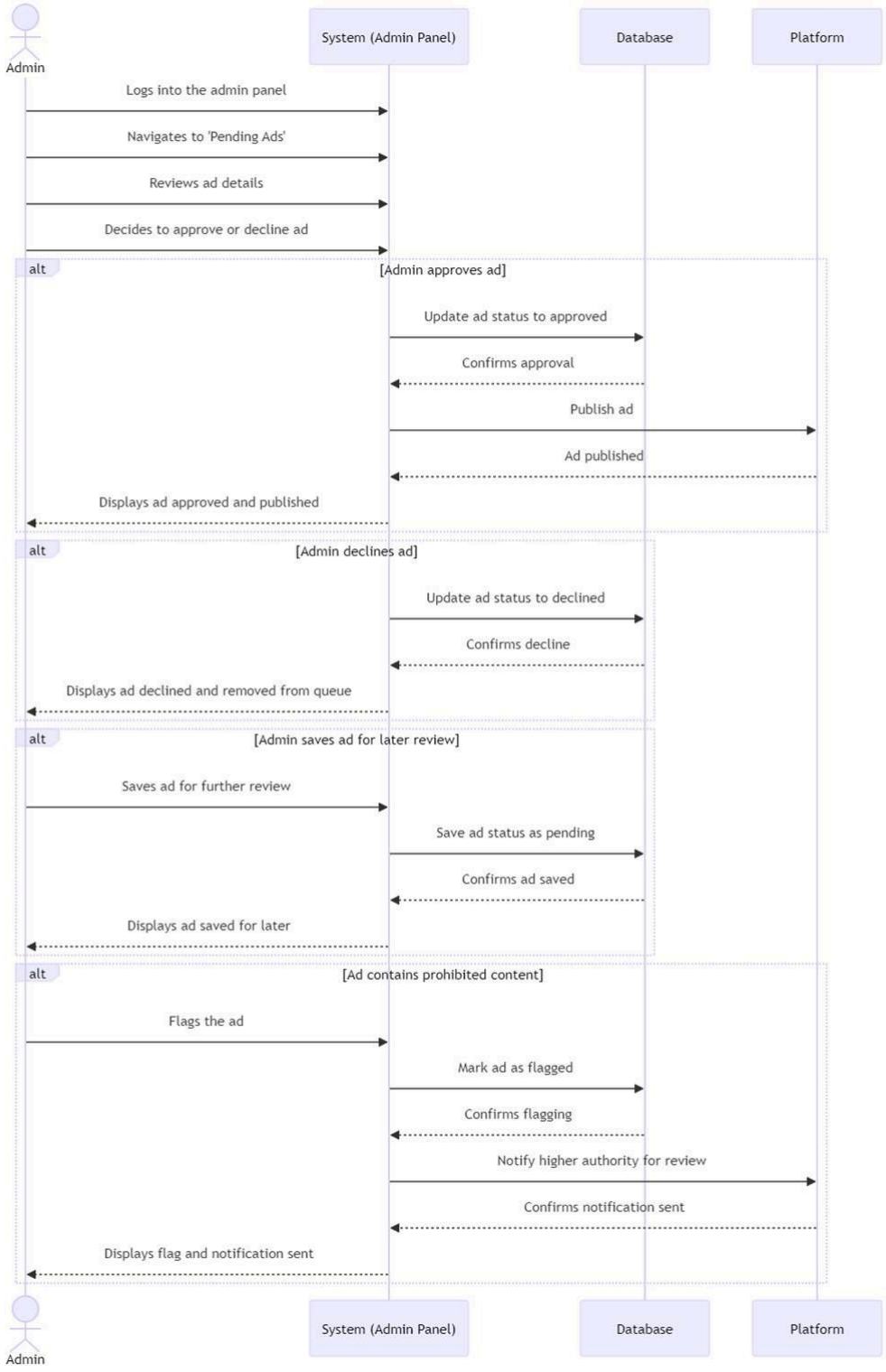
1 Manage Google Ads



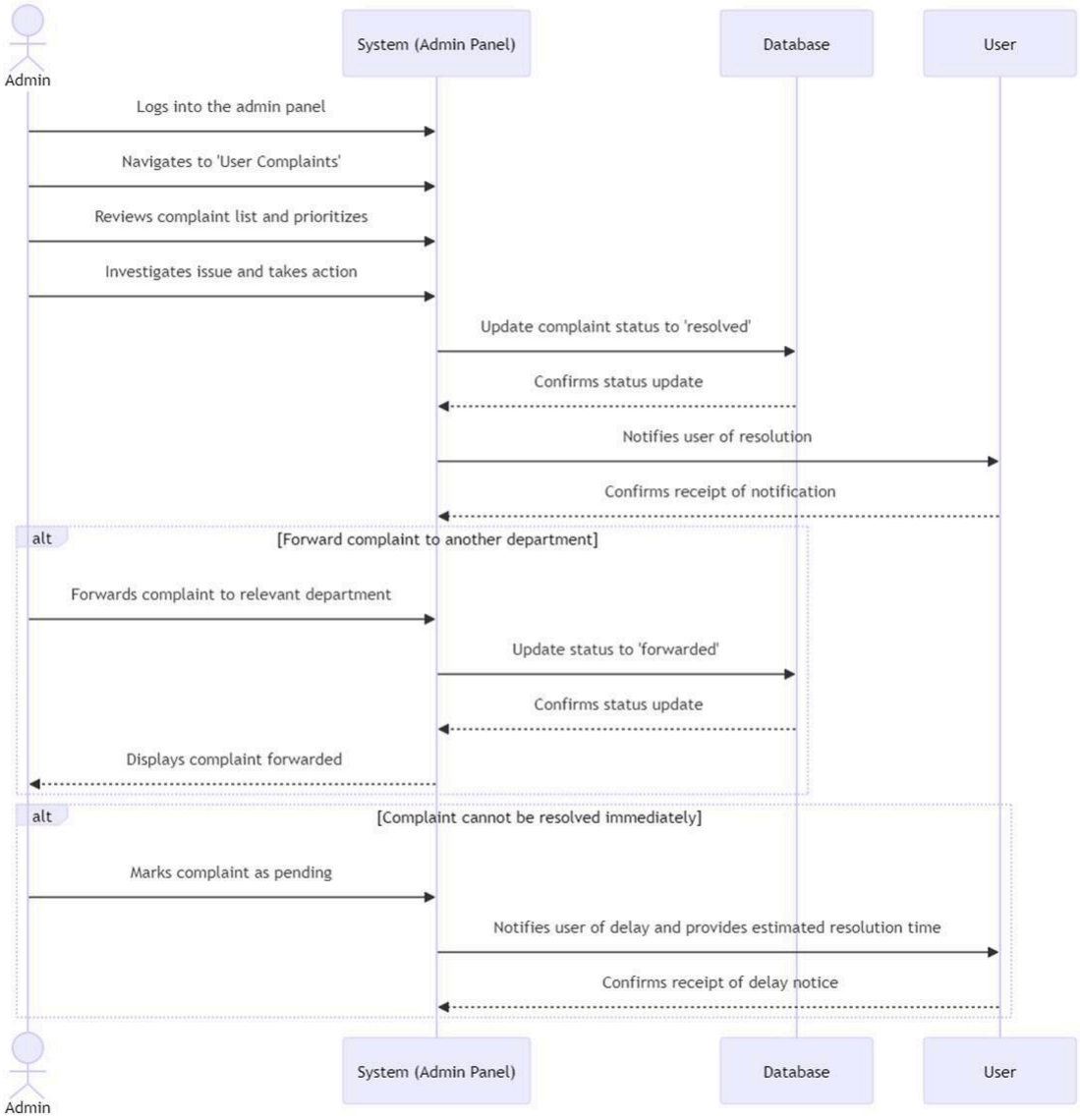
2 Update Website Content



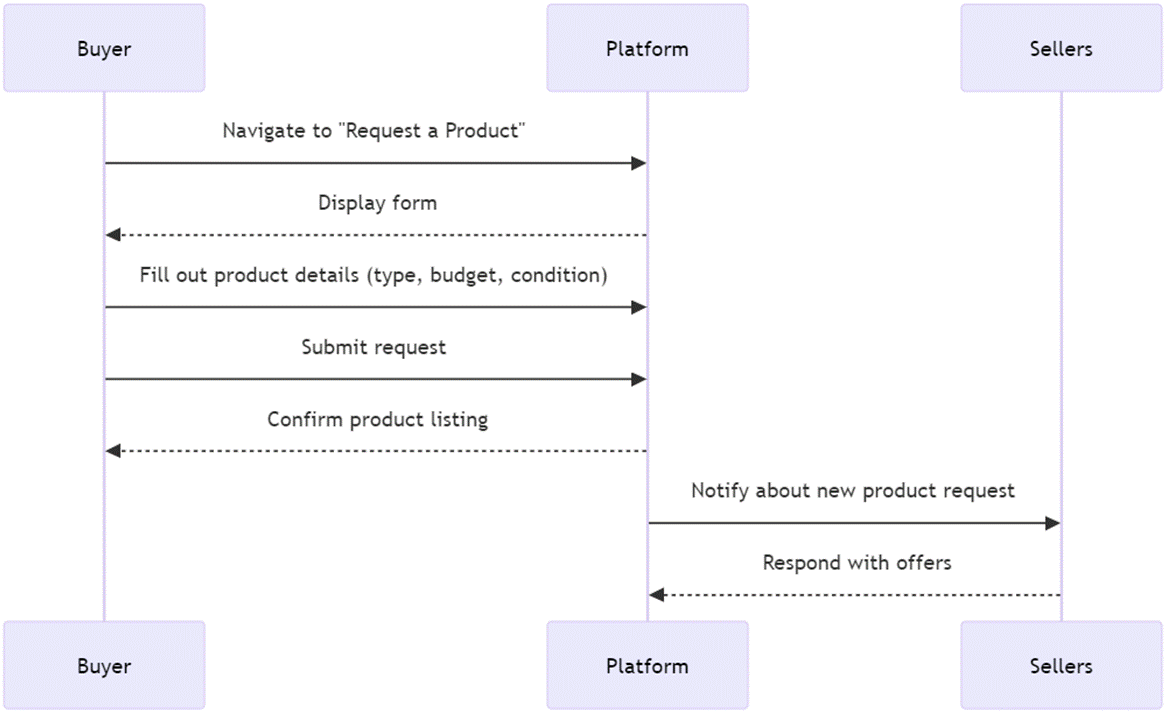
3 Approve or Decline Ads


Admin
System (Admin Panel)
Database
Platform
Logs into the admin panel
Navigates to 'Pending Ads'
Reviews ad details
Decides to approve or decline ad
alt
[Admin approves ad]
Update ad status to approved
Confirms approval
Publish ad
Ad published
Displays ad approved and published
alt
[Admin declines ad]
Update ad status to declined
Confirms decline
Displays ad declined and removed from queue
alt
[Admin saves ad for later review]
Saves ad for further review
Save ad status as pending
Confirms ad saved
Displays ad saved for later
alt
[Ad contains prohibited content]
Flags the ad
Mark ad as flagged
Confirms flagging
Notify higher authority for review
Confirms notification sent
Displays flag and notification sent
Admin
System (Admin Panel)
Database
Platform

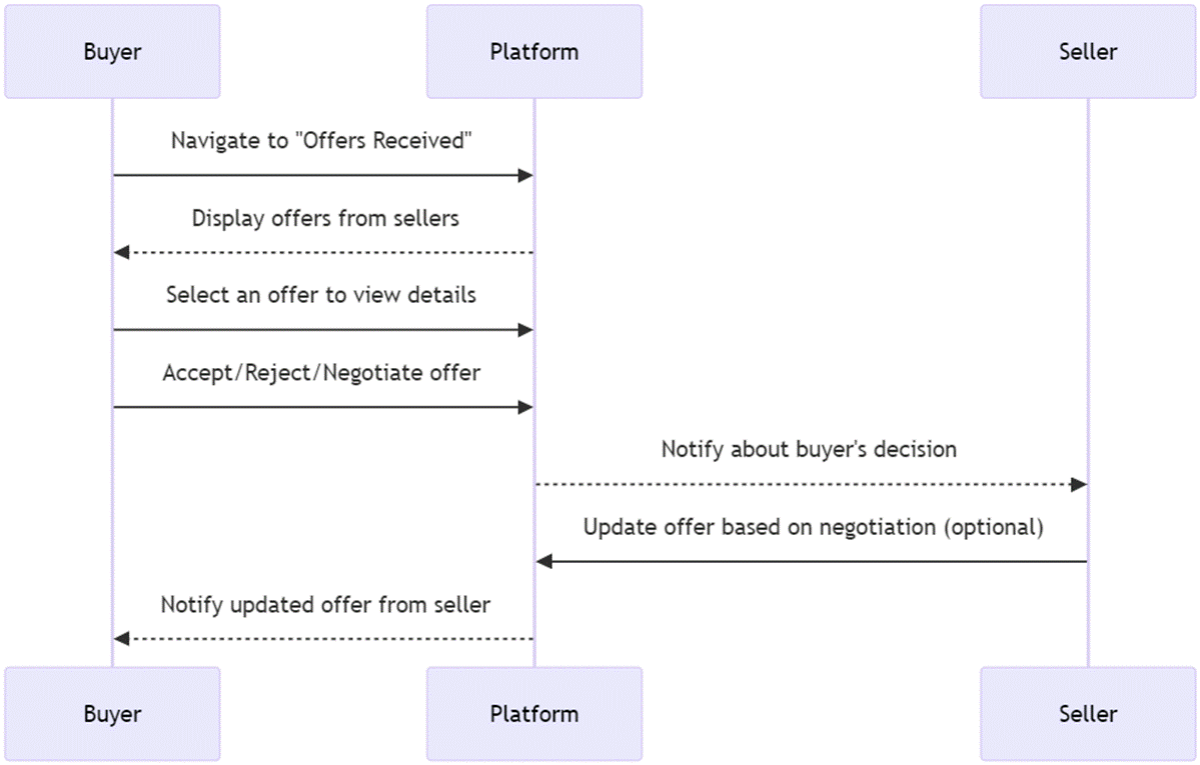
4 Resolve User Complaints



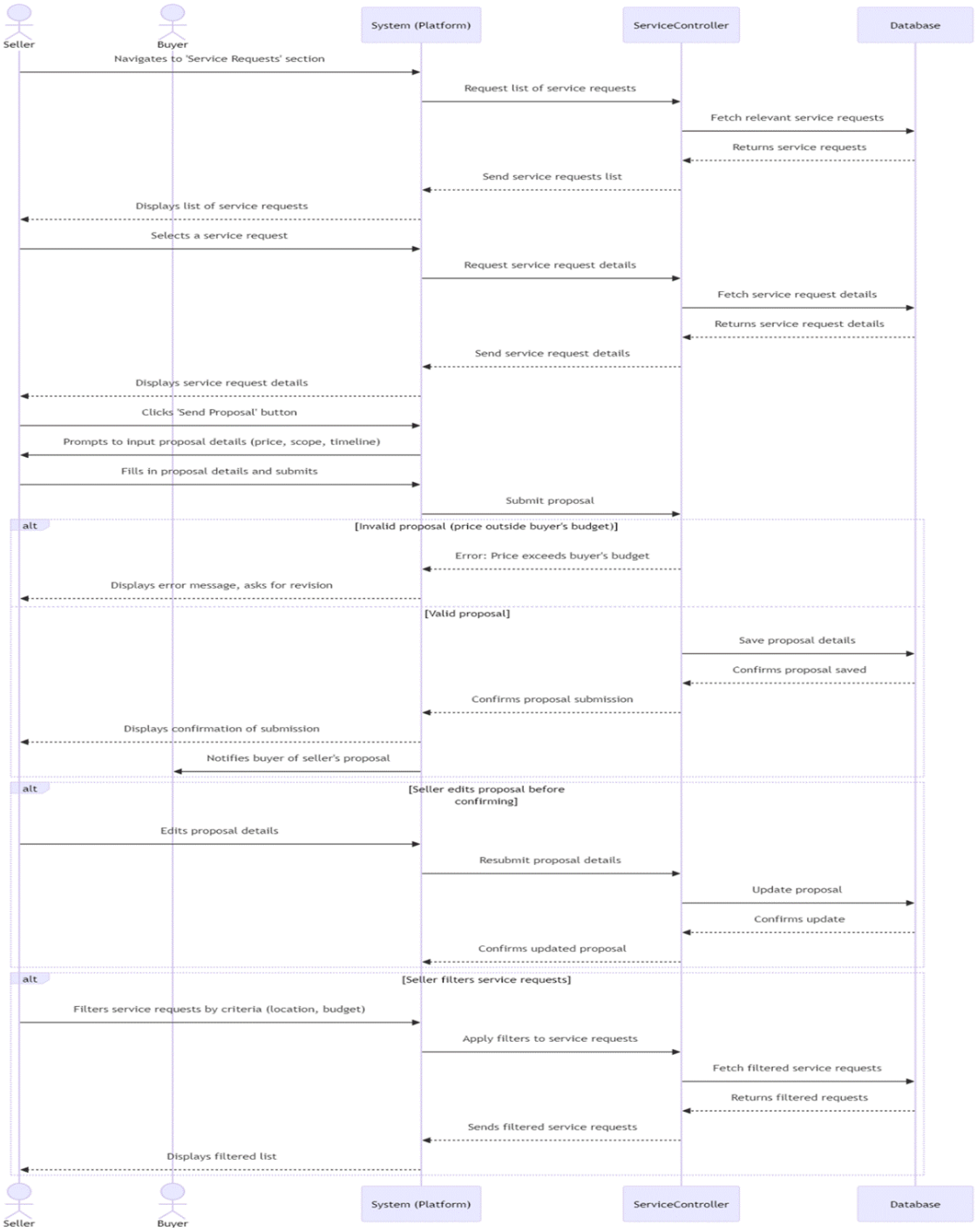
5 Buyers can make the listings



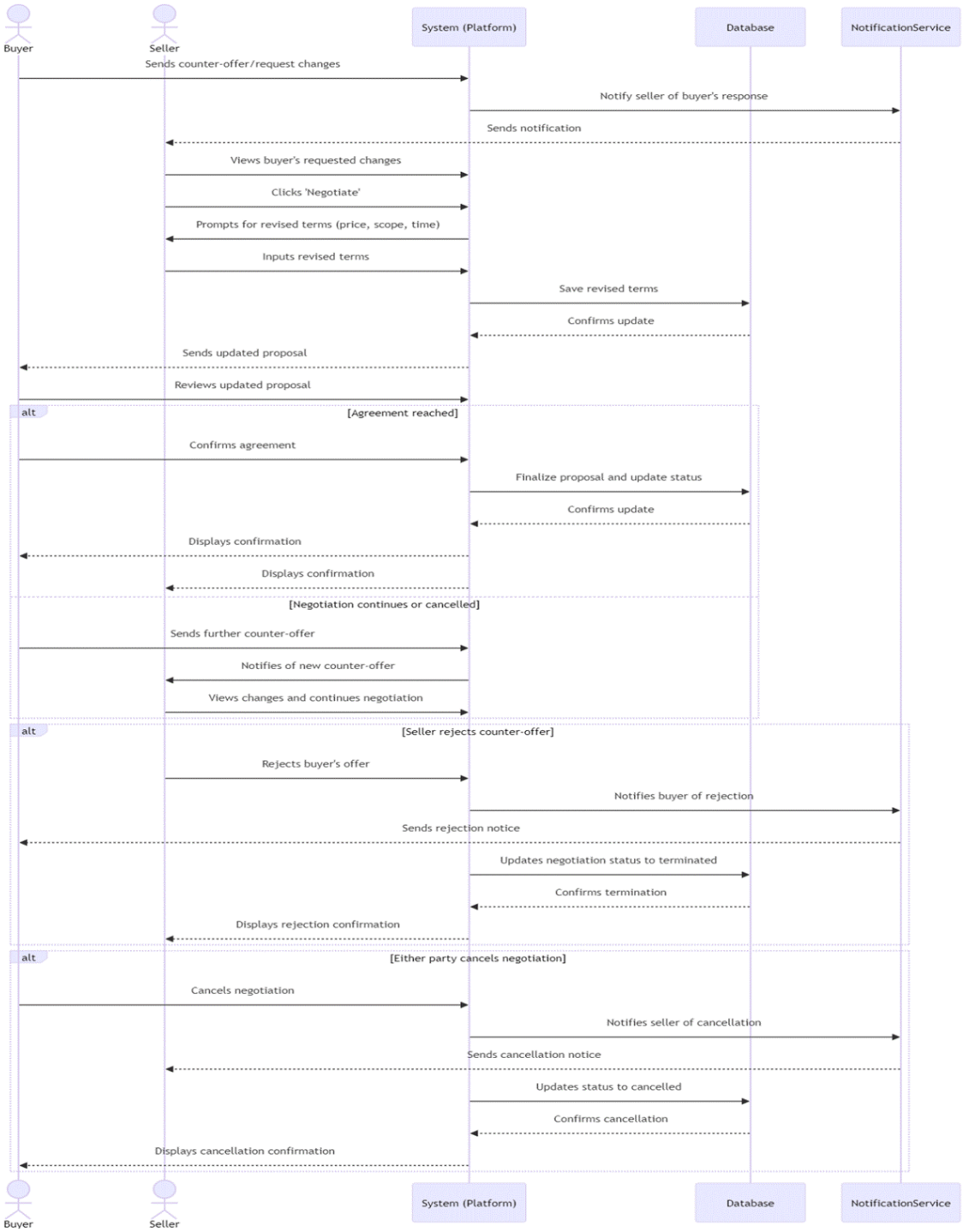
6 Buyers can respond to sellers' offers



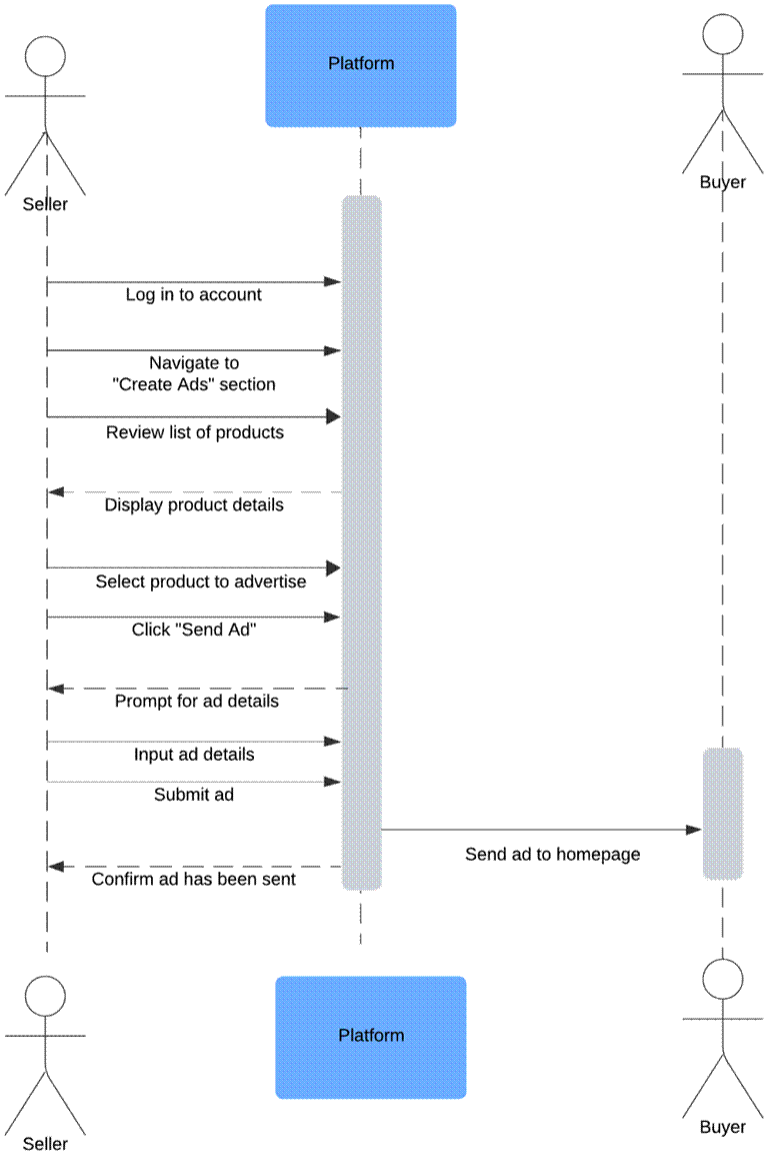
7 Send Proposals to Buyers for Service Requests



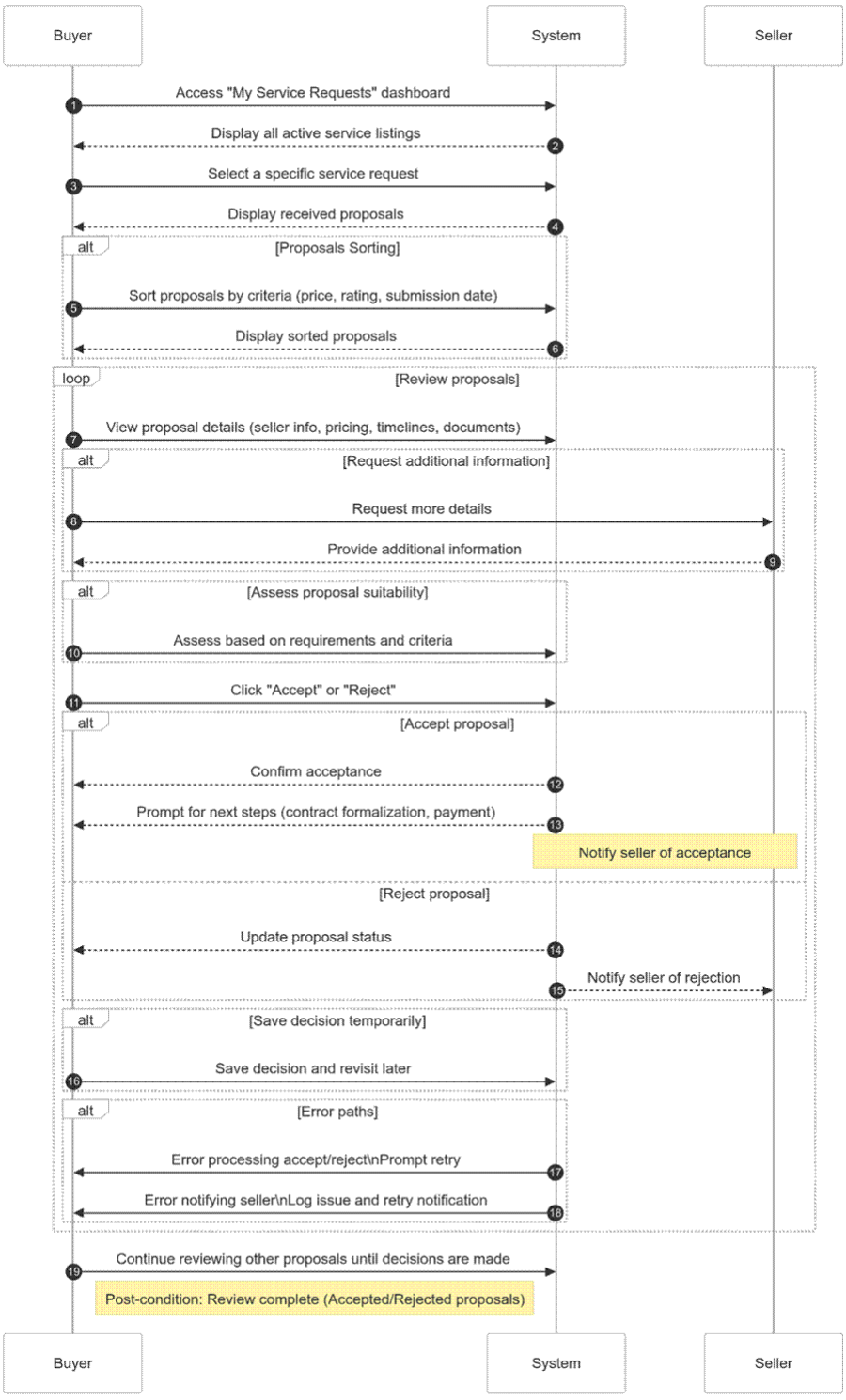
8 Negotiate Terms of Service with Buyer



9 Negotiate Terms of Service with Buyer



10 Negotiate Terms of Service with Buyer


Buyer
System
Seller
Access "My Service Requests" dashboard
Display all active service listings
Select a specific service request
Display received proposals
alt
[Proposals Sorting]
Sort proposals by criteria (price, rating, submission date)
Display sorted proposals
loop
[Review proposals]
View proposal details (seller info, pricing, timelines, documents)
alt
[Request additional information]
Request more details
Provide additional information
alt
[Assess proposal suitability]
Assess based on requirements and criteria
Click "Accept" or "Reject"
alt
[Accept proposal]
Confirm acceptance
Prompt for next steps (contract formalization, payment)
Notify seller of acceptance
[Reject proposal]
Update proposal status
Notify seller of rejection
alt
[Save decision temporarily]
Save decision and revisit later
alt
[Error paths]
Error processing accept/reject\nPrompt retry
Error notifying seller\nLog issue and retry notification
Continue reviewing other proposals until decisions are made
Post-condition: Review complete (Accepted/Rejected proposals)
Buyer
System
Seller

5. Software Development Methodology and Plan

Brief introduction of this chapter in a paragraph highlighting the content

a. Software Process Selection

1 Pros and Cons of Waterfall and Agile (Scrum) Processes

1.1 Waterfall Process:

- Pros:

  ❖ **Structured Approach**: The waterfall model is linear and sequential, making it easy to manage and understand. Each phase has specific deliverables and a review process, ensuring clarity in project progress.

  ❖ **Defined Requirements**: Since requirements are gathered at the beginning, there's a clear understanding of the project's scope from the start, which can help in planning and resource allocation.

- Cons:

  ❖ **Lack of Flexibility**: Once a phase is completed, it is difficult to go back and make changes. If the client or development team identifies issues or new requirements later, the rigid structure does not allow for easy modification.

  ❖ **Limited Client Visibility**: Clients don't see the final product until the end of the development cycle, which increases the risk of the product not meeting their expectations.

  ❖ **Delayed Feedback**: Since there is no incremental development or regular testing with clients, any issues or misalignments are only identified towards the end, making them harder and more expensive to fix.

1.2 Agile (Scrum) Process:

- Pros:

  - **Iterative Development**: Agile operates on short, iterative cycles (sprints), allowing teams to focus on manageable tasks and continuously improve based on client feedback.

  - **Frequent Feedback**: Clients are involved at the end of each sprint, providing feedback that can be incorporated into the next iteration. This ensures the product is more aligned with their needs and expectations.

  - **Flexibility**: Agile allows teams to adapt to changing requirements and make modifications during development, which is beneficial when project needs are uncertain or evolving.

  - **Transparency and Collaboration**: With daily stand-ups and weekly checkpoints (scrum), there is clear visibility of the project's progress, enhancing collaboration within the team and with stakeholders.

- Cons:

  - **Demanding for Team and Client**: Agile requires active participation from both the team and the client. Regular meetings and feedback sessions can be time-consuming and may require resources that some teams or clients might not be able to commit.

  - **Potential for Scope Creep**: Since Agile is flexible and open to changes, there is a risk of scope creep if the project team does not manage changes carefully. This could lead to extended timelines or additional costs if not controlled properly.

2 Selection of Agile (Scrum) Process

For the development of our MinarMarket e-commerce marketplace project, we have chosen the Agile (Scrum) process with a one-week sprint cycle.

3 Justification for Agile (Scrum) Process Selection

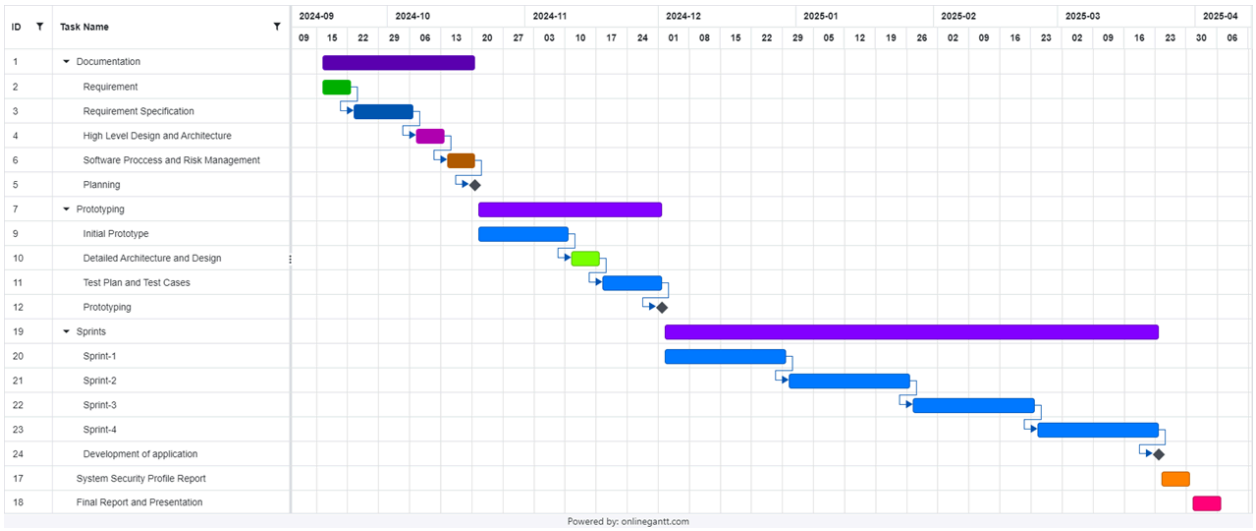
Given the dynamic nature of our project and the need to build a platform that actively engages buyers and sellers, the Agile (Scrum) methodology is the most suitable approach. Our marketplace aims to foster an interactive and responsive environment, which requires continuous iterations and client feedback to refine and improve the user experience.

By adopting Agile, we can work in short, focused sprints, ensuring that we receive feedback frequently from users and stakeholders. This is critical because our platform's success depends on meeting user needs and expectations, and their feedback will help us validate our features and functionality early on. Moreover, Agile's flexibility will allow us to adjust our plans if new requirements or challenges arise, ensuring that the final product is adaptable and optimized for the evolving needs of our target audience.

Additionally, the weekly sprint structure in Scrum aligns perfectly with our Agile approach, allowing us to break down tasks into manageable components that can be completed and reviewed within a short timeframe. This provides transparency to all stakeholders and helps keep the team focused on delivering specific functionalities efficiently.

In contrast, the waterfall approach would not be suitable for our project since it lacks the flexibility required for an evolving and user-driven platform. The risk of reaching the final stages of development only to find that the product does not align with user expectations is too high, making Agile the preferred choice for a dynamic, interactive marketplace.

b. Gantt Chart

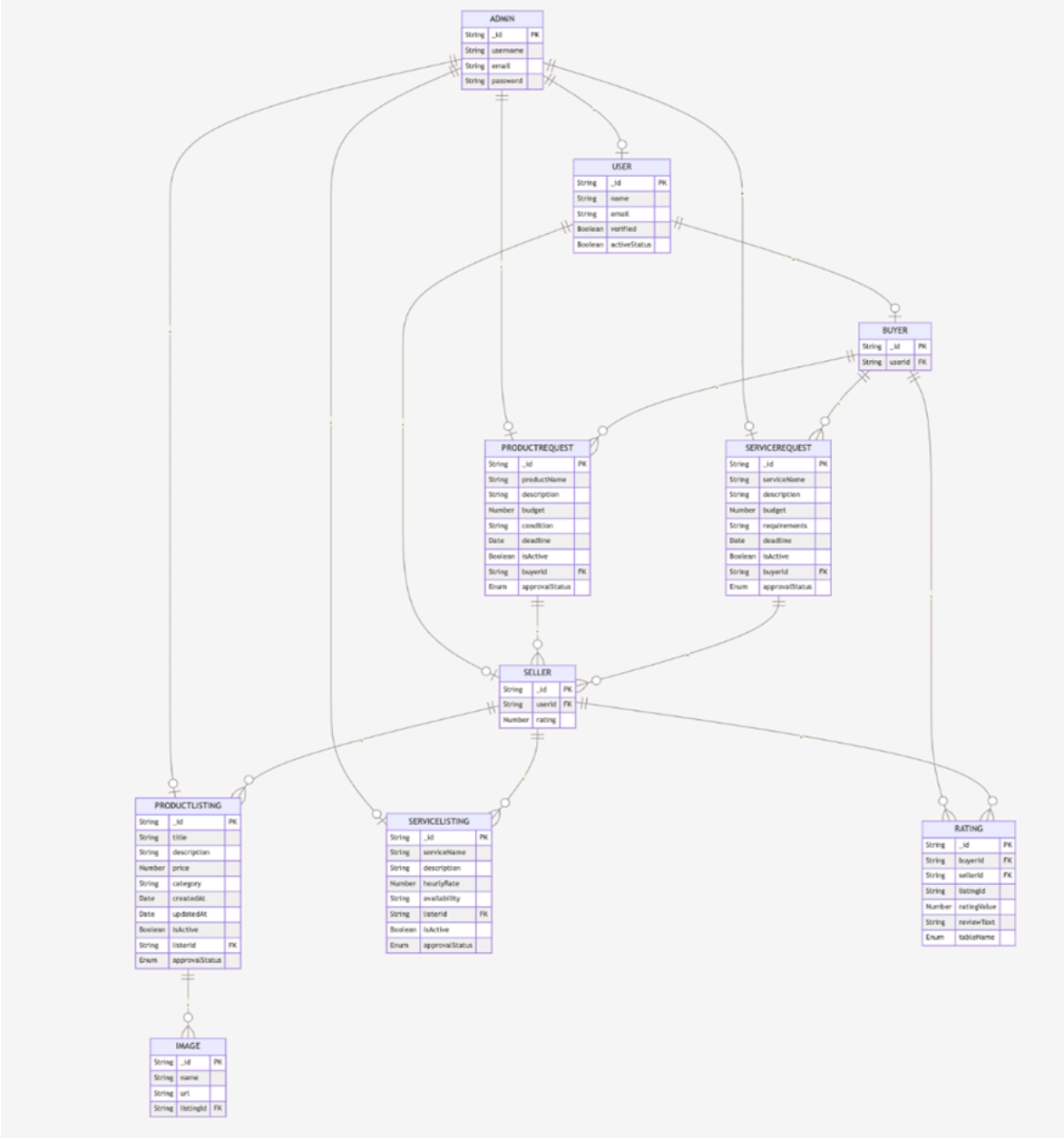


6. Database Design and Web Services

Brief introduction of this chapter in a paragraph

a. Database Design

Link: ER Diagram



Entities and Data Fields

1. USER

- **Purpose**: Represents a generic user (both buyers and sellers).
- **Fields**:
    - _id: Unique identifier for the user (primary key).
    - name: Full name of the user.
    - email: Email address of the user.
    - verified: Boolean indicating if the user's email or identity is verified.
    - activeStatus: Boolean indicating if the user's account is active.

2. ADMIN

- **Purpose**: Represents the admin with elevated privileges to manage the system.
- **Fields**:
    - _id: Unique identifier for the admin (primary key).
    - username: Admin's username.
    - email: Admin's email address.
    - password: Encrypted password for admin authentication.

3. BUYER

- **Purpose**: Represents a buyer account derived from a generic user.
- **Fields**:

- _id: Unique identifier for the buyer (primary key).
- userId: Foreign key linking to the USER table.

4. SELLER

- **Purpose**: Represents a seller account derived from a generic user.
- **Fields**:
  - _id: Unique identifier for the seller (primary key).
  - userId: Foreign key linking to the USER table.
  - rating: Average rating received by the seller.

5. PRODUCTLISTING

- **Purpose**: Represents a product listed by a seller.
- **Fields**:
  - _id: Unique identifier for the product listing (primary key).
  - title: Title of the product.
  - description: Detailed description of the product.
  - price: Price of the product.
  - category: Category to which the product belongs.
  - createdAt: Date and time the listing was created.
  - updatedAt: Date and time the listing was last updated.
  - isActive: Boolean indicating if the listing is active.
  - listerId: Foreign key linking to the SELLER table.

- approvalStatus: Enum to represent the approval state (approved, pending, rejected).

6. SERVICELISTING

- **Purpose**: Represents a service offered by a seller.
- **Fields**:
  - _id: Unique identifier for the service listing (primary key).
  - serviceName: Name of the service.
  - description: Detailed description of the service.
  - hourlyRate: Rate per hour for the service.
  - availability: Description of service availability (e.g., days or hours).
  - listerId: Foreign key linking to the SELLER table.
  - isActive: Boolean indicating if the service listing is active.
  - approvalStatus: Enum to represent the approval state (approved, pending, rejected).

7. PRODUCTREQUEST

- **Purpose**: Represents a request for a product posted by a buyer.
- **Fields**:
  - _id: Unique identifier for the product request (primary key).
  - productName: Name of the requested product.
  - description: Description of the requested product.
  - budget: Maximum budget for the requested product.

- condition: Preferred condition of the product (e.g., new, used).
- deadline: Deadline for fulfilling the request.
- isActive: Boolean indicating if the request is active.
- buyerId: Foreign key linking to the BUYER table.
- approvalStatus: Enum to represent the approval state (approved, pending, rejected).

8. SERVICEREQUEST

- **Purpose**: Represents a request for a service posted by a buyer.
- **Fields**:
  - _id: Unique identifier for the service request (primary key).
  - serviceName: Name of the requested service.
  - description: Detailed description of the requested service.
  - budget: Maximum budget for the service.
  - requirements: Additional requirements or details for the service.
  - deadline: Deadline for fulfilling the request.
  - isActive: Boolean indicating if the request is active.
  - buyerId: Foreign key linking to the BUYER table.
  - approvalStatus: Enum to represent the approval state (approved, pending, rejected).

9. IMAGE

- **Purpose**: Represents images associated with a product or service listing.

- **Fields**:
    - _id: Unique identifier for the image (primary key).
    - name: Name of the image file.
    - url: URL or path to access the image.
    - listingId: Foreign key linking to the PRODUCTLISTING or SERVICELISTING table.

10. RATING

- **Purpose**: Represents a rating and review for a seller or a specific listing.
- **Fields**:
    - _id: Unique identifier for the rating (primary key).
    - buyerId: Foreign key linking to the BUYER table (who gave the rating).
    - sellerId: Foreign key linking to the SELLER table (who received the rating).
    - listingId: Links to the listing the rating pertains to (product or service).
    - ratingValue: Numerical value of the rating (e.g., 1 to 5).
    - reviewText: Text review accompanying the rating.
    - tableName: Enum indicating whether the rating is for a product or service listing.

Relationships

- **ADMIN** has control over the **BUYER**, **SELLER**, **PRODUCTLISTING**, **SERVICELISTING**, **PRODUCTREQUEST**, and **SERVICEREQUEST** entities, with the ability to suspend, delete, and approve/reject.
- **SELLER** creates **PRODUCTLISTING** and **SERVICELISTING**.
- **BUYER** posts **PRODUCTREQUEST** and **SERVICEREQUEST**.
- **PRODUCTLISTING** can have associated **IMAGE** entries.
- **BUYER** gives **RATING** to **SELLER** based on their experience.
- **SELLER** responds to **PRODUCTREQUEST** and **SERVICEREQUEST**.

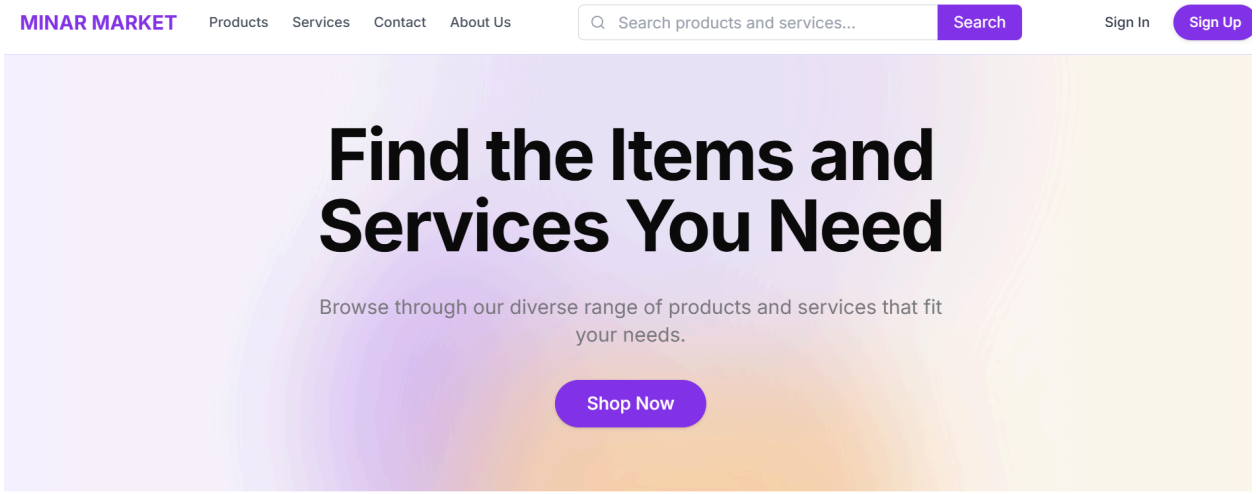
b. API Specification

No External APIs were used in this project.

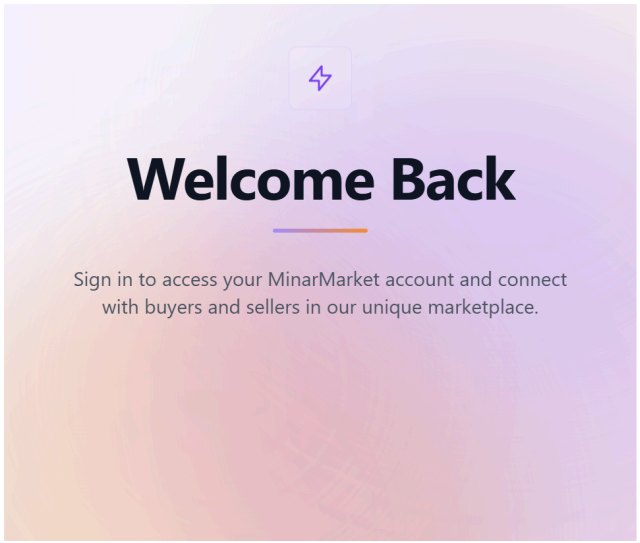
7. System User Interface

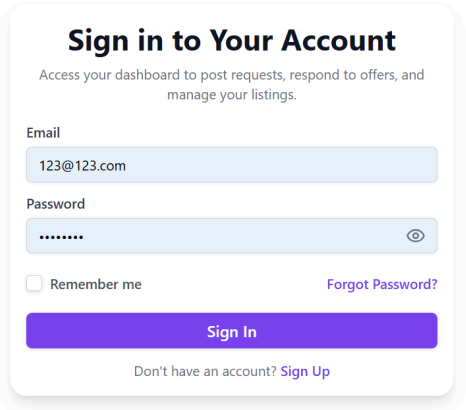
Brief introduction of this chapter in a paragraph

This sub-section should explain the functionality of your application to the end user with supporting screenshots.

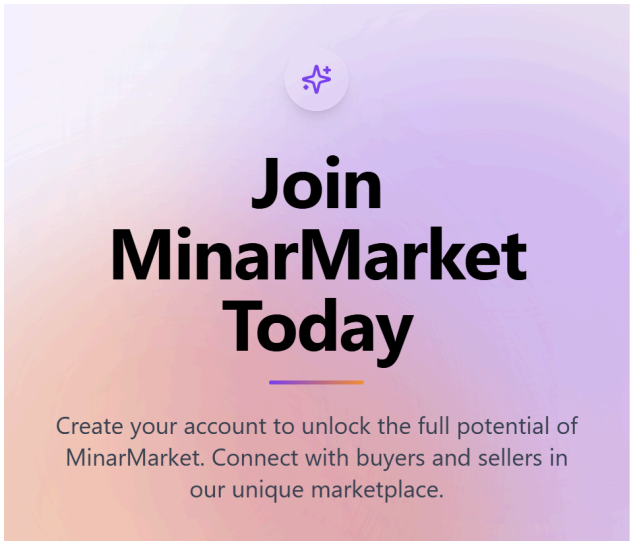


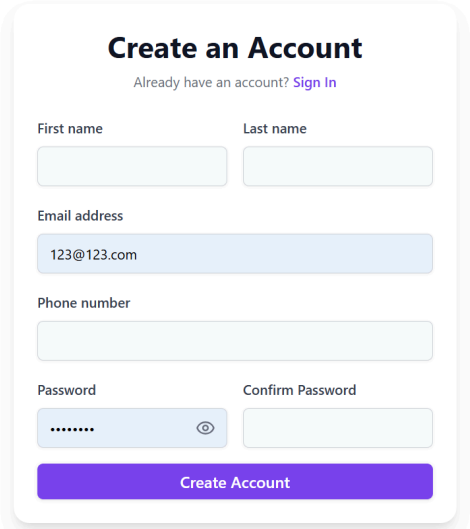
This is the landing page of the website. This page is accessible to all kinds of traffic, but to dive deeper into the website, you have to login.

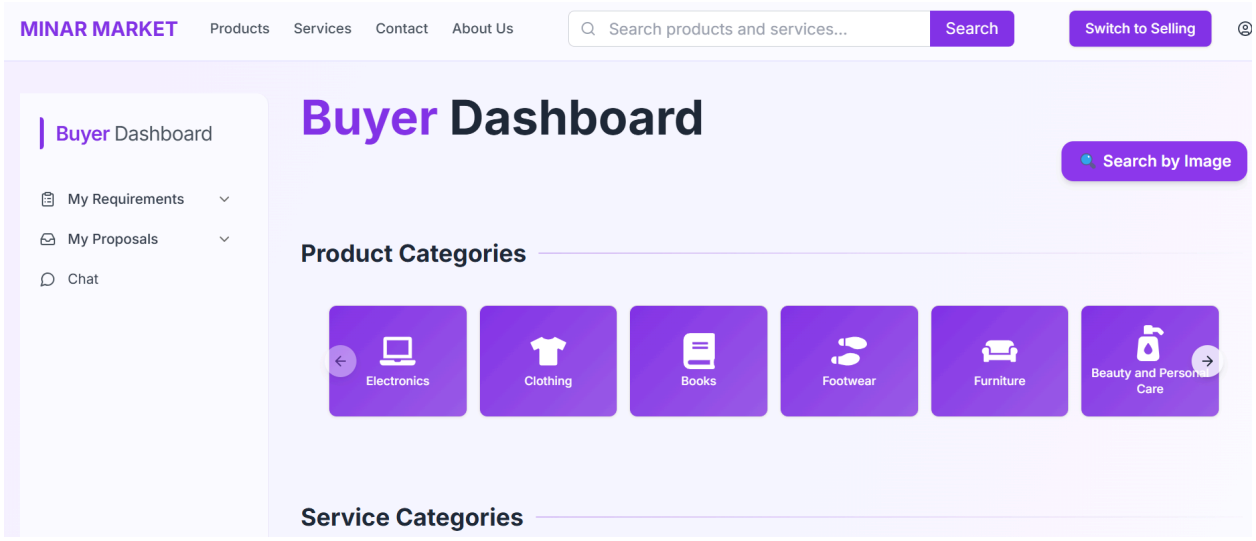




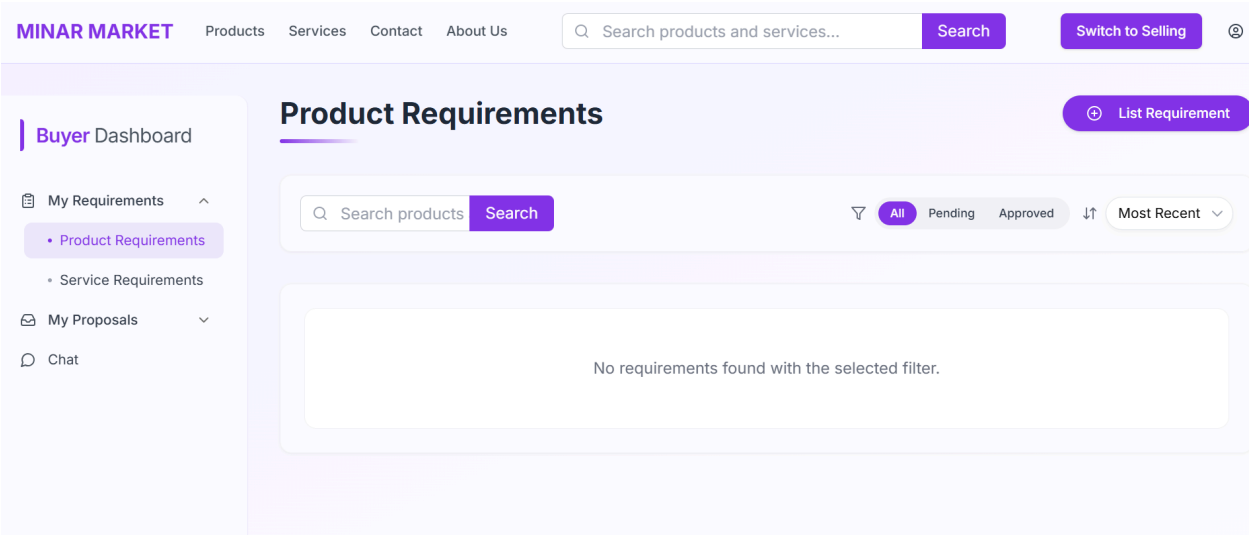
This is the login screen. With an intuitive frontend & UI/UX design. The login is safely secured using JWT and passwords are saved securely using bcrypt hashing.



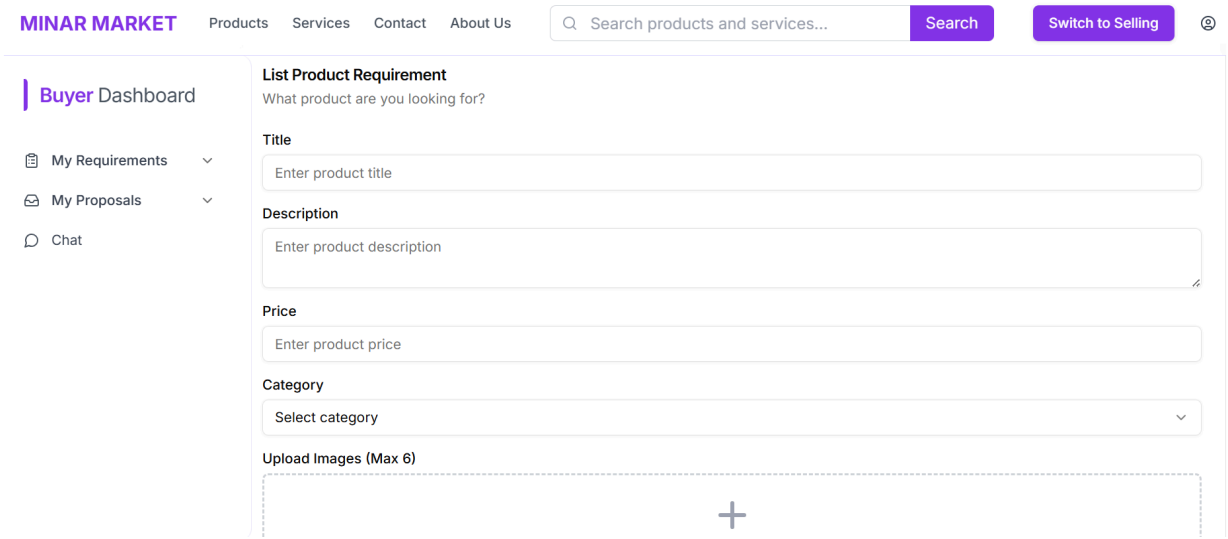




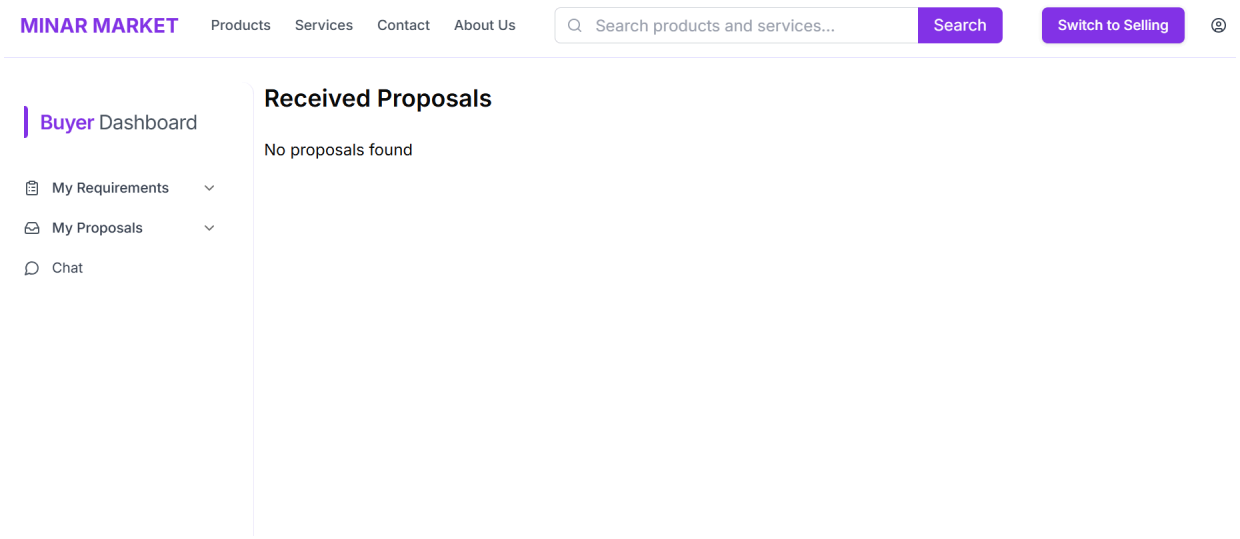
This is the Buyer Dashboard from where the user can navigate to the entire website. The buyer can look up different products and services. The user can also post their own requirements or see the proposals sent by the sellers themselves.



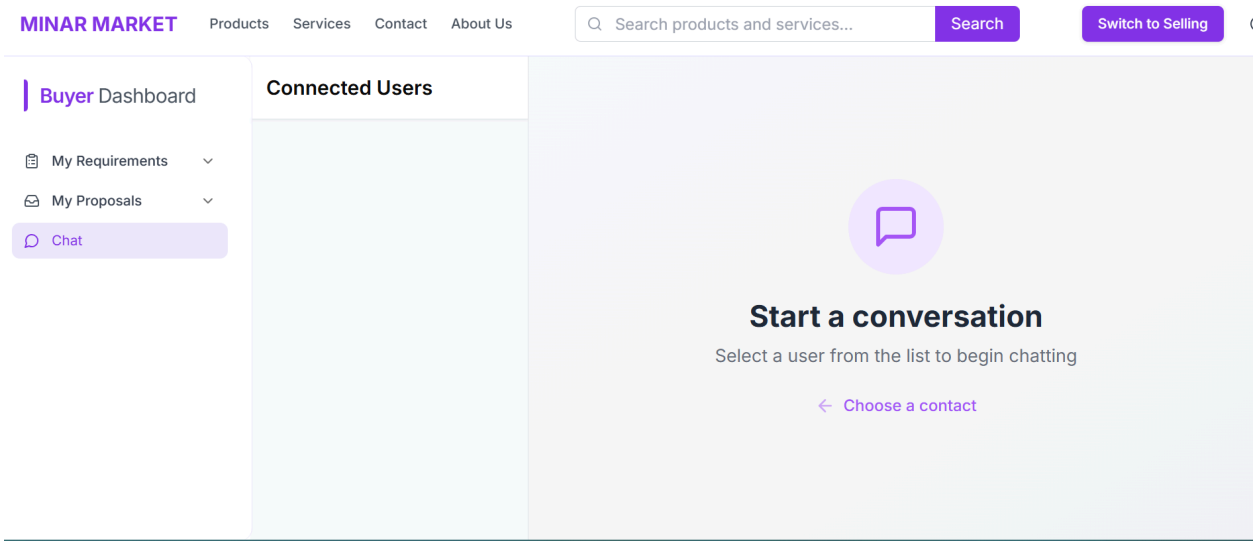
Here lies the USP of MinarMarket. A buyer sided website which allows the user to post their reqeuirements and let sellers approach them directly with their products or services.



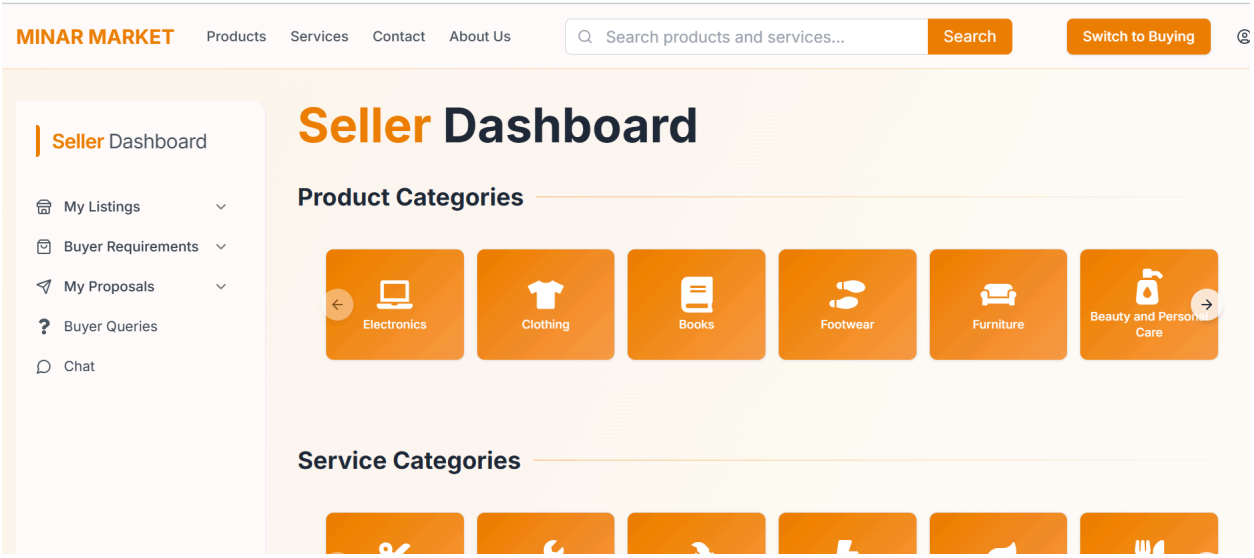
The form for the Product Requirement, which takes in comprehensive information for the seller to use and send their proposals.



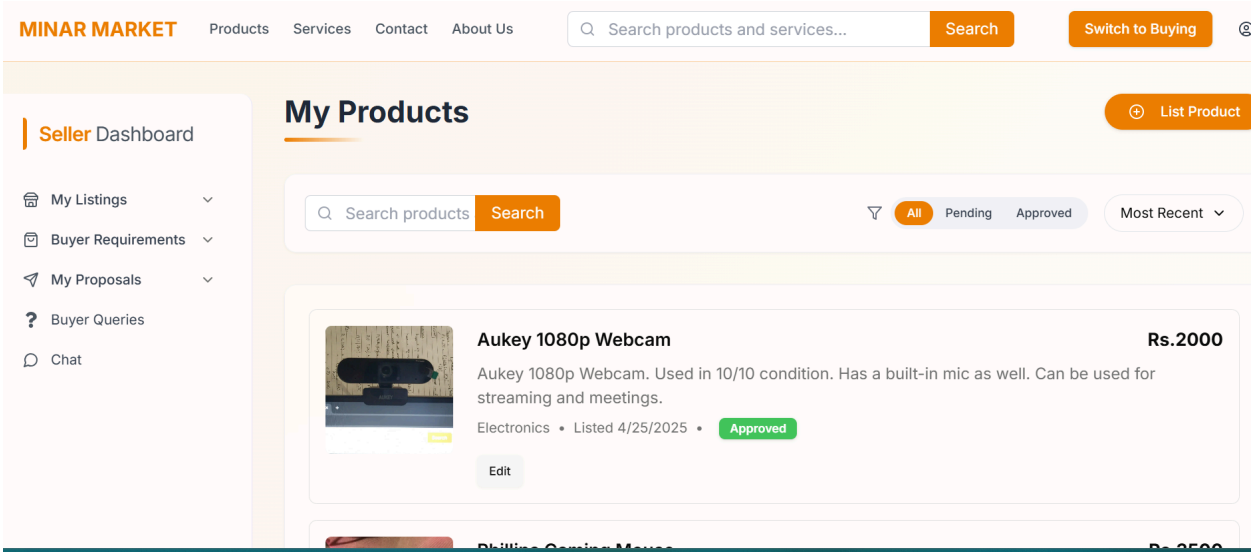
Here, the buyer can view the proposals they receive from sellers. After accepting it, the user can also chat with the seller



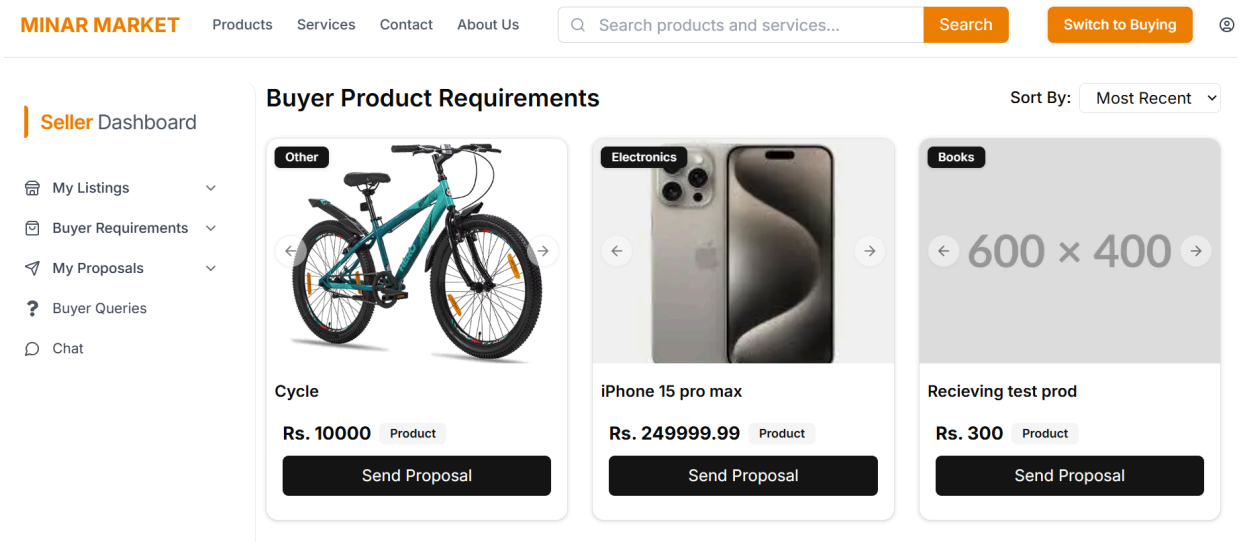
This is the chatting window where both the users can chat and exchange information about the product or service being sold.



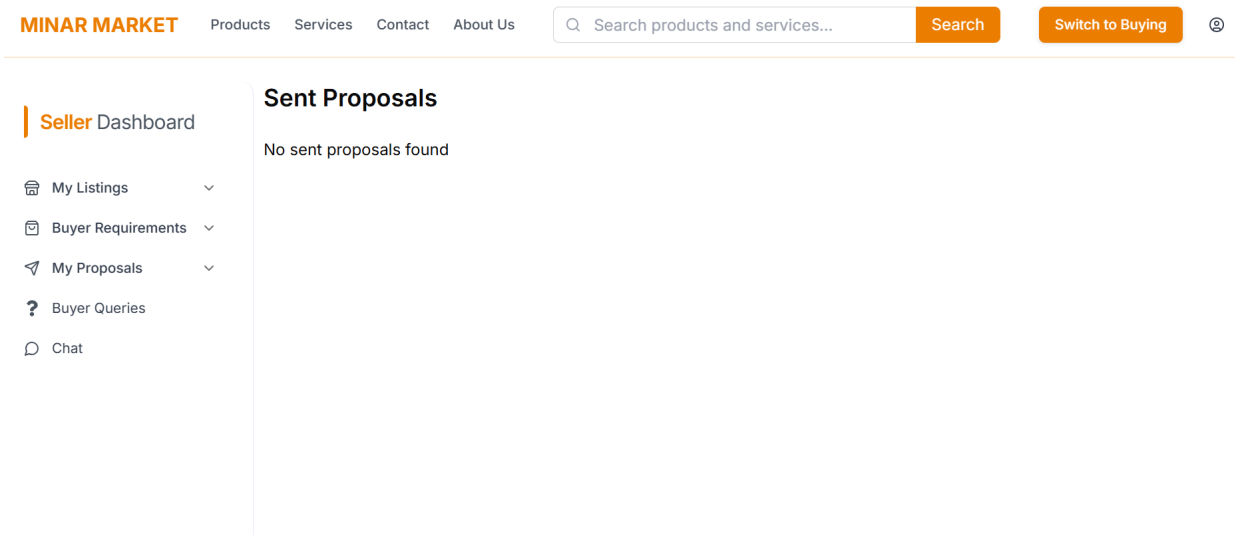
This is the Seller Dashboard which allows the usesr to become a seller and list the products or services they wish to sell.



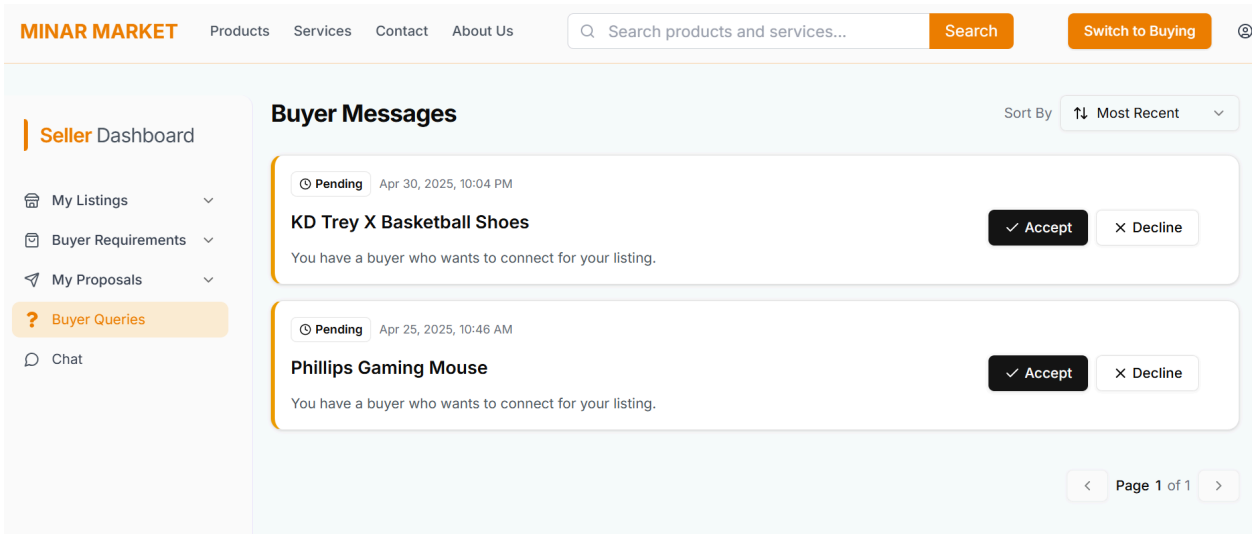
This is the My Products section where the products are shown which the seller wants to sell on MinarMarket. The Products have 3 different statuses, depending upon the approval of the Admin.



Here, the sellers can view the Requirements posted by the buyers, which they can send a proposal to. Proposals can only be sent of products which are already listed and approved.



This section includes the already sent proposals by the seller. It helps keep track of the sent proposals and their statuses as well.



This screen contains the messages from the buyers, for which they approach the sellers. After pressing accept, the seller is redirected to the chat window with that buyer.

8. Project Security

Security is a critical component of any web-based system, especially in platforms that handle user data, transactions, and sensitive business logic like *MinarMarket*. As a buyer-seller marketplace, our system collects personal information, stores product/service listings, supports real-time chat, and may involve financial operations in future iterations. This section outlines the potential threats, their associated risks, mitigation strategies, and tools employed to ensure a secure and reliable application environment.

a. Project Threats

The following security threats were identified for *MinarMarket*:

1. **Unauthorized Access:** Users attempting to access unauthorized sections (e.g., regular users accessing admin features).
2. **Injection Attacks (SQL/NoSQL):** Potential threats of malicious data being used to manipulate backend queries.
3. **Data Breach:** Compromised user data due to weak authentication or insecure data storage.
4. **Denial of Service (DoS):** Overloading the platform with fake requests to disrupt normal functioning.
5. **Broken Authentication:** Compromised or stolen credentials allowing account hijacking.

b. Potential Losses

Unauthorized Access

- Data leakage
- Reputation damage
- Potential legal issues

Injection Attacks

- Data corruption
- System compromise
- Service outages

Data Breach

- Legal penalties (e.g., violations of data protection laws)
- User churn
- Damage to brand reputation

Denial of Service (DoS) Attacks

- Temporary business outage
- Loss of user trust
- Financial losses

Broken Authentication

- Account theft
- User impersonation
- Unauthorized transactions

c. Security Controls

1. **Input Validation** *(Protective)*
   Sanitize all user inputs on both client and server sides to prevent XSS and injection attacks.
2. **Role-Based Access Control (RBAC)** *(Protective)*
   Restrict access based on user roles (e.g., admin, buyer, seller).

3. **JWT Authentication** *(Protective)*

   Use JSON Web Tokens for secure, stateless authentication.

4. **HTTPS Encryption** *(Protective)*

   Enforce HTTPS across the site to secure all data transmission.

5. **Password Hashing** *(Protective)*

   Securely store passwords using industry-standard hashing algorithms.

6. **Scheduled Backups** *(Recovery)*

   Regularly back up the database to enable recovery in case of data loss or corruption.

7. **Admin Approval of Listings** *(Protective / Detective)*

   Require administrator approval for new listings to prevent spam, fraud, or malicious content from being published.

d. Static and Dynamic Security Scanning Tools

Explore and select **static and dynamic security scanning tools** for your project. The tools should be selected considering languages and technologies being used in your project.

9. Risk Management

Sr.	Risk Description	Mitigation Strategy
1.	**Deployment Issues**: Errors during deployment or incorrect environment setup.	Perform a spike and research (RnD) on deployment tools beforehand. Use Docker to ensure consistency in environments.
2.	**Scope Creep**: Uncontrolled changes that increase project scope.	Regularly review project goals and adhere to Agile sprint planning to limit changes. Prioritize tasks and features.
3.	**Inadequate Testing**: Bugs or issues discovered late in the process.	Implement continuous integration with automated unit and integration testing. Perform testing at each sprint review.
4.	**Security Vulnerabilities**: Potential for data breaches or malicious attacks.	Conduct security audits and implement encryption and best security practices (e.g., HTTPS, secure database storage).
5.	**Poor User Adoption**: Users finding the platform difficult to use or not meeting expectations.	Conduct user research and usability testing to gather feedback and adjust the user interface based on user behavior.
6.	**Performance Issues**: The platform becomes slow or unresponsive with increasing users.	Optimize backend queries, implement load balancing, and conduct performance testing under simulated high traffic.

7.	**Data Loss**: Accidental loss of data due to technical or human error.	Regularly back-up databases and set up recovery strategies using automated cloud backups and version-controlled environments.
8.	**Insufficient Scalability**: The system architecture may not handle higher-than-expected loads or scaling to a larger user base.	Design the system with scalability in mind, using cloud-based services and load testing tools. Incorporate microservices and containerization (e.g., Docker).
9.	**Hardware/Software Compatibility Issues**: Compatibility problems between different development environments, software, or hardware.	Use containerization (e.g., Docker) and ensure standardized development environments. Conduct early testing on all target environments.
10.	**Time Management**: Inability to meet deadlines due to mismanagement of time or underestimation of task complexity.	Create a detailed Gantt chart and regularly monitor progress. Break down tasks into smaller milestones with clear deadlines.

10. Testing and Evaluation

To ensure the reliability and correctness of our system, we adopted a structured testing approach based on use-case-driven scenarios. Each core functionality was individually tested using well-defined test cases, designed to validate the application's behavior under both normal and edge conditions.

Testing Strategy

Our testing strategy followed a manual black-box testing approach where the focus was on evaluating the output based on various input conditions without examining the internal code structure. The test cases were created and executed for each critical user-facing feature of the system.

The process involved:

- Identifying key user actions and workflows.
- Writing test cases with defined objectives, preconditions, and expected outcomes.
- Executing the tests on a controlled platform (Windows 10 using Google Chrome).
- Recording actual outcomes and validating them against expectations.

Sample Test Cases

Here are a few examples of the test cases we created and executed:

Use Case 1 – Buyer can contact Seller

- **Test Case Objective:** Verify that a buyer can contact the seller to inquire about a product.
- **Priority:** High
- **Preconditions:** The user does not need to have an account.
- **Postconditions:** Message is sent successfully to the seller.
- **Tester:** M. Umer Jamil

Use Case 3 – Resolving User Complaints

- **Test Case Objective:** Ensure that the admin or support system can view and resolve user complaints.
- **Priority:** High
- **Preconditions:** The user must be logged in and must have submitted a complaint earlier.
- **Postconditions:** Complaint is marked as resolved or forwarded to the concerned party.
- **Tester:** Abdul Ahad Bin Ali

Use Case 5 – Seller can respond to Buyer

- **Test Case Objective:** Confirm that a seller is able to respond to buyer queries.
- **Priority:** High
- **Preconditions:** No login required.
- **Postconditions:** The buyer receives the seller's response.
- **Tester:** M. Umer Jamil

Each of these test cases was created with a specific user action in mind and validated accordingly during sprint 2 of development.

Tools Used

All tests were executed manually in this sprint. We did not use any automation testing tools for this phase. However, browser-based developer tools (Google Chrome DevTools) were utilized for inspecting UI behavior, form validation, and console errors.

11. Deployment Guidelines

List down the steps for deployment of your system. Start from where the code (link of the github repository) should be picked and then mention all the steps for deployment in a production

environment. Also mention the online link where your application is hosted along with access information (user/password etc.).

Domain Provider: GoDaddy

Deployment Platform: Hostinger VPS (KVM-2)

Production Domain: www.minarmarket.com

Source Code Repository: https://github.com/khurrumchaudhrty/MinarMarket_Development

Admin Login Credentials:

Email: **khurrum@gmail.com**

Password: **1122@1122**

For regular user accounts (buyers and sellers), users can sign up manually through the application interface.

Deployment Steps

The deployment of the Minar Market MERN (MongoDB, Express, React, Node.js) full-stack application on a production environment was carried out using Hostinger's VPS hosting services and a custom domain purchased from GoDaddy. The complete process, including server configuration, frontend and backend setup, and deployment, is detailed below.

1. Domain Configuration via GoDaddy

To begin, the domain name was configured to point toward the Hostinger VPS by setting up DNS records:

- Logged into the GoDaddy account.
- Navigated to the **DNS settings** for the domain.
- Created or edited the **A record**:

    - **Record Type:** A
    - **Name:** @
    - **Value:** The public IP address of the VPS (e.g., `91.108.102.224`).
    - **TTL:** Left unchanged (default value).
- Saved the DNS settings.

For further clarification, this process was followed as shown in the tutorial video:
DNS Setup Tutorial - Hostinger Academy

2. Preparing the VPS Environment (Hostinger VPS)

Once the domain was properly linked to the server:

- Logged into the **Hostinger VPS panel** and copied the **IP address** from the VPS information section.
- Connected to the VPS terminal using SSH:
    - `ssh root@91.108.102.224`
- Entered the **root password** to gain access to the VPS terminal.
- This allowed direct interaction with the server for all subsequent steps.

Reference tutorial followed:
Deploy MERN on VPS - GreatStack

3. Setting Up the MongoDB Database (Optional)

Setting up MongoDB on the VPS was an optional step. If needed, an external tutorial was available and referenced for installing MongoDB locally on the VPS. However, it was possible to use external database services as well.

4. Deploying the Backend (Express and Node.js)

The backend of the Minar Market application is built using Express and Node.js. The following deployment steps were carried out:

- Cloned the backend code from GitHub: https://github.com/umerjamil28/MinarMarket_Development.git
- Navigated into the backend directory.
- Installed necessary packages using `npm install`.
- The backend server was restarted using the service manager after any changes.

5. Deploying the React Frontend

The frontend is built with React and consists of separate applications for different user roles. The following steps were followed:

- Created a production build of the React frontend using `npm run build`.
- Ensured frontend packages were installed using `npm install`.
- Synced the frontend build to the correct location on the server using a custom sync script:
    - If synchronization was successful, proceeded.

    - If not, executed `./sync.sh` to fix the sync.

- Finally, the frontend was rebuilt and the Nginx server was restarted to serve the latest build.

6. Configuring Nginx as a Reverse Proxy

To efficiently manage routing and improve application performance, Nginx was used as a **reverse proxy** instead of Apache.

- Nginx handles reverse proxying by forwarding requests to the appropriate backend or frontend services.

- This setup helps in caching frequent requests, reducing server load, and increasing responsiveness.

- Two methods were attempted:

    1. **Symbolic link method** (as per tutorial): This approach did not work due to an unknown issue.

    2. **rsync method**: This method was successful and was used to complete the reverse proxy setup.

Tutorial used for Nginx setup: `rsync-nginx`.

7. Setting Up SSL Certificates

For secure communication over HTTPS, SSL certificates were set up. This process was carried out as part of the final steps of the deployment to ensure encrypted data transmission and user trust.

Final Notes on Deployment

After setting up the system on the VPS, the deployment workflow has become permanent and streamlined. For each deployment or update, the following quick steps are executed:

For Backend:

- Navigate to the backend directory.
- Run `npm install` to ensure all dependencies are updated.
- Restart the backend service.

For Frontend:

- Navigate to the frontend directory.
- Run `npm install`.
- Check if the sync is valid:
    - If synced, continue.
    - If not, execute `./sync.sh`.
- Run `npm run build` to generate the production build.
- Restart the Nginx server to reflect the changes.

Production Access

- **Website Link:** www.minarmarket.com
- **Admin Panel Login:**
    - **Email:** khurrum@gmail.com
    - **Password:** 1122@1122
- **Buyer/Seller Access:** Users can register on the website to use the application as buyers or sellers.

This comprehensive deployment ensures that Minar Market is running in a stable, production-ready environment, leveraging the flexibility and power of VPS hosting and full-stack web technologies.

12. Conclusion

a. Summary

Throughout the development of *MinarMarket*, we set out to build a dynamic and user-centric e-commerce platform that bridges the gap between buyers and sellers through a reverse marketplace model. Our approach emphasized iterative development using the Agile (Scrum) methodology, which allowed us to continuously incorporate feedback, prioritize critical features, and adapt to evolving requirements. From user research and UI/UX design in Figma to the implementation of a full-stack solution using the MERN stack, our team collaborated to deliver a scalable and intuitive platform. This journey not only strengthened our technical proficiency in frontend and backend development but also enhanced our skills in team coordination, agile planning, and system deployment in production environments.

b. Challenges

We encountered several technical and non-technical challenges during the course of this project:

c. **Integration Complexity:** Coordinating the interaction between the frontend and backend, especially in the chat system and real-time offer updates, required meticulous debugging and API consistency.

d. **Deployment Issues:** Initial deployment attempts on VPS hosting faced setbacks due to environment mismatches, Nginx misconfigurations, and synchronization problems between frontend builds and backend services. These were resolved through containerization practices, rsync methods, and repeated testing on Hostinger.

e. **Team Coordination:** Synchronizing efforts during parallel development and ensuring smooth code integration across sprints was a constant challenge. We addressed this through regular Scrum meetings, use of Git for version control, and documenting all tasks clearly on

project boards.

f. **Security and Access Control:** Implementing role-based access (e.g., admin vs regular users) and ensuring secure handling of sensitive data required careful planning. We incorporated input validation, HTTPS, and secure password handling mechanisms.

g. Future

The current version of *MinarMarket* lays the foundation for a feature-rich and scalable marketplace, but several enhancements can be introduced in future iterations:

- **AI-Powered Matching:** Implementing machine learning algorithms to match buyers with the most relevant sellers based on past activity and preferences.
- **Mobile Application:** Developing native mobile apps for Android and iOS to increase accessibility and improve user experience.
- **Payment Gateway Integration:** Adding secure and reliable online payment methods for smoother transactions.
- **Advanced Admin Dashboard:** Providing more granular analytics, user behavior insights, and platform health indicators to the admin for strategic decision-making.
- **Multi-language and Regional Support:** Expanding usability by localizing content and supporting users across different regions and languages.

Overall, this project has not only met its original objectives but also opened up new opportunities for innovation in demand-driven e-commerce platforms.

13. Review checklist

Before submission of this report, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

Chapter/Section Name	Reviewer Name(s)
1-3	Abdul Ahad
4-6	Hasan Malik
7-9	M. Umer Jamil
9-11	Saad Ilyas
12	Aniqa Aqeel