**High level Architecture**

**P02:MinarMarket**

| **Student ID** | **Name** |
| --- | --- |
| **25100016** | **Abdul ahad bin ali** |
| **25100017** | **M. Umer Jamil** |
| **25100211** | **Hasan Malik** |
| **25100181** | **M. Saad Ilyas** |
| **25100257** | **Aniqa Aqeel** |

**Table of Contents**

[1.](#_gjdgxs) Introduction 3

[2.](#_30j0zll) System Architecture 4

[2.1 Architecture Diagram 4](#_1fob9te)

[2.2 Architecture Description 5](#_3znysh7)

[2.3 Justification of the Architecture 6](#_2et92p0)

[3.](#_tyjcwt) Tools and Technologies 7

[4.](#_3dy6vkm) Hardware Requirements 8

[5.](#_1t3h5sf) Who Did What? 9

[6.](#_4d34og8) Review checklist 9

# Introduction

In the rapidly evolving world of e-commerce, the relationship between buyers and sellers is continuously being redefined. Traditional online marketplaces tend to operate in a seller-centric manner, where sellers list their products, and buyers browse through these listings to make their purchases. While this model has proven effective in many scenarios, it often leaves buyers with limited options when they have specific needs that don’t align perfectly with the available listings. As consumers increasingly demand personalization and convenience, there is a growing need for a marketplace that addresses this limitation and fosters a more collaborative relationship between buyers and sellers.

Our project introduces an innovative approach to online marketplaces by creating a platform that not only allows sellers to list their products but also gives buyers the power to post their specific requirements. This dual functionality transforms the marketplace into a more dynamic and interactive ecosystem where buyers actively express their needs, and sellers can respond by offering products that meet those exact requirements. This model reduces the gap between supply and demand, enabling sellers to more effectively target interested buyers and ensuring buyers find the products that truly fit their preferences.

The core objective of this project is to design and implement a marketplace that enhances the traditional e-commerce experience. Buyers will no longer be confined to searching through predefined listings but can instead list the products or services they are looking for. Sellers, in turn, will have visibility into these buyer requests and can engage directly with potential customers by offering relevant products or negotiating terms that meet the buyer’s expectations. This two-way interaction fosters a marketplace that is more responsive, transparent, and efficient.

A key focus of our project is to simplify and streamline communication between buyers and sellers. The platform will feature an intuitive user interface that allows both parties to post, search, and communicate with ease. Buyers will be able to track the offers they receive in response to their requests, compare different sellers, and make informed purchasing decisionUltimately, this project aims to revolutionize ths based on personalized recommendations. Sellers will benefit from real-time notifications of buyer requests that match their inventory, allowing them to act quickly to meet demand. This buyer-driven interaction introduces a new level of personalization and convenience, benefiting both parties involved in the transaction.

Our target audience for this platform includes individual consumers, small businesses, and larger enterprises. Individual consumers will appreciate the ability to request highly specific products, while businesses can leverage the platform to source bulk orders or specialized items.

Additionally, the platform can serve niche markets where product availability may be limited, empowering buyers with greater choice and sellers with direct access to a motivated customer base.

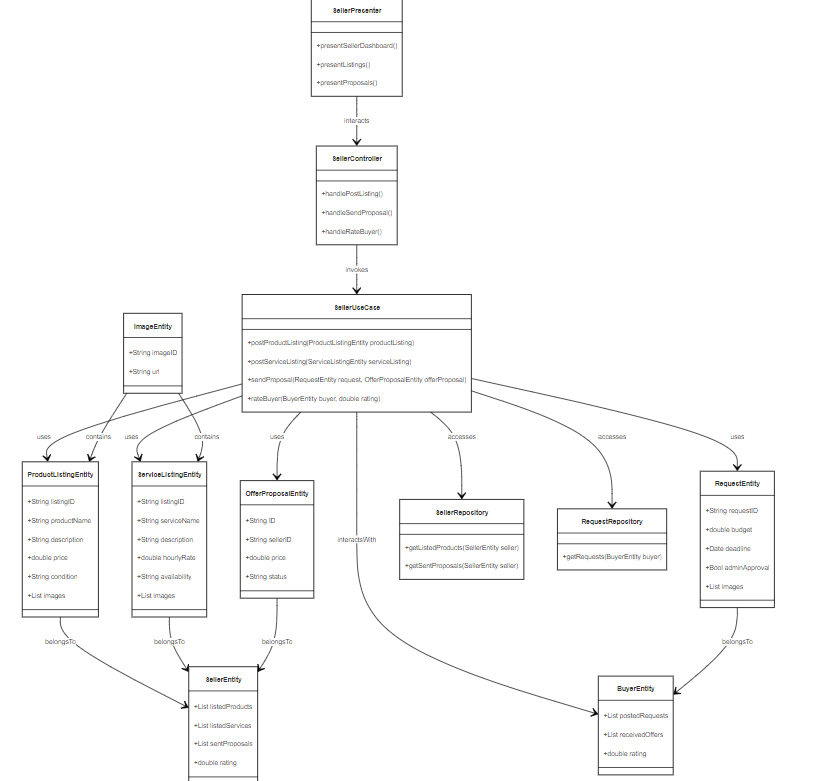
Furthermore, our marketplace is designed with scalability and flexibility in mind. As the platform grows, we plan to incorporate advanced features such as AI-driven product matching, where algorithms analyze buyer requests and suggest potential matches from a seller’s inventory. This feature will streamline the offer-making process for sellers and make it easier for buyers to receive relevant product suggestions. The system will also allow for future integrations with payment gateways, shipment tracking, and customer review systems to create a comprehensive and seamless e-commerce experience.

e traditional marketplace model by making it more buyer-driven, interactive, and efficient. By bridging the gap between buyer needs and seller offerings, our platform will create a more engaging and fulfilling experience for all users. This approach will not only improve transaction success rates but also foster stronger relationships between buyers and sellers, setting a new standard for online commerce in the modern digital era.

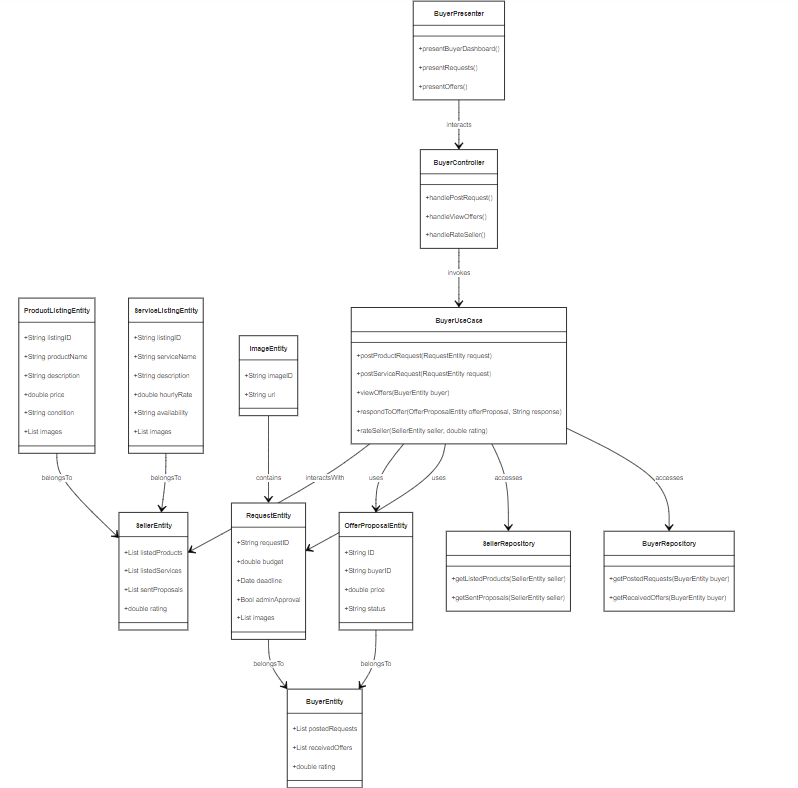
# System Architecture

## Architecture Diagram

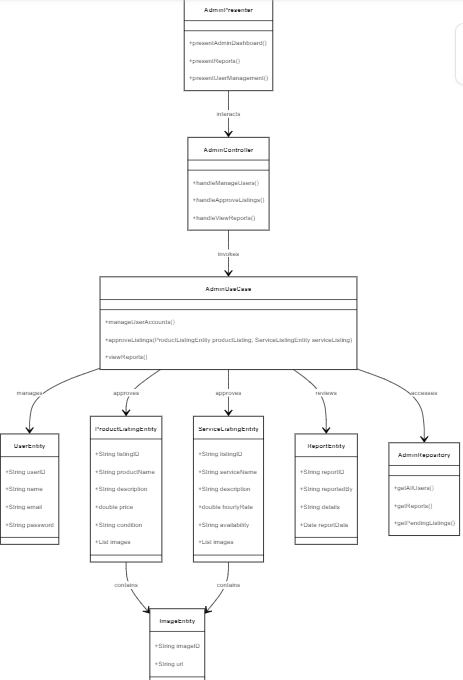
**Seller’s Use Case Architecture:** [LINK TO THE FOLLOWING DIAGRAM](https://www.mermaidchart.com/app/projects/4d87b42f-0db1-4c32-b583-bd8b16ca0fe4/diagrams/dfcc5296-f2b8-4921-ae3e-6035b17b5f42/version/v0.1/edit)



**Buyer’s Use Case Architecture:** [Link for the following Diagram](https://www.mermaidchart.com/app/projects/0c2ab960-f537-4aec-b945-3c65be4df284/diagrams/3bbe2885-d4f4-4e47-9a88-92d6fb99873a/version/v0.1/edit)



**Admin’s Use Case Architecture:** [Link for the following Diagram](https://www.mermaidchart.com/app/projects/4d87b42f-0db1-4c32-b583-bd8b16ca0fe4/diagrams/3a9d3a99-3c7c-46fd-9475-f2d7f32d32a8/version/v0.1/edit)

****

## Architecture Description

### **Description of Each Subsystem in the Architecture:**

1. **Front-End Subsystem:**
   * **UI Layer:** Consists of various user interfaces for the admin, seller, and buyer, along with login and landing pages. Each page is backed by its respective view model that handles user input and interactions. The front-end will have multiple view models (e.g., for API integration and pending actions). The view model wraps Data Transfer Objects (DTOs) to ensure that only relevant data is shown on the UI.
   * **View Models:** These are the middle layer between the user interface and backend logic, responsible for handling user requests, managing UI state, and packaging responses.
   * **Use Case Layer:** The third layer handles the main business logic. This is where different use cases (e.g., posting an ad, login, etc.) are implemented. The use cases communicate with the repository to fetch and process data.
   * **Repository Layer:** The repository is responsible for managing data from both local storage (offline capability) and remote API. It abstracts the source of data and ensures smooth communication between the front-end and back-end layers.
2. **Back-End Subsystem:**
   * **Application-Based Architecture:** The backend is designed to serve the front-end with the necessary data and logic. It follows a layered approach and includes functionalities for handling different use cases such as login, posting ads, and fetching user data.
   * **Session Management:** A session controller is integrated to ensure secure and continuous interactions. Before making an API call, the session's validity is checked. If the session is valid, the system proceeds, otherwise, the session is terminated.
   * **Repository:** On the backend, the repository serves as a gateway between the application and the data (both local and remote). The repository handles data aggregation, mapping DTOs to entities, and providing the necessary data to use cases.
3. **Data Handling Subsystem:**
   * **Entity/DTO Layer:** This layer converts backend data into a format that is consumable by the front-end. DTOs are used to reduce the exposure of unnecessary data to the UI, making sure only what’s needed for the specific use case is passed.
   * **Database (DB):** The master database is the single source of truth, storing all essential information. However, the system also has limited offline capabilities via local storage. This ensures the app can still function with basic features if the internet connection is unavailable.

### **Interaction Between Subsystems:**

* **Front-End Interaction:** The front-end UI interacts with the view models, which in turn invoke use cases depending on user actions. The view model ensures the proper DTOs are passed to the use case layer.
* **Use Case Layer Interaction:** The use cases, based on the user input or API calls, interact with the repository layer to fetch data. The use cases are the central logic hub where business logic is executed.
* **Repository Layer Interaction:** The repository aggregates data either from the local data source (offline) or the remote API. It ensures data consistency across the front-end and back-end by mapping DTOs to entities and vice versa.
* **Back-End Interaction:** The repository on the back-end interacts with the database and API to store, retrieve, and update data. In case of an API call, session management ensures secure access. Data from the back-end is transformed into DTOs before being passed to the front-end.

## Justification of the Architecture

### **Pros and Cons of the Architecture:**

#### **Pros:**

1. **Offline Functionality:** The ability to work with local data when the internet connection is not available increases user convenience and app reliability. The repository abstracts this complexity, ensuring the user can perform basic tasks offline.
2. **Scalability:** The architecture is designed with scalability in mind, making it easier to integrate additional functionalities like AI-driven product matching, payment gateways, and shipment tracking.
3. **Modularity:** The separation of layers (UI, view models, use cases, repositories) ensures a clean and maintainable codebase. Each component is responsible for a distinct part of the functionality, making future modifications easier.
4. **Security:** Session management ensures secure interaction between the client and server, preventing unauthorized access to APIs.
5. **Efficient Communication:** Using DTOs and entities ensures that only necessary data is exposed to the front-end, improving performance and reducing unnecessary data transmission.
6. **Real-time Notifications:** Sellers receive real-time notifications when buyer requests match their inventory, leading to faster response times.

#### **Cons:**

1. **Complexity:** The multi-layered architecture adds some complexity to the development process. It requires careful design and management of view models, DTOs, and repositories.
2. **Data Synchronization:** Managing local and remote data can lead to potential synchronization issues, especially when dealing with real-time data or if there’s a network interruption.
3. **Session Dependency:** If session management fails or is not implemented efficiently, the system could face issues with authentication and continuity in user interactions.

### **Justification for Architecture and Non-Functional Requirements**

#### **Justification:**

This architecture is ideal for an interactive marketplace platform that demands scalability, flexibility, and reliability. The separation of concerns ensures that future changes and updates to any component (like UI, APIs, or database structure) will not require rewriting the entire system. Additionally, the repository model provides a robust data management solution that ensures consistent data even when offline.

#### **Non-Functional Requirements:**

1. **Scalability:** The architecture supports scalability due to its modularity and the ability to add future integrations (e.g., AI product matching, payment gateways). This ensures that as the platform grows, the system can handle increased traffic and additional features.
2. **Performance:** The use of DTOs minimizes the data transferred between layers, improving performance. The system's ability to switch between local and remote data sources ensures that the app remains responsive even with intermittent internet connectivity.
3. **Maintainability:** By keeping concerns separated in distinct layers (UI, view models, use cases, repositories), the code is more maintainable. Changes in one part of the system (e.g., the database) will have minimal impact on others.
4. **Security:** Session management ensures that only valid users can access the system’s resources. This is essential for e-commerce platforms where sensitive information (e.g., user credentials, payment details) is exchanged.
5. **User Experience:** By incorporating real-time notifications and the ability to handle offline tasks, the architecture ensures a smooth user experience, which is crucial for e-commerce platforms.

In conclusion, this architecture is appropriate for the system due to its focus on modularity, scalability, and performance, all of which are necessary to provide a responsive, secure, and efficient marketplace experience.

# Tools and Technologies

#### **3.1 Front-end Development**

**3.1.1 React**

**3.1.1.1 Version**: 18.2.0

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

**3.1.2 Next.js**

**3.1.2.1 Version**: 14.1.4

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

**3.1.3 Tailwind CSS**

**3.1.3.1 Version**: 3.4.3

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

#### **3.2 Back-end Development**

**3.2.1 MongoDB**

**3.2.1.1 Version**: 8.0

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

**3.2.2 Node.js**

**3.2.2.1 Version**: 21.0.0

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

**3.2.3 Express.js**

**3.2.3.1 Version**: 5.0.0

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

#### **3.3 API Development and Testing**

**3.3.1 Postman**

**3.3.1.1 Version**: 11.0.0

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

#### **3.4 UI/UX Design**

**3.4.1 Figma**

**3.4.1.1 Version**: Latest (cloud-based)

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

#### **3.5 Development Environment**

**3.5.1 Visual Studio Code**

**3.5.1.1 Version**: 1.93.1

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

#### **3.6 Additional Tools**

**3.6.1 Git**

**3.6.1.1 Version**: 2.47

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

**3.6.2 Jest**

**3.6.2.1 Version**: 29.7.0

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

**3.7 Deployment**

**3.7.1 Git:**

**3.7.1.1 Version:** 2.47

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

**3.7.2 NPM (Node Package Manager):**

**3.7.2.1 Version:** 10.1.0

**3.7.2.2 Description:** Manages packages for Node.js applications.

### **3.7.3 Supervisord:**

**3.7.3.1 Version:** 4.2.4.

**3.7.3.2 Description:** Monitors and manages application processes.

### **3.7.4 Symbolic Links:**

**3.7.4.1 Description:** Links files and directories for easy access.

# Hardware Requirements

### **4.1 Development Machines**

**4.1.1 Processor (CPU)**: The minimum requirement is a Dual-core processor (e.g., Intel i3 or AMD Ryzen 3). However, the recommended one is a Quad-core processor (e.g., Intel i5 or AMD Ryzen 5) for better performance.

**4.1.2 Memory (RAM)**: The minimum requirement is 8 GB RAM. However, the recommended one is 16 GB RAM or more.

**4.1.3 Storage**: The minimum requirement is 256 GB SSD (Solid State Drive). However, the recommended 512 GB SSD or larger to ensure faster load times and ample space for development tools and databases.

**4.1.4 Graphics**: Integrated graphics are typically sufficient for development, but a dedicated GPU (e.g., NVIDIA or AMD) may benefit design work or certain applications.

**4.1.5 Operating System**: Compatible with development tools, generally Windows 10/11, macOS Monterey or later, or a modern Linux distribution (Ubuntu, Fedora, etc.).

### **4.2 Deployment Servers**

**4.2.1** **Use an AWS EC2 instance or a comparable cloud provider instance:** It should meet performance and scalability needs, such as adequate CPU cores, memory (RAM), SSD storage, and network throughput, based on application demands. Ensure compatibility with Linux-based operating systems (e.g., Ubuntu Server, or Debian) for stability and security.

# Who Did What?

| **Name of the Team Member** | **Tasks done** |
| --- | --- |
| Aniqa Aqeel` | 2.2, 2.3 |
| Muhammad Umer Jamil | Seller’s Diagram |
| Abdul Ahad Bin Ali | Buyer’s Diagram |
| Saad Ilyas | Tools and technologies, hardware requirement |
| Hasan Malik | Admin Diagram |

# Review checklist

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

| **Section** **Title** | **Reviewer Name(s)** |
| --- | --- |
| Tools and Technologies | Aniqa Aqeel |
| Buyer’s Diagram | Muhammad Umer Jamil |
| Admin Diagram | Saad Ilyas |
| 2.2 and 2.3 | Hasan Malik |
| Seller’s Diagram | Abdul Ahad |