**Low-Level Architecture and Data Models**

**P02:MinarMarket**

**<team member names & ids>**

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

|  |  |  |
| --- | --- | --- |
| **Content** | **Totals** | **Obtained** |
| Architecture diagram | 20 | 20 |
| Architecture justification | 20 | 20 |
| E/R diagram | 20 | 20 |
| E/R diagram description | 20 | 20 |
| Tools and Technologies | 10 | 10 |
| Who did what | 3 | 3 |
| Review checklist | 2 | 2 |
| Overall formatting/template | 5 | 5 |
| GitHub folder structure penalty | -15 | - |
| Late submission penalty | -20 | - |
| **Grand Total** | **100** | **100** |
| **General Comments/Individual Grading:** Update as the high-level architecture. Discuss security, buyer/seller. | | |

**Table of Contents**

[1.](#_gjdgxs) Introduction 3

[2.](#_30j0zll) System Architecture 4

[2.1 Architecture Diagram—](#_1fob9te)As it is in the prototype code 4

[2.2 Architecture Diagram—](#_3znysh7)As it should-be 4

[3.](#_2et92p0) Data Models 5

[4.](#_tyjcwt) Tools and Technologies 6

[5.](#_3dy6vkm) Who Did What? 7

[6.](#_1t3h5sf) Review checklist 7

# 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 decisions 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.

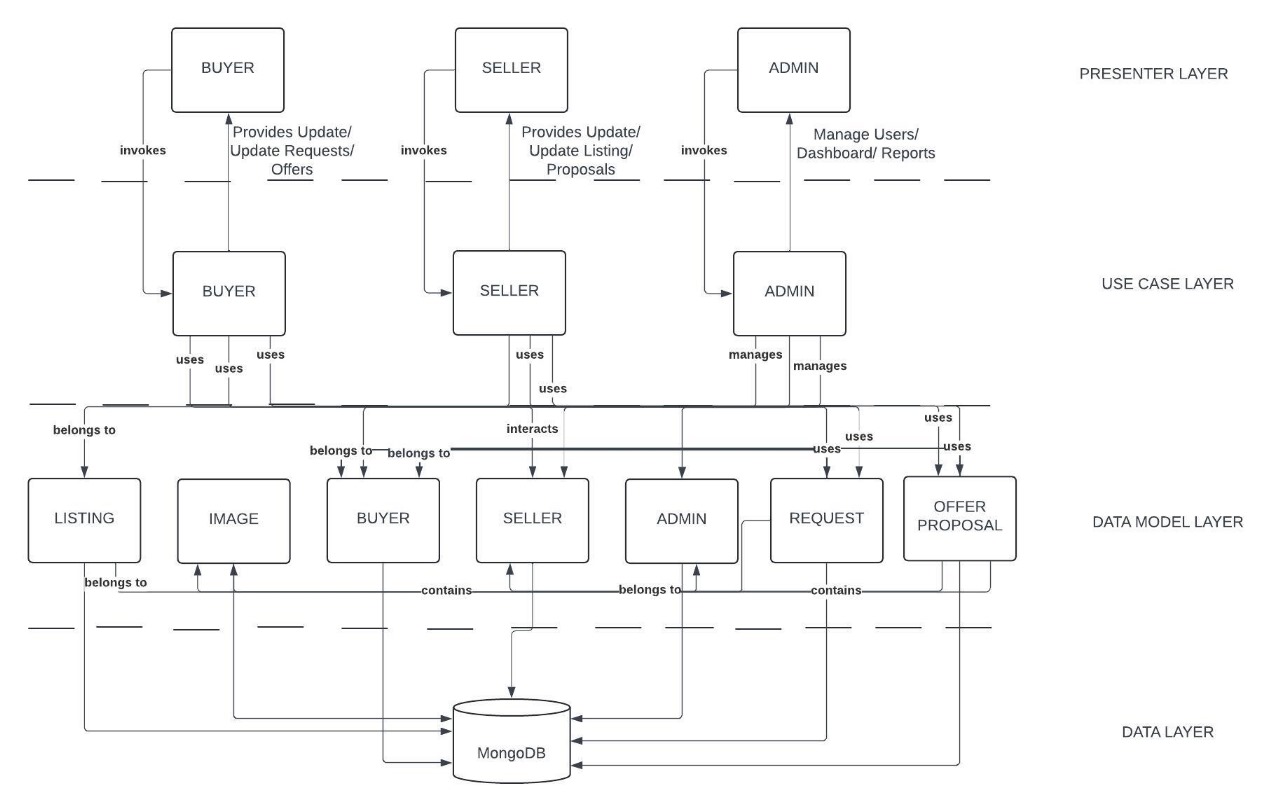
Furthermore, our marketplace is designed with scalability and flexibility in mind. As the platform grows, we plan to incorporate advanced features such as AI-driven product matching, where algorithms analyze buyer requests and suggest potential matches from a seller’s inventory. This feature will streamline 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.

Ultimately, this project aims to revolutionize the 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

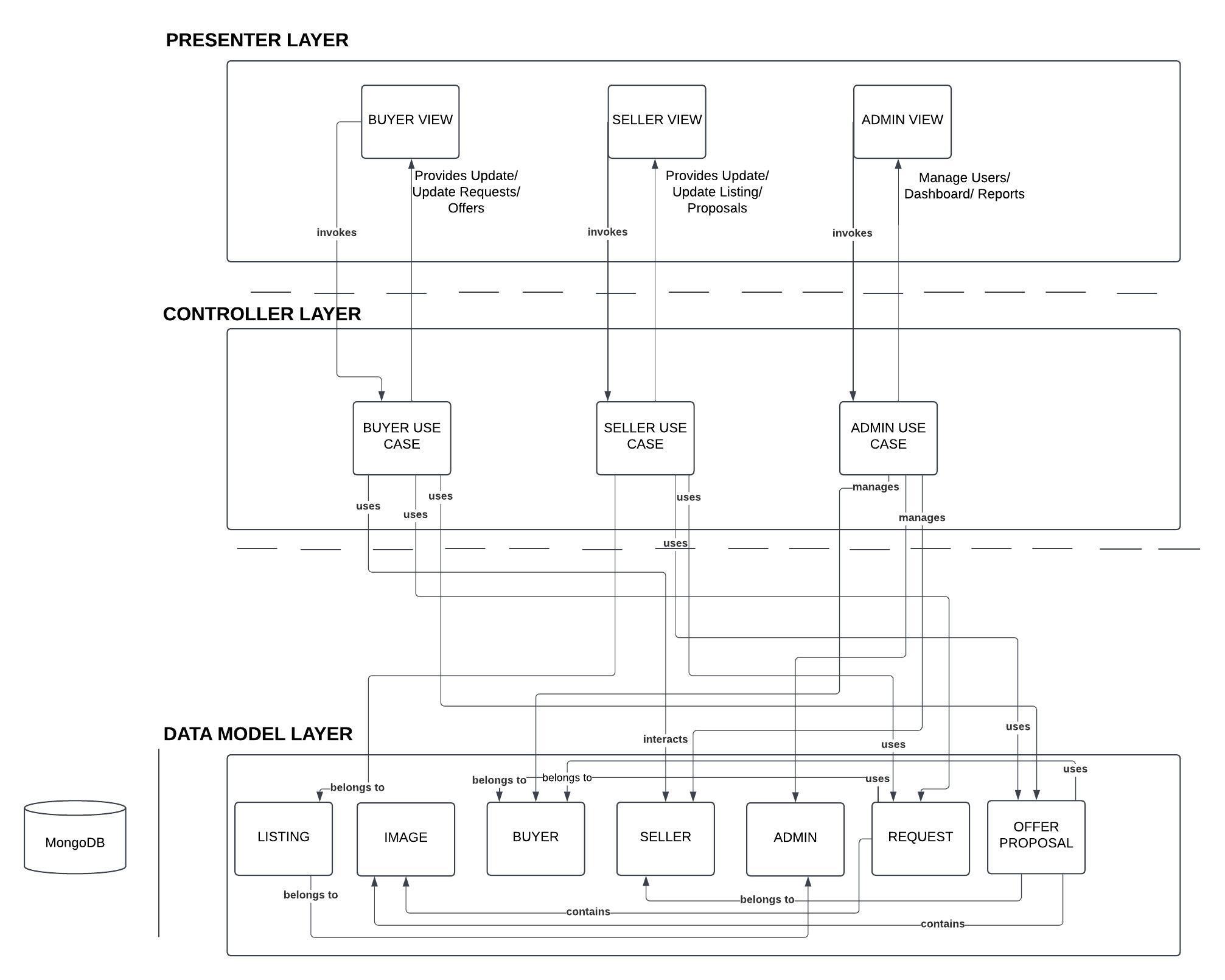
## Architecture Diagram—As it is in the prototype code

<Draw a diagram of the system architecture. The diagram must reflect the architecture of the code that you have written for prototype. For instance, if you are following layered architecture, your diagram must distinguish all layers along with classes/modules in each layer. You must mention exact names of classes/modules inside the layers as they are in your prototype code. If there are too many classes, your diagram must show classes/modules involved in at least two use cases. >



## Architecture Diagram—As it should-be

Link: [Click here](https://lucid.app/lucidchart/de5552d3-2506-48d0-a863-ee51b9610904/edit?viewport_loc=-603%2C-514%2C2986%2C1452%2C0_0&invitationId=inv_c59fcda0-16ce-4263-a7f8-537f4660db0a)



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.

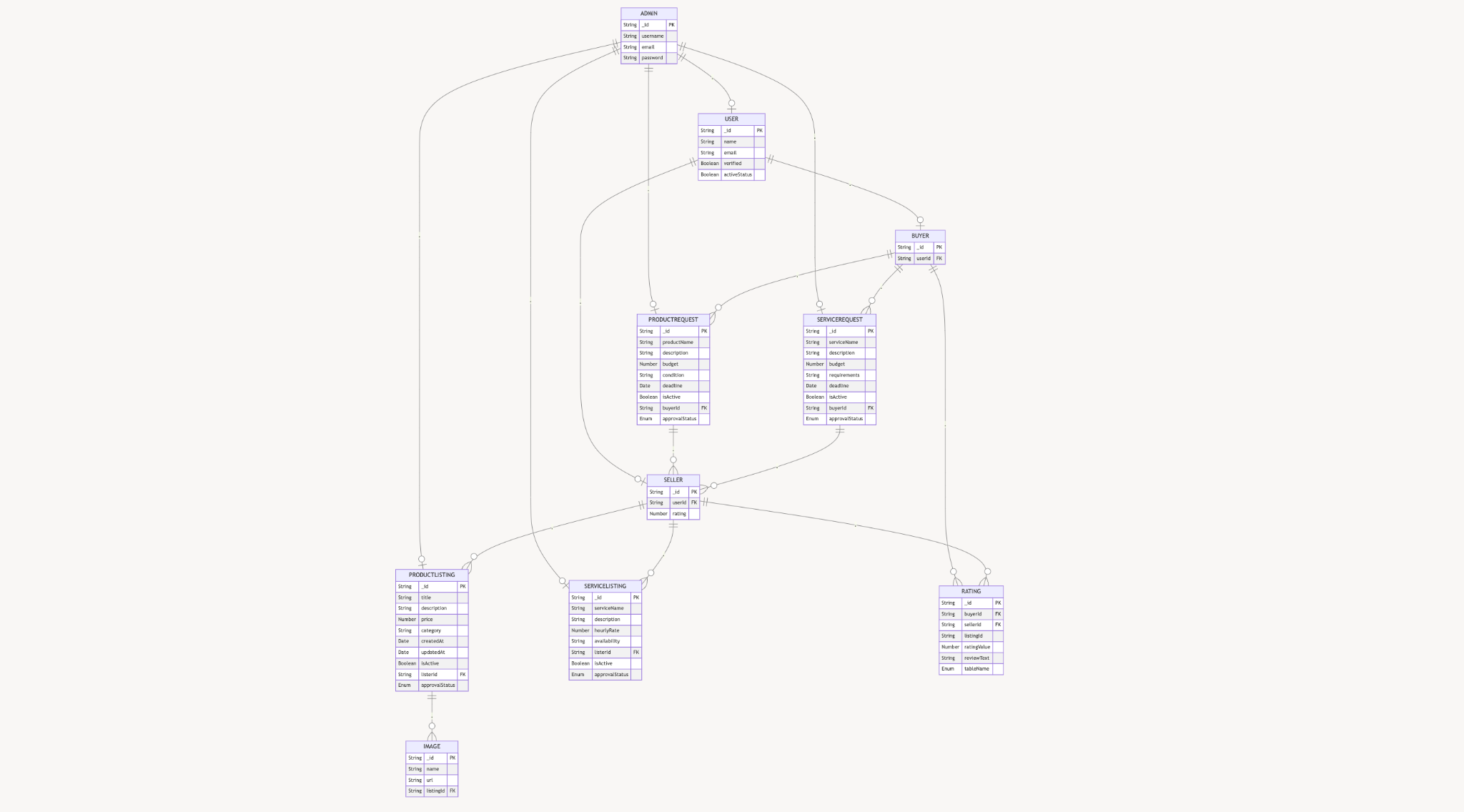
By encapsulating logic in well-defined layers, components from one layer can be reused across multiple parts of the application or even in different projects. For instance, a business logic layer can serve different frontends like web and mobile apps, reducing duplication and speeding up development.

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.

# Data Models

Link: [ER Diagram](https://mermaid.ink/img/pako:eNrFVttu4jAQ_RXLUt8ootCWJm-0ZCvUlu0SqLQrpJWJp2DJibOOTcsC_74mSaG5UWgf1g-5zByPZ8bHM15iT1DANgbZZWQqiT8OEELxA41cZ4CWyfdmuEqyYIp-M4oe7wrigPhQEIJPGN9Jr4XgQAI0B8meGdCihniKzcFVROko0a7HQfLR6T70-of7oyOQh_iUSkMSRS9C0tyi16OfxyRhs2iPom93OTOuc3__BTub0df-BCSSRBlAzvzj4Ht3dDO877nDXv_28GUUU7yYIAqRJ1momAgKy4eSecUZHlEwFXKxU3SNBHkSzIt2VE6uQ5qXv-0_izoxAwpLcBapQk6cQPuIhKEUc8JLSWMo_NS7cY7OjMn_3ETaLyPQvvzMhJZ8MTDhFaaRueEdmTDO1OKw8KqT8nHgKSUGzo-R4w4PD9yYpNpTRwc-0XQKqsgMEVCWnRBTgAKhnAVwBAMmevF5Ahydh88SoCIPEv5oJsGHQEX_KxW9h86t88WCbthdyl3zLqt7g85xx64ssO2GcF6l23lQUS-fCNdQsilzBi9DeFW5NCoy4e-2fhvOyQkaACebjY9mLIzeN6bV6vRUrJKmaaMxro_xuzaaapN2UqlO28RWX2I_V-ozpjLAXOWrBuYqxYcWC8BMi4uxy71uZpH7_EzSlTVZ6mcGuN_PnGfpnORslEPfDG3NZzeppMpUIEt8TQ_InvzkEbiGfZDmDkPNtS0-WGOsZqayjPEGQ-GZaK42yLWBEq2Euwg8bCupoYal0NMZtp8Jj8xf0ojTm99WGpLglxC7fzAVXMiH5KIY3xdjDLaX-BXbp82GVW81Ghdn7fP2pXXRuGrV8ALbVr1tWdaV1Wy1m5eN8-a6hv_GVpv1ZiMeBn7Vap9drv8Bv8wSCg?type=png)%5D(https://mermaid.live/edit#pako:eNrFVttu4jAQ_RXLUt8ootCWJm-0ZCvUlu0SqLQrpJWJp2DJibOOTcsC_74mSaG5UWgf1g-5zByPZ8bHM15iT1DANgbZZWQqiT8OEELxA41cZ4CWyfdmuEqyYIp-M4oe7wrigPhQEIJPGN9Jr4XgQAI0B8meGdCihniKzcFVROko0a7HQfLR6T70-of7oyOQh_iUSkMSRS9C0tyi16OfxyRhs2iPom93OTOuc3__BTub0df-BCSSRBlAzvzj4Ht3dDO877nDXv_28GUUU7yYIAqRJ1momAgKy4eSecUZHlEwFXKxU3SNBHkSzIt2VE6uQ5qXv-0_izoxAwpLcBapQk6cQPuIhKEUc8JLSWMo_NS7cY7OjMn_3ETaLyPQvvzMhJZ8MTDhFaaRueEdmTDO1OKw8KqT8nHgKSUGzo-R4w4PD9yYpNpTRwc-0XQKqsgMEVCWnRBTgAKhnAVwBAMmevF5Ahydh88SoCIPEv5oJsGHQEX_KxW9h86t88WCbthdyl3zLqt7g85xx64ssO2GcF6l23lQUS-fCNdQsilzBi9DeFW5NCoy4e-2fhvOyQkaACebjY9mLIzeN6bV6vRUrJKmaaMxro_xuzaaapN2UqlO28RWX2I_V-ozpjLAXOWrBuYqxYcWC8BMi4uxy71uZpH7_EzSlTVZ6mcGuN_PnGfpnORslEPfDG3NZzeppMpUIEt8TQ_InvzkEbiGfZDmDkPNtS0-WGOsZqayjPEGQ-GZaK42yLWBEq2Euwg8bCupoYal0NMZtp8Jj8xf0ojTm99WGpLglxC7fzAVXMiH5KIY3xdjDLaX-BXbp82GVW81Ghdn7fP2pXXRuGrV8ALbVr1tWdaV1Wy1m5eN8-a6hv_GVpv1ZiMeBn7Vap9drv8Bv8wSCg)



### **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**.

# Tools and Technologies

#### **4.1 Front-end Development**

**4.1.1 React**

**4.1.1.1 Version**: 18.2.0

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

**4.1.2 Next.js**

**4.1.2.1 Version**: 14.1.4

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

**4.1.3 Tailwind CSS**

**4.1.3.1 Version**: 3.4.3

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

#### **4.2 Back-end Development**

**4.2.1 MongoDB**

**4.2.1.1 Version**: 8.0

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

**4.2.2 Node.js**

**4.2.2.1 Version**: 21.0.0

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

**4.2.3 Express.js**

**4.2.3.1 Version**: 5.0.0

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

#### **4.3 API Development and Testing**

**4.3.1 Postman**

**4.3.1.1 Version**: 11.0.0

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

#### **4.4 UI/UX Design**

**4.4.1 Figma**

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

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

#### **4.5 Development Environment**

**4.5.1 Visual Studio Code**

**4.5.1.1 Version**: 1.93.1

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

#### **4.6 Additional Tools**

**4.6.1 Git**

**4.6.1.1 Version**: 2.47

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

**4.6.2 Jest**

**4.6.2.1 Version**: 29.7.0

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

**4.7 Deployment**

**4.7.1 Git:**

**4.7.1.1 Version:** 2.47

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

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

**4.7.2.1 Version:** 10.1.0

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

### **4.7.3 Supervisord:**

**4.7.3.1 Version:** 4.2.4.

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

### **4.7.4 Symbolic Links:**

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

# Who Did What?

|  |  |
| --- | --- |
| **Name of the Team Member** | **Tasks done** |
|  |  |
| M. Saad Ilyas | Data Models |
| M. Umer Jamil | Tools and technologies / as it should be |
| Abdul Ahad | Architecture as it is now / as it should be |

# 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)** |
| Architecture Diagram | M. Saad Ilyas |
| As it is | Umer Jamil |
| as it should be | Abdul Ahad |
|  |  |