**High level Architecture**

**P14:Shop Savvy**

| **Student ID** | **Name** |
| --- | --- |
| 25100002 | Ahmad Kashif Jabbar |
| 25100105 | Husnain Ali |
| 25100208 | Zainab Fatima |
| 25100229 | Syed Messam Ali |
| 25100179 | Musa Aftab Ahmad |

**Table of Contents**

1. Introduction

2. System Architecture

2.1 Architecture Diagram

2.2 Architecture Description

2.3 Justification of the Architecture

3. Tools and Technologies

4. Hardware Requirements

5. Who Did What?

6. Review checklist

**1.** **Introduction**

The AI-driven Personalized Clothing Recommendation Platform is designed to enhance the online shopping experience for Pakistani consumers by bringing together a curated selection of local clothing brands on a single platform. The platform will feature at least 10 prominent Pakistani clothing brands, allowing users to explore and shop from a variety of options conveniently in one place. The main objective is to simplify the shopping process by offering personalized recommendations tailored to individual user preferences, all while showcasing local fashion.

The platform’s primary users are Pakistani consumers who are looking for a seamless, convenient, and personalized shopping experience. By aggregating clothing options from multiple brands, the platform eliminates the need for shoppers to visit multiple websites. The AI-powered recommendation engine will leverage user data—such as browsing history, past purchases, and personal preferences—to suggest relevant products, making the experience more engaging and efficient.

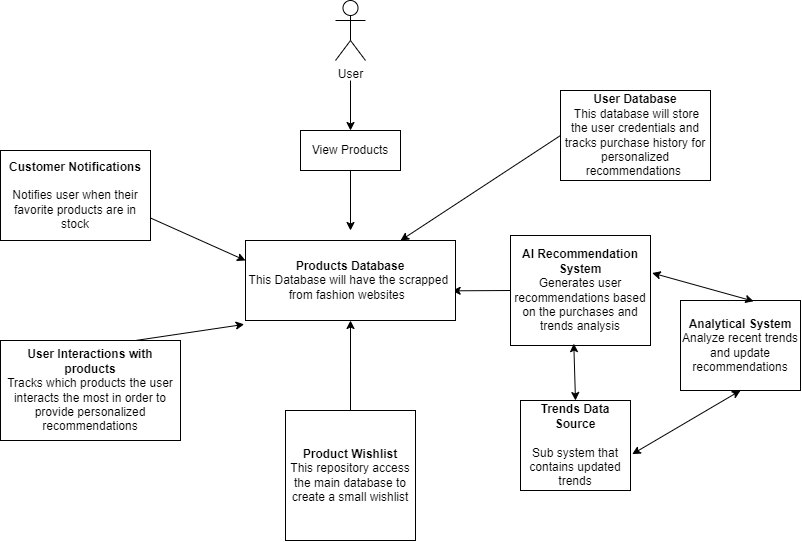
In the long term, the platform has the potential to generate revenue through affiliate marketing by partnering with local brands, earning commissions on purchases made via the platform. This creates a win-win situation, providing visibility for the brands and a tailored shopping experience for consumers.

The platform aims to provide a personalized, convenient, and enjoyable online shopping experience, helping Pakistani consumers discover and purchase clothing from a range of local brands in a simple, AI-enhanced environment.

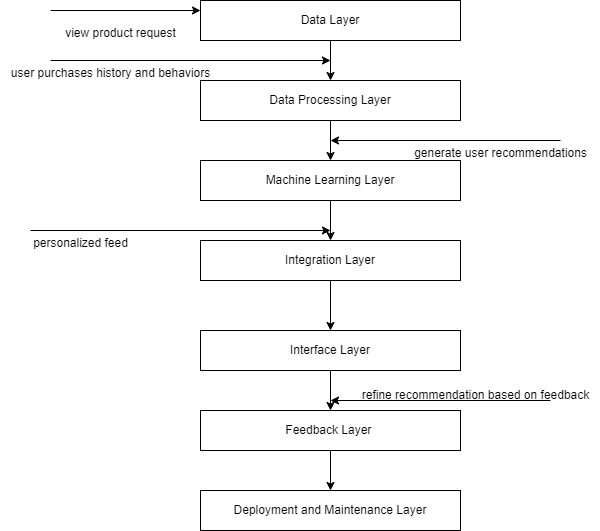
**2.** **System Architecture**

**2.1** **Architecture Diagram**

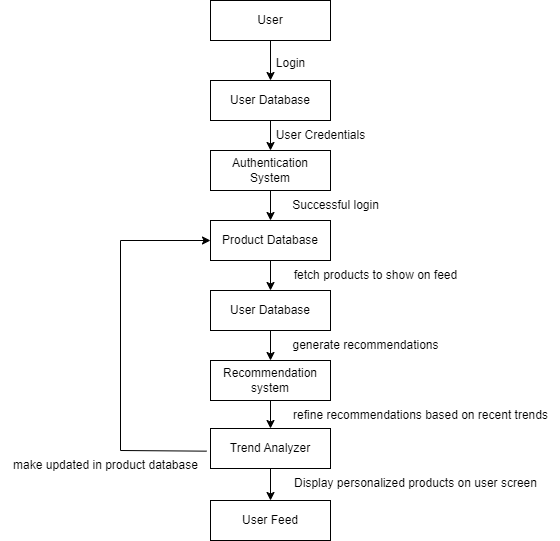
**Repository Architecture:**

****

**Layered Architecture:**

****

**Pipe and Filter Architecture:**

****

### 2.2 Architecture Description

#### 1. Repository Architecture

This architecture diagram presents a repository-based architecture for an AI-powered shopping assistant system. Below is a description of each subsystem and the interactions among them:

**Subsystem Descriptions**:

* **User Database**:
  + Stores user credentials, purchase history, and preferences.
  + Generates personalized recommendations by tracking past purchases and behaviors.
  + Feeds relevant user data to the AI Recommendation System.
* **Products Database**:
  + Contains product data scraped from various fashion websites.
  + Stores information such as product descriptions, prices, availability, and more.
  + Acts as the central repository for product information, providing data for recommendations, notifications, and wishlists.
* **AI Recommendation System**:
  + Utilizes data from the User Database, Products Database, and Trends Data Source to generate personalized product recommendations.
  + Employs machine learning models to analyze user purchase history and trends.
  + Outputs tailored suggestions to users based on their behavior and current trends.
* **Trends Data Source**:
  + Contains updated information on recent trends, collected from sources like social media and fashion websites.
  + Provides input to the AI Recommendation System to align recommendations with current trends.
  + Continuously updated via the Analytical System.
* **Analytical System**:
  + Monitors recent trends and updates the Trends Data Source regularly.
  + Ensures that recommendations reflect the latest market trends.
  + Collaborates with the AI Recommendation System to keep suggestions relevant.
* **Product Wishlist**:
  + Allows users to create a wishlist by selecting items from the Products Database.
  + Provides a focused subset of products based on user preferences.
* **Customer Notifications**:
  + Alerts users when items on their wishlist are available or back in stock.
  + Uses product availability data from the Products Database to trigger notifications.
* **User Interactions with Products**:
  + Tracks the products that users interact with the most.
  + Helps refine the AI Recommendation System by identifying user preferences and usage patterns.

#### 2. Layered Architecture

**Layer Descriptions and Responsibilities**:

* **Data Layer**:
  + **Purpose**: Acts as the foundation of the system, storing essential data such as user purchase history, behaviors, and product information.
  + **Example Data**: User browsing history, product catalogs, user profile data, and purchase records.
  + **Interaction**: Receives product requests from users and provides raw data to the Data Processing Layer for further analysis.
* **Data Processing Layer**:
  + **Purpose**: Processes and organizes data for use by other components in the system. Cleans, transforms, and structures data to ensure it is suitable for machine learning algorithms.
  + **Interaction**: Extracts relevant user purchase history and behavior from the Data Layer and feeds structured data to the Machine Learning Layer.
* **Machine Learning Layer**:
  + **Purpose**: Generates personalized recommendations using machine learning algorithms such as collaborative filtering or content-based filtering.
  + **Tasks**: Analyzes user behavior, historical data, and trends to create tailored recommendations.
  + **Interaction**: Receives processed data from the Data Processing Layer and returns personalized suggestions to the Integration Layer.
* **Integration Layer**:
  + **Purpose**: Acts as the intermediary, ensuring all components (recommendation system, data processing, interfaces) communicate smoothly.
  + **Interaction**: Takes personalized feeds from the Machine Learning Layer and makes them available to the Interface Layer.
* **Interface Layer**:
  + **Purpose**: Provides the front-end interface through which users interact with the system, including product searches and recommendations.
  + **Tasks**: Displays recommendations to the users and collects interactions or preferences.
  + **Interaction**: Sends feedback from users to the Feedback Layer for refinement of the recommendations.
* **Feedback Layer**:
  + **Purpose**: Gathers and processes feedback from users based on their interactions with the system (e.g., likes, dislikes, purchases).
  + **Tasks**: Helps refine the recommendation models to improve accuracy and relevance.
  + **Interaction**: Sends refined insights back to the Machine Learning Layer to improve future recommendations.
* **Deployment and Maintenance Layer**:
  + **Purpose**: Ensures the system is up-to-date and performs efficiently by managing deployments, monitoring, and updates.
  + **Tasks**: Manages code releases, bug fixes, and feature updates.
  + **Interaction**: Works with the entire system to ensure smooth deployment of changes and continuous improvement.

**Subsystem Interactions**:

* **Data Requests Flow**: Users send product requests, which are processed by the Data Layer and forwarded through the Data Processing Layer for structuring.
* **Recommendation Flow**: Data flows to the Machine Learning Layer, which generates recommendations that pass through the Integration Layer to reach the Interface Layer.
* **Feedback Loop**: User interactions are captured through the Interface Layer and passed to the Feedback Layer for refinement.
* **Model Improvement**: Feedback flows back to the Machine Learning Layer to fine-tune the recommendations.
* **System Updates**: The Deployment and Maintenance Layer ensures continuous improvement by releasing updates and monitoring system performance.

#### 3. Pipe and Filter Architecture

**Filter Descriptions and Responsibilities**:

* **User**:
  + The entry point to the system.
  + Initiates the interaction by logging into the system and viewing the personalized feed.
* **User Database**:
  + Stores user credentials, profiles, preferences, and behavioral data.
  + Provides authentication data for the login process and historical data to generate recommendations.
  + After authentication, the system uses this data to inform the Recommendation System.
* **Authentication System**:
  + Validates the user's login credentials.
  + Ensures only authenticated users can access the system's features.
  + If login is successful, the authenticated session allows access to the product data.
* **Product Database**:
  + Contains detailed product data, including descriptions, prices, availability, and categories.
  + Fetches products to be displayed on the personalized feed.
  + Works with the User Database to provide relevant product information for recommendation generation.
* **Recommendation System**:
  + Analyzes user preferences, historical purchases, and product availability to generate personalized product suggestions.
  + Uses algorithms such as collaborative filtering and content-based filtering to suggest relevant products.
  + Passes recommendations to the Trend Analyzer for refinement.
* **Trend Analyzer**:
  + Refines recommendations by incorporating the latest product trends.
  + Updates the Product Database to ensure the system reflects recent trends and changes.
  + Ensures that the recommendations align with both the user's preferences and current trends.
* **User Feed**:
  + Displays the personalized product feed on the user interface.
  + Reflects the final recommendations generated by the Recommendation System and refined by the Trend Analyzer.
  + Provides a real-time view of available and recommended products to the user.

**Subsystem Interactions (Pipes)**:

* **Login Request Flow**: The user initiates a login request, which flows from the User Database to the Authentication System.
* **Recommendation Generation**: The User Database provides user preferences and historical data, which flows to the Recommendation System. The Product Database also feeds product information into the Recommendation System to generate suggestions.
* **Trend Refinement**: The Trend Analyzer refines the generated recommendations based on current market trends. Any new insights or trends are updated in the Product Database to ensure consistency across the system.
* **Displaying Personalized Feed**: After trend analysis, the recommendations flow to the User Feed to be displayed on the user’s interface.

### 2.3 Justification of the Architecture

#### 1. Repository Architecture

**Pros of the Architecture**:

* **Modularity**:
  + The architecture is divided into distinct subsystems, each focusing on a specific task (e.g., recommendation generation, product tracking, notifications). This separation simplifies development and maintenance.
* **Scalability**:
  + Each subsystem can be scaled independently. For example, if the product catalog grows, only the Products Database needs to be expanded.
* **Real-time Updates**:
  + The use of an Analytical System ensures that trends and recommendations remain up-to-date, improving user satisfaction.
* **Personalization**:
  + The combination of the User Database and AI Recommendation System enables highly personalized suggestions, enhancing the shopping experience.
* **User Engagement**:
  + Customer Notifications and Product Wishlists promote engagement by keeping users informed and offering tailored product selections.
* **Data-driven Recommendations**:
  + User interactions and behavior are tracked systematically to continuously refine recommendations and align with user preferences.

**Cons of the Architecture**:

* **Complex Data Management**:
  + Integrating multiple databases (User, Products, Trends) requires robust data synchronization and management strategies.
* **Dependency on External Sources**:
  + Scraping product data and collecting trends from external sources introduces potential reliability risks if sources change or restrict access.
* **Resource-intensive Recommendation System**:
  + Running an AI recommendation engine, especially with real-time updates, can be resource-intensive and may require optimization.
* **Notification Overload Risk**:
  + If notifications are not well-managed, users may feel overwhelmed and disengage from the system.

**Justification of the Architecture**:

This architecture is appropriate for the AI-powered virtual shopping assistant because it supports both **non-functional** and **functional requirements**.

* **Non-functional Requirements**:
  + **Scalability**: Each subsystem is modular, allowing individual components to scale as needed.
  + **Maintainability**: Clear separation between databases and functional components makes it easier to troubleshoot and maintain.
  + **Performance**: Real-time notifications and recommendations enhance responsiveness and improve the user experience.
  + **Adaptability**: The Analytical System ensures that recommendations remain relevant by updating trend data regularly.
* **Functional Requirements**:
  + **Alignment with Functional Requirements**: The architecture enables tracking of user behavior, personalized recommendations, and real-time notifications, aligning perfectly with the goal of providing a tailored shopping experience.
  + **Data-driven Insights**: The use of AI for recommendations and trend analysis ensures that users receive high-quality, relevant suggestions, increasing engagement and satisfaction.
  + **Improved User Engagement**: The inclusion of notifications, wishlists, and personalized recommendations ensures that users stay engaged with the platform, increasing retention.

#### 2. Layered Architecture

**Pros of the Architecture**:

* **Modular and Layered Structure**:
  + Each layer is functionally independent, allowing for easier maintenance and scalability. Any layer can be modified or upgraded without impacting others, improving flexibility.
* **Efficient Feedback Loop**:
  + The inclusion of a Feedback Layer ensures continuous improvement in recommendation quality based on user preferences and interactions.
* **Seamless Integration**:
  + The Integration Layer ensures smooth data flow and interoperability between subsystems, reducing communication bottlenecks.
* **Supports Real-Time Recommendations**:
  + The architecture enables real-time recommendations, enhancing the user experience with timely and relevant suggestions.
* **Scalability and Maintainability**:
  + The Deployment and Maintenance Layer supports easy scaling and continuous deployment of new features or updates.
* **Clear Data Flow**:
  + The structured flow from the Data Layer to the Interface Layer ensures logical data movement, improving system transparency and debugging.

**Cons of the Architecture**:

* **Complex Feedback Handling**:
  + The feedback mechanism can become complex, requiring efficient tracking and handling to ensure meaningful improvements.
* **High Resource Consumption**:
  + Machine learning models, especially when combined with real-time processing, may consume significant computational resources.
* **Dependency on Integration Layer**:
  + If the Integration Layer experiences issues, the entire system's communication could be affected.
* **Latency Risks**:
  + If the machine learning processes are not optimized, the system may experience delays in generating recommendations.

**Justification for the Architecture**:

This layered architecture is well-suited for the AI-powered shopping assistant because it provides a structured, scalable, and maintainable framework. Each layer has a distinct responsibility, ensuring the separation of concerns and ease of management.

* **Support for Non-Functional Requirements**:
  + **Scalability**: The modular nature allows for independent scaling of components.
  + **Maintainability**: Clear separation of layers facilitates easier troubleshooting and upgrades.
  + **Performance**: Real-time recommendations ensure quick responses, enhancing user satisfaction.
  + **Flexibility**: The Feedback Layer allows continuous refinement based on user input, ensuring adaptability to changing preferences.
* **Alignment with Functional Requirements**:
  + This architecture supports personalized recommendations, user interaction tracking, and real-time feedback processing, making it a perfect fit for the system’s goals.
* **Improves User Experience**:
  + The well-defined feedback loop and real-time integration ensure that users receive relevant and timely recommendations, increasing engagement and retention.

#### 3. Pipe and Filter Architecture

**Pros of the Architecture**:

* **Modular and Maintainable**:
  + Each stage (filter) is functionally independent, making the system modular and easier to maintain. New filters (e.g., additional analytics modules) can be added without disrupting the existing architecture.
* **Scalable**:
  + The pipe-and-filter structure supports scalability, as each filter can be optimized or scaled independently to handle increased load.
* **Efficient Data Flow**:
  + Data flows logically through each stage, ensuring a well-defined transformation from raw data to final recommendations.
* **Real-Time Trend Updates**:
  + The Trend Analyzer ensures that recommendations are aligned with both user preferences and the latest product trends, improving relevance.
* **Security via Authentication**:
  + The Authentication System ensures that only authorized users can access personalized feeds, enhancing security and privacy.

**Cons of the Architecture**:

* **High Resource Consumption**:
  + Running multiple filters (e.g., recommendation generation and trend analysis) can be resource-intensive.
* **Latency Issues**:
  + If not optimized, the multiple stages of processing may introduce delays, affecting real-time responsiveness.
* **Data Synchronization Complexity**:
  + Keeping the User Database and Product Database synchronized with real-time updates from the Trend Analyzer can be challenging.
* **Dependency on Authentication System**:
  + If the Authentication System fails, users cannot access the system, which may impact availability.

**Justification of the Architecture**:

This pipe-and-filter architecture is well-suited for the AI-powered product recommendation system due to the following reasons:

* **Separation of Concerns**:
  + Each filter handles a specific task, ensuring a clear separation of responsibilities. For example, authentication is handled independently from recommendation generation, improving system modularity.
* **Support for Non-functional Requirements**:
  + **Scalability**: Each filter can be scaled or optimized independently to handle growth.
  + **Maintainability**: The modular nature ensures that issues can be isolated and fixed within individual filters without affecting the entire system.
  + **Security**: The Authentication System ensures that access is restricted to authorized users only.
  + **Performance**: Trend updates are incorporated in real-time to keep recommendations relevant and engaging.
* **Enhanced User Experience**:
  + The architecture ensures that users receive timely, relevant product recommendations based on their preferences and current trends. The User Feed offers a personalized interface, keeping users engaged and increasing satisfaction.

**3.** **Tools and Technologies**

**MongoDB (v6.x or later)**

MongoDB will function as the central database for the platform, efficiently managing diverse data types such as user profiles, browsing histories, purchase records, and product catalogs from various local clothing brands. Its document-oriented design offers the flexibility required to handle complex, nested data structures without enforcing rigid schemas, making it ideal for accommodating the dynamic nature of user preferences and product offerings. Additionally, MongoDB's inherent scalability ensures the platform can adeptly handle increasing user traffic and expanding datasets as the business grows.

**Vercel**

Vercel will be utilized to deploy and host the platform's front-end React application. It provides a serverless architecture that delivers fast, secure, and scalable performance, essential for ensuring a seamless user experience. With Vercel's optimized deployment pipeline, the platform can implement continuous integration and delivery, facilitating rapid updates and maintaining high availability for users.

**React (v18.x or later)**

React will serve as the primary framework for constructing the platform's user interface. Its component-based architecture allows for the creation of reusable UI elements, enabling a consistent and responsive design across the platform. React's virtual DOM efficiently updates and renders components, ensuring a smooth and interactive user experience as users navigate through various brands and product listings.

**Node.js (v20.x or later)**

Node.js will power the platform's backend server, handling tasks such as API requests, user authentication, data processing, and integration with the AI-driven recommendation system. Its non-blocking, event-driven architecture is well-suited for managing high-volume, concurrent operations, ensuring the platform remains responsive and efficient even under substantial user load.

**Pinecone** **(v2.x)**

We plan to use Pinecone (v2.x) as a managed vector database for efficient real-time similarity search and vector storage. It will help us store and query user preference embeddings and clothing feature vectors, critical for generating personalized recommendations. However, we need to further vet Pinecone's integration with our AI recommendation system to ensure it meets the performance and scalability needs of our platform.

**Hugging Face (V4.x)**

We are considering Hugging Face (v4.x) for generating text-based embeddings using pre-trained models like BERT or GPT. These embeddings will enhance our AI system's ability to recommend clothing based on user behavior and inputs. Further analysis is required to finalize the best-suited model for our use case, ensuring the recommendations are accurate and efficient for our platform.

**4.** **Hardware Requirements**

**Development Machines (Recommended Specs):**

* **Processor**:
  + **Intel Core i7 or higher / AMD Ryzen 7 or higher** (Hexa-core or above)
    - *Reason*: Enhanced processing power for handling ML models and development environments.
* **RAM**:
  + **16 GB** (or more, if working with large datasets or multiple applications simultaneously)
    - *Reason*: To handle larger data processing and multiple applications more efficiently.
* **Storage**:
  + **1 TB SSD**
    - *Reason*: storage for datasets, model weights, and various development tools.
* **Graphics**:
  + **Dedicated GPU (e.g., NVIDIA GeForce RTX series)**
    - *Reason*: Useful for ML model inference, especially when working with embeddings and other computational tasks.
* **Operating System**:
  + **Windows 10/11, macOS, or Linux (Ubuntu 20.04 or higher)**
* **Internet Connection**:
  + **High-speed internet** (for access to cloud services, repositories, and APIs).
* **Development Tools**:
  + Node.js, Express, React Native, MongoDB, Firebase, Git, Visual Studio Code, or any preferred IDE, along with ML libraries (e.g., TensorFlow, PyTorch) and NLP libraries (e.g., spaCy, Hugging Face Transformers).

**Deployment Servers (Recommended Specs):**

* **AWS EC2 Instance**:
  + **Type**:
    - **t3.medium or t3.large** for production (for better handling of backend requests and ML workloads).
  + **vCPU**:
    - **2-4 vCPUs** (scalable based on traffic and ML inference demands).
  + **RAM**:
    - **8 GB or 16 GB** (to support the additional load from NLP and ML processing).
  + **Storage**:
    - **100-200 GB EBS** (for application deployment, logs, temporary files, and data storage).
  + **OS**:
    - Ubuntu 20.04 LTS or Amazon Linux 2.
* **MongoDB Atlas**:
  + **Cluster**:
    - M10 cluster or higher (to handle increased data volume and queries).
  + **Storage**:
    - **20-50 GB** (scalable based on data growth).
* **Firebase**:
  + For real-time database and authentication, as well as managing user sessions.
* **Load Balancer**:
  + **AWS Elastic Load Balancer** (optional for auto-scaling to handle high traffic).
* **Machine Learning Services**:
  + **AWS SageMaker** or similar services for deploying ML models and managing inference tasks.
* **Backup and Recovery**:
  + **Cloud storage (S3 or equivalent)** for periodic backups of data and logs.

### 

### Additional Considerations

1. **NLP and Vector Embeddings**:
   * Implement a service for managing and serving embeddings, possibly using services like **Amazon SageMaker** or **Hugging Face's Inference API**.
2. **Web Scraping**:
   * Consider using a cloud-based scraping service or serverless architecture (e.g., **AWS Lambda**) to periodically gather fashion items.
3. **Search Optimization**:
   * Utilize search engines like **Elasticsearch** or **Amazon OpenSearch Service** for efficient handling of NLP and semantic searches.
4. **Recommendation Systems**:
   * Implement additional services for collaborative filtering or content-based filtering to enhance user experience.
5. **Monitoring and Analytics**:
   * Integrate monitoring solutions like **AWS CloudWatch** and analytics tools to track user behavior, search queries, and system performance.

By scaling up the development and deployment hardware requirements and integrating additional services tailored to NLP and e-commerce functionalities, we aim to create a robust environment to support our AI powered shopping platform.

**5.** **Who Did What?**

| **Name of the Team Member** | **Tasks done** |
| --- | --- |
| Ahmad Kashif Jabbar | Tools and technologies |
| Zainab Fatima | Architecture Diagrams |
| Messam Ali | Architecture descriptions |
| Husnain Ali | Justifications |
| Musa Aftab Ahmed | Hardware Requirements |

**6.** **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 Descriptions | Ahmad Jabbar |
| Architecture Diagrams | Husnain Ali |
| Architecture Diagrams | Messam Ali |
| Hardware Requirements | Zainab Fatima |
| Tools and Technologies | Musa Aftab |