**Low-Level Architecture and Data Models**

**<P14>:<Shop Savvy>**

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

|  |  |  |
| --- | --- | --- |
| **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 this if the high level architecture is updated. | | |

**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

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.

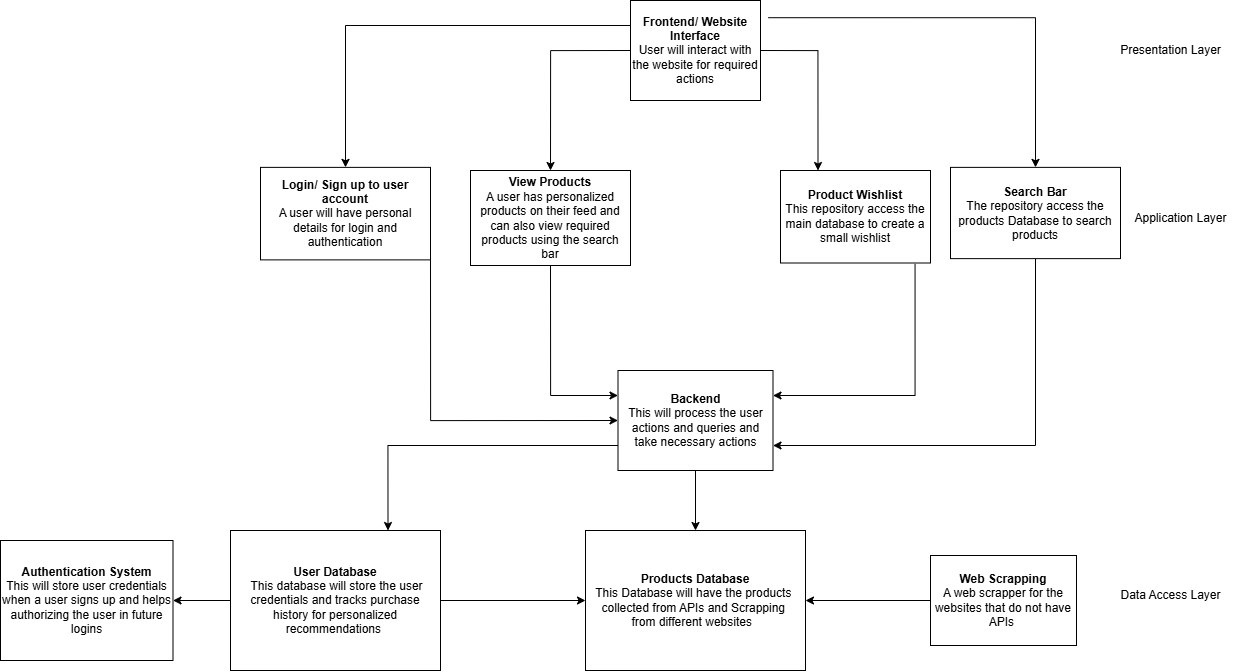
The platform’s primary users are Pakistani consumers who are looking for a seamless, convenient, and personalized shopping experience. By aggregating clothing options from multiple brands, the platform eliminates the need for shoppers to visit multiple websites. The AI-powered recommendation engine will 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.

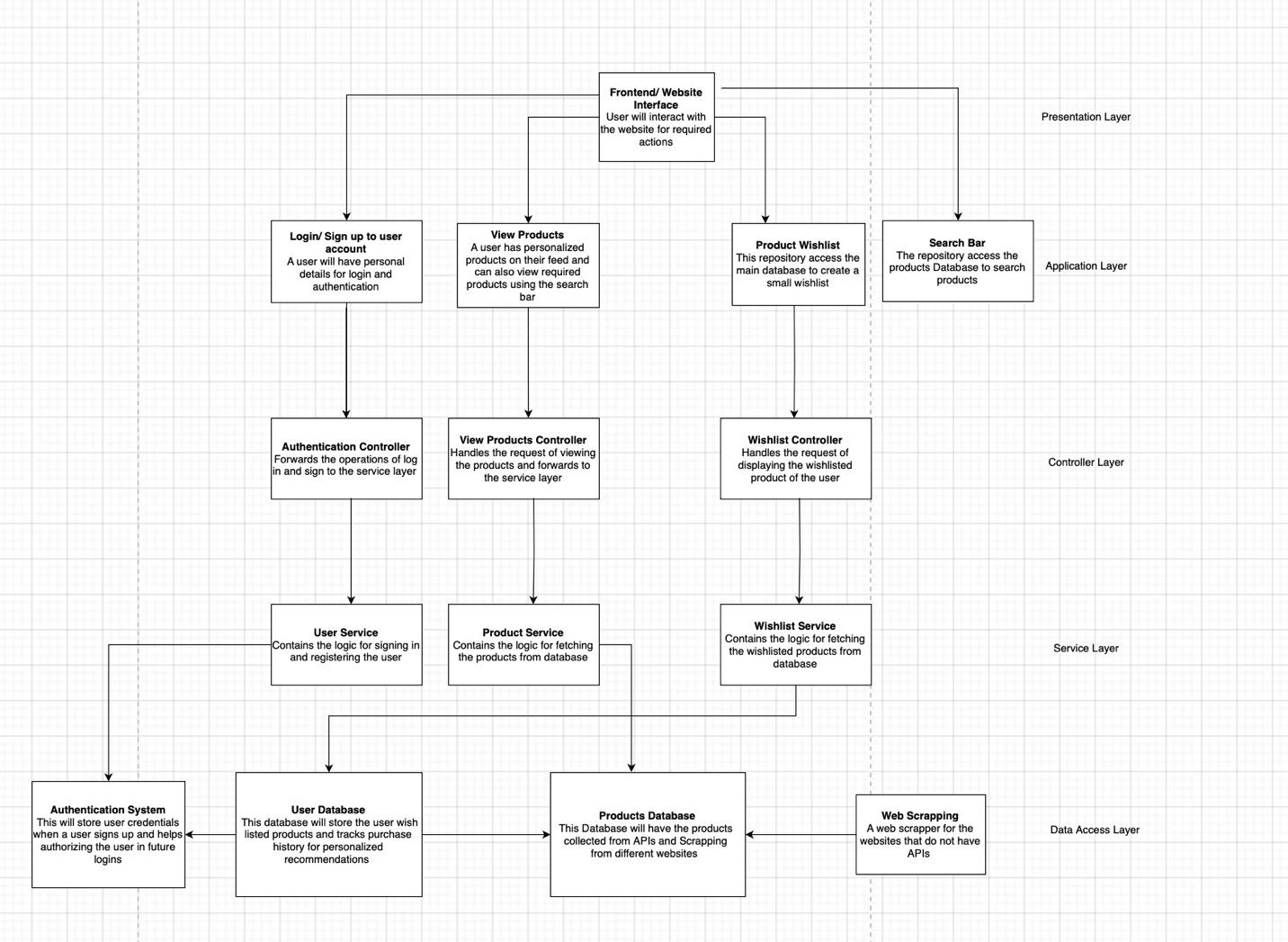
The platform aims to provide a personalized, convenient, and enjoyable online shopping experience, helping Pakistani consumers discover and purchase clothing from a range of local brands in a simple, AI-enhanced environment.

# System Architecture

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



## Architecture Diagram—As it should-be



**Updated Architecture**

**Key Adjustments:**

1. **Introduce a Controller Layer**:
   * Add controller classes to handle routing and logic for specific features. For example, AccountController for login/signup logic and ProductController for product management.
2. **Add a Service Layer**:
   * Encapsulate the business logic (e.g., validating inputs, querying databases). This separates concerns and makes testing and scaling easier.
3. **Refactor Database Interactions**:
   * Use a Data Access Object (DAO) pattern to abstract direct database calls for models like Account and Product.
4. **Frontend Integration**:
   * Ensure the frontend communicates with backend APIs (RESTful routes) rather than directly interacting with the database.

**Layered Architecture:**

1. **Presentation Layer**:
   * **Frontend (React)**: Contains components like LoginForm, ProductList, and SearchBar.
2. **Application Layer**:
   * **Controllers**: Handle HTTP requests and route them to the corresponding service logic.
     + Example: AccountController, ProductController.
   * **Services**: Implement the core logic, process data, and communicate with DAOs.
     + Example: AccountService, ProductService.
3. **Data Access Layer**:
   * Models and DAOs:
     + Models (e.g., Account, Product) define the database schema.
     + DAOs (e.g., AccountDAO, ProductDAO) handle queries to MongoDB.

**Benefits of Updated Architecture**

1. **Maintainability**:
   * Separate layers make it easy to locate and modify code. For instance, database changes only affect DAOs, not controllers or services.
2. **Reusability**:
   * Logic in services (e.g., validating an email) can be reused across controllers.
3. **Extensibility**:
   * New features, like adding categories to products, can be implemented by extending existing services or DAOs without disrupting other layers.
4. **Separation of Concerns**:
   * Each layer handles distinct responsibilities, making debugging and scaling more straightforward.

**Classes/Modules in Each Layer**

Below are examples of classes/modules based on your current models and the updated architecture:

**Presentation Layer:**

* **Frontend Components**:
  + LoginForm: Handles user login.
  + ProductList: Displays products to the user.
  + SearchBar: Allows users to search for products.

**Application Layer:**

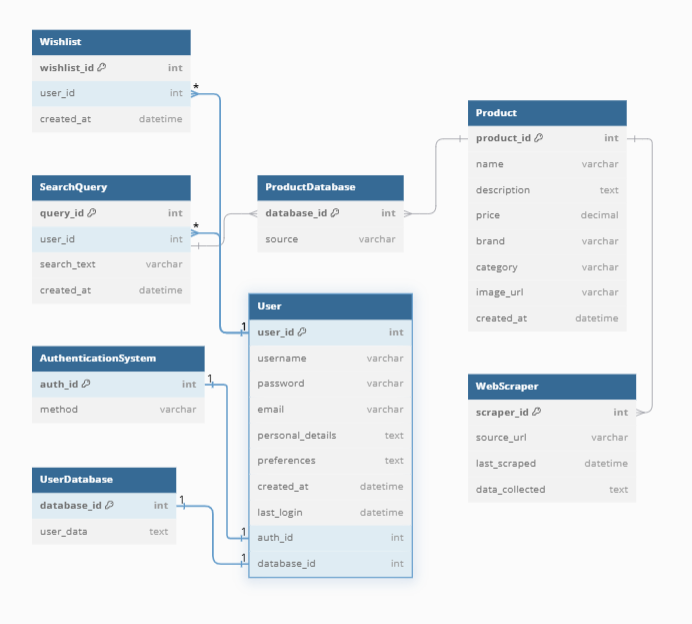
* **Controllers**:
  + AccountController.js: Handles routes like /login and /signup.
  + ProductController.js: Handles routes like /products and /products/search.
* **Services**:
  + AccountService.js: Validates user credentials and passes data to the DAO.
  + ProductService.js: Processes product search and wishlist logic.

**Data Access Layer:**

* **Models**:
  + Account.js: Defines schema for user accounts.
  + Product.js: Defines schema for products.
* **DAOs**:
  + AccountDAO.js: Contains database operations for accounts (e.g., createUser, findUserByEmail).
  + ProductDAO.js: Handles queries for products (e.g., searchProducts, addProduct).

.

# Data Models



### 

### **1. User**

* **user\_id**: Unique identifier for each user (Primary Key).
* **username**: Username chosen by the user.
* **email**: Email address associated with the user account.

### **2. Product**

* **product\_id**: Unique identifier for each product (Primary Key).
* **name**: Name or title of the product.
* **price**: Price of the product.
* **gender:** For men or women
* **Sizes:** Array of available sizes
* **Colours :** Array of colours
* **Description**: Brief description of the products

### **3. Wishlist**

* **wishlist\_id**: Unique identifier for each wishlist (Primary Key).
* **user\_id**: Foreign key referencing **User**; associates the wishlist with a specific user.
* **created\_at**: Timestamp of when the wishlist was created.

### **4. SearchQuery**

* **query\_id**: Unique identifier for each search query (Primary Key).
* **user\_id**: Foreign key referencing **User**; indicates which user made the search.
* **search\_text**: Text of the search query submitted by the user.

### **5. ProductDatabase**

* **database\_id**: Unique identifier for the product database (Primary Key).
* **source**: Source of the product data (e.g., API, web scraping).
* **created\_at**: Timestamp of when the product data was added to the database.

### **6. UserDatabase**

* **database\_id**: Unique identifier for the user database (Primary Key).
* **user\_id**: Foreign key referencing **User**; links the user to their stored data.
* **user\_data**: Serialized or structured data for user preferences or purchase history.

### **7. AuthenticationSystem**

* **auth\_id**: Unique identifier for each authentication method (Primary Key).
* **method**: Authentication method (e.g., password, OAuth).
* **created\_at**: Timestamp of when the authentication method was added.

### **8. WebScraper**

* **scraper\_id**: Unique identifier for each web scraper instance (Primary Key).
* **source\_url**: URL of the website being scrapped for product data.
* **last\_scraped**: Timestamp of the last data scrape from the source.

### **Relationships Overview**

* **User and Wishlist**: Each user can have multiple wishlists, where each wishlist contains multiple products.
* **User and AuthenticationSystem**: Links each user to an authentication method, showing how they log in.
* **User and UserDatabase**: Links user accounts to user-related data storage, including preferences and history.
* **Wishlist and Product**: A wishlist consists of multiple products, and products can appear in multiple wishlists.
* **User and SearchQuery**: Each search query is associated with a user who performs it.
* **Product and ProductDatabase**: All products are stored in the central product database, sourced from APIs or web scraping.
* **Product and WebScraper**: Some products are collected through a web scraper for sites without direct APIs.

# 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.

**Amazon S3**

Amazon S3 will serve as the primary storage solution for images associated with each product, reliably handling large volumes of image files with high availability and durability. Each product’s images will be stored in S3 buckets, with unique keys tied to product IDs, enabling straightforward retrieval and efficient organization. This setup allows for flexible scalability, as S3 can easily accommodate an expanding catalog of products and associated images without performance degradation. Additionally, S3's integration with other AWS services enables advanced features like image processing, optimization, and access control, ensuring images are securely managed, optimized for delivery, and quickly accessible for a seamless user experience on the platform.

# Who Did What?

|  |  |
| --- | --- |
| **Name of the Team Member** | **Tasks done** |
| Ahmad Kashif Jabbar | System Architecture - as it should be |
| Syed Messam Ali | Tools and Technologies |
| Husnain Ali | Introduction |
| Zainab Fatima | Architecture Diagram—As it is in the prototype code |
| Musa Aftab | Data Models |

# 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)** |
| Musa Aftab | System Architecture - as it should be |
| Syed Messam Ali | Data Model |
| Ahmed Jabbar | Tools and Technologies |
| Zainab Fatima | Introduction |
| Husnain Ali | Architecture Diagram - as it is |