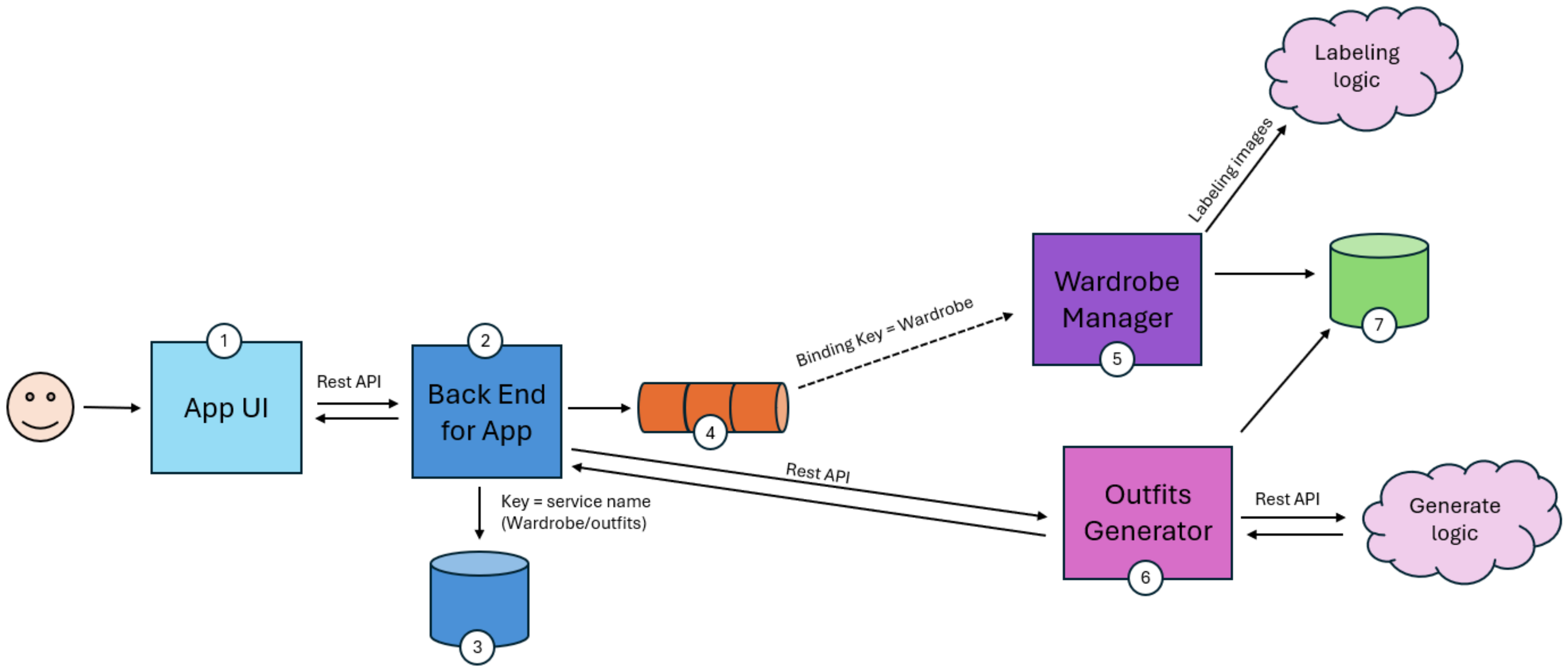
StyleMate: High Level Design

Team: StyleMate
Members: Stav Duck, Tal Dayan

Many people struggle with putting together stylish and appropriate daily outfits for various occasions. This challenge often leads to frustration, wasted time, and missed opportunities to make a good impression or feel confident.
We are developing an intuitive app that helps users quickly choose the perfect outfit based on the weather, occasion, and personal style preferences.

1. Main Components Diagram

The following diagram outlines the components planned for implementation in our application, providing a high-level overview of the connections and technologies involved.



1.1 App UI:
The user interface of the application will serve as the primary interaction point for users, providing access to various features and functionalities. It is planned to be implemented using **React**, The UI will be designed to ensure a seamless and intuitive user experience, incorporating modern design principles and best practices.

1.2 Back End for App:
The back end will act as the core of the application, handling business logic, authentication, and data processing. It will listen for requests from the UI via a RESTful. The authentication system will include a sign-up and login flow, where users can register and authenticate using their credentials.

1.3 Users Data Base:
The Users Database will be a **SQL-based relational database** responsible for securely storing and managing user information.
The database will be managed using a relational database management system (RDBMS) such as **PostgreSQL** or **MySQL**, ensuring data integrity, scalability, and efficient querying.

1.4 Message Broker:
The Message Broker will act as an intermediary for **asynchronous communication** between different services in the application. Instead of direct service-to-service communication, the message broker will decouple components, ensuring better scalability, fault tolerance, and reliability. We are considering using **RabbitMQ** as the Advanced Message Queuing Protocol (AMQP).

1.5 Wardrobe Manager:
The Wardrobe Manager will serve as a core service responsible for storing, organizing, and managing users' clothing items. It will allow users to **add, update, and delete** clothing items while also tracking their condition
To automate clothing categorization, the Wardrobe Manager will use an AI model for processing the images of the clothes and labeling them accordingly and store the results inside the DB.
By implementing the Wardrobe Manager, the application will offer an intelligent and organized way for users to manage their clothing items efficiently.

1.6 Outfit Generator:
The Outfit Generator will be a smart service that creates personalized outfit recommendations for users based on the clothing items available in their Wardrobe Database. It will use customizable logic and AI-driven techniques to generate outfits that match the user's preferences, weather conditions, and specific occasions.
The Outfit Generator will enhance the user experience by providing personalized, weather-aware, and event-specific outfit recommendations, making daily clothing choices effortless and efficient.

1.7 Wardrobe Data Base:
The Wardrobe Database will serve as a **NoSQL storage solution** for managing all clothing-related data. It will store metadata extracted from user-uploaded images and manual inputs, enabling efficient retrieval and outfit generation. The database will be implemented using **MongoDB**, a document-oriented database, which provides flexibility in handling unstructured and semi-structured data.

2. Data Model:

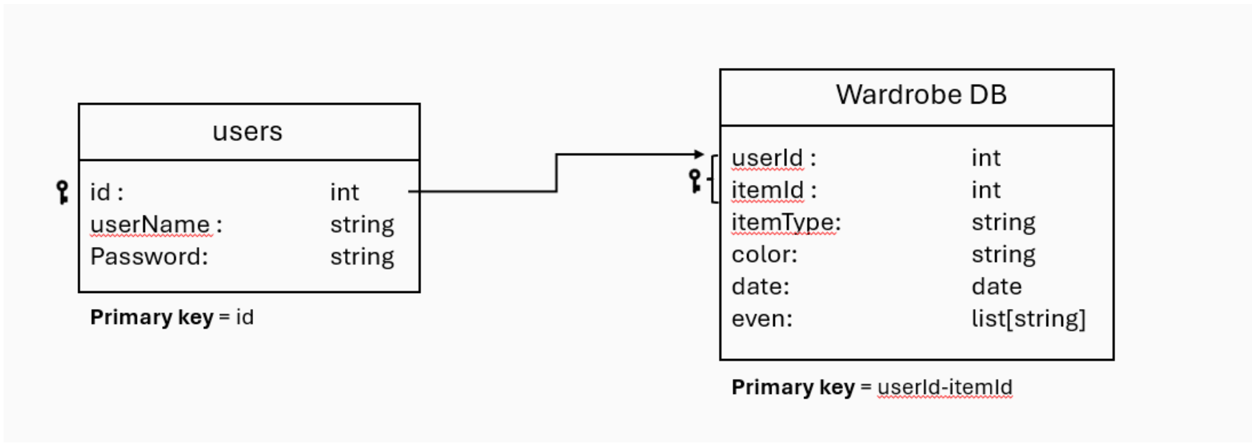
The following diagram illustrates the relationship between the **User Database Schema** and the **Wardrobe Database Schema**. This design ensures that each user's personal wardrobe data remains distinct and easily accessible

2.1 User Database Schema

- Every user will have a **unique user ID (`userId`)**, which serves as the primary key.
- This ID will be used to establish a relationship between the user and their wardrobe items.
- Other user-related attributes, such as **name, password** will be stored in this schema.

2.2 Wardrobe Database Schema

- Each wardrobe item (e.g., shirts, pants, shoes) will have:
  A unique `itemId` to differentiate between individual clothing pieces.
  An owner's `userId`, linking the item back to its respective user.
- This means that every wardrobe entry will be uniquely identified by a combination of `userId` and `itemId` to ensure:
- Separation of user data – No user can access another's wardrobe.
- Uniqueness of items – Two users can have identical clothing items, but each instance remains unique to its respective owner.



3. API Description:

3.1 Backend:

- /items/{userId}:
    - GET - get all wardrobe for user id

```
Output: {
          "Shirts": [ {
                     "item_id": int,
                      "image": image,
                      "color": string,
                     "last_used": date
          } ],
           "Pants": [ {
                     "item_id": int,
                      "image": image,
                      "color": string,
                     "last_used": date
          } ],
           "Shoes": [ {
                      "item_id": int,
                      "image": image,
                      "color": string,
                     "last_used": date
           ] }
}
```

- /item/{userId}/{itemId}:
    - GET - get specific item

      ```
      Output: {
              "item":  {
                      “Item_type”: string,
                      "item_id": int,
                       "image": image,
                       "color": string,
                      "last_used": date
                      }
              }
      ```
    - DELETE - delete specific item from wardrobe
- /item/{userId}:
    - POST - create new item for user id

      ```
      Output: {
              "new_item":  {
                      “Item_type”: string,
                      "item_id": int,
                       "image": image,
                       "color": string,
                      "last_used": date
                      }
              }
      ```
- /outfit/{userId}:
    - POST - get generated outfit by user id

      ```
      Parameters: {
              “Date”: date,
              “Weather”: string,
              “Is_raining”: bool,
              “Event”: string
      }
      Output: {
              “Shirt_id”: int,
              “Pants_id”: int,
              “Shoes_id”: int
      }
      ```

3.2 Messaging Queue:

- Sending a message for creating a new item as a json format. It will contain the userId and item's image.

```
{
        message_id: {int}
        time_stamp: {time}
        user_id: {int}
        Image: {serialized image}
}
```

- Sending a messageas a json format for deleting an item from the wardrobe. It will contain the userId and itemId.

```
{
        message_id: {int}
        time_stamp: {time}
        user_id: {int}
        Item_id: {int}
}
```

3.3 Outfit Generator:

- /outfit/{userId}:
  - POST - get generated outfit

```
Parameters: {
                "Date": date,
                "Weather": string,
                "Is_raining": bool,
                "Event": string
        }
Output: {
                "Shirt_id": int,
                "Pants_id": int,
                "Shoes_id": int
        }
```

4. **Use Case Diagram:**

The user interface of the application provides three primary options:

4.1 **Add New Item:**

- When adding a new clothing item, the user will upload an image of the clothing. The image will undergo AI-powered labeling technology, which processes the image to identify key features of the item (e.g., type of clothing, color, patterns, material). The identified information will then be analyzed and stored in a database for future use.
- This data will be tagged with relevant keywords and properties to allow the system to recognize the item in future interactions and make recommendations.

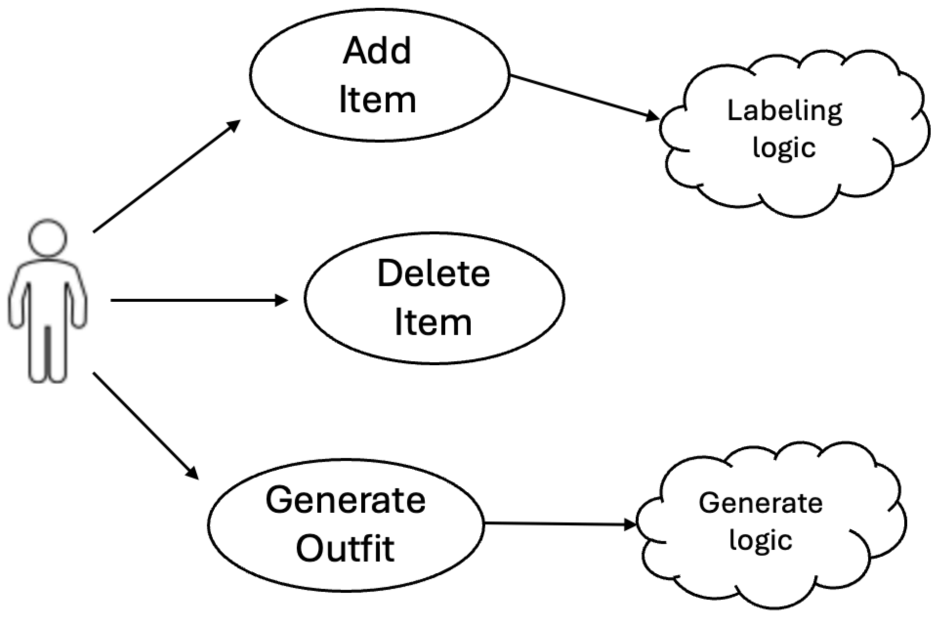
4.2 Delete Item:

- The user can remove an existing clothing item from their personal collection within the application. This will update the database, ensuring the item no longer appears in their available wardrobe list.

4.3 Generate Outfit:

- The user can input their personal preferences (e.g., style, color, occasion) and the system will use AI algorithms to generate a complete outfit using the clothing items the user has previously added.
- The AI will analyze the existing items, match them according to the user's personal data and preferences, and suggest an outfit that is visually and contextually appropriate. This allows the user to receive a tailored recommendation based on their personal wardrobe.

The overall goal of the application is to enhance the user's wardrobe experience by leveraging AI to personalize the customization, addition, and removal of clothing items, while also generating fashion-forward outfit suggestions based on their unique data.



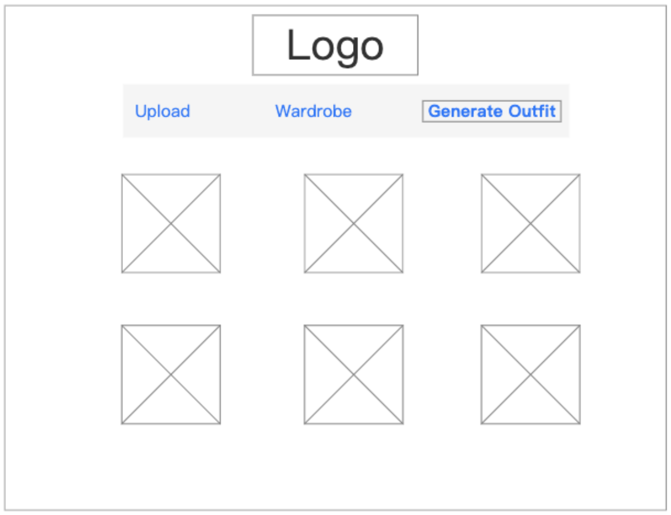
5. **<u>UI Implementation</u>**:
   The user interface (UI) of the application will be implemented using **React**,
   React's component-based architecture will llow for modular, reusable UI elements and seamless state management, ensuring a smooth user experience.

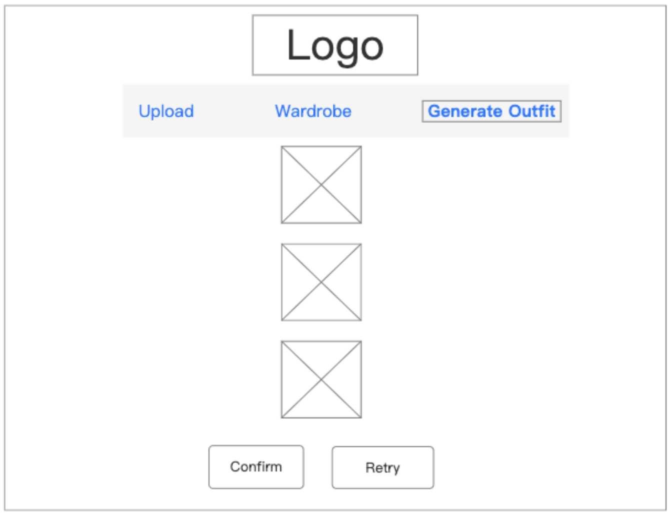
6. Main Pages In The Application

The following mockups of the high-level layout for our UI pages will be presented to the user within our application.

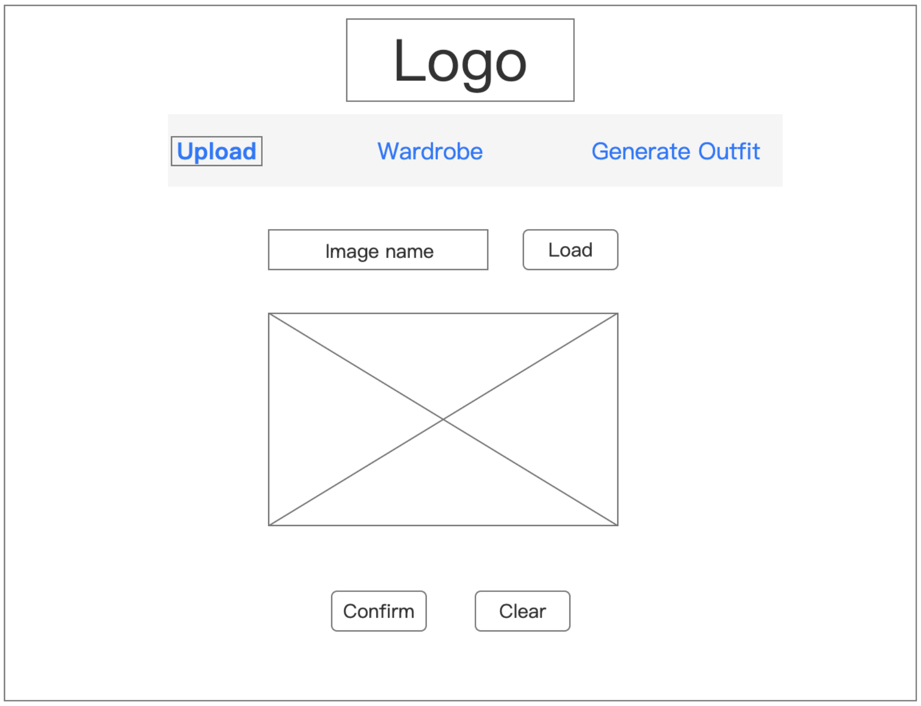
6.1 Starts Page:

When the user first opens the app, the main screen will be “Generate Outfit” screen:
It will have a 'Generate' button that will provide a new outfit suggestion. The user can continue generating outfits until they are satisfied.

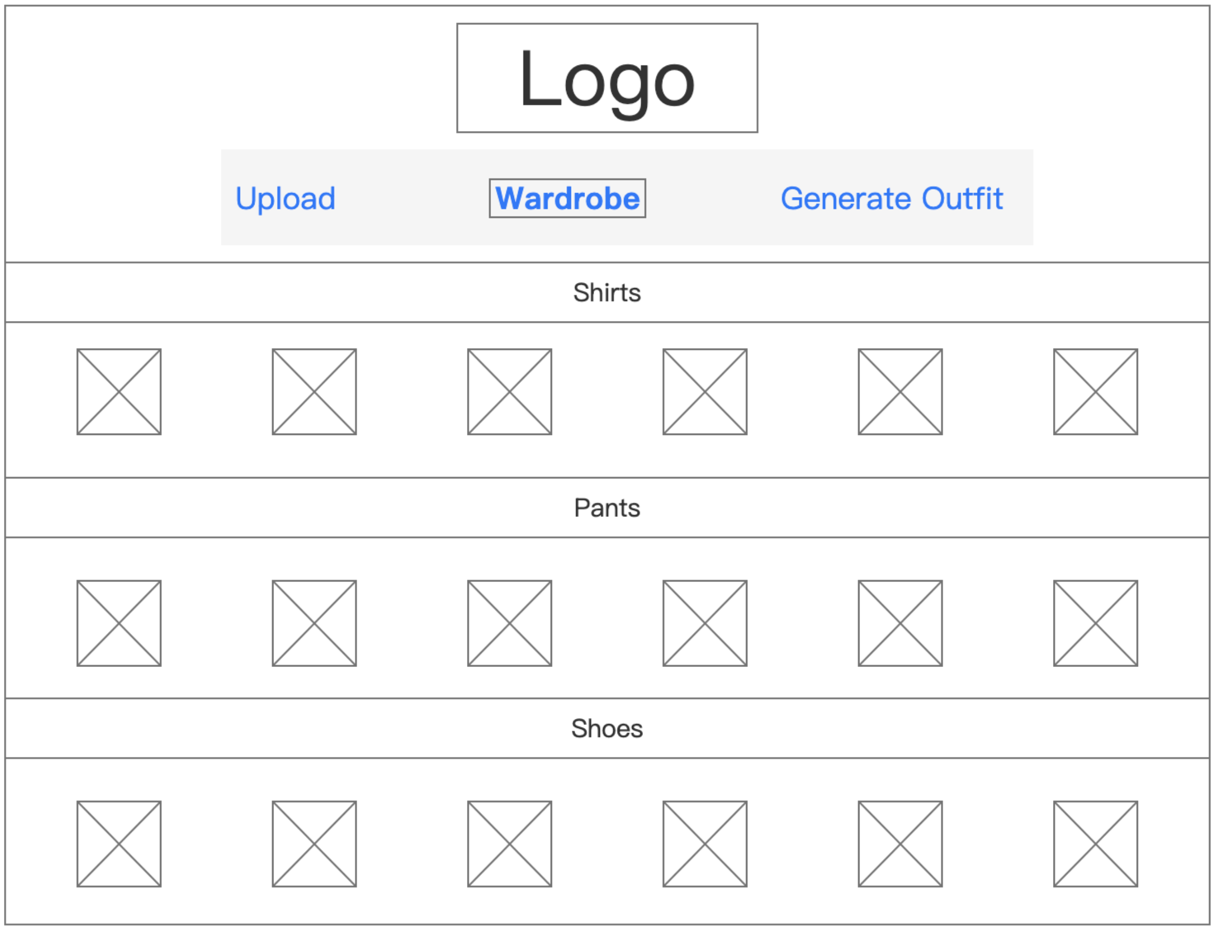




6.2 Upload screen:
The user will be able to take a picture or load a new image of the new item and upload it to their profile. It will then be used later in the outfit generation feature.



6.3 Wardrobe screen:
This page will present the user with information about all the items they currently have, organized by item type (e.g., shirt, pants, etc.). The user will be able to manage, update, and delete the items as desired.


Logo
Upload
Wardrobe
Generate Outfit
Shirts
Pants
Shoes