Product Design

Team 42

Intelligent Wardrobe Management System

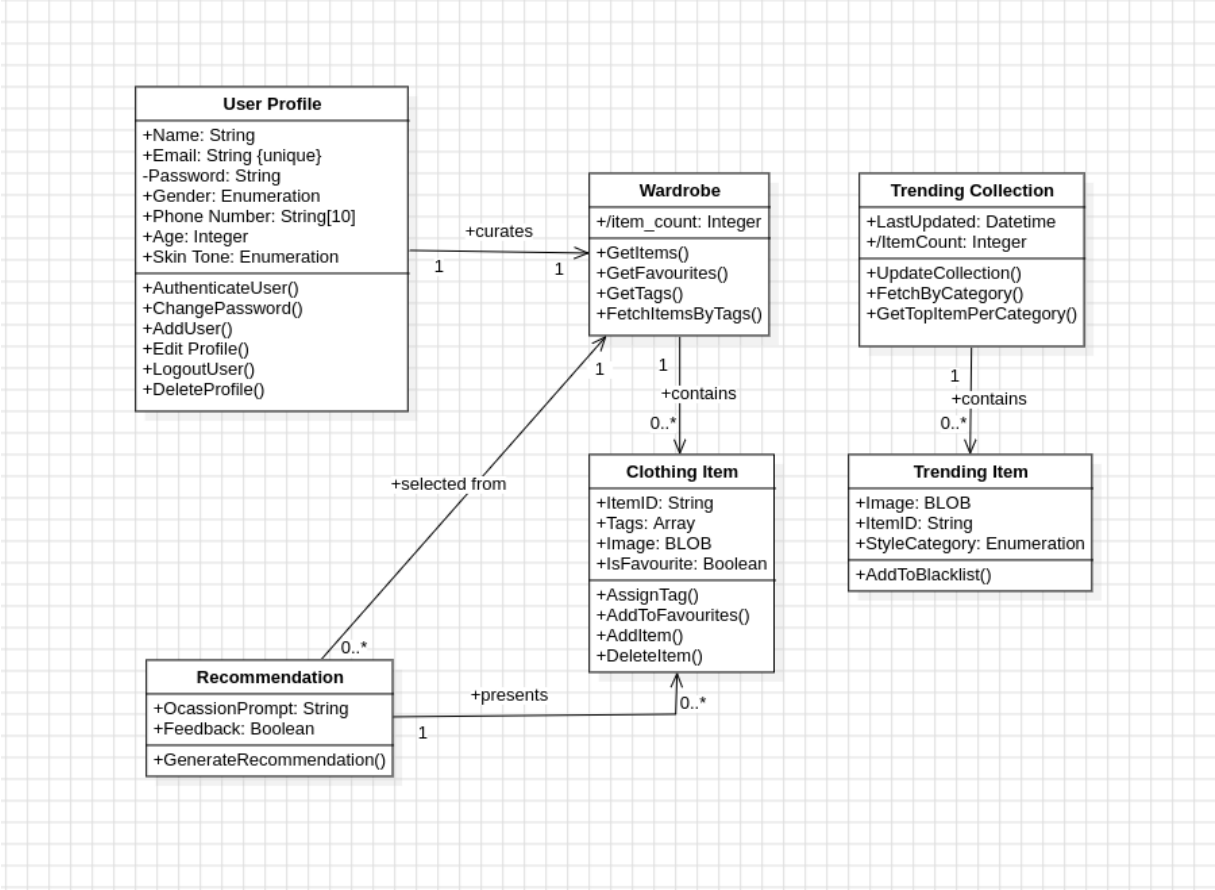
Shaunak Biswas, Vansh Motwani, Harpreet Singh,

Jayesh Rajesh Sutar, Raveesh Vyas

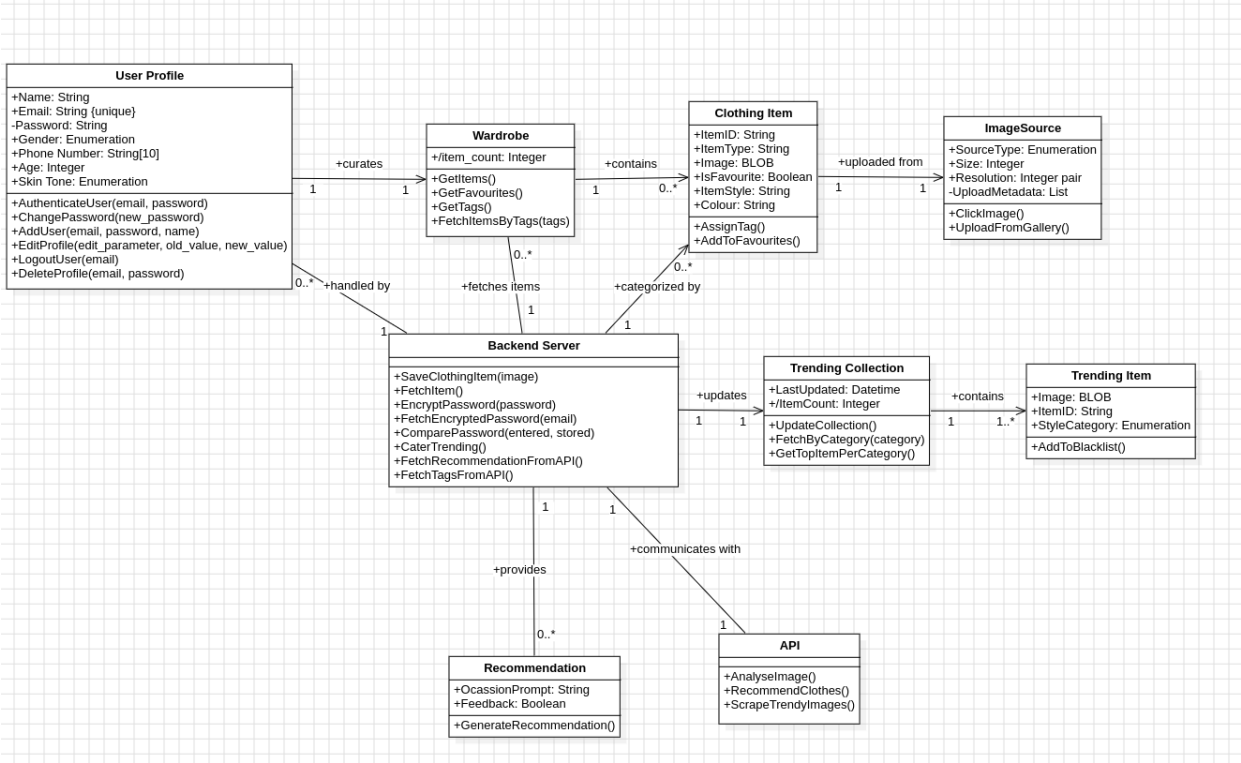
Design Model

We have presented two class diagrams at different levels of abstraction: the architecture-level diagram provides an overview of system components, while the low-level design diagram delves into implementation level details. The Sequence diagrams are based on the low-level class design.

Architecture Level Design:



Low Level Design:



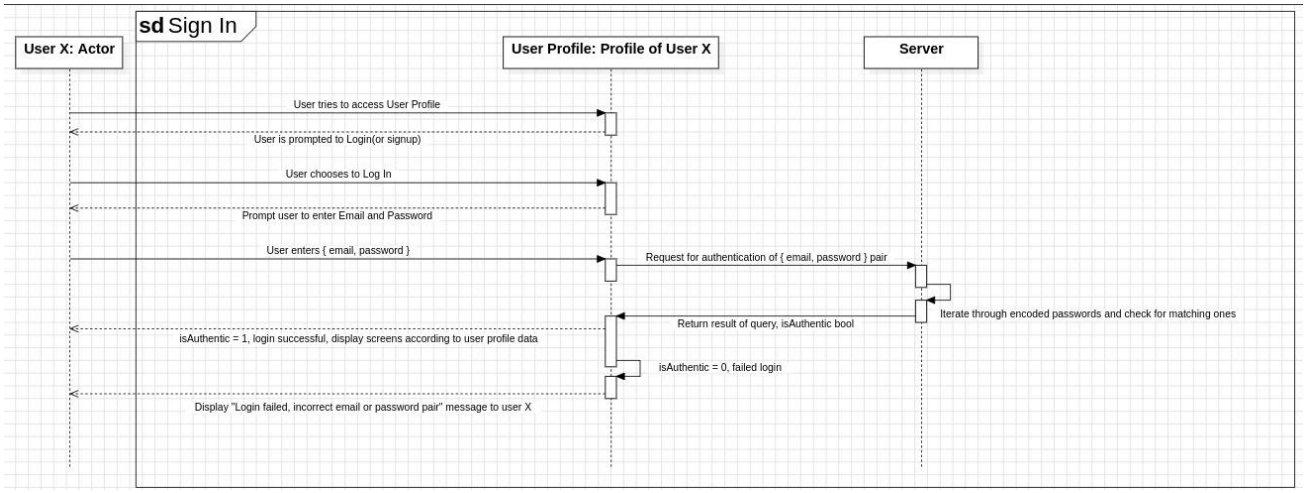
User Profile	Class state • The class stores all information about a user's profile that he entered while signing in. • This includes name, email id, password, gender, phone number, age, skin tone Class behavior • Allows users to sign in and sign out • Change password and other attributes as well • Delete their profile
Wardrobe	Class state • Number of items in the wardrobe Class behavior • Allows user to see all items • Allows user to see favourites • Allows user to get the item tags (casual, black t-shirt) • Allows user to get items based on tags he enters
Clothing item	Class state • All attributes of the item, such as type, style and colour. • Whether the user has added it to favourites. • Image and ID Class behavior • System can allocate tags to the image. • User can add to favourites
Image source	Class state • Size and resolution of image • Type of source • Other metadata associated

	Class behaviour • Click image, or upload from gallery
Backend server	Class behavior • Communicate with API to get tags of clothes, and recommendations for users • Shows the user trending items • Saves clothing images uploaded by user • Fetches clothes that the user wants to see • Encrypts passwords • Compares the password entered by the user to the encrypted password.
Trending collection	Class state • When was the collection last updated • Number of items Class behavior • Update the collection • Get the items in the collections according to category • Get the top items of all categories
Trending item	Class state • Image • ID • Category Class behavior • Add the item to blacklist
Recommendations	Class state • Occasion of the wear • Feedback on the item Class behavior • Generates a recommendation based on user inputs
API	Class behavior • Analysing Images • Recommend clothes • Scrape trending clothes from the internet

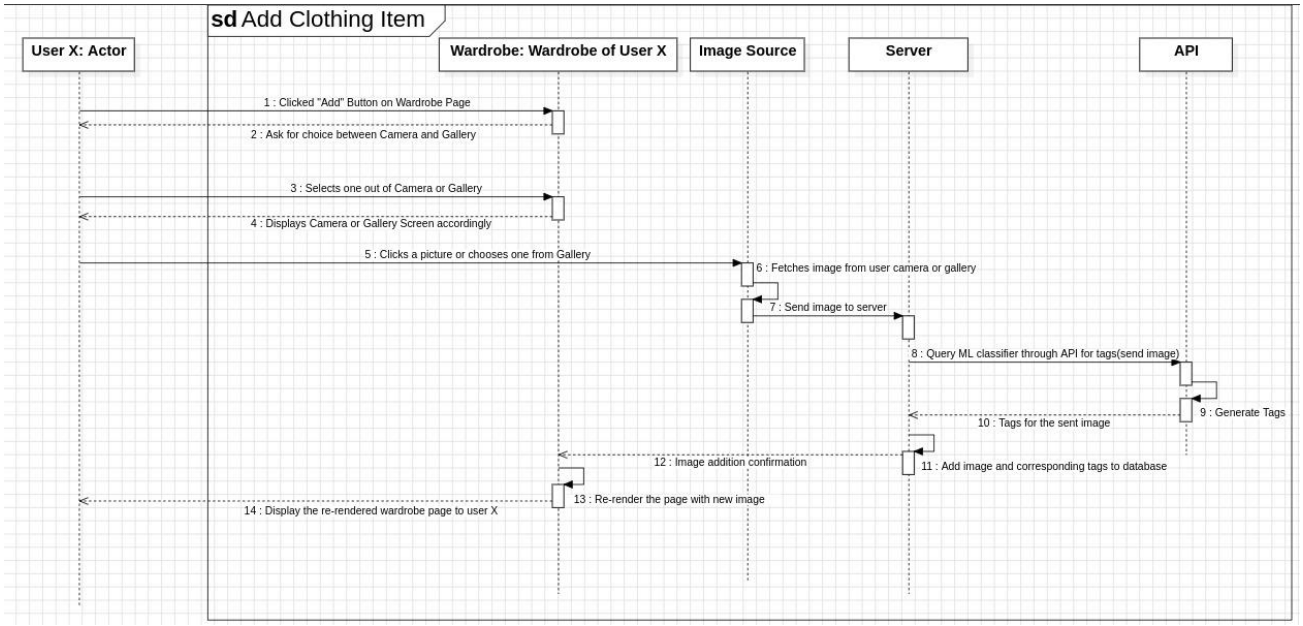
.

Sequence Diagram(s)

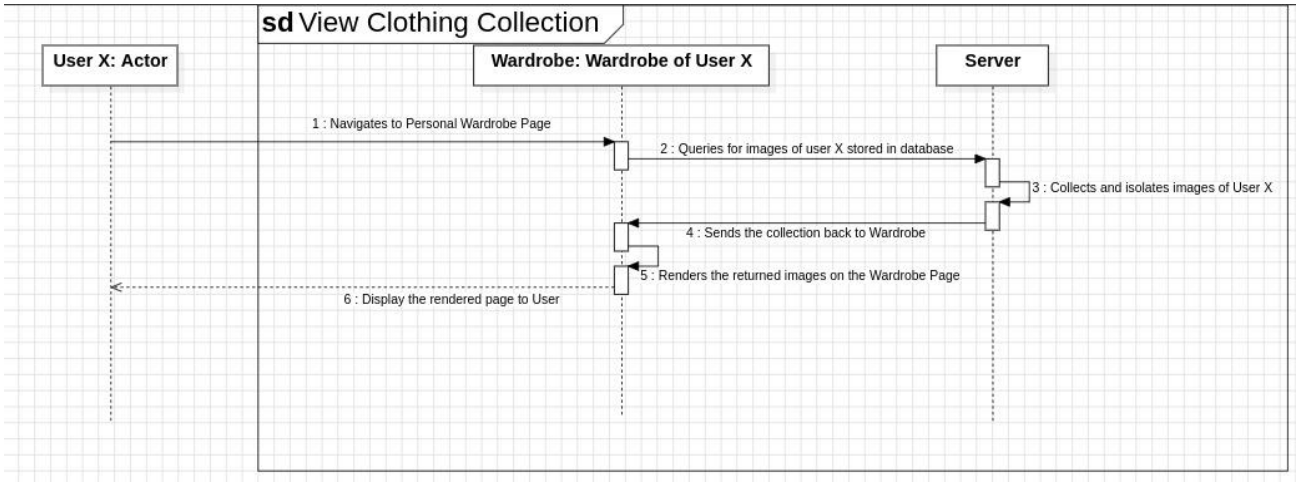
1.Sign In:



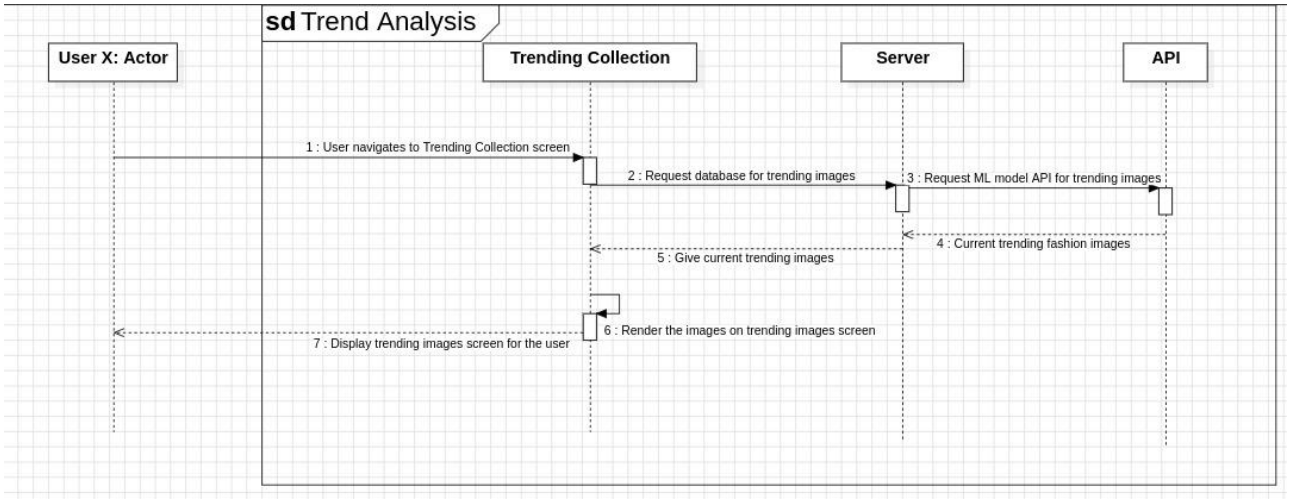
2.Adding Clothing Item:



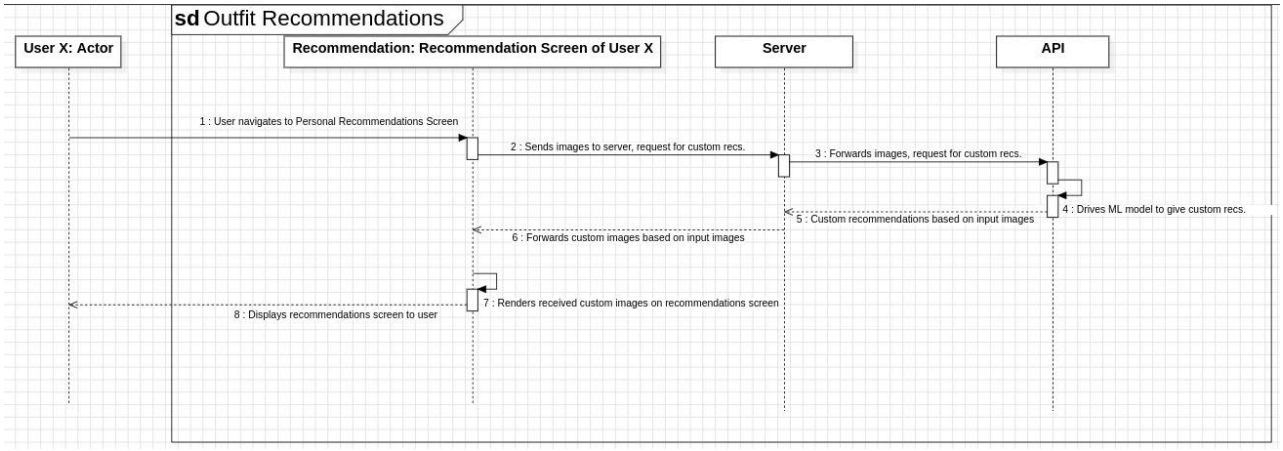
3.View Clothing Collection:



4.Trend Analysis:



5.Outfit Recommendation:



Design Rationale

1. Issue: Efficient Storage and Management of Clothing Item Attributes

The issue at hand was finding a more efficient way to store, manage, retrieve, and display attributes for each clothing item. Initially, tags were assigned to each item, but this approach proved to be inefficient for the system's requirements.

Decision: Replace Tags with Color, Type, and Style Attributes

To address the inefficiency in storage and management, we decided to replace the tags assigned to each clothing item with three distinct attributes: color (e.g., yellow), type (e.g., t-shirt), and style (e.g., ethnic, partywear). This new approach allows for more efficient storage, management, retrieval, and display of clothing item attributes.

Rejected: Alternative Approaches

1. **List of Tags:** Initially, we considered continuing with the use of a list of tags for each clothing item. However, this approach was rejected due to its limitations in filtering clothing items efficiently.

2. **Hierarchical Tagging System**: Another rejected solution was the implementation of a hierarchical tagging system, where tags would be organized into a tree-like structure. While this approach could potentially offer more granularity in categorizing attributes, it was deemed overly complex and difficult to manage.

3. **Combination of Tags and Attributes:** We also explored the possibility of combining the use of tags with the new attributes (color, type, style). However, this approach was rejected as it would introduce redundancy and complicate the management of clothing item attributes.

By replacing tags with color, type, and style attributes, we were able to streamline the storage, management, retrieval, and display of clothing item attributes, improving the overall efficiency of the system.

2. **Issue: Collecting and Analyzing User Feedback for Recommendations**

Initially, the issue arose regarding the collection and analysis of user feedback for recommendations. The approach involved implementing complex forms and applying Natural Language Processing techniques to understand the sentiment behind user feedback. However, this approach proved to be cumbersome and resource-intensive, leading to the exploration of alternative solutions.

Decision: Implementation of a Thumbs-Up, Thumbs-Down System

After careful consideration, we determined that the best solution for collecting user feedback on recommendations is to implement a simple thumbs-up, thumbs-down system. This straightforward approach allows users to easily express their satisfaction or dissatisfaction with recommendations without the need for complex forms or NLP analysis.

Rejected: Alternative Approaches

1. **Complex Forms and NLP Analysis:** The initial approach involved using complex forms to gather detailed feedback from users and applying NLP techniques to analyze the sentiment behind the feedback. However, this approach was rejected due to its complexity and resource requirements, which could potentially lead to poor user experience.

2. **Star Ratings:** Another alternative considered was implementing star ratings or Likert scales to gather user feedback. While these methods offer a more nuanced understanding of user sentiment compared to a simple thumbs-up, thumbs-down system, they also introduce complexity and may deter users from providing feedback.

By implementing a thumbs-up, thumbs-down system for collecting user feedback on recommendations, we aim to streamline the feedback process and enhance user engagement while maintaining simplicity and efficiency.

3. **Issue: Integrating Python and JavaScript Code**

The issue revolved around integrating Python code, specifically for machine learning models, with JavaScript code. Our aim was to establish an interface between the two languages. The integration was between backend logic and ML models being developed by another team.

Decision: Abstraction and Separation of Concerns with FastAPI

The proposed solution involved adopting an abstraction and separation of concerns approach by creating API endpoints for the Python code, which houses the machine learning models. This interface can be implemented using frameworks like FastAPI.

Rejected: Direct Python Backend and Alternative Interface Methods

We rejected the initial solution of directly building the backend in Python. Additionally, alternative methods of interfacing, such as using traditional REST APIs or WebSocket connections, were also considered but ultimately rejected in favor of the abstraction layer provided by FastAPI.