Table of Contents

1.0 Preliminary Project Proposal Document

2.0 Proposal Document and Presentation Slides

4.0 Software Development Plan

4.1 Plan Introduction

This Software Development Plan provides the details of the planned development for the induō Software CSCI which provides an iOS application to help you manage your wardrobe virtually, allowing you to easily traverse your clothes and effortlessly generate AI-powered outfit recommendations for personalized ranking

induō is an AI-powered iOS application designed to revolutionize wardrobe management by streamlining outfit selection and promoting sustainable fashion habits. It enables users to upload and categorize their clothing items, then generate AI-based outfit recommendations tailored to their preferences, events, and weather conditions. The app also features a virtual try-on system that allows users to visualize AI-generated outfit combinations on themselves or another individual of their choosing. Developed using Swift and Firebase, induō integrates advanced AI tools for clothing descriptions and outfit generation, while prioritizing user personalization and data security. The development will involve activities such as designing the user interface, integrating Firebase for data management, building AI-driven outfit generation, testing the system, and ensuring security compliance. The rationale behind induō's development is to address the time-consuming and inefficient process of selecting outfits, offering a modern, tech-driven solution that encourages creativity and sustainable consumption. All induō project subtask completion dates (milestones) can be found in the *Deliverables and Due Dates* Table under section 4.1.1 Project Deliverables.

4.1.1 Project Deliverables:

Table: Deliverables and Due Dates		
Number	**Deliverables**	**Due Date**
D#01	**Develop visually pleasing and fully functional App Login and Profile Page:** ● Design and implement the user interface (UI) for the Welcome/Sign-in Page and Profile Page. ● Integrate Firebase Authentication for user login, third-party login, and profile management. ● Set up and maintain the Firebase backend for	10/17/2024

	storing profiles in a database.	
D#02	**Develop visually pleasing Virtual Closet Page with Clothing Upload Functionality:** ● Design and implement the user interface (UI) for the Virtual Closet Page. ● Develop the functionality for users to upload clothing images implementing background removal. ● Set up and maintain the Firebase backend for storing clothing images in a database. ● Display uploaded Clothing Images in an organized fashion on Virtual Closet Page.	10/25/2024
D#03	**Generate Outfit Matches onto user:** ● Develop and integrate the AI workflow for rendering the AI-generated outfit matches using fal-ai/cat-vton based on the user's virtual closet. ● Set up and maintain the Firebase backend for storing outfit match images in a database.	10/28/2024
D#04	**Generate and Store Descriptions of each uploaded clothing item and outfit match:** ● Setup local LLM using Olama and develop prompt engineering for generating detailed descriptions of clothing and outfits, including event-specific details. ● Store in respective databases via Firebase.	11/2/2024
D#05	**Develop visually pleasing and fully functional Outfit Match Page:** ● Implement the swipe-based interface and display the necessary pre-rendered AI-generated outfits for the clothing item sent to the matching page or for the outfits that match the user's input event prompt.	11/7/2024
D#06	**Implement Chatbot on Virtual Closet Page:** ● Utilize local LLM using Olama to offer answers for clothing and outfit suggestions based on previously generated descriptions of the clothes from Firebase.	11/13/2024
D#07	**Establish method for Outfit Prioritization based on Personalized Style Rankings:** ● Gather user interactions (likes/dislikes) to determine style scores of each outfit, and analyze data for improving AI suggestions. These scores will be stored within the outfits Firebase database.	11/19/2024
D#08	**Develop increased User Management:**	11/21/2024

	● Implement profile management features, including profile edits, and logout functionality.	
D#09	**Ensure Security:** ● Implement security measures like data encryption and ensure compliance with data privacy regulations.	11/25/2024
D#10	**Verification Testing:** ● Test all features for usability, performance, and security, including both manual and automated testing.	11/28/2024

4.2 Project Resources

The induō project team consists of Caroline Ellis, Breea Toomey, and Jack Seymore.

4.2.1 Hardware Resources

The Hardware Resources section of the software development plan for induō will outline the specific hardware requirements needed for development, database storage, and app execution. For development, the team will utilize a machine equipped with an M2 processor and 32GB of RAM, alongside a cloud-hosted A100 GPU with 40GB of VRAM and 10 CPUs provided by fal.ai PaaS. A system swap mechanism will also be included to ensure smooth transitions between different hardware configurations as required. This setup enables high-performance processing capabilities, particularly for tasks requiring advanced machine learning and AI computations, and the cloud-hosted GPU allows scalable access to powerful hardware as needed. The database will be stored on a Firestore distributed database utilizing solid-state drives. Finally, the app will be designed to run on mobile iOS devices with an A11 Bionic processor or newer and at least 3GB of RAM. These hardware specifications will ensure that the development process runs smoothly and that the final app can be executed efficiently on target devices.

4.2.2 Software Resources

The Software Resources section of the induō development plan will detail the specific software needed for development, database management, and app execution. During development, the team will utilize macOS Monterey 12 as the operating system. The primary development environment will be XCode 13.3, which serves as the IDE (Integrated Development Environment) and compiler. For AI functionalities, the plan involves utilizing a multimodal LLM for image to text generation, managed through the Ollama application. Image generation will be handled by the virtual try-on (VTON) model, running on the fal.ai platform (PaaS). Swift Core ML library will be used for image masking tasks including background removal. Version control and code repository will be facilitated by Github. For the database, the Firebase SDK will be employed to manage the NoSQL document database. Finally, app execution will require iOS 12.4 or later on the user's device, with an active internet connection. However, limited offline functionality will be available using pre-loaded data.

4.3 Project Organization

The Induo application development project has been divided into three major sub-teams, each responsible for distinct aspects of the application. These sub-teams collaborate closely to ensure that the application development process runs smoothly and all project requirements are met. Below is a detailed description of each sub-team, their responsibilities, and how they contribute to the overall success of the project.

Front-End Development Sub-Team

Team Lead: Jack Seymour

The Front-End Development Sub-Team is responsible for designing and implementing the user interface (UI) and enhancing the overall user experience (UX). This sub-team ensures that all front-end components are visually appealing, intuitive to navigate, and responsive across different devices. Key deliverables include developing the UI for the Welcome Page, Profile Page, Virtual Closet Page, and Outfit Match Page, as well as implementing interactive elements and ensuring seamless integration with the backend.

Back-End Development Sub-Team

Team Lead: Breea Toomey
The Back-End Development Sub-Team manages the underlying server, database, and application logic. This includes setting up and maintaining Firebase for storing user profiles, clothing images, and outfit matches, as well as ensuring data security and compliance with privacy regulations. The team is also responsible for implementing user management features and integrating AI models that support the core functionality of the application.

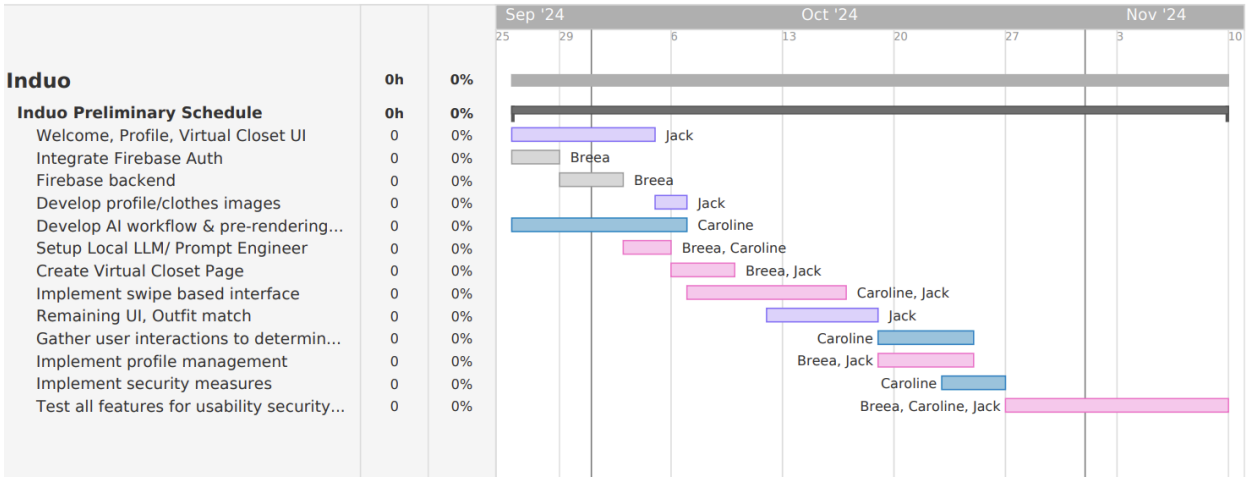
AI Development Sub-Team

Team Lead: Caroline Ellis
The AI Development Sub-Team focuses on integrating AI-based functionalities into the application. This includes developing and refining models that generate outfit recommendations, categorizing clothing items, and personalizing suggestions based on user feedback. The team is also responsible for integrating the chatbot that provides users with clothing and outfit recommendations.

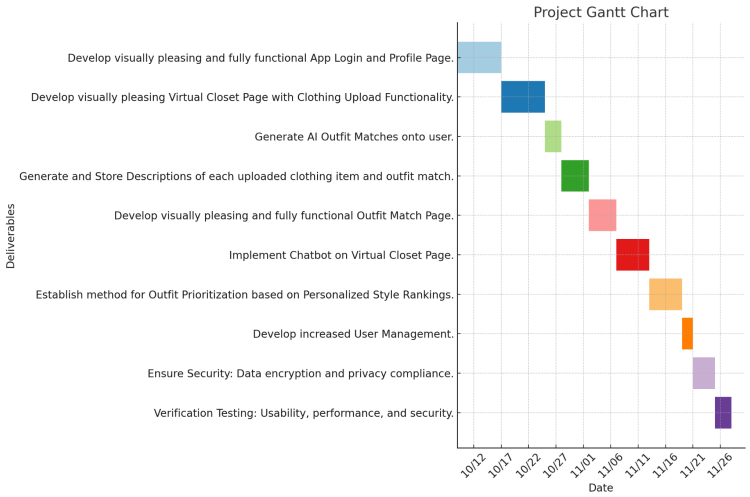
4.4 Project Schedule

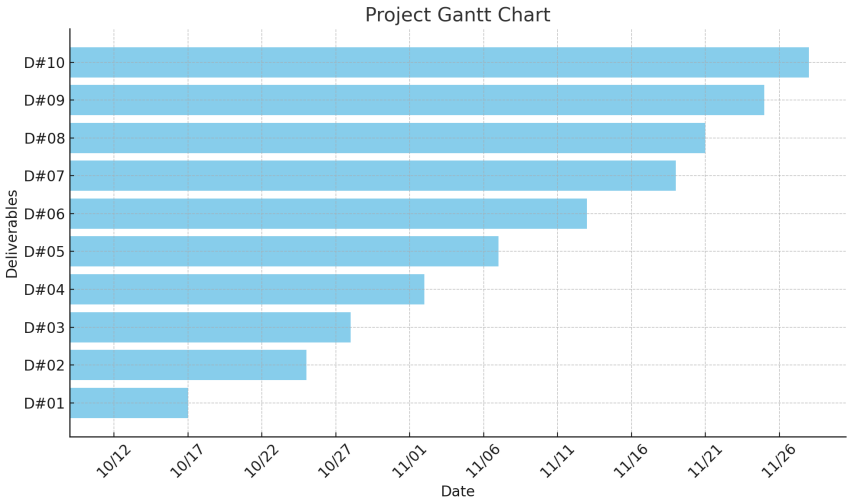
This section provides schedule information for the induō project.

4.4.1.1 Initial GANTT Chart



4.4.1.2 Master GANTT Chart





4.4.2 Task / Resource Table

Task	Assigned To	Hardware	Software
Design and implement the user interface (UI) for all pages (Welcome Page, Profile Page, Virtual Closet, Outfit Match).	Jack	MacBooks, iOS devices for testing	Xcode, Swift, GitHub for version control
Integrate Firebase Authentication for user login, third-party login, and profile	Breea	MacBooks, iOS devices	Firebase, Swift, Xcode

management.			
Develop the functionality for users to upload a profile image as well as clothing images and implement background removal.	Jack	MacBooks, iOS devices	Swift, Xcode, Firebase, image processing libraries
Set up and maintain the Firebase backend for storing profiles, clothing images, and metadata.	Breea	MacBooks	Firebase, Swift, Xcode
Create the Virtual Closet page that displays and manages user clothing items, including search and categorization.	Jack + Breea	MacBooks, iOS devices	Swift, Xcode, Firebase, AI image categorization tools
Setup local LLM and develop prompt engineering for generating detailed descriptions of clothing and outfits, including event-specific details.	Breea + Caroline	MacBooks M3 Pro (for LLM)	Ollama, Firebase, Swift
Develop and integrate the AI workflow for rendering the AI-generated outfit matches using fal-ai/cat-vton based on the user's virtual closet.	Caroline	MacBooks M3 Pro for LLM hosting, iOS devices	fal-ai/cat-vton, Ollama, LLM, Firebase
Implement the swipe-based interface and display the necessary pre-rendered AI-generated outfits for the clothing item sent to the matching page or for the outfits that match the user's input event prompt.	Jack + Caroline	MacBooks, iOS devices	Firebase, Swift, Xcode
Gather user interactions (likes/dislikes) to determine style scores of each outfit, and analyze data for improving AI suggestions.	Caroline	MacBooks, iOS devices	Firebase, Swift
Implement profile management features, including profile picture uploads, edits, and logout functionality.	Jack + Breea	MacBooks, iOS devices	Firebase, Swift, Xcode
Implement security measures like data encryption and ensure	Caroline	MacBooks	Firebase

compliance with data privacy regulations.			
Test all features for usability, performance, and security, including both manual and automated testing.	Jack + Breea + Caroline	iOS devices, MacBooks	Xcode, testing frameworks (e.g., XCTest)

5.0 Requirements Specification

5.1 Introduction

The induō system is a mobile application designed to assist users in managing their wardrobe and enhancing their fashion choices. It enables users to upload images of their clothing items, generate AI-based outfit recommendations, and visualize these outfits through a virtual try-on interface. By incorporating advanced AI technologies, such as hosting a large language model (LLM) for generating detailed clothing descriptions and integrating with an API-based virtual try-on feature, induō aims to streamline wardrobe management, increase creativity in outfit selection, and encourage sustainable fashion consumption. The application allows users to swipe through AI-generated outfits, upload clothing images with background removal, and store user preferences and wardrobe data securely using Firebase. The system is built as an iOS application using the Swift programming language and integrates Firebase as the backend to store user profiles and clothing metadata.

The system is composed of several key components, including the Graphical User Interface (GUI) for user interaction, AI processing for outfit generation and clothing descriptions including AI Outfit Matching Subsystem and AI Clothing Description Subsystem, and Firebase Backend Subsystem for storing data. Each component has specific functional and performance requirements, which are detailed in the subsequent sections of this document. The high-level diagram below represents the primary system components and their interactions.

The high-level system diagram below outlines the main components of the induō application:

System Components Overview

1. **Graphical User Interface (GUI)**: This includes various user-facing screens where users interact with their wardrobe, upload clothing items, and assess AI-generated outfits.
2. **AI Outfit Matching Subsystem**: The AI-driven subsystem responsible for generating outfits and managing the swipe-based interface for user selections.
3. **AI Clothing Description Subsystem**: Generates detailed descriptions of individual clothing items and complete outfits using the locally hosted LLM.
4. **Firebase Backend Subsystem**: Manages the storage of all user data, including profile information, wardrobe data, and AI-generated outfits, securely in Firebase.

5.2. CSCI Component Breakdown

CSCI induō is composed of the following CSCs:

5.2.1 Graphical User Interface (GUI) CSC

This CSC is responsible for the user-facing portion of the app, where users interact with their wardrobe, upload clothing items, and prompt/assess generated images of themselves in outfits.

- **5.2.1.1 Virtual Closet Main Screen CSU**
  Displays the user's wardrobe and facilitates interaction with individual clothing items.
  - **5.2.1.1.1 WardrobeDisplay module**: Displays all clothing items as scrollable images.
  - **5.2.1.1.2 ClothingItem module**: Represents individual clothing items, enabling selection for matching.
  - **5.2.1.1.3 SearchBar module**: Allows users to input event/weather prompts to find suitable outfits.
- **5.2.1.2 Clothing Item Upload Panel CSU**
  Manages image uploading, background removal, and local storage.
  - **5.2.1.2.1 ImageUpload module**: Handles image selection and upload.
  - **5.2.1.2.2 BackgroundRemoval module**: Removes backgrounds using image processing.
  - **5.2.1.2.3 FirebaseStorage module**: Stores filenames and metadata in Firebase.
- **5.2.1.3 Profile Page CSU**
  Handles user profiles, including uploading a full-body image and managing profile data.
  - **5.2.1.3.1 ProfileInput module**: Manages full-body image uploads.
  - **5.2.1.3.2 ProfileEdit module**: Allows users to update personal information.
- **5.2.1.4 Swipe Screen UI CSU**
  This CSU is responsible for managing the swipe interaction where users like or dislike AI-generated outfits.
  - **5.2.1.4.1 SwipeGestureHandler module**: Manages the gesture input for swipe left (dislike) or swipe right (like) interactions.
  - **5.2.1.4.2 SwipeFeedback module**: Visually highlights swipes, provides feedback for liked/disliked outfits (e.g., animations, color change).

5.2.2 AI Outfit Matching CSC

This CSC handles the AI-generated outfit generation and the "Tinder-style" swiping interface.

- **5.2.2.1 AI-Generated Outfit Rendering CSU**
  Handles API calls to fal-ai/cat-vton for AI outfit generation.
  - **5.2.2.1.1 APIRequest module**: Sends API requests to fal-ai/cat-vton with user data and clothing images.
  - **5.2.2.1.2 ImageProcessing module**: Receives and processes generated outfit images.

- **5.2.2.2 Outfit Matching Swipe Screen CSU**
  Displays pre-rendered AI-generated outfit combinations for users to swipe through.
    - **5.2.2.2.1 OutfitDisplay module**: Collects and displays pre-rendered AI outfits for swiping display.
    - **5.2.2.2.2 StyleScore module**: Updates style scores in database table based on user interactions.

5.2.3 AI Clothing Description CSC

Responsible for generating detailed descriptions of clothing and outfits using Ollama-hosted LLM locally on a Macbook M3 Pro acting as our server.

- **5.2.3.1 Clothing Description Generation CSU**
  Generates descriptions for individual clothing items.
    - **5.2.3.1.1 LLMRequest module**: Sends images to the LLM for clothing descriptions.
    - **5.2.3.1.2 FirebaseSync module**: Syncs descriptions with Firebase.
- **5.2.3.2 Outfit Description Generation CSU**
  Generates descriptions for complete outfits and provides event-specific suggestions.
    - **5.2.3.2.1 LLMOutfitRequest module**: Requests descriptions and event/environment recommendations for full AI-generated outfits.
    - **5.2.3.2.2 EventRecommendation module**: Sends user search bar prompt to LLM with descriptions of all outfits to determine which outfits best align with the user's request.

5.2.4 Firebase Backend CSC

Handles the storage of images, metadata, user profiles, and other application data in Firebase.

- **5.2.4.1 User Profile Management CSU**
  Manages user profiles, including full-body images and wardrobe data.
    - **5.2.4.1.1 FirebaseAuth module**: Handles user authentication.
    - **5.2.4.1.2 ProfileStorage module**: Stores user profiles, including clothing data.
- **5.2.4.2 Wardrobe Database Storage CSU**
  Manages the storage of clothing and outfit image filenames, and associated metadata.
    - **5.2.4.2.1 ClothingImage module**: Stores table of uploaded clothing item image filenames and descriptions.
    - **5.2.4.2.2 OutfitImage module**: Stores table of AI-generated outfit image filenames and style scores.

5.3 Functional Requirements by CSC

5.3.1 Graphical User Interface (GUI)

The GUI subsystem serves as the primary interface through which users interact with the induō application. It enables users to navigate between different features, upload clothing items, view outfit suggestions, and manage their profiles. The GUI will provide intuitive interactions and visual feedback to enhance user experience.

5.3.1.1 The GUI subsystem shall be displayed as an iOS application using the Swift programming language.

5.3.1.2 The subsystem shall consist of four key pages: Welcome Page, Profile Page, Virtual Closet Page, Outfit Match Page

5.3.1.3 The GUI subsystem shall provide clear navigation elements (e.g., buttons, tabs) for moving between pages of the application..

5.3.1.4 The GUI subsystem shall include a Welcome Page with the app name “induō” prominently displayed.

5.3.1.4.1 The subsystem shall display a login page with fields for both username and password inputs.

5.3.1.4.2 The subsystem shall provide a “forgot password” feature, allowing users to reset their password via email.

5.3.1.4.3 The subsystem shall validate user credentials and provide error messages for incorrect input.

5.3.1.4.4 The subsystem shall allow users to log in via third-party services (e.g., Google, Apple).

5.3.1.4.5 The subsystem shall display a welcome message upon successful login.

5.3.1.4.6 The subsystem shall provide an option for users to create a new account.

5.3.1.5 The GUI subsystem shall include a Profile Page that has input for and displays the user’s name, account information, and profile picture.

5.3.1.5.1 The subsystem shall allow users to upload a full-body picture, which will remain displayed on the profile.

5.3.1.5.2 The subsystem shall provide an option for users to edit their profile information (e.g., name, email).

5.3.1.5.3 The GUI subsystem shall provide a logout option from the Profile Page.

5.3.1.6 The GUI subsystem shall display the Virtual Closet Page with a scrollable view of the user's clothing items.

5.3.1.6.1 The GUI subsystem shall react to user gestures, including swiping and tapping, for interaction with clothing items.

5.3.1.6.2 The GUI subsystem shall include input fields for users to enter search queries for specific outfit requests.

5.3.1.6.3 The GUI subsystem shall provide error messages for invalid inputs in search fields.

5.3.1.6.4 The GUI subsystem shall display loading indicators during image uploads and AI processing.

5.3.1.6.5 The subsystem shall provide a button for users to upload images of their clothing items in a pop-up window.

5.3.1.6.6 Each clothing item in the virtual closet shall be clickable, launching a pop-up window with a view of the selected item.

5.3.1.6.6.1 The GUI subsystem shall allow users to view detailed information about each clothing item in the pop-up when clicked.

5.3.1.6.6.2 The GUI subsystem shall include a “Match” button on the pop-up that will take users to the Outfit Match Page where you will swipe suggested outfits using that article of clothing.

5.3.1.6.7 The subsystem shall provide a categorization feature, allowing users to navigate clothing by type (e.g., shirts, pants).

5.3.1.6.8 The subsystem shall include a tab that displays the user's previously saved outfits.

5.3.1.6.9 The GUI subsystem shall display help information or tooltips to guide users in using the app.

5.3.1.6.10

5.3.1.6.11 The GUI subsystem shall provide an option for users to pin their favorite clothing items.

5.3.1.6.12 The GUI subsystem shall allow users to edit or delete clothing items from their wardrobe.

5.3.1.6.13 The GUI subsystem shall provide user prompts to confirm actions such as deletion of items.

5.3.1.7 The GUI subsystem shall include an Outfit Match Page that displays clothing matches based on user preferences and uploaded items.

5.3.1.7.1 The GUI subsystem shall provide feedback animations when users swipe left (dislike) or right (like) on outfits.

5.3.1.7.2 The user shall be able to swipe through various outfit matches generated by the app.

5.3.1.7.3 The AI Outfit Matching subsystem shall display a pop-up of the clothes within the user's virtual closet that can be used to create a similar look for the outfits they swipe right on (like).

5.3.2 AI Outfit Matching Subsystem

The AI Outfit Matching subsystem is responsible for generating outfit combinations from the virtual closet onto the user, using their profile photo, and recommending outfits based on user preferences. This subsystem incorporates a swipe-based interface that allows users to easily express their likes and dislikes regarding the suggested outfits.

5.3.2.1 The AI Outfit Matching subsystem shall generate outfit suggestions based on user-uploaded clothing items, style scores of outfits based on swipe history, and if the user included an event prompt.

5.3.2.2 The AI Outfit Matching subsystem shall allow users to swipe through generated outfit suggestions.

5.3.2.3 The AI Outfit Matching subsystem shall record user preferences for outfits selected as “liked.”

5.3.2.4 The AI Outfit Matching subsystem shall save the user's favorite outfits for future reference.

5.3.2.5 The AI Outfit Matching subsystem shall update the outfit recommendations based on user interactions over time.

5.3.2.6 The AI Outfit Matching subsystem shall integrate weather and event data to provide relevant outfit suggestions.

5.3.2.7 The AI Outfit Matching subsystem shall display style scores for each outfit based on user feedback.

5.3.2.8 The AI Outfit Matching subsystem shall allow users to filter outfit suggestions by clothing type or occasion.

5.3.2.9 The AI Outfit Matching subsystem shall analyze user preferences to improve outfit recommendations.

5.3.3 AI Clothing Description Subsystem

The AI Clothing Description subsystem generates detailed descriptions for individual clothing items and outfits. This subsystem uses a locally hosted large language model to provide users with contextual information and recommendations.

5.3.3.1 The AI Clothing Description subsystem shall generate descriptions for each clothing item uploaded by the user.

5.3.3.2 The AI Clothing Description subsystem shall provide outfit descriptions based on selected clothing items.

5.3.3.3 The AI Clothing Description subsystem shall incorporate event and weather information into outfit descriptions.

5.3.3.4 The AI Clothing Description subsystem shall suggest styling tips based on the generated outfit descriptions.

5.3.3.5 The AI Clothing Description subsystem shall provide feedback to users on the suitability of outfits for specific events.

5.3.3.6 The AI Clothing Description subsystem shall sync generated descriptions with the Firebase backend for user access.

5.3.3.7 The AI Clothing Description subsystem shall support requests for outfit recommendations based on user preferences.

5.3.3.8 The AI Clothing Description subsystem shall utilize user feedback to refine and improve clothing descriptions over time.

5.3.3.9 The AI Clothing Description subsystem shall maintain a history of generated descriptions for user review.

5.3.3.10 The AI Clothing Description subsystem shall allow users to edit or update clothing descriptions manually.

5.3.3.11 The AI Clothing Description subsystem shall provide users with seasonal outfit suggestions.

5.3.4 Firebase Backend Subsystem

The Firebase Backend subsystem is responsible for managing user data, including profile information, wardrobe data, AI-generated outfit data, and local image files' location/name. This subsystem ensures secure storage and retrieval of all application data.

5.3.4.1 The Firebase Backend subsystem shall authenticate user profiles securely using Firebase Authentication.

5.3.4.2 The Firebase Backend subsystem shall store user profiles, including full-body image filename and wardrobe data.

5.3.4.3 The Firebase Backend subsystem shall securely store filenames and metadata for uploaded clothing images.

5.3.4.4 The Firebase Backend subsystem shall sync user data across devices to ensure data consistency.

5.3.4.5 The Firebase Backend subsystem shall allow users to update their profile information as needed.

5.3.4.6 The Firebase Backend subsystem shall manage the retrieval of saved outfits for user access.

5.3.4.7 The Firebase Backend subsystem shall support efficient querying of wardrobe items based on user preferences.

5.3.4.8 The Firebase Backend subsystem shall maintain logs of user interactions with the system for analytics.

5.3.4.9 The Firebase Backend subsystem shall ensure data privacy and compliance with relevant regulations.

5.3.4.10 The Firebase Backend subsystem shall provide backup and recovery options for user data.

5.3.4.11 The Firebase Backend subsystem shall provide data encryption for user information stored in the database.

5.4 Performance Requirements by CSC

5.4.1 Graphical User Interface (GUI)

5.4.1.1 The GUI shall be intuitively designed in order to support seamless user interaction.

5.4.1.2 The application should be responsive and function smoothly on various iOS devices (iPhones and iPads).

5.4.1.3 Image loading and outfit generation should occur within a reasonable timeframe given backend processing capabilities. (e.g., about 30 seconds per outfit).

5.4.1.4 The application should be memory efficient (100-150 MB RAM) and not drain the users battery excessively (5-10 percent per hour).

5.4.1.5 The application should be compatible with a wide range of iOS devices, including both iPhones and iPads, as well as older models (iPhone 8 or later) and different screen sizes.

5.4.2 AI Outfit Matching

5.4.2.1 Generated outfit images should be realistic and visually appealing.

5.4.2.2 The outfit generator should be able to handle a wide range of clothing item types (tops, shirts, pants, skirts, etc.) and styles.

5.4.2.3 The ML model should be adaptable to evolving fashion trends and styles.

5.4.3 AI Image Generation

5.4.3.1 Generated images should be consistent with the user's selected environment and outfit style.

5.4.3.2 The AI image generation tool should be able to produce high-quality images even on mobile devices with limited processing power.

5.4.4 Firebase Backend Subsystem

5.4.4.1 The database (Firebase) should be optimized for efficient data retrieval and storage. Under a reliable network connection, users should receive data to their device in 2-5 seconds.

5.4.4.2 The Firebase backend should have robust security measures in place to protect user data. Built-in Firebase authentication will be used as well as the implementation of granular security rules in order to control who can read/write data.

5.4.4.3 The Firebase backend database shall accommodate up to 10,000 monthly active users for authentication.

5.4.4.4 The Firebase backend database shall allow up to 50,000 reads, 20,000 writes, and 20,000 deletes per day.

5.4.4.5 The Firebase backend will accommodate up to 100 simultaneous connections.

5.4.4.6 Firebase cloud functionality shall accommodate up to 2 million invocations per month.

5.5 Project Environment Requirements

5.5.1 Development/Deployment

5.5.1.1 Hardware:

Category	Requirement
Processor	M2
GPU Memory	Optional
RAM	32GB
System Swap	Required

5.5.1.2 Software:

Category	Requirement
Operating System	macOS Monterey 12
IDE/Compiler	XCode 13.3
LLM (Text Generation)	ClaudeAI
LLM (Image Generation)	Stable Diffusion
Backend	Firebase SDK
Image Masking	Swift Core ML
VC/Repository	Github

5.5.2 Execution

5.5.2.1 Hardware:

Category	Requirement
Device Type	Mobile iOS
Processor	A11 Bionic
RAM	3GB

5.5.2.2 Software:

Category	Requirement
Operating System	iOS 12.4
Network	Active Internet
Offline Functionality	Limited to pre-loaded data.

6.0 Software Design Description

6.1. Introduction

This document presents the architecture and detailed design for the software for the induō project. The project provides an iOS application that enables users to upload, categorize, and visualize their wardrobe digitally, while generating AI-based outfit recommendations tailored to individual style preferences, events, and weather conditions. It incorporates virtual try-on capabilities for users to view AI-generated outfit combinations on themselves or others, supporting efficient wardrobe management, creative styling, and sustainable fashion habits.

6.1.1 System Objectives

The objective of the induō application is to streamline and enhance users' wardrobe management through a user-friendly and intuitive iOS interface. induō will allow users to efficiently organize and visualize their clothing options by uploading items with backgrounds removed, providing a digital space for users to manage their closets. Through AI-based outfit recommendations, users can make clothing choices tailored to their individual preferences, weather conditions, and upcoming events, enhancing both creativity and practicality in everyday attire. A swipe-style outfit matching feature will help users discover new styling ideas, while the integrated Virtual Try-On tool will offer a realistic preview of potential outfits. With this structure,

induō seeks to empower users to make informed, sustainable, and personalized fashion choices.

6.1.2 Hardware, Software, and Human Interfaces

This section identifies the hardware, software, and human interfaces that are integral to the induō application. Each interface is detailed to provide a clear understanding of how they interrelate to the base system.

6.1.2.1 Hardware Interfaces

- **6.1.2.1.1 Development Machine**
  The team will utilize a development machine equipped with an M2 processor and 32GB of RAM. This machine will serve as the primary development environment for coding, testing, and debugging the application.
- **6.1.2.1.2 Cloud-hosted GPU**
  The application will leverage a cloud-hosted NVIDIA A100 GPU with 40GB of VRAM and 10 CPUs, provided by fal.ai PaaS. This powerful hardware will facilitate advanced machine learning tasks, including image processing and AI computations. The system swap mechanism will allow for seamless transitions between different hardware configurations, ensuring optimal performance as development needs change.
- **6.1.2.1.3 Mobile Devices**
  The final application will be designed to run on mobile iOS devices that have an A11 Bionic processor or newer, with a minimum of 3GB of RAM. This ensures compatibility with a wide range of modern iPhones and iPads.

6.1.2.2 Software Interfaces

- **6.1.2.2.1 Operating System**
  The development will take place on macOS Monterey 12, providing a stable environment for application development.
- **6.1.2.2.2 Integrated Development Environment (IDE)**
  Xcode 13.3 will be used as the primary IDE and compiler for the Swift programming language. This tool will facilitate code development, debugging, and deployment.
- **6.1.2.2.3 AI Functionality**
  The application will utilize a multimodal LLM for image-to-text generation, managed through the Ollama application. This integration allows for effective processing of user-uploaded images.
- **6.1.2.2.4 Virtual Try-On (VTON) Model**
  Image generation will be handled by the VTON model running on the fal.ai platform. This software will enable users to visualize outfits on their uploaded images.

- **6.1.2.2.5 Image Processing Library**
  The Swift Core ML library will be employed for image masking tasks, ensuring efficient handling of image data for outfit recommendations.
- **6.1.2.2.6 Version Control**
  GitHub will be used for version control and code repository management, allowing for collaborative development and tracking of code changes.
- **6.1.2.2.7 Database Management**
  The Firebase SDK will be employed for managing the NoSQL document database, enabling efficient data storage and retrieval for user profiles and outfit information.
- **6.1.2.2.8 App Execution**
  The application will require iOS 12.4 or later on the user's device, with an active internet connection for full functionality. Limited offline capabilities will be available using pre-loaded data.

6.1.2.3 Human Interfaces

- **6.1.2.3.1 Graphical User Interface (GUI)**
  The application will feature a user-friendly GUI designed to facilitate easy navigation and interaction. Key screens will include the Welcome Page, Profile Page, Virtual Closet Page, and Outfit Match Page. Each page will be designed with an intuitive layout and controls, allowing users of all experience levels to engage with the app effectively.
- **6.1.2.3.2 User Input Methods**
  Users will interact with the application using touch gestures (tapping, swiping) on mobile devices. Additionally, image uploads will be facilitated through standard file selection dialogs, allowing users to easily add clothing images to their profiles.

6.2 Architectural Design

The architectural design of the induō application focuses on creating a robust and flexible system that aligns with the functional requirements identified in the earlier sections of the Software Development Plan (SDP). The architecture will support modularity, maintainability, and scalability, ensuring that the application can evolve over time with changing user needs and technological advancements.

The system will be designed with a clear separation of concerns, allowing distinct modules to handle specific functionalities, thus enhancing cohesion and minimizing coupling. This design approach aims to facilitate a smooth development process and ensure that individual components can be implemented and tested independently.

The induō application is organized into several logical class definitions based on their functionalities, including but not limited to user interface management, profile handling, outfit recommendations, data management, and AI processing.

6.2.1 Major Software Components

The induō application comprises several major software components, each addressing specific functional requirements:

- **6.2.1.1 User Interface Module:** This component is responsible for managing all user interactions with the application. It includes the Welcome Page, Profile Page, Virtual Closet Page, and Outfit Match Page. Each page is designed to provide a seamless user experience, allowing users to easily navigate through their wardrobe and receive outfit recommendations.
- **6.2.1.2 Profile Management Module:** This subsystem handles user profile creation and management. It enables users to upload their clothing items, including images with background removal, and manage their preferences for outfit recommendations. This module interfaces with the database to store user data securely.
- **6.2.1.3 Recommendation Engine Module:** This component utilizes AI algorithms to generate personalized outfit recommendations based on user preferences, clothing availability, and contextual inputs like weather. The engine integrates with the virtual try-on feature to provide users with a visual representation of outfit combinations.
- **6.2.1.4 Database Management Module:** This subsystem is responsible for managing data storage and retrieval through the Firestore distributed database. It ensures that user data, clothing items, and generated outfit recommendations are stored efficiently and can be accessed quickly by other modules.
- **6.2.1.5 AI Processing Module:** This module encompasses all AI-related functionalities, including image processing for clothing uploads and outfit visualization. It leverages the cloud-hosted A100 GPU for high-performance computing and integrates with the Ollama application for multimodal LLM tasks.

6.2.2 Major Software Interactions

The interactions between the major software components of the induō application are designed to ensure smooth data flow and functional coherence:

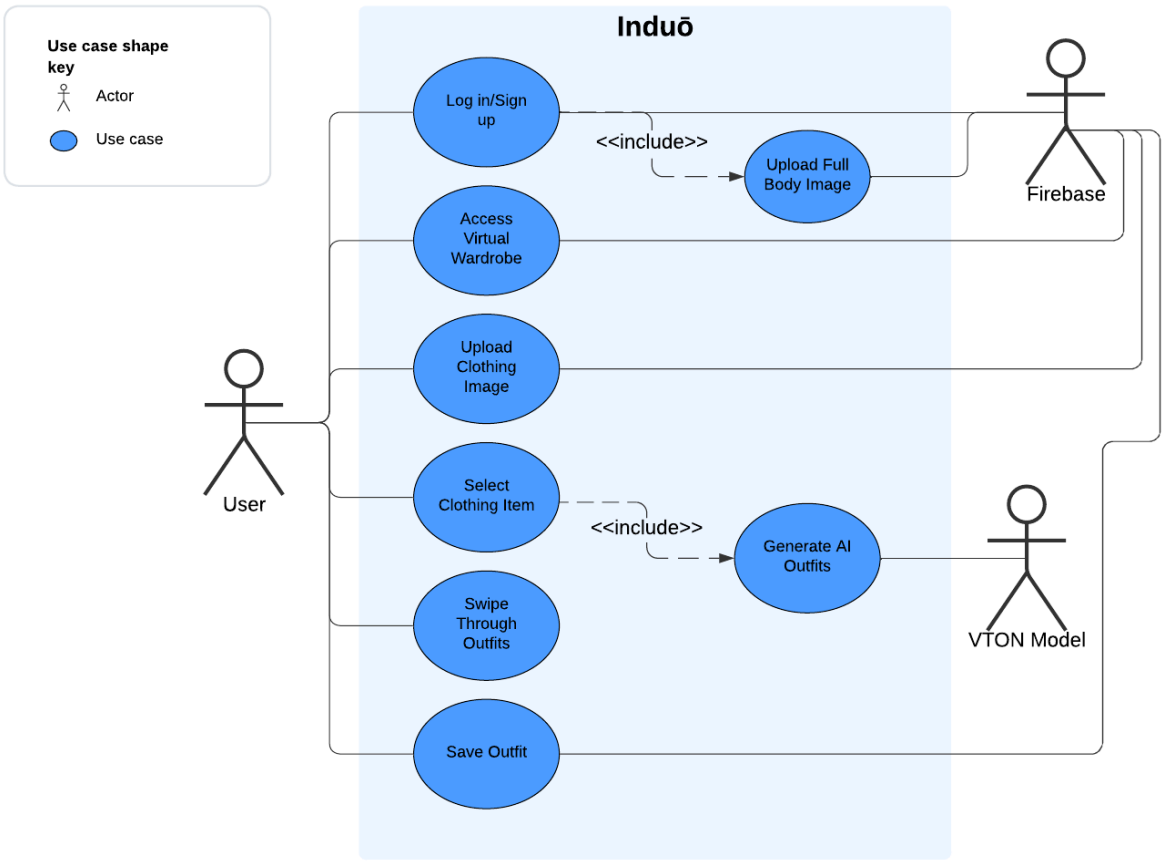
- **6.2.2.1 User Interface and Profile Management:** The User Interface Module communicates with the Profile Management Module to send and retrieve user profile data, allowing for dynamic updates as users modify their profiles or upload clothing items.
- **6.2.2.2 Profile Management and Database Management:** The Profile Management Module interacts with the Database Management Module to store user data and clothing information. This interaction is performed using the Firebase SDK to facilitate secure data transactions.

- **6.2.2.3 Recommendation Engine and AI Processing Module:** The Recommendation Engine Module sends requests to the AI Processing Module for outfit generation and image processing tasks. The AI Processing Module processes these requests, utilizing the cloud GPU for computations, and returns the results back to the Recommendation Engine.
- **6.2.2.4 User Interface and Recommendation Engine:** The User Interface Module displays outfit recommendations generated by the Recommendation Engine. It handles user inputs and preferences, relaying them to the Recommendation Engine to generate relevant suggestions.
- **6.2.2.5 AI Processing and Virtual Try-On Integration:** The AI Processing Module interfaces with the virtual try-on model running on the fal.ai platform. This interaction allows users to visualize outfits before making decisions, enhancing their experience with the app.

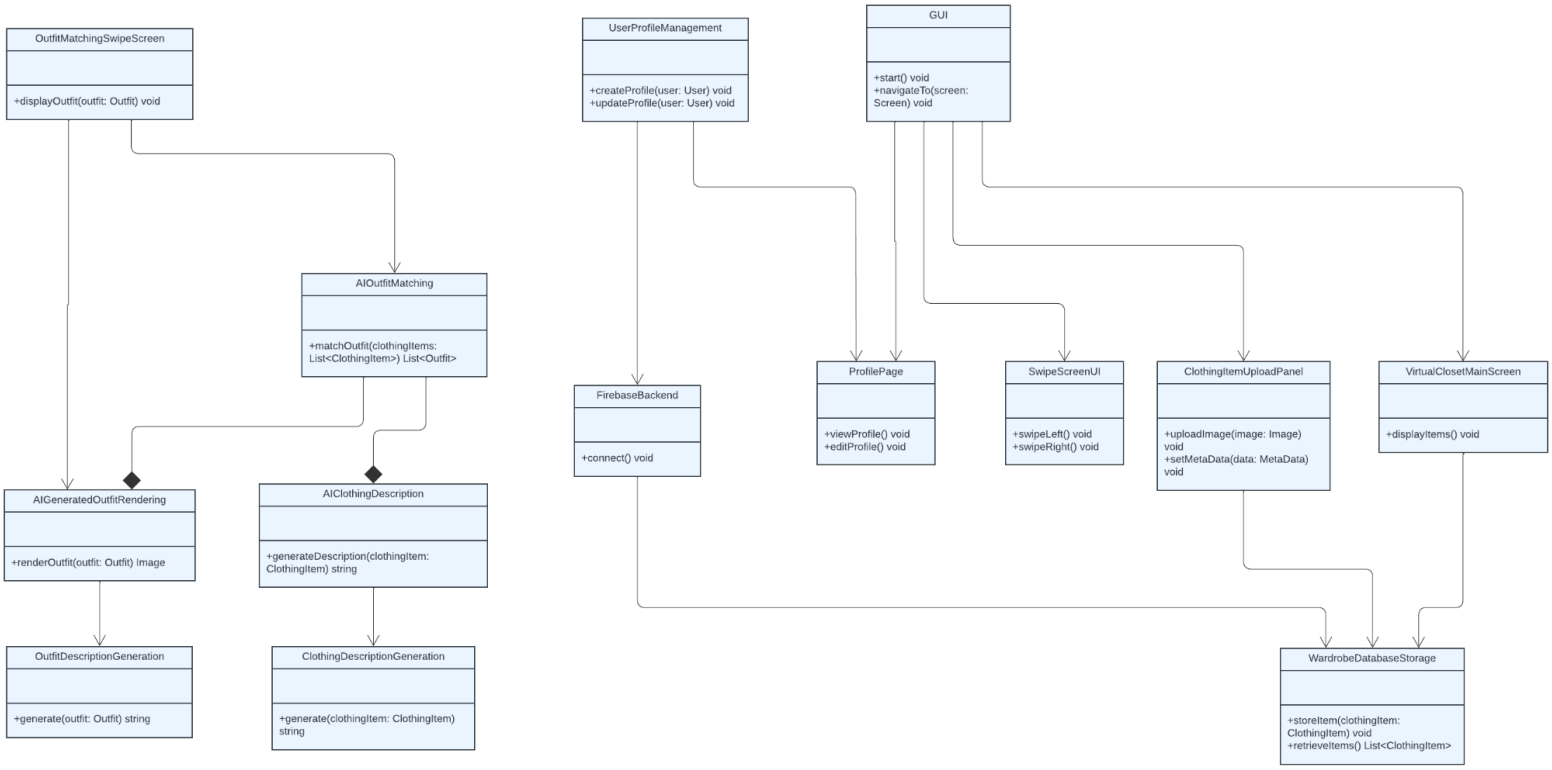
6.2.3 Architectural Design Diagrams

To illustrate the architectural design of the induō application, several UML-style diagrams will be included. These diagrams will visually represent the system's structure and interactions, enhancing the understanding of the application’s architecture. The following diagrams will be provided:

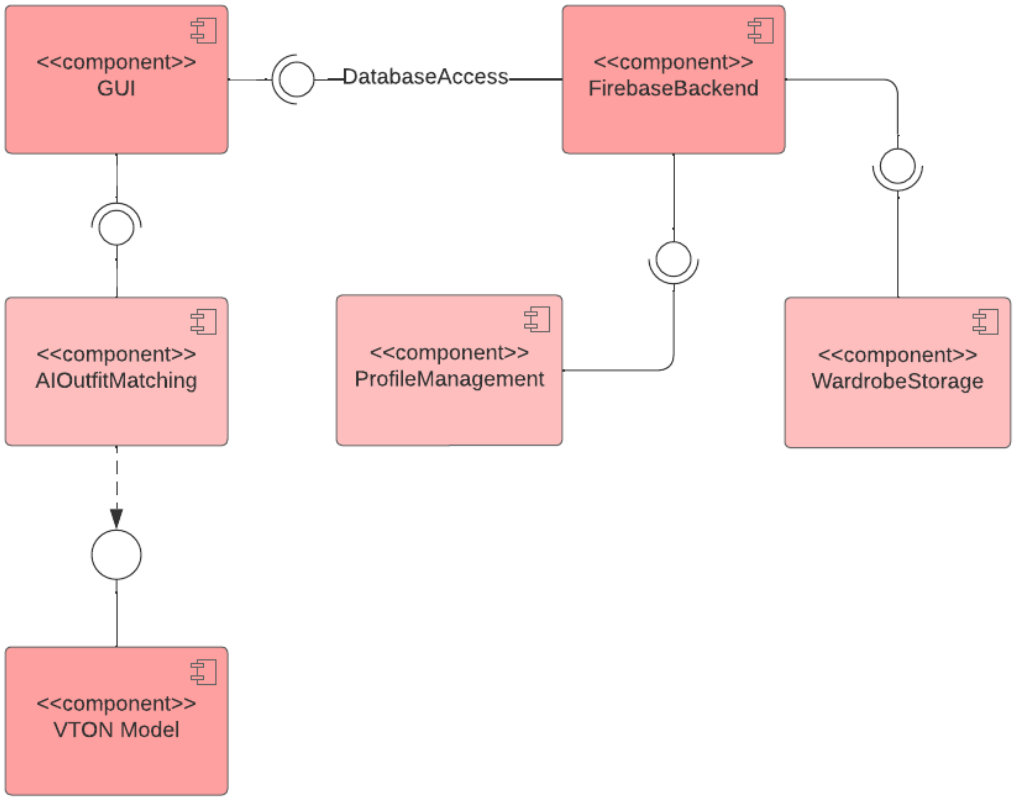
- **6.2.3.1 Use Case Diagram:** This diagram will showcase the primary interactions between users and the system, detailing various user scenarios such as creating a profile, uploading clothing items, and receiving outfit recommendations.



- **6.2.3.2 Top-Level Class Diagram:** This diagram will depict the major classes within the application and their relationships, highlighting how different components interact with one another.



- **6.2.3.3 Component Diagram:** This diagram will illustrate the communication between the software components, showcasing how data flows between modules and external systems like the cloud GPU and database.



6.3. CSC and CSU Descriptions

This section begins the Detailed Design phase of the document, outlining the Computer Software Components (CSCs) and Computer Software Units (CSUs) that comprise the induō application. The induō application itself is a Computer Software Configuration Item (CSCI), composed of multiple CSCs, each encompassing various CSUs.

6.3.1 Detailed Class Descriptions

The following sections provide the details of all classes used in the induō application. Each class serves a specific function within the overall architecture and is categorized under relevant Computer Software Units (CSUs).

6.3.1.1 Virtual Closet Main Screen CSU

- **6.3.1.1.1 ClothingItem Class**
  - **Description:** Represents individual clothing items uploaded by users.
  - **Fields:**
    - `itemId`: Unique identifier for the clothing item.
    - `image`: Image URL of the clothing item.
    - `category`: Category of the clothing (e.g., tops, bottoms).
  - **Methods:**
    - `getImage()`: Returns the string URL of the clothing item image.
    - `getCategory()`: Returns a string representing the clothing category.
- **6.3.1.1.2 WardrobeDisplay Class**
  - **Description:** Manages the display of all clothing items in a scrollable format.
  - **Fields:**
    - `clothingItems`: List of ClothingItem objects to be displayed.
  - **Methods:**
    - `renderItems()`: Renders the clothing items on the screen.
- **6.3.1.1.3 SearchBar Class**
  - **Description:** Handles user input for searching clothing items based on prompts.
  - **Fields:**
    - `searchTerm`: The term entered by the user for searching.
  - **Methods:**
    - `performSearch()`: Executes a search based on the entered term.

6.3.1.2 Clothing Item Upload Panel CSU

- **6.3.1.2.1 ImageUpload Class**
  - **Description:** Manages the image selection and uploading process.
  - **Fields:**
    - `selectedImage`: The image selected by the user for upload.
  - **Methods:**
    - `uploadImage()`: Uploads the selected image to the server.
- **6.3.1.2.2 BackgroundRemoval Class**
  - **Description:** Handles the background removal from uploaded images.
  - **Fields:**
    - `originalImage`: The image before background removal.
    - `processedImage`: The image after background removal.

    - **Methods:**
        - `removeBackground()`: Processes the image to remove its background.
- **6.3.1.2.3 FirebaseStorage Class**
    - **Description:** Manages the storage of image filenames and metadata in Firebase.
    - **Fields:**
        - `storageReference`: Reference to Firebase storage.
    - **Methods:**
        - `storeImageMetadata()`: Stores the image metadata in Firebase.

6.3.1.3 Profile Page CSU

- **6.3.1.3.1 ProfileInput Class**
    - **Description:** Manages the input of the full-body image for user profiles.
    - **Fields:**
        - `profileImage`: The user's full-body image.
    - **Methods:**
        - `setProfileImage()`: Sets the user's profile image.
- **6.3.1.3.2 ProfileEdit Class**
    - **Description:** Handles user profile updates for personal information.
    - **Fields:**
        - `userData`: Object containing user data to be updated.
    - **Methods:**
        - `updateUserInfo()`: Updates the user's personal information in the profile.
- **6.3.1.3.3 ProfileDisplay Class**
    - **Description:** Displays the user's profile information and full-body image.
    - **Fields:**
        - `displayedProfile`: The user profile data to be displayed.
    - **Methods:**
        - `renderProfile()`: Renders the user's profile information on the screen.

6.3.1.4 Swipe Screen UI CSU

- **6.3.1.4.1 SwipeGestureHandler Class**
    - **Description:** Manages swipe gestures for liking or disliking outfits.
    - **Fields:**
        - `gestureDirection`: Direction of the swipe (left or right).

    - **Methods:**
        - `handleSwipe()`: Processes swipe gestures and determines user action.
- **6.3.1.4.2 SwipeFeedback Class**
    - **Description:** Provides visual feedback for user interactions with swipes.
    - **Fields:**
        - `swipeAnimation`: Animation object for swipe feedback.
    - **Methods:**
        - `showFeedback()`: Displays visual feedback for liked or disliked outfits.
- **6.3.1.4.3 OutfitFeedback Class**
    - **Description:** Manages feedback data for AI-generated outfits.
    - **Fields:**
        - `outfitId`: Identifier for the outfit being reviewed.
        - `feedbackScore`: Score representing the user's feedback.
    - **Methods:**
        - `submitFeedback()`: Submits feedback score for the outfit.

6.3.1.5 AI-Generated Outfit Rendering CSU

- **6.3.1.5.1 APIRequest Class**
    - **Description:** Handles API requests for outfit generation.
    - **Fields:**
        - `apiEndpoint`: URL endpoint for the API.
    - **Methods:**
        - `sendRequest()`: Sends a request to the API with user data.
- **6.3.1.5.2 ImageProcessing Class**
    - **Description:** Processes generated outfit images for display.
    - **Fields:**
        - `generatedImage`: The image returned by the API.
    - **Methods:**
        - `processImage()`: Processes the image for display in the app.
- **6.3.1.5.3 OutfitRenderer Class**
    - **Description:** Renders AI-generated outfits for display in the app.
    - **Fields:**
        - `outfitId`: Identifier for the outfit being rendered.
    - **Methods:**
        - `renderOutfit()`: Renders the outfit for user interaction.

6.3.1.6 Outfit Matching Swipe Screen CSU

- **6.3.1.6.1 OutfitDisplay Class**
    - **Description:** Collects and displays pre-rendered AI outfits for swiping.

    - **Fields:**
        - `outfits`: List of Outfit objects to display.
    - **Methods:**
        - `displayOutfits()`: Displays the list of AI-generated outfits for user interaction.
- **6.3.1.6.2 StyleScore Class**
    - **Description:** Updates style scores in the database table based on user interactions.
    - **Fields:**
        - `styleScore`: The score to be updated in the database.
    - **Methods:**
        - `updateScore()`: Updates the style score based on user feedback.

6.3.1.7 Clothing Description Generation CSU

- **6.3.1.7.1 LLMRequest Class**
    - **Description:** Sends images to the LLM for clothing descriptions.
    - **Fields:**
        - `imageData`: The image data being sent to the LLM.
    - **Methods:**
        - `requestDescription()`: Requests a description from the LLM.
- **6.3.1.7.2 FirebaseSync Class**
    - **Description:** Syncs clothing descriptions with Firebase.
    - **Fields:**
        - `descriptionData`: The description data to be synced.
    - **Methods:**
        - `syncWithFirebase()`: Syncs the generated descriptions with Firebase.

6.3.1.8 Outfit Description Generation CSU

- **6.3.1.8.1 LLMOutfitRequest Class**
    - **Description:** Requests descriptions and event/environment recommendations for full AI-generated outfits.
    - **Fields:**
        - `outfitImages`: Images of the outfits to be described.
    - **Methods:**
        - `requestOutfitDescription()`: Requests descriptions for outfits from the LLM.
- **6.3.1.8.2 EventRecommendation Class**
    - **Description:** Sends user search bar prompts to the LLM with descriptions of all outfits to determine the best outfit suggestions.

- **Fields:**
  - `userPrompt`: The prompt entered by the user for recommendations.
- **Methods:**
  - `generateRecommendations()`: Generates outfit recommendations based on the user's input.

6.3.1.9 User Profile Management CSU

- **6.3.1.9.1 FirebaseAuth Class**
  - **Description:** Handles user authentication for the app.
  - **Fields:**
    - `userId`: Unique identifier for the authenticated user.
  - **Methods:**
    - `loginUser()`: Logs in the user with credentials.
    - `logoutUser()`: Logs out the authenticated user.
- **6.3.1.9.2 ProfileStorage Class**
  - **Description:** Stores user profiles, including clothing data.
  - **Fields:**
    - `profileData`: Data structure holding user profile information.
    - `name`: The user's full name.
    - `preferences`: Array of user's clothing preferences
    - `recommendedOutfits`: List of outfits recommended for the user.
  - **Methods:**
    - `storeProfile()`: Stores the user's profile data in Firebase.
    - `updatePreferences(newPreferences)`**:** Updates the user's clothing preferences.

6.3.1.10 Wardrobe Database Storage CSU

- **6.3.1.10.1 ClothingImage Class**
  - **Description:** Stores a table of uploaded clothing item image filenames and descriptions.
  - **Fields:**
    - `imageFilename`: Filename of the clothing item image.
    - `description`: Textual description of the clothing item (e.g., color, style).
    - `category`: The category to which the clothing item belongs (e.g., tops, bottoms).
    - `uploadedDate`: Timestamp indicating when the item was uploaded.
    - `userId`: Unique identifier for the user who uploaded the item.

- **Methods:**
  - `getImageFilename()`: Returns the filename of the clothing item image.
  - `getDescription()`: Returns the description of the clothing item.
  - `getCategory()`: Returns the category of the clothing item.
  - `getUploadedDate()`: Returns the upload date of the clothing item.
  - `getUserId()`: Returns the unique identifier for the user associated with the clothing item.

- **6.3.1.10.2 OutfitImage Class**
  - **Description:** Represents an AI-generated outfit and its associated metadata in the wardrobe database.
  - **Fields:**
    - `outfitId`: Unique identifier for the outfit.
    - `imageFilenames`: List of filenames of the outfit's component clothing images.
    - `styleScore`: Score representing the style quality of the outfit.
    - `generatedDate`: Timestamp indicating when the outfit was generated.
  - **Methods:**
    - `getOutfitId()`: Returns the unique identifier for the outfit.
    - `getImageFilenames()`: Returns the list of clothing image file names associated with the outfit.
    - `getStyleScore()`: Returns the style score of the outfit.
    - `getGeneratedDate()`: Returns the date when the outfit was generated.

6.3.2 Detailed Interface Descriptions

The following sections provide details of all interfaces used in the induō application. These interfaces facilitate communication and data exchange between various components of the system, ensuring seamless integration and functionality.

6.3.2.1 User Interface to Profile Management Interface

- **Purpose:** The UI Module provides an accessible, visual way for users to input and update their profile and clothing items, which are then stored and managed in the Profile Management subsystem.

- **Data Flow:**
  - *Input*: When a user updates their profile or uploads a new clothing item, the UI sends this data to Profile Management for processing and storage.
  - *Output*: Profile Management sends feedback messages back to the UI

6.3.2.2 Profile Management to Database Management Interface

- **Purpose:** This interface allows Profile Management to store and retrieve user data in the Firebase database.
- **Data Flow:**
  - *Input*: Profile Management sends data structured by the user, such as user profiles and clothing items, to Data Management for storage.
  - *Output*: Database Management returns data requested by Profile Management, including user profiles or specific clothing items.

6.3.2.3 Recommendation Engine to AI Processing Interface

- **Purpose:** The Recommendation Engine interfaces with the AI Processing module to generate tailored outfit recommendations and process visual data for virtual try-on.
- **Data Flow:**
  - *Input*: Recommendation Engine sends requests containing user preferences and contextual information to the AI Processing module.
  - *Output*: The AI Processing module returns generated outfit images and metadata to the Recommendation Engine for further use.

6.3.2.4 User Interface to Recommendation Engine Interface

- **Purpose:** This interface manages communication between the UI and the Recommendation Engine, allowing users to request and receive outfit recommendations.
- **Data Flow:**
  - *Input*: User preferences and settings are sent from the UI to the Recommendation Engine.
  - *Output*: The Recommendation Engine provides recommended outfits back to the UI.

6.3.2.5 AI Processing to Virtual Try-On Interface

- **Purpose:** This Interface enables interaction between the AI Processing module and the Virtual Try-On (VTON) system, allowing users to visualize outfits on themselves
- **Data Flow:**
  - *Input*: The AI Processing module sends processed images to the VTOn model.
  - *Output*: VTON returns the augmented images, displaying outfits on a user's full-body photo.

6.3.2.6 Database Management to AI Processing Interface

- **Purpose:** The Database Management system interfaces with AI Processing to retrieve stored images and data for generating outfit recommendations.
- **Data Flow:**
  - *Input*: AI Processing requested stored clothing images with metadata.
  - *Output*: Database Management retrieves and sends the requested data back to AI Processing.

6.3.3 Detailed Data Structure Descriptions

The following sections detail the core data structures implemented within the induō application. Each structure is documented with its internal composition, operations and specific use cases within the system. The descriptions are organized under each relevant CSU to map the relationship between system functionality and its supporting data structures. This documentation serves as both a technical reference and an architectural overview of how data is managed and flows through the induō system.

6.3.3.1 Virtual Closet Main Screen CSU

- **6.3.3.1.1 Circular Buffer for WardrobeDisplay**
  - **Purpose:** Manages the currently visible clothing items during scrolling.
  - **Structure:** Fixed-size circular buffer storing image references.
  - **Operations:**
    - `push()`: Adds new items as user scrolls.
    - `pop()`: Removes items that scroll out of view.
    - `shift()`: Moves window of visible items.
  - **Usage:** Used in WardrobeDisplay module for efficient image loading/unloading.

6.3.3.2 Clothing Item Upload Panel CSU

- **6.3.3.2.1 Upload Queue**
  - **Purpose:** Manages pending image uploads.
  - **Structure:** FIFO queue with retry tracking.
  - **Operations:**
    - `enqueue(imageData)`: Adds new upload.
    - `dequeue()`: Removes completed upload.
    - `retry(failedUpload)`: Moves failed upload to front.
    - `getStatus()`: Returns queue statistics.
  - **Usage:** ImageUpload module uses this to handle multiple concurrent uploads.
- **6.3.3.2.2 LRU Cache for Background Removal**
  - **Purpose:** Caches recently processed images.
  - **Structure**: Doubly-linked list + hash map.
  - **Operations:**
    - `get(key)`: O(1) retrieval.
    - `put(key, value)`: O(1) insertion.
    - `evict()`: Removes least recently used.
  - **Usage**: BackgroundRemoval module uses this to avoid reprocessing.

6.3.3.3 Profile Page CSU

- **6.3.3.3.1 Image Processing Queue**
  - **Purpose:** Manages full-body image upload processing tasks.
  - **Structure:** Priority Queue implementation using a Binary Heap.
  - **Operations:**
    - `enqueue(imageData,priority)`: O(log n) - Adds new image processing task.
    - `dequeue()`: O(log n) - Retrieves highest priority task.
    - `peek()`: O(1) - Views next task without removing.
    - `reprioritize(taskId, newPriority)`: O(log n) - Updates task priority.
  - **Usage:** Image upload preprocessing and background removal task scheduling.
- **6.3.3.3.2 Image Buffer Cache**
  - **Purpose**: Caches processed image data to avoid reprocessing.
  - **Structure**: LRU (Least Recently Used) Cache using Doubly-Linked List + HashMap.
  - **Operations:**
    - `get(key)`: O(1) - Retrieves cached image.
    - `put(key, value)`: O(1) - Stores processed image.

        - `evict()`: O(1) - Removes least recently used item.
        - `clear()`: O(1) - Empties cache.
    - **Usage:** Temporary storage of processed images, quick retrieval of recently uploaded images and memory management for image processing.

6.3.3.4 Swipe Screen UI CSU

- **6.3.3.4.1 Ring Buffer for Animation States**
    - **Purpose**: Manages swipe animation states.
    - **Structure**: Circular buffer of fixed size.
    - **Operations:**
        - `write(state)`: O(log n) - Adds new animation state.
        - `read()`: Gets current state.
        - `advance()`: Moves to the next state.
    - **Usage:** SwipeFeedback module uses this for smooth animations.

6.3.3.5 AI-Generated Outfit Rendering CSU

- **6.3.3.5.1 Task Queue for API Requests**
    - **Purpose**: Manages pending AI generation requests.
    - **Structure**: Priority queue with timeout tracking.
    - **Operations:**
        - `schedule(request, priority)`: Adds request.
        - `process()`: Handles next request.
        - `timeout()`: Handles failed requests.
    - **Usage:** APIRequest module uses this to manage API rate limits.
- **6.3.3.5.2 Image Buffer Pool**
    - **Purpose**: Reusable memory for image processing.
    - **Structure**: Pool of pre-allocated buffers.
    - **Operations:**
        - `acquire()`: Get available buffer.
        - `release(buffer)`: Return buffer to pool.
        - `resize(size)`: Adjust buffer size.
    - **Usage:** ImageProcessing module uses this for efficient memory management.

6.3.3.6 Outfit Matching Swipe Screen CSU

- **6.3.3.6.1 Outfit Preview Queue**
  - **Purpose**: Manages outfit cards available for swiping.
  - **Structure**: Double-Ended Queue (Deque) with Pre-fetching.
  - **Operations:**
    - `pushFront(outfit)`: Adds new outfit to front.
    - `pushBack(outfit)`: Adds new outfit to back.
    - `popFront()`: Removes front outfit after swipe.
    - `peek(direction)`: Views next/previous outfit.
  - **Usage:** Managing visible and upcoming outfits, ensuring smooth transitions between swipes and pre-loading next outfits.

6.3.3.7 Clothing Description Generation CSU

- **6.3.3.7.1 Trie for Description Keywords**
  - **Purpose**: Fast lookup of common clothing descriptors.
  - **Structure**: Prefix tree of clothing terms.
  - **Operations:**
    - `insert(word)`: Adds new descriptor.
    - `search(prefix)`: Finds matching terms.
    - `delete(word)`: Removes descriptor.
  - **Usage:** LLMRequest module uses this for prompt generation.
- **6.3.3.7.2 Message Queue for LLM Requests**
  - **Purpose**: Batches description requests.
  - **Structure**: FIFO queue with batching.
  - **Operations:**
    - `enqueue(request)`: Adds request.
    - `batchDequeue(size)`: Gets a batch of requests.
    - `flush()`: Processes remaining requests.
  - **Usage**: FirebaseSync uses this for efficient API usage.

6.3.3.8 Outfit Description Generation CSU

- **6.3.3.8.1 Request Batch Queue**
  - **Purpose**: Optimizes LLM API calls by batching similar requests.
  - **Structure**: Priority Queue with Batch Processing.
  - **Operations:**
    - `enqueueBatch(requests)`: Adds new request batch.
    - `dequeueBatch()`: Retrieves highest priority batch.

        - `mergeBatches(batch1, batch2)`: Combines compatible batches.
        - `splitBatch(batch)`: Splits oversized batch.
    - **Usage:** Managing API rate limits and optimizing token usage.

6.3.3.9 User Profile Management CSU

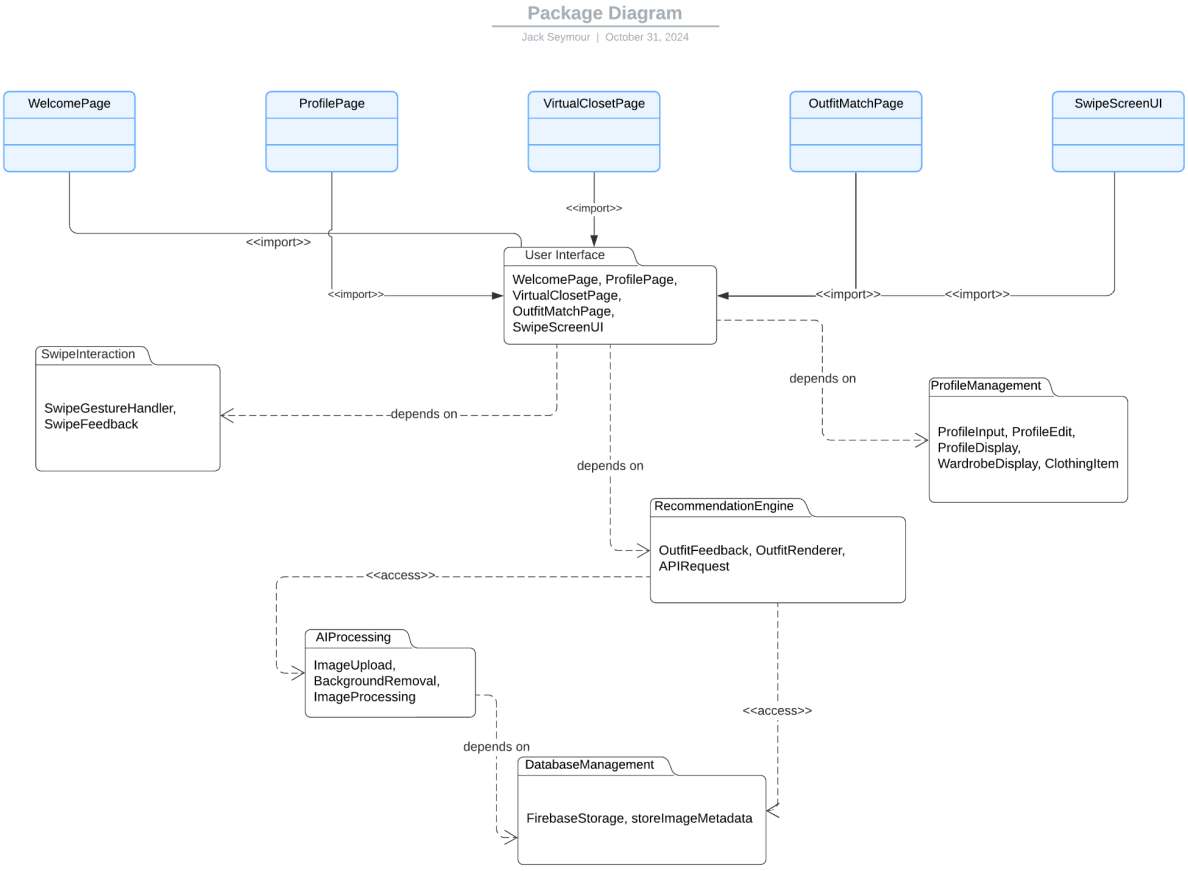
- **6.3.3.9.1 Auth State Queue**
    - **Purpose**: Manages authentication state transitions and callbacks.
    - **Structure**: State Machine Queue.
    - **Operations:**
        - `enqueue(state)`: Adds new auth state.
        - `dequeue()`: Processes next state.
        - `peek()`: Views current state.
        - `rollback()`: Reverts to previous state.
    - **Usage:** Managing auth state transitions, handling auth callbacks and error recovery.
- **6.3.3.9.2 Token Cache**
    - **Purpose**: Manages authentication tokens and refresh timing.
    - **Structure**: LRU Cache with Time-Based Expiration.
    - **Operations:**
        - `set(token, expiry)`: Stores new token.
        - `get(tokenId)`: Retrieves token.
        - `refresh(tokenId)`: Updates token expiry.
        - `evict()`: Removes expired tokens.
    - **Usage:** Token management, auto-refresh timing and session management
- **6.3.3.9.3 Profile Cache**
    - **Purpose**: Optimizes frequently accessed profile data.
    - **Structure**: Multi-Level Cache with Write-Through Policy.
    - **Operations:**
        - `read(key)`: Retrieves cached data.
        - `write(key, value)`: Updates cache and storage.
        - `invalidate(key)`: Removes from cache.
        - `promoteLevel(key)`: Moves to higher cache level.
    - **Usage:** Frequent data access optimization, write synchronization and cache coherency maintenance.

6.3.3.10 Wardrobe Database Storage CSU

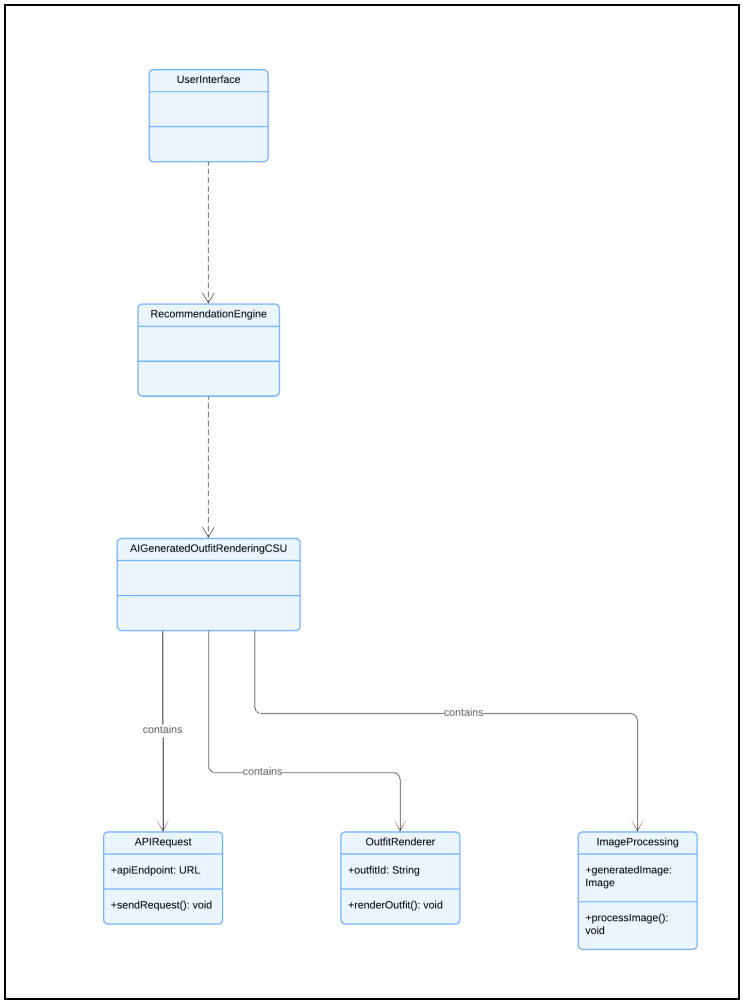
- **6.3.3.10.1 B-Tree Index for Image Lookups**
  - **Purpose**: Fast filename/metadata queries.
  - **Structure**: B-tree storing filename keys.
  - **Operations:**
    - `find(key)`: O(log n) lookup.
    - `insert(key, value)`: O(log n) insertion.
    - `delete(key)`: O(log n) deletion.
  - **Usage**: ClothingImage module uses this for fast file lookups.
- **6.3.3.10.2 Bloom Filter for Duplicate Detection**
  - **Purpose**: Quick check for duplicate images.
  - **Structure**: Probabilistic bit array.
  - **Operations:**
    - `add(imageHash)`: Adds image hash.
    - `mightContain(imageHash)`: Checks for duplicate.
    - `clear()`: Resets filter.
  - **Usage**: OutfitImage module uses this to prevent duplicate storage.

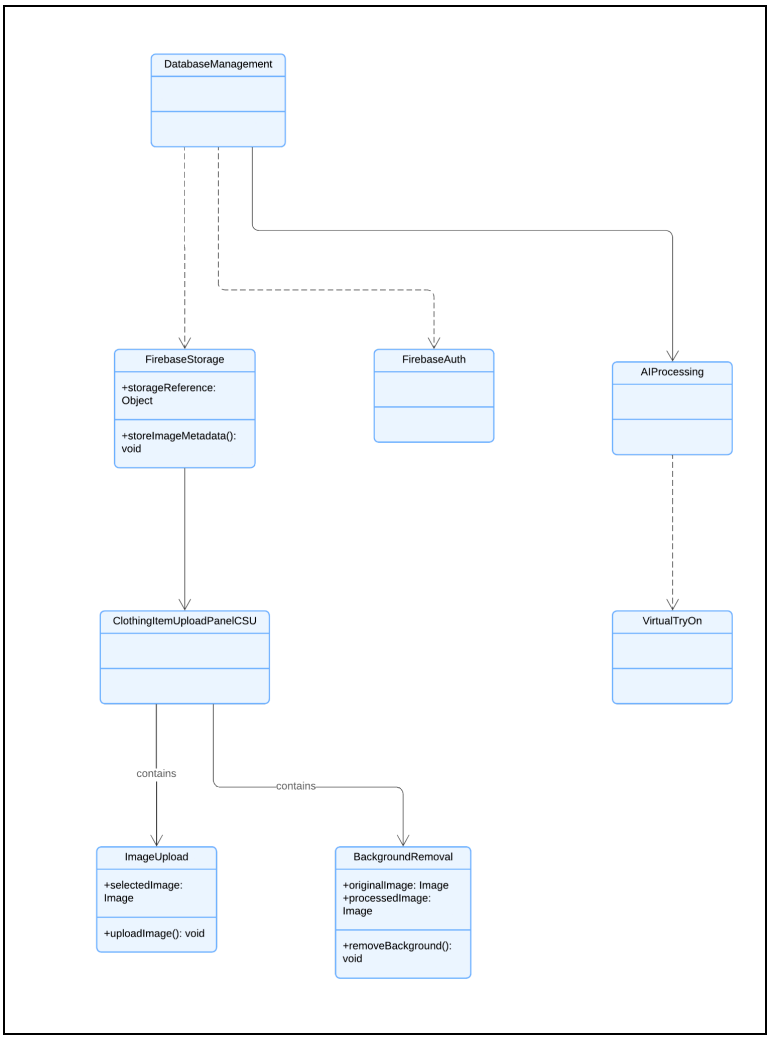
6.3.4 Detailed Design Diagrams

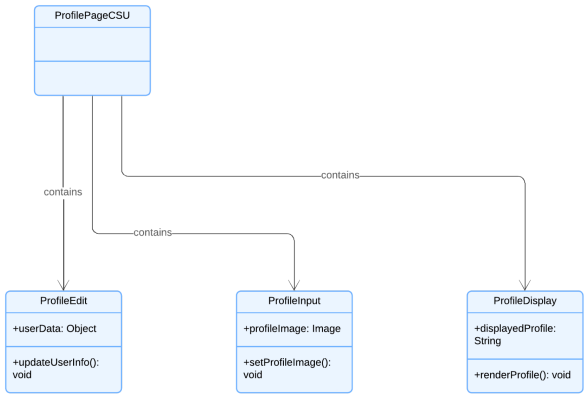
6.3.4.1 Package Diagram

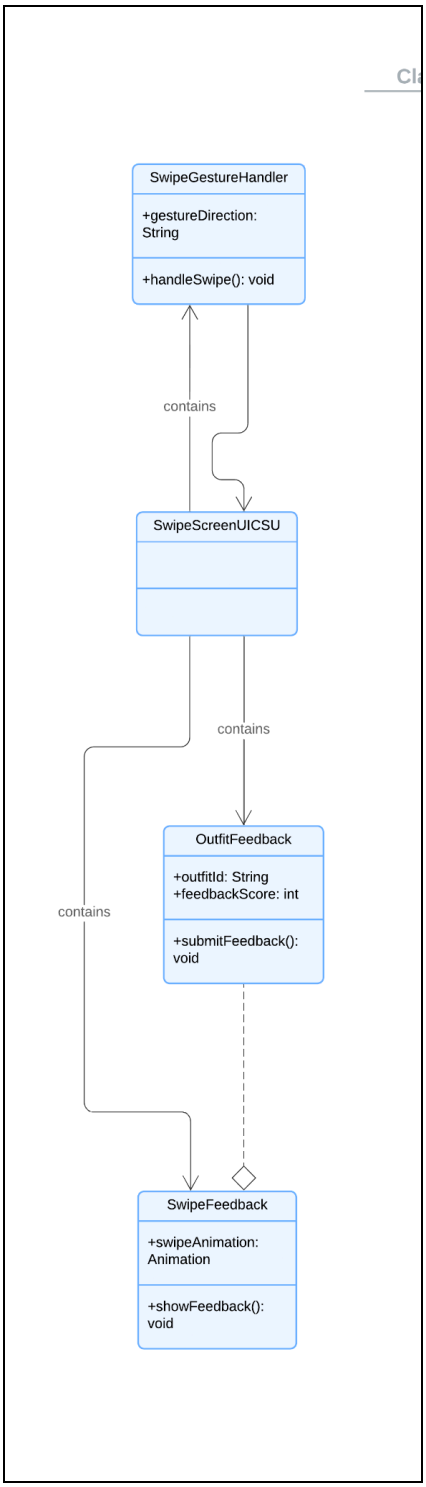


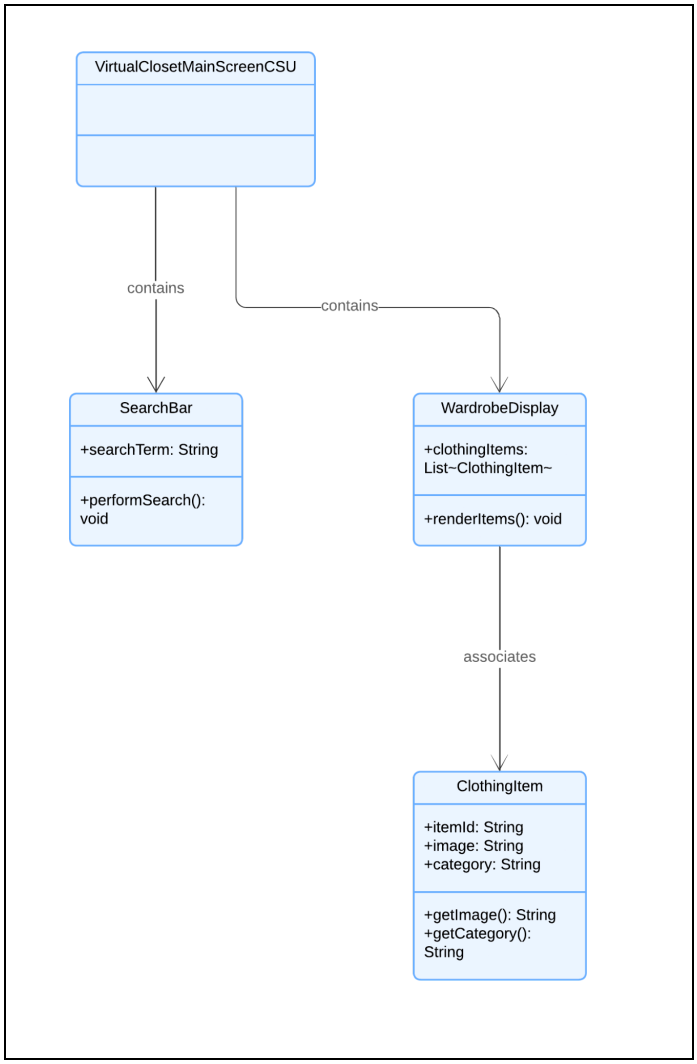
6.3.4.2 Class Diagrams



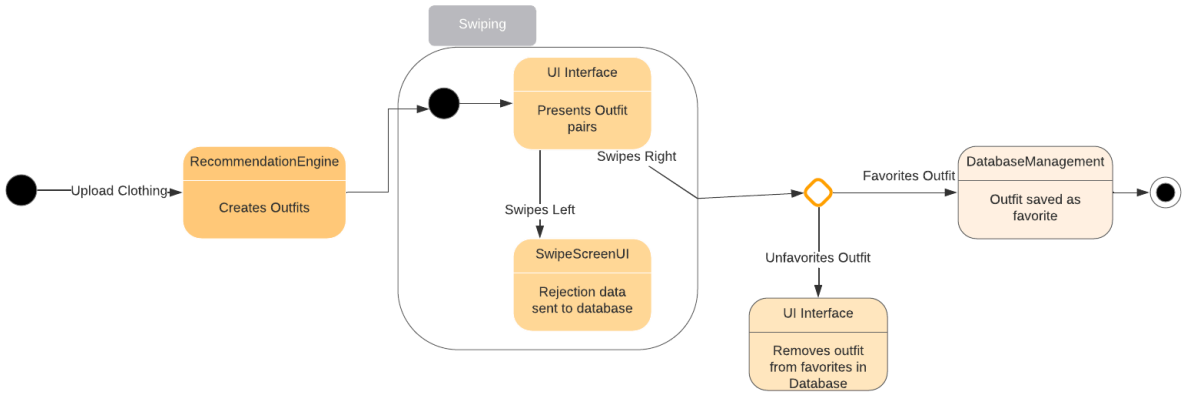



ProfilePageCSU
contains
contains
contains
ProfileEdit
+userData: Object
+updateUserInfo(): void
ProfileInput
+profileImage: Image
+setProfileImage(): void
ProfileDisplay
+displayedProfile: String
+renderProfile(): void


SwipeGestureHandler
+gestureDirection: String
+handleSwipe(): void
contains
SwipeScreenUICSU
contains
contains
OutfitFeedback
+outfitId: String
+feedbackScore: int
+submitFeedback(): void
SwipeFeedback
+swipeAnimation: Animation
+showFeedback(): void


VirtualClosetMainScreenCSU
contains
contains
SearchBar
+searchTerm: String
+performSearch(): void
WardrobeDisplay
+clothingItems: List~ClothingItem~
+renderItems(): void
associates
ClothingItem
+itemId: String
+image: String
+category: String
+getImage(): String
+getCategory(): String

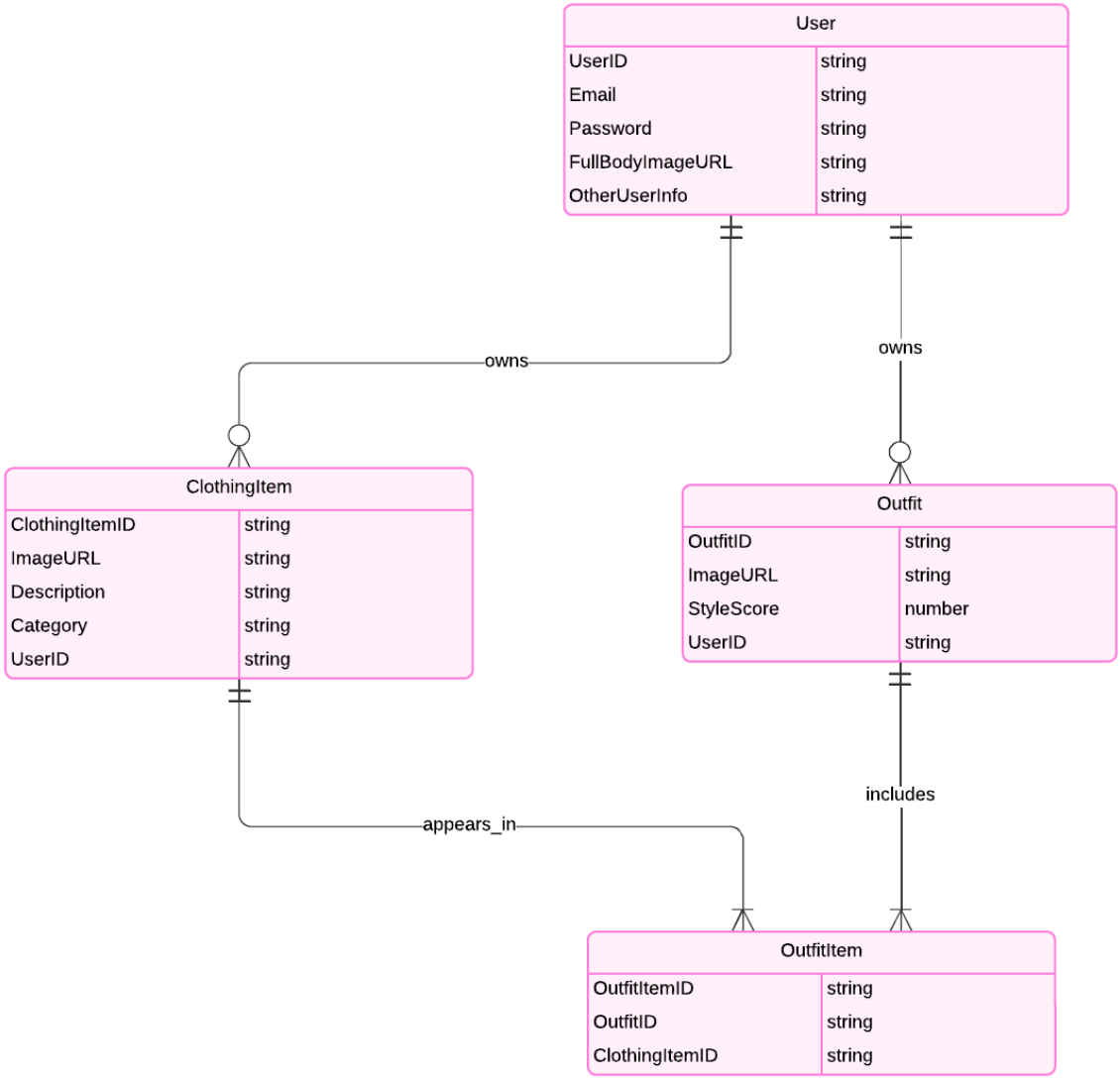
6.3.4.3 State Diagram



6.4 Database Design and Description

This section outlines the design of induōs backend database. Induō's database design leverages Firebase to store and manage critical application data, including user profiles, clothing items, generated outfits, and user preferences. Key entities in the database include Users, ClothingItems, and Outfits. A many-to-many relationship exists between Outfits and ClothingItems, managed through the OutfitItem junction table. Firebase Realtime Database is used for real-time updates, Firebase Storage for image storage, and Firebase Authentication for user authentication. The design prioritizes data security, integrity, scalability, performance, and backup/recovery strategies to ensure a reliable and efficient user experience.

6.4.1 Database Design ER Diagram



6.4.2 Database Access

Induō's database will be accessed primarily through Firebase's SDKs. These SDKs provide a robust and efficient way to interact with the database, allowing for real-time data synchronization, offline capabilities, and secure authentication. The development team will utilize these SDKs to perform CRUD operations (Create, Read, Update, Delete) on the database.

For example, when a user uploads a clothing item, the app will use the Firebase SDK to store the image and its metadata in Firebase Storage. The metadata, including the image URL and description, will be stored in the Realtime Database. Similarly, when the AI generates outfits, the generated images and style scores will be stored in Firebase Storage and the Realtime

Database, respectively. By leveraging Firebase's powerful features, induō ensures seamless data access and updates across different platforms and devices.

6.4.3 Database Security

To ensure the security of user data, induō employs a multi-layered security approach. Robust authentication mechanisms, including password-based authentication and potential social media authentication integration (Google, Facebook, Github, etc.), are implemented using Firebase Authentication. Sensitive data, such as user passwords and personal information, is encrypted both at rest and in transit, leveraging industry-standard encryption algorithms. Firebase's granular access control system is utilized to restrict access to specific data based on user roles and permissions. The development team will conduct regular security audits and will promptly apply security updates to address vulnerabilities.