Bilkent University
Department of Computer Engineering

Senior Design Project
T2327
Capsule

Final Report

22003186, Ali Emir Güzey, emir.guzey@ug.bilkent.edu.tr;
22003229, Alp Afyonluoğlu, alp.afyonluoglu@ug.bilkent.edu.tr;
22003158, Ceren Akyar, ceren.akyar@ug.bilkent.edu.tr;
22003530, Deniz Mert Dilaverler, mert.dilaverler@ug.bilkent.edu.tr;
22002379, Mehmet Kağan İlbak, kagan.ilbak@ug.bilkent.edu.tr

Supervisor: Asst. Prof. Hamdi Dibeklioğlu
Course Instructors: Dr. Atakan Erdem & Mert Bıçakçı

13.05.2024

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

1. Introduction

Our clothing choices reflect our personalities and often don't get the credit they deserve. People, ranging from high school students to middle-aged adults, are increasingly paying attention to their clothing choices. This comes with a set of challenges, one of which is the overwhelmingness of tracking daily outfits and finding clothes to dress differently every day. As a result, people are prone to shopping beyond their needs despite having perfectly good outfit options in their wardrobes. Research shows that the average U.S. citizen annually spends 1434$ on clothing, which has its own economic and environmental drawbacks [1]. Capsule aims to solve this problem.

We embarked on this journey with the idea of providing people with a “capsule wardrobe,” which is described as “a collection of clothing composed of thoughtfully curated, easily interchangeable items designed to maximize the number of outfits that [1] can create.” [2] Our objective is to enable users to optimize their wardrobes by curating stylish outfits without the burden of possessing an unnecessary amount of clothing.

2. Requirements Details

2.1. Functional Requirements

This subsection defines the core capabilities and functionalities of Capsule. These functional requirements encompass the application's ability to track wardrobe usage, generate outfit recommendations based on user preferences, and facilitate collaborative dress code decisions.

2.1.1. User Profile Creation and Management

- Users must be able to create and manage their profiles, including adding personal information, profile pictures, and wardrobe details.

2.1.2 Sharing Outfits

- Users can share their outfits and clothes to message clients like Whatsapp.

2.1.3 Privacy Controls

- Users should have the ability to control the visibility of their activities and wardrobe details.
- Users can set privacy preferences for their profile, events, and dress code discussions.

2.1.4 Outfit Recommendation

- The system must be able to combine clothes to form personalized outfit recommendations that can be accessed by the user within the app.
- Either a completely algorithm-generated outfit must be suggested, or the user must be able to enter specific items that they would like to wear at the moment, and the system must be able to provide suggestions that go well with the selected item(s).

2.1.5. Personalization

- Outfit recommendations must be personalized based on the user's behavior and personal information, which may include the user's age, gender, body type, and behaviors learned via machine learning models.
- Outfit recommendations must also be personalized based on the instance for which the recommendation will be made, which may include weather, dress code, and user's preferences regarding what type of clothes to wear at that specific time.
- For behavior-analysis-led personalization, some features must be tracked, which may include the type of clothes worn around the current time.

2.1.6 Image Recognition

- If an item cannot be recognized or is not present in the database, the user must be able to take photos of an item, and it must be automatically labeled by the system before adding it to the database.
- Photos of the user that are either shared in the app or uploaded as mirror-selfie must be processed and matched with items available in the user's wardrobe to track the last-worn information of items, as well as how long the item is used for.

2.1.7. Wardrobe Tracking

- The system must be able to track and estimate outfit items that are available for users to use at any given time.
- The system must add outfit items to the wardrobe as the user takes photos of clothes.

2.1.8. Wardrobe Processing

- The system must be able to make suggestions to make use of the least-worn clothes. The suggestions may include creating outfit combinations which include that specific item or offering to sell the item.
- The system must be able to track and detect when a cloth is worn out too much.

2.1.9 User Analytics

- The system must be able to show weekly and monthly wardrobe reports, which will have a digest of the following information. These reports will be shown to the user like a pop-up event.
    - Most/Least worn clothes
    - Most worn colors
    - View the pie chart of outfit categories in wardrobe
    - New outfits / New clothes added to wardrobe (3 examples most and count of items (weekly))
    - Wardrobe utilization percentage (weekly/monthly)

2.1.10. Building Outfits Manually

- In addition to system-recommended outfits, users must be able to manually create outfits by using the clothes already available in their wardrobes.
- User must build an outfit by selecting a clothing item for each item type.

2.1.11 Outfit Management

- Users should be able to save either manually or automatically created outfits and access them later.
- Users should be able to add outfit properties optionally, such as names and tags.

2.1.12 Signup and Login

- Users should be able to authenticate via Google, Apple, or email and password.
- Users should be able to verify their email address by entering a verification code sent to their email address by the system.
- During the signup process, users should be able to enter information such as full name, date of birth, and username, upload profile picture, and select their style preference, which includes masculine, neutral, or feminine, to shape their profile.

2.1.13 Reset Password

- In case of forgetting their password, users should be able to reset their password by entering their email address and using the link mailed to them to set a new password.

2.1.14 Selecting Outfits For a Day

- Select “Today” or a day from the calendar view
- The user can select an outfit
    - Select a saved outfit from the built outfits in the wardrobe
    - Build an outfit for the day
    - Take a mirror selfie to match worn outfits to the ones in the system
- User can view selected outfits from the calendar view and be reminded of their daily outfit

2.1.15 Viewing/Managing Wardrobe

- User selects between outfits and clothes
- Filtering clothes by color, favorited clothes and type (top, overwear, bottom, full-body, shoes)
- Adding clothing or outfit to favorites
- See outfit/clothing properties
    - ML-recommended
        - Colors
        - Tags
        - Seasons
        - Texture
    - Usage counts
    - Last worn date
    - Favorited or not
    - Create an outfit with this clothing item
- Delete outfit/clothing

    - If clothing is in an outfit, the user is prompted with “You have outfits with this item do you also want to delete them?” This prompt will only be asked once; this selection can be changed from the settings page.
        - Yes: the clothing is deleted, and the outfit has an empty slot at the palace of the clothing
        - No: The clothing is not deleted
- Add outfit/clothing to wardrobe

2.1.16 Clothing Matching

- User can take a mirror selfie for outfit logs, the matching engine then matches the clothes in the database

2.2. Non-Functional Requirements

In this subsection, we outline the critical performance, security, and quality criteria that Capsule must adhere to. These non-functional requirements encompass aspects such as data privacy, system responsiveness, scalability, and usability.

2.2.1. Usability

The application should be intuitive to use. It should not have complex user interface elements and paths that could confuse the user. The user should be able to learn and navigate the basics of the app in under 5 minutes. Furthermore, some features of the application should be usable without an internet connection. Features such as viewing wardrobe and previous outfits should be available offline.

2.2.2. Reliability

The application should have a reliable infrastructure and should not crash. In the worst case, the backend should not be down for more than 10 minutes. The application should also produce reliable outfits. Test accuracy results of the machine learning models should be higher than 90%.

2.2.3. Performance

The user interface of the application should be fast in order to maximize the user experience. Rendering each component should take less than 100ms. The machine learning models should produce decent results in under 1 second. Furthermore, backend resource consumption should be less than 80% at all times. This would be a measure for usage spikes, which could halt the backend.

2.2.4. Supportability

The application should be supportable, such that it could receive further development in the future. It should be supportable for at least a year due to potential contracts with brands. The application should not have predefined instructions that could hinder its development in the future.

2.2.5. Scalability

The application should be scalable to more than 100,000 users without a major financial burden. It should be easy to expand the capacity to over 1,000,000 users in order to serve the incoming users.

3. Final Architecture and Design Details

3.1 Current Software Architecture

App Name	Capsule (Our Project)	Whering	Pronti	Pureple
Outfit Recommendation	Yes	Yes	Yes	No
Manual Outfit Builder	Yes	Yes	Yes	Yes
UI Quality	Good	Good	Cluttered and Unintuitive	Bad
Wardrobe Analytics	Yes	Yes	No	No
Shopping Recommendation	Yes (with Turkish retailers)	Yes, but inaccurate and no Turkish retailers	Yes, but no Turkish retailers	No
Outfit Collaboration for Events	Yes	No	No	No
Business Plan	Affiliate Marketing, Sponsored Content	No ads or subscriptions, likely affiliate marketing	No ads or subscriptions, likely affiliate marketing	₺28/month on Android, ₺99/month on iOS, Unsubscribed use is nearly impossible due to overwhelming ads

Table 1: Competitive market analysis for Capsule

While the smart wardrobe market isn't new, we believe that Capsule can do things better than the competitors, especially in the Turkish market. We have picked 3 of the most prevalent competitors in the market. Pureple was the worst out of all the smart wardrobe solutions we have seen. They went with a subscription model, which isn't worth the price, and without a subscription, it is impossible to use the app as every click causes a 5 to 30-second advertisement to be run.

Another competitor of ours is Pronti. Unlike Pureple, they have redeemable qualities, and it is usable. However, Pronti's UI and UX aren't desirable. It is hard to navigate and all around ugly.

Whering is the main concern of ours when it comes to market competition. They have a clean UI with a good feature set; however, with the event feature and the better implementation of the core features, we believe we can steal a significant market share from them. Despite its qualities, Whering doesn't have much presence or support for the Turkish market, which allows us to start from the Turkish market and grow our customer base.

3.2 Final Architecture Overview

Capsule follows two architectural styles: layered architecture and microservices architecture. User Interface Layer, Business Layer, and Data Layer are the three layers that form the application. The ML engines for outfit recommendation, clothing item segmentation and labeling, and matching, together with the Django web server, form a microservices architecture inside the Business Layer.

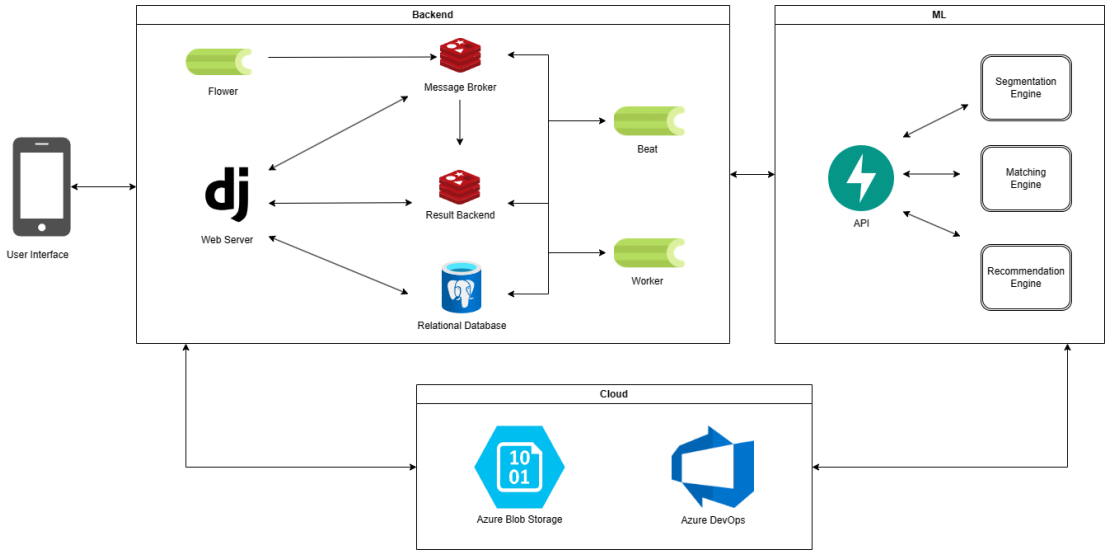

Figure 1: Architecture Overview of Capsule

3.3 Final Backend Architecture

3.3.1 Django Web Server

The Django web server is the main business layer component that interacts with the external engines, APIs, and the database. As far as the mobile interface is concerned, the Django web server is the only communication option. All the logic about Authorization and Authentication, users, clothes, and outfits will be handled here. This will also be used as a proxy to interact with the ML engines.

3.3.2 PostgreSQL Relational Database

As the primary database, a PostgreSQL relational database will be used because of the team's familiarity with the system and due to economic reasons, as it is an open-source system. All the non-image data will be stored here.

3.3.3 Azure Blob Storage

An Azure Blob Storage account will store and organize the clothing and profile images uploaded by users and those generated by the ML engines.

3.3.4 Result Backend

Redis is used for the message queue backend. It is used as the cache in order to improve performance in the backend.

3.3.5 Redis Message Broker

Redis is also used for message broking. For long running tasks, the message broker schedules them to different Celery workers.

3.3.6 Celery Worker

Celery Worker allows concurrency by running code on a designated parallel container. These containers are running long duration tasks like clothing recommendation requests.

3.3.7 Celery Beat

Celery Beat schedules cron jobs. They are tasks that need to be ran periodically. Celery Beat schedules these jobs on Celery Worker instances. An example to this is the recommendation algorithm running periodically to cache recommendation results.

3.3.8 Flower

Flower is a manager for celery clusters. In our system we use it for viewing tasks history, running times. It is also used for manually running or stopping tasks.

3.4 User Interface Architecture

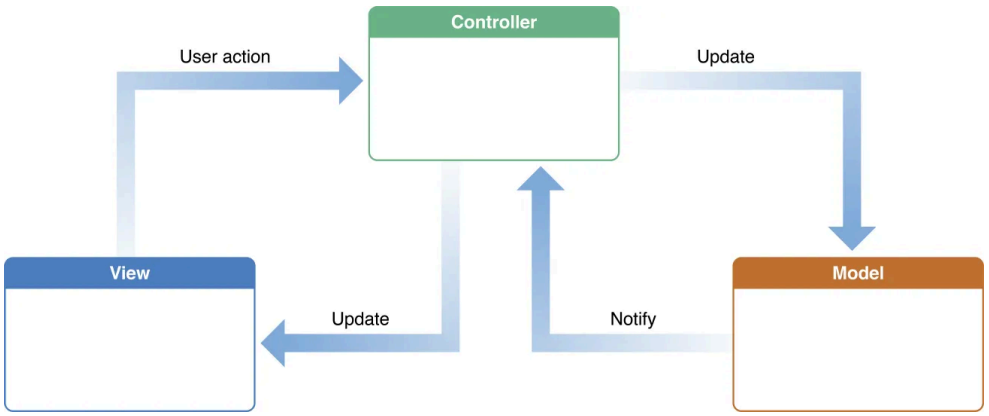

Figure 2: User Interface Architecture [3]

We used the Model-View-Controller (MVC) approach as our architecture for the user interface. In our Swift-based iOS app, Capsule, we used the model approach to create structs like ClothingItems, Outfits, OutfitLogs, and RecommendedOutfits to encapsulate the data we work with. for the views, SwiftUI is used mainly with an addition to UIKit and two user interface

libraries ConfettiSwiftUI and WeeklyCalendar. The controller acted as an intermediary between the models and views. It handles user input, updates the models as necessary, especially with the requests to the server, and updates the view to reflect changes in the model.
Overall, we selected the MVC architecture since This separation makes the code easier to understand, maintain, and test.

3.5 Segmentation Engine

The segmentation engine consists of two models: one for segmenting unworn single-item clothing items and one for extracting multiple clothing items worn by a person. Segmented images are then passed to a classification model for labeling.

3.5.1 Single-item Segmentation Model

The single-item model processes the image of a single clothing item and removes the background and any foreground object that does not relate to clothing in order to output the masked version of the clothing item with a transparent background, as seen in Figure 3. This model is used when adding a new clothing item to the user's wardrobe.

Figure 3: Sample input image and corresponding output of the single-item segmentation model

The model is based on the U-net structure, which is an encoder-decoder network that first creates embedding of the input image and then uses it to generate an output mask. The model utilized in the project uses pre-trained weights already fine-tuned for binary segmentation of clothing items over the iMaterialist Fashion dataset. The dataset contains 46 categories of clothing items, as seen in Table 2, allowing the model to detect and segment a variety of items [4]. The model uses "timm-efficientnet-b3" encoder and can semantically segment clothing items without classification into a single class [5]. After segmentation, the resultant mask is used to cut the clothes out of the input image. The resultant masked image is then passed to the classification model.

Table 2: Clothing item categories included in the iMaterialist Fashion dataset [4]

ID	Name	Category
0	shirt, blouse	upperbody
1	top, t-shirt, sweatshirt	upperbody
2	sweater	upperbody
3	cardigan	upperbody
4	jacket	upperbody
5	vest	upperbody
6	pants	lowerbody
7	shorts	lowerbody
8	skirt	lowerbody
9	coat	wholebody
10	dress	wholebody
11	jumpsuit	wholebody
12	cape	wholebody
13	glasses	head
14	hat	head
15	headband, head covering, hair accessory	head
16	tie	neck
17	glove	arms and hands
18	watch	arms and hands
19	belt	waist
20	leg warmer	legs and feet
21	tights, stockings	legs and feet
22	sock	legs and feet
23	shoe	legs and feet
24	bag, wallet	others
25	scarf	others
26	umbrella	others
27	hood	garment parts
28	collar	garment parts
29	lapel	garment parts
30	epaulette	garment parts
31	sleeve	garment parts
32	pocket	garment parts
33	neckline	garment parts
34	buckle	closures
35	zipper	closures
36	applique	decorations
37	bead	decorations
38	bow	decorations
39	flower	decorations
40	fringe	decorations

41	ribbon	decorations
42	rivet	decorations
43	ruffle	decorations

44	sequin	decorations
45	tassel	decorations

3.5.2 Multi-item Segmentation Model

Compared to the single-item model, the multi-item model carries a more difficult task, as it requires high-resolution processing of images to prevent loss of details of the smaller clothing items in the image. Additionally, this task requires instance segmentation rather than semantic segmentation, meaning the model has to be able to differentiate between different objects of the same category. Multiple attempts were made while trying to achieve an efficient, high-accuracy model. The final version is used when logging what the user wore for a given day via a selfie image, for which the model segments the input image as seen in Figure 4.

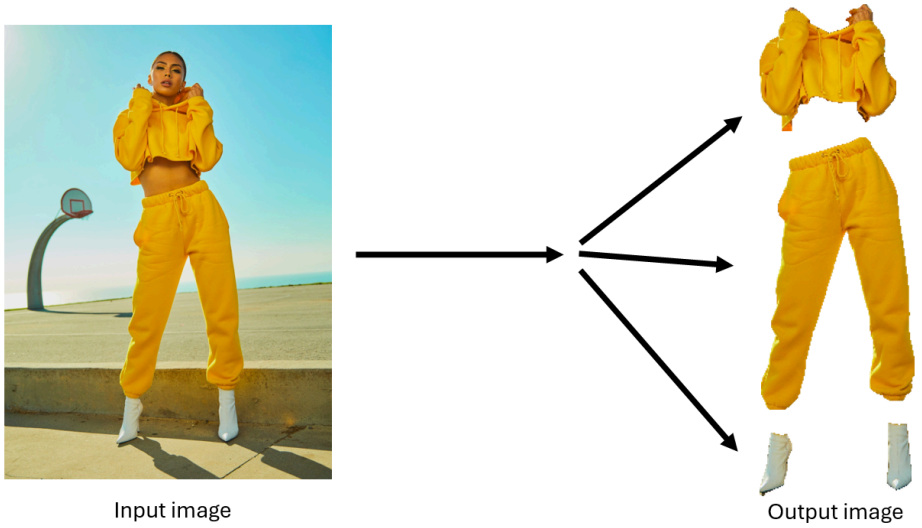

Figure 4: Sample input image and corresponding output of the multi-item segmentation model

3.5.2.1 Previous Attempts

The initial approach was to train a U-net model, just like the single-item model. The Clothing Co-Parsing (CCP) Dataset was chosen for training, which included 59 separate categories [6]. Upon initial attempts, it was quickly realized that a single U-net was not enough to fully capture the patterns that allow accurate segmentation of all 59 classes. After multiple incremental changes, this evolved to a collection of segmentation sub-models following a hierarchical structure. In this model, Each model fine-tuned a U-net structure that uses “resnet152” as an encoder and was pre-trained on the imagenet dataset. The number of required sub-models was determined according to the selected dataset. Upon manual analysis of the CCP dataset, the 59 original labels were categorized, and categories were fit into a hierarchical tree structure, as seen in Figure 5, in which red labels represent ignored and removed parts, green labels represent segmented and usable parts, and blue labels represent intermediate images

that require further processing before being used. The mapping of dataset labels and chosen hierarchical category labels can be seen in Table 3.

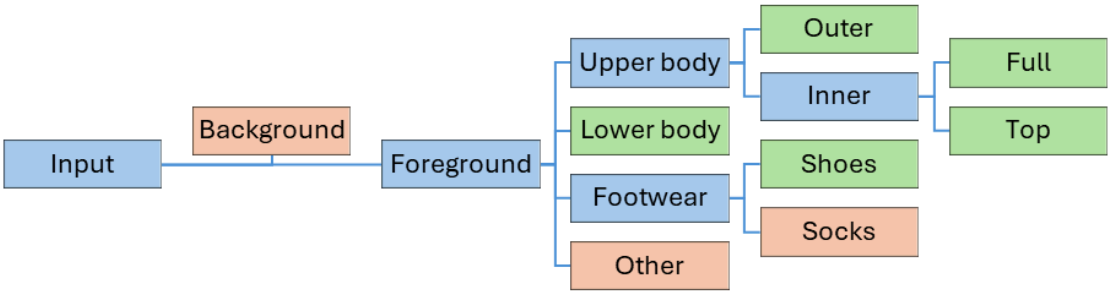

Figure 5: Hierarchical label category structure of the CCP Dataset

Table 3: Mapping of dataset labels and hierarchical category labels

Hierarchical category	Dataset labels
Background	null
Foreground/Upperbody/Outer	cape, blazer, coat, jacket
Foreground/Upperbody/Inner/Full	bodysuit, dress, intimate, romper, swimwear
Foreground/Upperbody/Inner/Top	blouse, bra, cardigan, hoodie, jumper, shirt, suit, sweater, sweatshirt, t-shirt, top, vest
Foreground/Lowerbody	jeans, leggings, panties, pants, shorts, skirt, tights
Foreground/Footwear/Shoes	boots, clogs, flats, heels, loafers, pumps, sandals, shoes, sneakers, wedges
Foreground/Footwear/Socks	socks, stockings
Foreground/Other	hat, hair, bag, purse, wallet, accessories, belt, gloves, scarf, tie, glasses, sunglasses, bracelet, earrings, necklace, ring, watch, skin

A separate sub-model was created for every point in Figure 5, where input was divided into multiple branches. For the CCP dataset, five models were created, and the dataset, consisting of 1004 images and corresponding annotations, was processed for each model [6]. Sample output of layers can be seen in Figure 6, which also reflects how the dataset is processed for each sub-model.

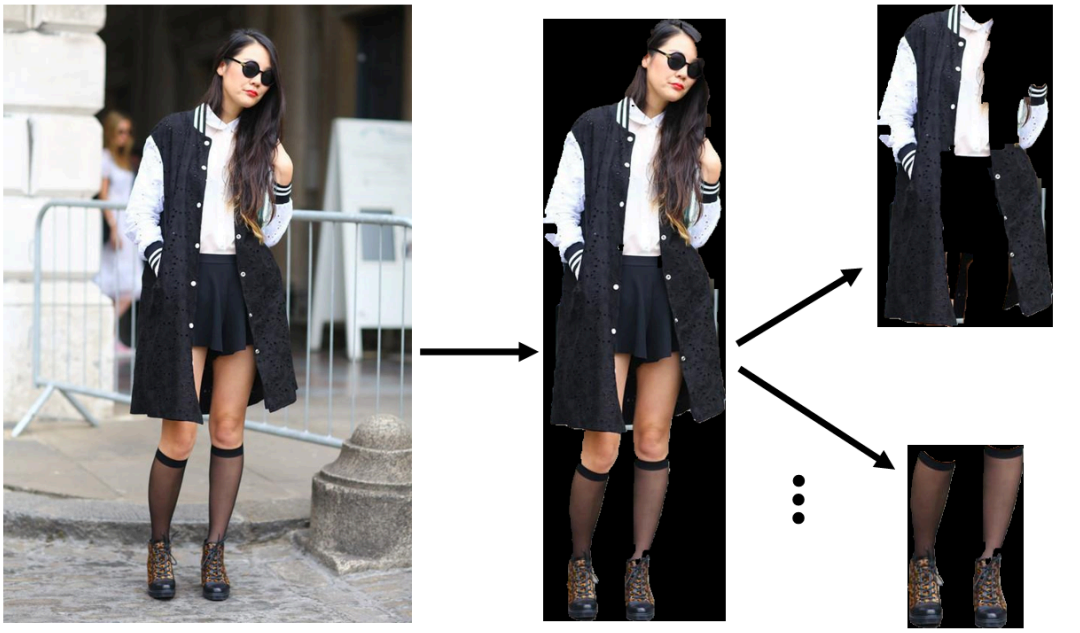

Figure 6: Progression of sample input image through the hierarchical model

Even after grouping labels and implementing hierarchical segmentation models, training models in high resolution was too costly and finding correct hyperparameters was too time-consuming, leading to the model's inability to produce high-accuracy predictions. After multiple failed attempts of different model structures and training attempts, the idea of training a model from scratch was replaced by using a pre-trained model.

3.5.2.2 Final Model

In the final version, the Segment Anything Model (SAM) is used for the multi-item segmentation task, which is pre-trained on the SA-1B dataset. When compared with the U-net structure, this model has two main differences that makes it outperform the competitor models. First, it has a prompt encoder that takes in text, points, or boxes on the input image to direct the model's focus, as seen in Figure 7. Secondly, unlike ResNet or EfficientNet, SAM uses a Vision Transformer (ViT) for image encoding, which yields better pattern capturing and higher model scores [7]. The base vision transformer (ViT-b) backbone is used in this project for faster processing with less memory requirement. With this model structure, the multi-item segmentation task became easier and more accurate, even with 1024x1024 high-resolution input images.

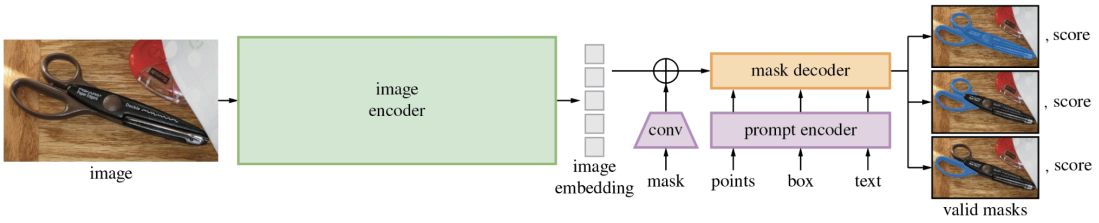

Figure 7: Segment Anything Model structure [7]

This model allows segmenting in both automatic and manual modes. Automatic segmentation processes the image to extract all clothing items automatically. Manual segmentation accepts points on the input image to be fed into the model to extract only the items that are limited by the point prompt, which produces results faster and, in some cases, more accurately than the automatic mode.

3.6 Classification Engine

The output of both segmentation models passes through the classification model for labeling clothing items in greater detail. For this purpose, a fine-grained classification model is constructed, the accuracy of which continues improving as different network structures are used to experiment. The model is trained on a custom dataset collected from the web via our custom web scraper. The dataset contains 4400 images having 5 categories and 22 subcategories, where each subcategory contains 200 images. Categories and subcategories are used to create a hierarchical labeling tree, which is used to train the model on different layers, which directs the behavior of different parts of the model and increases overall accuracy. The test accuracy of the model is 73.64%, and the related confusion matrix can be seen in Figure 8.

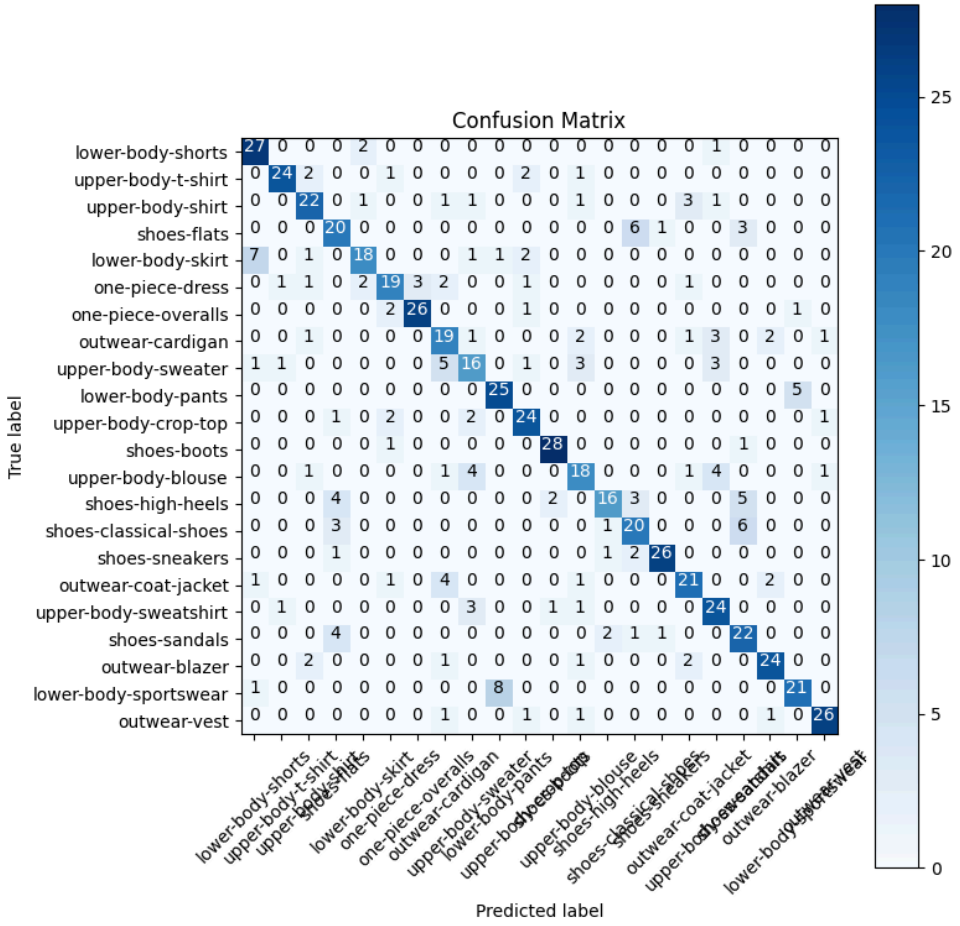

Figure 8: Confusion matrix of the fine-grained classification model

Model structure can be seen in Figure 9, in which the ResNet block represents a resnet18 model pre-trained on the imagenet dataset; all parameters except “layer4” and forward of which are frozen. The last fully-connected layer of the model is removed, making the model headless. Features extracted via the ResNet block are then passed to Module 1, which uses fully connected layers to output 5 nodes, each representing a category. Module 1 output is concatenated with the ResNet output and passed to Module 2, which passes the input through other fully connected layers to output 22 nodes, each representing a subcategory. Both category output and subcategory output are used for loss calculation, making Module 1 and Module 2 specialized in category and subcategory detection, respectively. The model contains 4 fully connected layers in Module 1 and Module 2, having 1024, 512, and 256 neurons in the first three layers, respectively, 5 nodes for Module 1 and 22 nodes for Module 2 in the 4th layer. In between the fully connected layers are Relu activation functions and dropout layers for regularization.

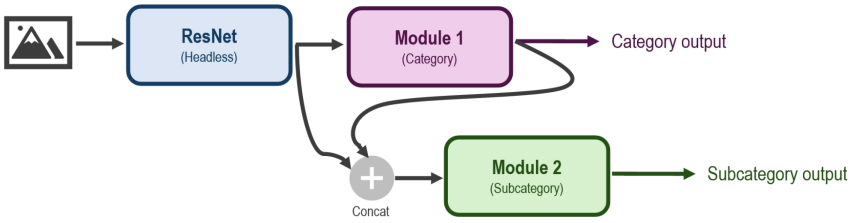

Figure 9: Structure of the fine-grained classification model

Despite the actual model accuracy being around 73%, the acceptable correctness of the model is higher. As seen in Figure 10, some categories are similar, like crop-tops and t-shirts, and having the model possibility mixing those categories out would not be a problem for the users. Similarly, some items are suitable for multiple categories, like the boots with heels seen below, which can be categorized as either boots or heels.

Figure 10: Sample segmented images from our custom dataset

3.7 Recommendation Engine

The recommendation engine is composed of two main components. The first component is a transformer based outfit scoring model. The second component is the ranking algorithm which combines the outfit score with constraints such as occasion, weather, season, or natural language.

The transformer based outfit scoring model was trained on the PolyVore dataset [9]. The dataset consists of user created outfits from polyvore.com which includes individual outfits' descriptions, views, and likes. In addition, it includes images and descriptions of each clothing item. The model's main goal is to predict the compatibility of all items in an outfit. Given an outfit of N items $\{(I_i, T_i)\}_{i=1}^{N}$ where I_i is the image and T_i is its text description, the model predicts a compatibility score between 0 and 1 where 1 is the perfect compatibility [10]. The image and its description is fed into image and text feature extractors to obtain feature vectors. Then, both image and text vectors are concatenated to obtain a clothing feature vector. In addition, an outfit token is introduced whose state represents the global outfit representation [10]. These clothing feature vectors are fed into the transformer encoder with the outfit token to get a global outfit representation [10]. Then, the global outfit representation is fed into a

multi-layer perceptron to predict an outfit compatibility score [10]. This score is then used by the ranking algorithm.

The ranking algorithm essentially performs similarity search between the compatible outfits and the given text query. There are various algorithms to achieve this. However, we decided to settle on two heuristic based algorithms since brute force was unusable due to the exponential runtime.

The first heuristic is beam search which uses breadth-first search to generate outfits from user's clothings. At each step, it generates the successor sequence by first generating partial outfits based on the transformer model. Then, similarity search is applied with the text query to select the best clothings. Each successor sequence is based on previous B sequences which is also called the beam width.

The second heuristic is the nucleus sampling which uses a probabilistic approach to select the next clothing from the user's clothings [11]. Similar to beam search, compatibility score and the text query is used to create outfits. At each step, it selects clothings whose cumulative probability exceeds the chosen threshold p [11]. This allows sequences to be dynamic unlike the beam search [11]. Therefore, the generated outfits have more variety.

Finally, the recommended outfits are classified by users with right or left swipes where a right swipe indicates 1 and a left swipe indicates 0. This data is then collected and utilized to train the outfit transformer model.

3.8 Matching Engine

The outputs of the multi-item segmentation model are passed to the matching engine to determine which cloths the user is wearing from their wardrobe. To accomplish this, the engine compares all images in the wardrobe with the inputted clothing item. For feature extraction, the matching engine uses OpenAI's CLIP model, which gathers text and image embeddings in a single latent space, allowing general-purpose labelling of images and creating suitable image embeddings [8]. CLIP is based on a Vision Transformer backbone, just like SAM. In addition to feature vectors, information such as the dominant color, category and subcategory labels are used by the engine. As images are added to the user's wardrobe, all input images are passed through CLIP for feature and information, such as color, extraction. To match a new image with one in the user's wardrobe, the engine compares the feature vectors to find the most similar vector, while considering color and other information as constraints. In a sense, the feature vectors are converted to corresponding points on a multidimensional hyperspace, and when a new image is inputted, the image is processed just like other items in the user's wardrobe to extract a feature vector, which is then projected on the hyperspace. Based on the distance between the new image and other points, all clothes in the user's wardrobe are ordered according to similarity. The matching engine returns the closest point as the matching result and several other points as suggestions within the determined limit of max distance.

To measure the accuracy of the engine, the Multi View Clothing (MVC) dataset is used to fetch 151 clothing pairs and run the engine over them [12]. While the engine yields an accuracy of

19.38%, some different clothing items that look very similar are matched by the engine, as seen on Figure 11, which can be considered correct from the user's point of view, making the perceived correctness of the engine higher.

Item A Item B

Figure 11: Two different clothing items that look alike from the MVC dataset, matched by the engine

4. Development/Implementation Details

4.1 Backend Development Details

For the backend of the project we used the Django Rest Framework with the Python language. The reason behind this was that the two backend developers on the project had familiarity with both of the technologies and the simplicity of the Django framework. During the development and production, we dockerized the entire backend system as it made it easier to run the project from any machine necessary. Also, Dockerization allowed us to orchestrate all 7 containers of ours. Furthermore, we used pre-commit to check the code formatting before every commit regarding the backend. This allowed our codebase to be more maintainable and consistent.

In order to have test data readily available, we exported the data of a test user called "cerenimo" into a JSON file. This allowed us to set up a backend instance with a user to test the application within a matter of seconds. This also provided a backup for whenever the database data got corrupted and needed to be deleted during development.

4.2 Frontend Development Details

We designed the user interface using the Figma application for frontend development. For the implementation part, we started the implementation using React Native. However, some issues were presented, such as the photo gallery not being loaded correctly or the lack of capability to use the native features provided.

Thus, we rewrote the front end in Swift. With the native option, we could use the native properties such as removing the background of images and reaching the photo gallery and camera oh iPhones. In detail, we used Swift, SwiftUI, and UIKit. Additionally, we used two

libraries, ConfettiSwiftUI, to throw confetti when the user likes a recommended outfit and WeeklyCalendar to view the calendar.

4.3 Machine Learning Models Development Details

On the machine learning side, a Python environment is used for training and testing models, and serving model outputs via an API. All models used in this project are based on PyTorch, due to wide availability of popular and novel models on PyTorch framework and the familiarity o the team's ML developers with the framework. Models are trained on both free and paid Google Colab GPUs as well as the GPUs of the team members' computers. For efficient processing, vectorized operations are used as much as possible with the help of Numpy and similar libraries.

After initially trying to use Git Large File Storage (Git LFS), the trained model instances are decided to be stored on Hıgging Face. Bash scripts are used to fetch datasets, download trained model instances, and run the ML API. FastAPI is used to serve RESTful endpoints to be accessed by the backend.

5. Test Cases and Results

5.1 Functional Test Cases

FT0001
Title: Create Outfit Manually
type/category: Functional
Severity: Major

Step	Outcome	Pass/ Fail
Go to add wardrobe page	Wardrobe page opens	**P**
Press the Add New Outfit button	Opens add new outfit dropdown	**P**
Select create a manual outfit	Opens manual outfit creation page	**P**
Select outfit size: top-bottom and shoes with coat option	Relevant outfits are shown to the user according to the selected size preference	**P**
Select clothing preference for each segment	Clothings are set	**P**
Press continue	The full outfit is displayed	**P**
Enter name and tag	Entered details are set	**P**

Press save	Outfit is saved	**P**

Table 5: Manual Outfit Creation Test

FT0002
Title: Automatic Outfit Generation
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
Go to add wardrobe page	Wardrobe page opens	**P**
Press add new outfit button	Opens add new outfit dropdown	**P**
Select create AI-generated outfit	AI-generated outfit creation page open	**P**
Select occasion	Outfit recommendations are shown in a swiping view (Tinder-esque)	**P**
Swipe left to say that you didn't like that outfit	Outfit is removed from the top of the stack with an indication that it is disliked	**P**
Press not interested	Outfit is removed from the top of the stack with a pop up indicating that this outfit won't be recommended any more	**P**
Swipe right	Outfit is saved and selected	**P**
Enter tag and outfit name	Added properties are set	**P**
Press save icon	Outfit is saved	**P**

Table 6: Automatic Outfit Generation Test

FT0003
Title: Register Clothing
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
Open wardrobe tab and	Opens dropdown of camera and	**P**

press add clothing	camera roll	
Select camera and take a picture	App asks for approval of the picture	**P**
Approve the picture	Segmented image is received and with recommended tags and the subcategory the outfit is in	**P**
Edit properties and hit save	The clothing item is saved	**P**
Do the same for addition by camera roll	The clothing item is saved	**P**

Table 10: Register Clothing Test

FT0004
Title: Login
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
Enter user credentials to username and password fields	Signed into the app	**P**
Signout	Opens login page	**P**
Press sign in with Google and select the account prepared for the test	Signed into the app	**P**
Signout	Opens login page	**P**
Press sign in with Apple and select account	Signed into the app	**P**

Table 11: Login Test

FT0005
Title: Signup by email
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
In the login page select signup and choose signup	Opens signup by email page	**P**

by email		
Enter Email and password and press create	Systems asks for email authentication	**P**
Open the email account and click the auth link	User is directed to personal details step	**P**
Fill the properties and click continue	Account created	**P**

Table 12: Signup by email Test

FT0006
Title: Signup with Google/Apple
type/category: Functional
Severity: Minor

Step	Outcome	Pass/ Fail
In the login page select signup and choose signup by Google/Apple (Do the following steps for both) and select your account	Opens signup by email page with name fields filled according to your Single Sign-on provider	**P**
Fill the properties and click continue	Account created	**P**

Table 13: Signup with Google/Apple Test

FT0007
Title: Forgot password
type/category: Functional
Severity: Major

Step	Outcome	Pass/ Fail
On the login page, select I forgot my password.	App prompts you with email	**P**
Enter email and press reset	An email for password reset is sent to your account	**P**
Go to the mail account and press the reset link	Directs the user to a password change page	**P**
Enter the new password and click reset	Password is reset	**P**

Sign In with the old password	Access is denied	**P**
Sign In with the new password	Login to the app	**P**

Table 14: Forgot password Test

FT0008
Title: Select outfit for a day
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
Select today or a future date from the home page	Home page for the set date is opened	**P**
Click add outfit	Dropdown of outfit selection options	**P**
Click select from outfits	List of your outfits are shown	**P**
select one of the outfits	Outfit is added to the date as worn	**P**
Click add outfit	Dropdown of outfit selection options	**P**
Click select from outfits	List of your outfits are shown	**P**
Select build outfit	Redirects you to the outfit creation tab	**P**
Do the same steps in both #T0001 and #T0002	Outfit is created and added as an outfit log to the date	**P**
Click add outfit	Dropdown of outfit selection options	**P**
Click mirror selfie	Opens the camera	**P**
Take a picture of your whole outfit	Shows you the outfits you are wearing and shows you the clothing items in the picture	**P**
Approve the outfit	New outfit is created and added as an outfit log to the said date	**P**

Table 15: Select outfit for a day Test

FT0009
Title: Viewing the wardrobe
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
Press wardrobe tab	Opens user wardrobe, should be able to see clothing items (separated by categories) and outfits	**P**
Press favorites icon	Only favorited items and outfits shown	**P**
Press favorites icon again	All items and outfits are again shown	**P**
Click on an outfit	The detailed page of the outfit is opened. Must see tags, category, date it was last worn, is_favorited, how many times it was worn, colors, seasons, texture	**P**

Table 16: Viewing the wardrobe Test

FT0010
Title: Deleting Clothing
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
Press wardrobe tab	Opens user wardrobe, should be able to see clothing items (separated by categories) and outfits	**P**
Click on an clothing that is used in an outfit	Opens outfit detailed view	**P**
Press delete icon	User is prompted with the clothing being in an outfit.	**P**
Click yes	Clothing is deleted	**P**

Table 17: Deleting Clothing Test

FT0011

Title: View analytics
type/category: Functional
Severity: Major

Step	Outcome	Pass/ Fail
Go to profile	Profile opened	**P**
Press analytics icon	Analytics page opens with the week view. Users should be able to see most/least worn clothes, most worn colors, pie chart of outfit categories in wardrobe, view new outfits/clothes added, wardrobe utilization for the week.	**P**
Press monthly tab	Analytics page opened with a month view, and users should be able to see most/least worn clothes, most worn colors, pie chart of outfit categories in wardrobe, view new outfits/clothes added, wardrobe utilization for the week.	**P**

Table 18: View analytics Test

FT0012
Title: Manage profile
type/category: Functional
Severity: Minor

Step	Outcome	Pass/ Fail
Go to profile	Profile opened	**P**
Press edit	Profile fields are now editable	**P**
Edit body type and style gender and press save	Fields should now be updated	**P**
Click the profile picture	Profile picture opened in the detailed view	**P**
Click edit and add a new picture and hit save	The profile picture is updated	**P**

Table 19: Manage profile Test

FT0013
Title: Clothing Item Recommendation
type/category: Functional
Severity: Major

Step	Outcome	Pass/ Fail
Open the outfit recommendation tab	The outfit recommendation tab is opened	**P**
Select pieces of clothing and hit recommend	The app suggests items that would go well with those items.	**P**
Edit body type and style gender and press save	Fields should now be updated	**P**
Click the profile picture	Profile picture opened in the detailed view	**P**
Click edit add a new picture and hit save	The profile picture is updated	**P**

Table 20: Clothing Item Recommendation Test

FT0014
Title: Outfit view
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
Open wardrobe tab	Wardrobe tab opened	**P**
Press outfits	Outfits of user are shown	**P**
Click the topmost outfit	Outfit details are opened	**P**
Press edit and change the tags of the outfit than hit save	Outfit details are edited	**P**
Go back to the outfits view	Outfits of user are shown	**P**
Click the favorited icon	Only the favorited outfits are shown	**P**
Click favorited icon	All outfits are shown	**P**

Table 21: Outfit view Test

FT0015
Title: Settings
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
Press settings icon	Opens settings page	**P**
Turn off/on notifications	Notifications are turned off/on	**P**
Click contact information	Contact information is shown	**P**
Click security preferences and make profile public	Profile is visible to everyone	**P**
Click terms and conditions	Shows terms and conditions	**P**

Table 22: Settings Test

FT0016
Title: Polling
type/category: Functional
Severity: Major

Step	Outcome	Pass/ Fail
Open events tab	Event tab opened	**P**
Click the most recent event and open chat.	Event chat opened	**P**
Click poll and select 4 dresscodes for the poll	Poll is created	**P**
Vote in the poll	Vote added to the selection	**P**

Table 23: PollingTest

FT0017
Title: Deleting Outfits
type/category: Functional
Severity: Critical

Step	Outcome	Pass/ Fail
Go to wardrobe page	Open wardrobe page	**P**

Click outfits	Outfits tab open	**P**
Click the topmost outfit	Outfit detailed paged opens	**P**
Click the delete icon and approve it	Outfit deleted	**P**

Table 24: Deleting Outfits Test

5.2 Non-functional Tests

5.2.1 Usability

All usability tests will be evaluated according to the number of tasks completed successfully, and according to the task times.

$$Effectiveness = \frac{\text{Number of tasks completed successfully}}{\text{Total number of tasks undertaken}} \times 100\%$$

$$Time\ Based\ Efficiency = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N}\frac{n_{ij}}{t_{ij}}}{NR}$$

Fig. 7: Usability Metrics [13]

NFT001
Title: Usability Test
type/category: Non-functional
Severity: Major

Step	Outcome	Pass/ Fail
100 users start investigating our 17 functional requirements*	At least 90 out of 100 users must complete all 17.	**P**
Calculate time spent for each functional requirement	Users must complete under 5 minutes each task.	**P**

Table 25: Usability Test

* Done with 20 people instead.

5.2.2 Reliability

NFT002

Title: Reliability Test for Backend
type/category: Non-functional
Severity: Major

Step	Outcome	Pass/ Fail
Check reliability information of uptime and reliability of AWS server.	The backend should not be down for more than 10 minutes.	**P**

Table 26: Reliability Test for Backend

NFT003
Title: Reliability Test for Machine Learning Models
type/category: Non-functional
Severity: Major

Step	Outcome	Pass/ Fail
Run the test batches for each machine learning.	Test accuracy results of the machine learning models should be higher than 85%.*	**P**

Table 27: Reliability Test for Machine Learning Models

* The accuracy of some models is below 85%, but the team agreed that it is acceptable, considering the nature of those specific models, as discussed in section 4.3.

5.2.3 Performance

NFT004
Title: Performance test for Outfit recommendation
type/category: Non-functional
Severity: Major

Step	Outcome	Pass/ Fail
Setup 100 clothing items and 20 outfits	-	**P**
Start a timer and request an outfit recommendation	Outfits must arrive in less than 2.5s	**P**

Table 28: Performance test for Outfit recommendation

NFT005
Title: Performance test for Wardrobe view
type/category: Non-functional
Severity: Major

Step	Outcome	Pass/ Fail
Setup 100 clothing items and 20 outfits	-	**P**
Start a timer and open wardrobe page	Items should load in less than 1.5 seconds	**P**
Rapidly swipe up and down on the page to load different items continuously	The app should scroll smoothly and the new items should be loaded at most in 1 seconds	**P**

Table 29: Performance test for Wardrobe view

NFT006
Title: Performance test for Single item clothing registration
type/category: Non-functional
Severity: Major

Step	Outcome	Pass/ Fail
Open clothing registration page	Page opened	**P**
Take a picture of an outfit lying on the bed and send the segmentation request	The segmented and labeled image should return in less than 3 seconds	**P**
Start the timer again and hit save	The item should be saved in less than 1 seconds	**P**

Table 30: Performance test for Single item clothing registration

6. Maintenance Plan and Details

The maintenance plan is critical for the continuous development of Capsule. This section provides approaches we adopted to support and develop the services and the infrastructure of Capsule.

6.1 Continuous Integration and Development

We have created multiple tests for validating the functionality and the performance of the overall system. When the new code is pushed, it first goes through tests to make sure it does not break any existing functionality. Then, it must be reviewed by a team member to land on the main branch. Deployment pipeline is set up in a way that it automatically deploys the stable version. Furthermore, it supports rollbacks when necessary.

6.2 Failure Prevention

We have set up tests to prevent failure. However, we also have continuous monitoring if something breaks or behaves in an unexpected way.

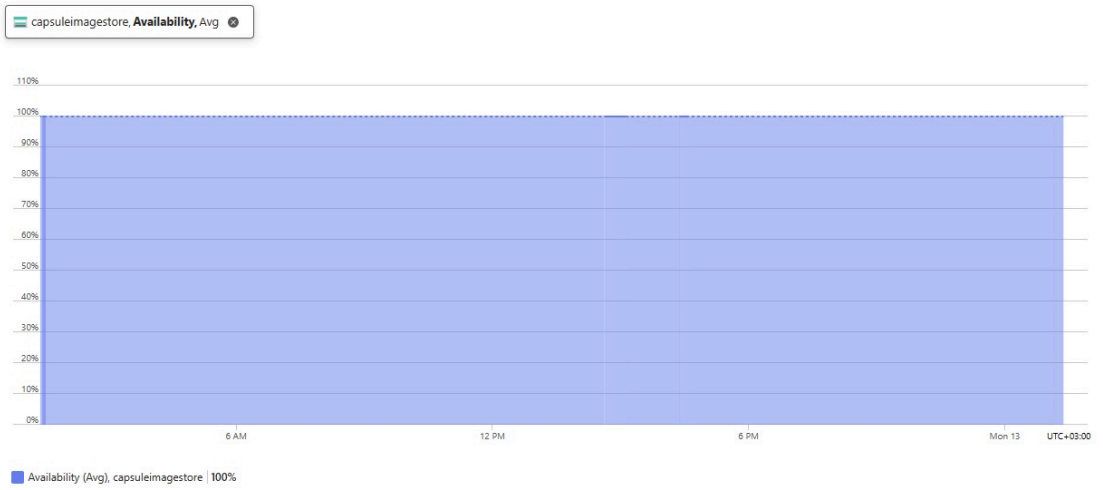

Figure 12: Image Store Availability Monitor

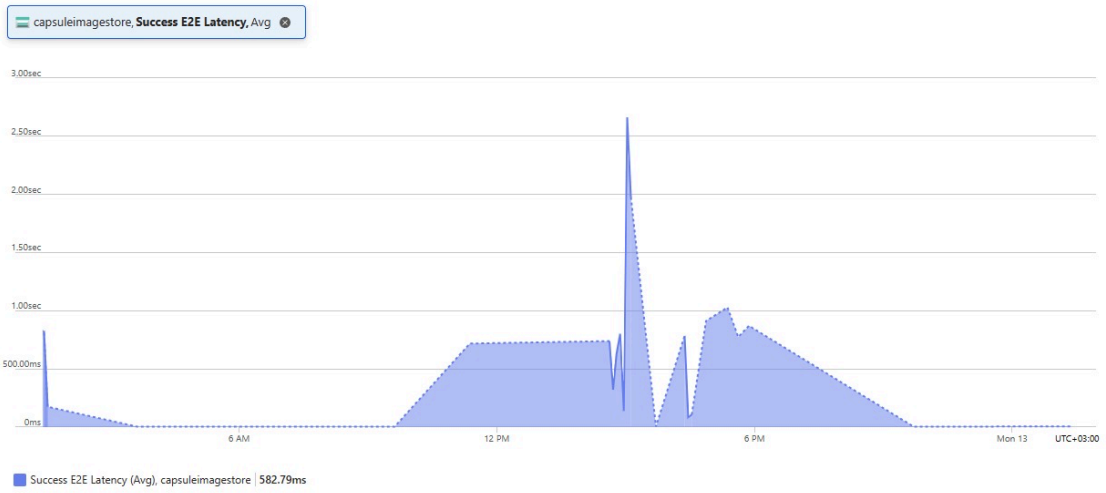

Figure 13: Image Store E2E Latency Monitor

With these monitors, we are able to react quickly to any failures or overloads.

6.3 Scaling Adjustments

Usage and health monitors allow us to track usage statistics, delays, and performance of the system every minute. In the case of overloads or performance issues, we are able to deploy more servers and utilize load balancing. This method ensures that the system is robust and responsive all the time. We also utilize caching on both backend and frontend to minimize the load on the servers. In the future, we are considering database sharding to improve response time across regions. We are also considering automated scaling based on load to reduce time spent on monitoring.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

As the owner of a B2C product, we need to take into consideration what constraints are enforced upon us and what limitations we should keep in mind. Furthermore, we must also standardize our work in order to maintain uniformity across multiple members of the project.

7.1.1 Constraints

7.1.1.1 Public Health and Safety

We must ensure that our application does not compromise the safety and well-being of our users. For example, we should ensure that user data, including images, are handled carefully and that privacy and security measures are in place.

7.1.1.2 Data Security

To prevent unauthorized access to user data, data need to be protected both at transit during client-server communication and at rest on the server side. Necessary authentication and authorization systems need to be implemented, and data should be sent to the client after ensuring the client is the owner of the requested data.

7.1.1.3 Data Privacy and Regulations

Our app should respect the privacy of users' data and comply with related local regulations, such as the General Data Protection Regulation (GDPR) for Europe and, for example, the localized Turkish version, KVKK. Personally Identifiable Information (PII) of users should not be collected unless required to provide our service, and only a minimal amount of information needs to be collected and stored. In addition to the transparency about which data is collected and how it is used, collected information should be permanently deleted upon the request of a user.

7.1.1.4 Ethical Limitations

Any data collection or processing that may lead to ethical uncertainties should be prevented, and ethical considerations should be in focus throughout the project's development. For example, while users need to send their photos to servers for processing and extraction of clothing-related data, having personal photos of users on the server in a human-readable form would lead to ethical and privacy-related concerns. To overcome this issue, faces can be blurred before being transferred to the server for processing in order to minimize the collected PII, increase user privacy, and resolve ethical concerns..

7.1.1.5 Computational Limitations

As the processing of user data will either be done locally on the user's device or remotely on our servers, the amount of data to process will be limited by the computational capabilities of those machines. Provided service will depend on the time of computation and the virtual space, memory, or disk space spared for the computations. Recognizing that the AI engine will perform demanding tasks to recognize and categorize user images, this constraint will be significant throughout the development of Capsule.

7.1.2 Standards

Standards are important for a long lasting software project as it makes it more robust and consistent. Furthermore, it makes it so that all members of the team are on the same page. For Capsule we are using various standards for the reports, diagrams and the codebase.

We are following IEEE standards for requirements documentation, employing IEEE 830-1998 to systematically capture and manage project requirements. We are using IEEE for our citations as well. Additionally, we utilize UML 2.5.1 as the standard modeling language for visualizing, specifying, and documenting various aspects of system design. In terms of code formatting, the "Black" formating standard is used for Python code, ensuring adherence to a strict coding style guide for enhanced readability and maintainability.

7.2. Ethics and Professional Responsibilities

In an app like Capsule where users constantly upload their pictures, there is an important question on ethics. Capsule should be a platform where users shouldn't worry about how their data is used. Therefore it is our utmost ethical and professional responsibility to handle user data securely and privately. One of the main frameworks that we follow for this is GDPR. We do not use the user's data in any context other than the app itself. Any viewing of the data is done purely on maintenance and debug purposes and whilst maintaining user anonymity. The live database and the Azure S3 instances have the principle of least privilege. Hence, such data access is only possible for the developers that need to access the data.

Apart from the privacy aspects, our main goal should be to not discriminate against any ethnicities and cultures. Capsule should be a platform that works for all. So we have collected feedback from people with a variety of differing backgrounds. This allows us to see the needs of different types of users and have the opportunity to fix the problem they are having with our platform.

7.3. Teamwork Details

7.3.1. Contributing and functioning effectively on the team

Ali Emir Güzey
He contributes to the backend development of the project. He is also responsible for the infrastructure-related tasks such as Docker deployment and choosing and integrating necessary cloud products.

Alp Afyonluoğlu
He contributed to the machine learning side, working on segmentation, classification and matching models, structuring the hierarchical multi-item segmentation model and related dataset-processing tools, and fine-grained classification model.

Ceren Akyar
She contributed as the UI/UX designer and designed both the landing page and the user interface of the mobile app. Also, she contributes to the development of the mobile app as a frontend developer.

Deniz Mert Dilaverler
Deniz worked on the backend of the system, designing the schema and APIs for the frontend to consume. He worked on the endpoints for user registration, authentication and also wardrobe side of the app like CRUD requests of clothings and outfits.

Mehmet Kağan İlbak
Kağan worked on machine learning tasks and on the mobile application. On the machine learning side, he contributed to the initial segmentation pipeline and the recommendation pipeline. He also helped rewrite the application in platform native APIs.

7.3.2. Helping creating a collaborative and inclusive environment

Our top priority has always been creating a collaborative and inclusive environment since diverse perspectives and experiences will bring new ideas to light. We have implemented several strategies to foster this environment.

We have set up recurring weekly meetings where each member freely expresses their ideas and feelings about the project. These meetings served as a dedicated space for dialogue where each team member actively participated. Furthermore, we also picked a physical headquarters for our meetings, which is in Dorm 82. This physical gathering place allowed us to have a sense of belonging since we have spent most of our time here. We distributed the workload according to everyone's relative expertise so that everyone was working on a task they liked.

Overall, with these measures, we have achieved our initial goal of creating a collaborative and inclusive environment. Each team member felt valued, empowered, and motivated to contribute their best work.

Ali Emir Güzey
He helps the team through code reviews. He also encourages his teammates with positive feedback. He has a camera, which he uses to make his teammates happier by recording their memories. He also created a shared keychain on Bitwarden where we keep our secrets (e.g. API keys, environment variables) securely. Lastly, he created and shared a custom Neovim configuration to increase his teammates' productivity.

Alp Afyonluoğlu
He worked 1-on-1 with team members to collaboratively integrate ML models with other modular app parts, such as the backend and the web scraper, He documented ML-related parts of the repo with readme files for maintainability and easier understanding by the teammates. He also helped the team with transportation when possible to prevent unproductive time loss during transportation for the whole team.

Ceren Akyar
She designed the user interface for the team to implement collaboratively. She also tries to motivate unmotivated team members and tries to help them with their problems.

Deniz Mert Dilaverler

Deniz distributes tasks for the backend team and communicates with the frontend team for API creation. Furthermore, he documents the endpoints the team creates for the requests to be easily consumable. He also distributes tasks to the team for each report.

Mehmet Kağan İlbak
Kağan helped with the overall synchronization of various subsystems. He contributed to implementation strategies of subsystems such as ML and mobile. He encouraged team members to conduct code reviews with cute GIFs. He also has a camera, allowing him to contribute to Ali Emir's perspective. He also increased the happiness of team members by removing JavaScript from the project.

7.3.3. Taking lead role and sharing leadership on the team

Ali Emir Güzey
He is the lead responsible for system infrastructure tasks.

Alp Afyonluoğlu
He is the lead responsible for segmentation and matching parts of ML.

Ceren Akyar
She is the lead front-end developer and UI/UX designer.

Deniz Mert Dilaverler
Working as the lead backend engineer and report manager of the team.

Mehmet Kağan İlbak
He is the lead responsible for ML infrastructure and the recommendation system.

7.3.4. Meeting objectives

Throughout the development process of this project, including planning, implementation and testing, the team arranged regular meetings to track progress and be in touch continuously. In addition to weekly team meetings, the team regularly met with their supervisor once in every two weeks, and met with a course instructor advisor once every month.Meetings with the supervisor and the advisor provided the chance to showcase the progress and gather continuous feedback and possible improvement recommendations at every stage of the project.

The team met at least once a week, sometimes increasing the meeting frequency up to once everyday. Those meetings focused on planning the project, providing feedback on teammates' work, and working on the implementation together. Coding in the same environment allowed anyone to ask any question to anyone else and get instant replies, making development faster and preventing blocking issues.

As a result of these regular meetings and hard work, the final product got many of the originally planned features implemented. As expected, the product presents the following general features: hold a record of the user's wardrobe, track wardrobe analytics, add a clothing item to the user's wardrobe by simply taking a photo of it, group all items in the wardrobe

automatically, suggest outfit recommendations from the user's wardrobe or our partners while considering the occasion and weather condition, and log what the user wore on which day, either manually or via a mirror selfie. Upon requirement and feedback analysis, some of the team's initial objectives are adjusted, such as changing the in-app messaging feature with sharing outfits via third party messaging apps, which turned out to be more convenient and preferred by users.

7.4 New Knowledge Acquired and Applied

We initially implemented the front end in React Native. However, we realized that implementation in React Native is unreliable since it relies on user libraries, and we cannot properly use the native features provided. Hence, although we previously did not know Swift, we switched. With its easy documentation, learning the language and implementing complex views wasn't hard. In addition, Swift provided us with a reliable environment so that we didn't have to worry about bugs due to user libraries and security issues.

Another knowledge we acquired was the Django framework. While we had prior experience with the framework, this was the first time that we have used it to such extent. We also learned how to handle long running tasks by using message queues and cron jobs. Learning these tools allowed us to have a more performant application despite having long running ML models.

On machine learning side, we learned that training a new model, especially a generative model like U-net that generates masks, requires too much computational power, access to powerful GPUs and high-capacity RAMs. Despite trying to train such models, it was difficult to correctly determine usable hyperparameters and we decided to use pretrained networks instead. We also realized loading multiple models to memory, making them ready to run, occupied too much memory, even when the models were not running, which forced us to find new solutions, such as using lighter models or changing when some models are loaded to the memory. No teammember had a previous extensive ML experience, which led us to learn many ML concepts as this project progressed. This included even trivial ML knowledge, such as modiftying data flow throughout a network to implement, for example, a fine grained classification model,

8. Conclusion and Future Work

To conclude, Capsule is a smart solution to individuals' challenges in managing their wardrobes, tracking their outfits, and making stylish outfit choices. Using advanced technologies such as image recognition and machine learning, Capsule provides an interface for users to optimize outfit combinations and make informed decisions about their clothing selections.

We valued a user-friendly interface and robust functionality the most and provided the users to use everything without choosing nearly nothing., Also, the emphasis on privacy controls and data security underscores our commitment to safeguarding user information. Moreover, Capsule's scalability and supportability ensure that it can accommodate the needs of a growing user base while continuing to evolve and adapt to future developments in fashion and technology.

For future work, we plan to cache the images and save them to the users' phones to avoid loading screens as much as possible. One other plan we have in mind is to run our machine-learning models on the users' phones. With this, we can avoid the long pipelines of network requests and decrease user wait times. Finally, we want to re-adjust some feature locations on the user interface. For instance, the detailed analysis page is currently on the profile view, which is not very reachable for users since they cannot view it immediately. We want to re-adjust the user interface of the home page and add a shortcut for the user interface to view analytics faster. We also want to implement a weekly/monthly analysis indicator to provide users a better analysis.

Overall, Capsule represents a paradigm shift in how individuals approach wardrobe management, offering a sustainable and efficient alternative. With Capsule, users can reduce unnecessary spending and embrace a more mindful approach to fashion, all while looking effortlessly stylish.

9. User Manual

The below subsections explain how to use Capsule by examining each display the user can view.

9.1 Sign Up and Log in

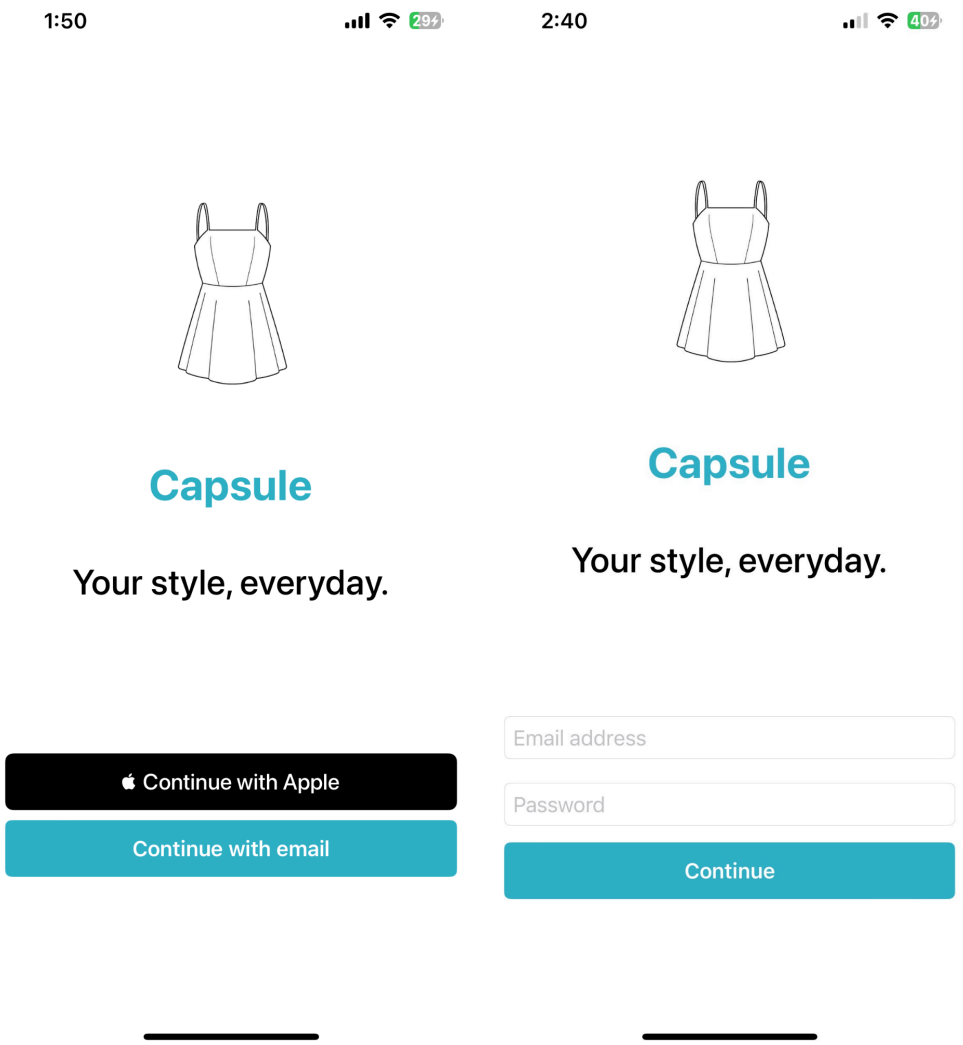

Figure 14: Landing and Login Page

The user has two options to log into the application. “Continue with Apple” button automatically performs the login or sign up action. “Continue with email” option allows users to enter their email and password. Their account is automatically created if it does not exist already. This design allows users to spend the minimum amount of time with the authentication process.

9.2 Profile and Detailed Analysis Page

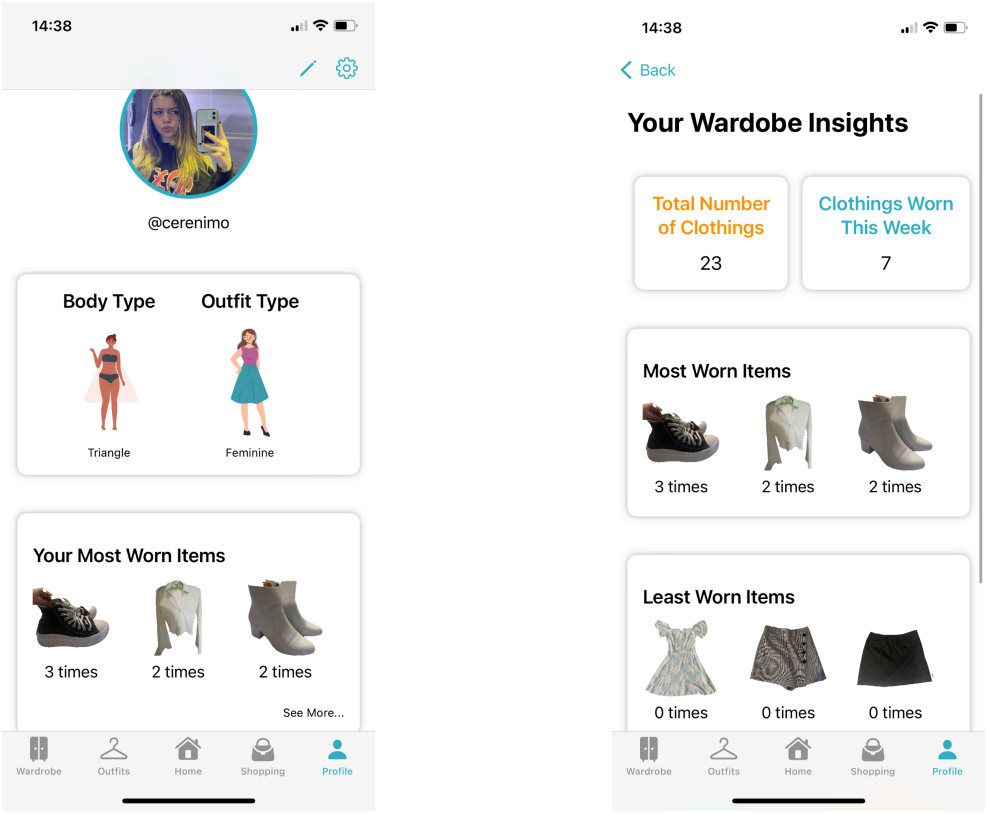

Figure 15: Profile and Detailed Analysis Pages

The user can view the profile information, body type, and outfit type on the profile page. The user can edit the profile information by pressing the edit button in the right top corner. The user can view the settings and KVKK requirements by pressing the settings button. The user can view the wardrobe insights by pressing on the see more button located at the bottom.

On the wardrobe insights, the user can view information like the total number of clothes owned, clothes worn this week, most worn items, least worn items, and outfit tag distribution.

9.3 Shopping Recommendation Page

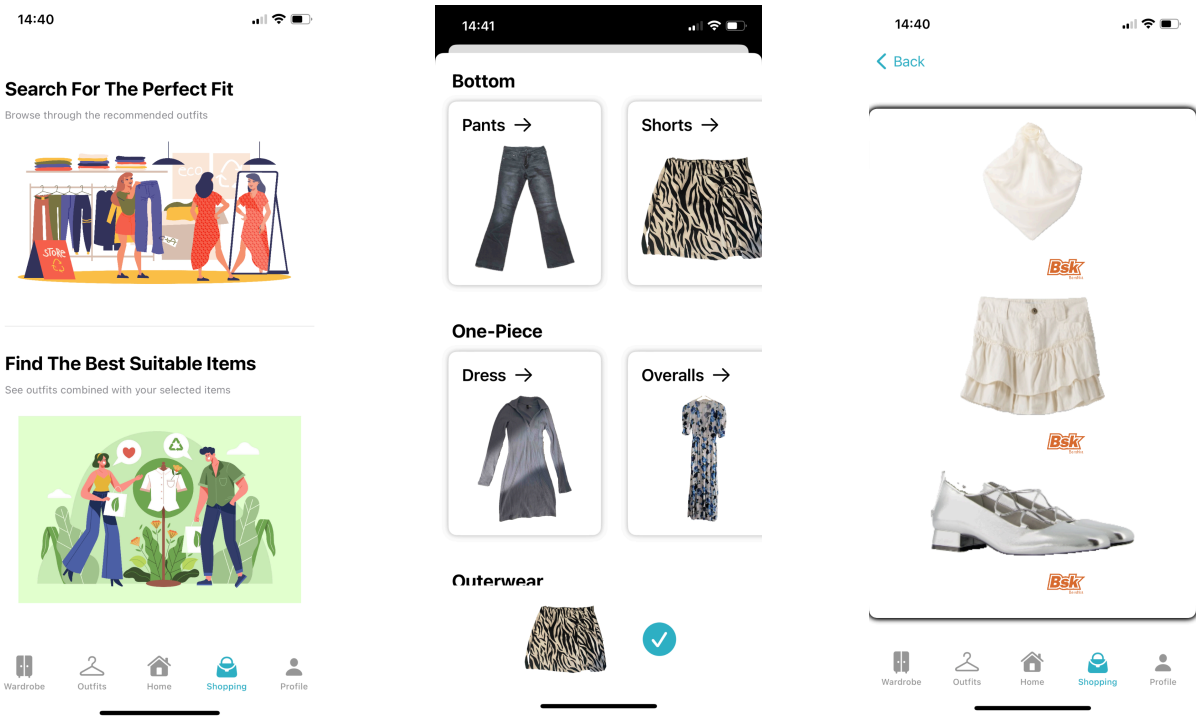

Figure 16: Shopping Recommendation Page Views

The user can reach this page by pressing on the shopping section on the toolbar. Then, the user can browse recommended outfits or select some items first. If the user chooses the first option, they can browse the outfits by swiping them to the left. If the user likes an item, they can press on the item to go to the sellers' page.
If the user decides to lock some items first, the user can select some items from their wardrobe. Then, by pressing the check symbol, they can browse the recommended outfits.

9.4 Home Page and Outfit Tracking

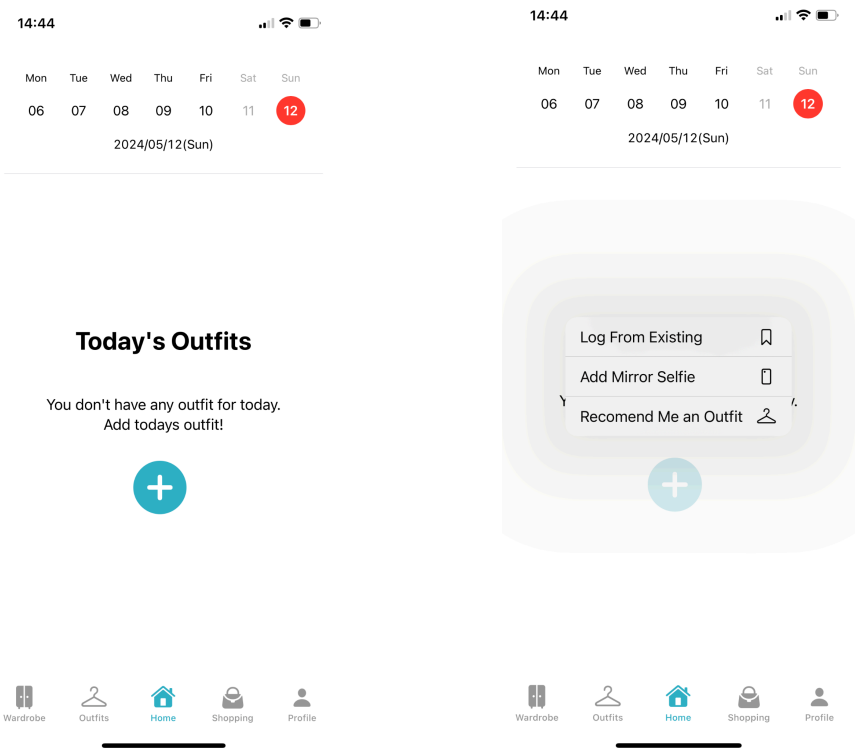

Figure 17: Home View

The user can reach the home page by pressing the home button on the toolbar menu. They can go to other days by pressing the dates on the calendar view. They can wipe the days to see other weeks. Users can add their outfits of the day by pressing the plus button. There are three options to add: “Log from existing,” “add mirror selfie,” and “recommend me an outfit.”

9.4.1 Log Outfit From Existing Outfits

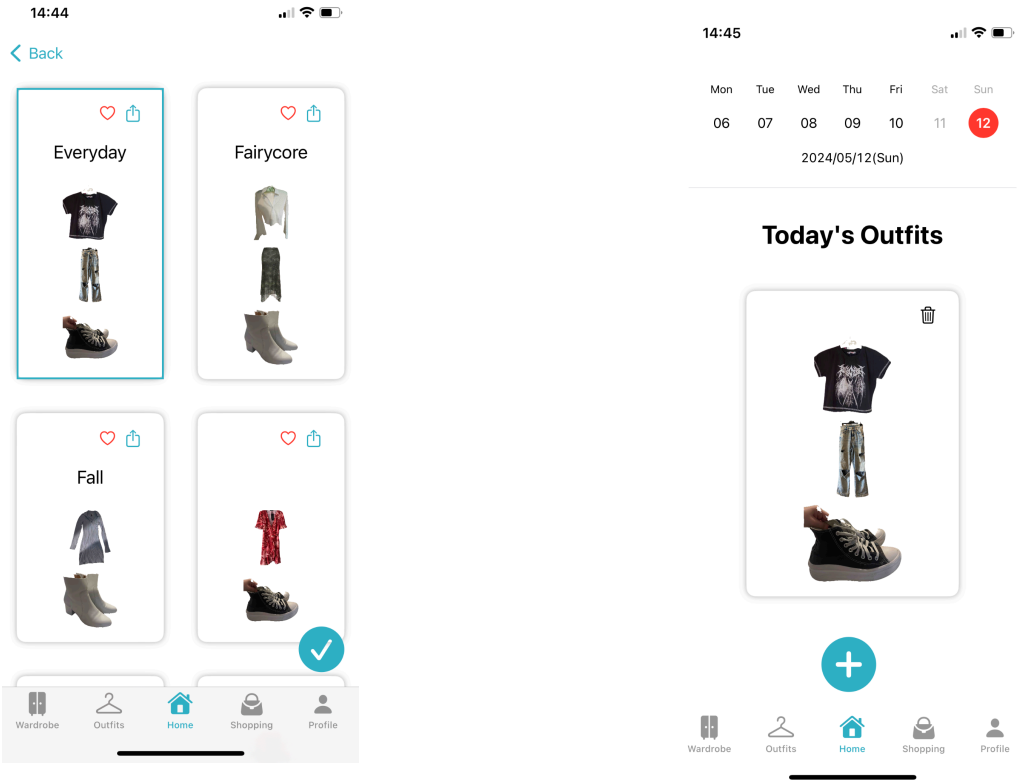

Figure 18: Outfit logging and existing outfits pages

The user can create new outfit logs using their outfits in their wardrobes. In order to enter the manual outfit logging page, click the plus icon on the start page, then select log from existing. This will show your outfits for you. Select which outfit you are wearing today and approve it with the check mark button. You have successfully logged your outfit!

9.4.2 Add Mirror Selfie

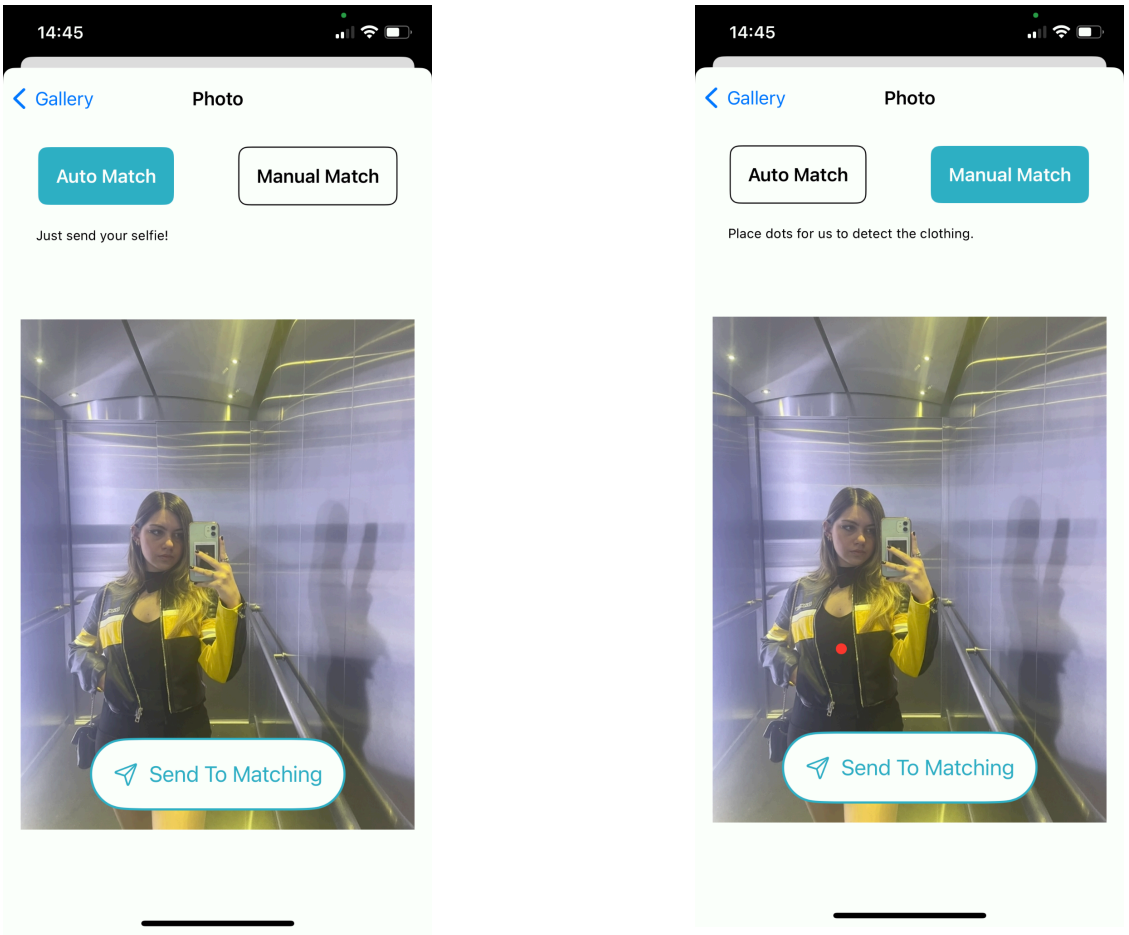

Figure 19: Auto and manual mirror selfie adding pages

Figure 20: Matching results page

On the home page, the user can press the plus button and select “Add mirror selfie” to upload an image of themselves from the device gallery or the camera. Upon image selection, the user can select “Auto Match” and click on “Send to Matching” to make the app automatically detect the clothes worn by the user in the image and match them with the clothes in the user’s wardrobe.

Alternatively, to direct the app in the right way, the user can select “Manual Match” instead, at which point the user can click on the clothing items on the image which they want to be logged. In this scenario, only the items selected by the user are extracted from the image and sent to the matching engine.

Behind the curtains, detected clothing items are segmented and compared with all other items in the user’s wardrobe to find the best matching items. After matching is completed, a list of matched items are displayed to get the user’s approval, at which point the user can simply click on the “Log Outfit” button to log the detected items as what the user is wearing on the selected day.

9.5 Wardrobe Page

9.5.1 View Wardrobe

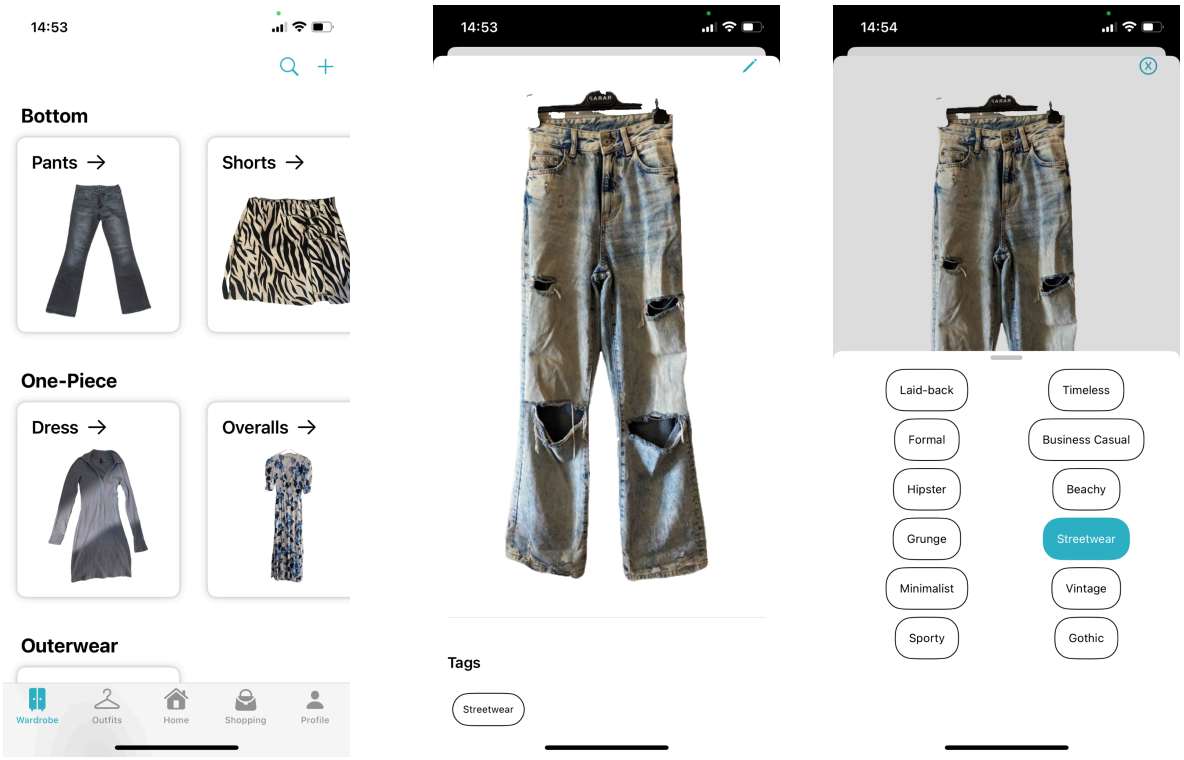

Figure 21: Wardrobe view

In the wardrobe tab, clothes in the user's wardrobe are shown under categories grouped by their subcategories. Users can view all the clothes in a subcategory by tapping on the respective subcategory's card. When the user taps on a clothing item, they can view and modify the tags of that clothing item.

9.5.2 Add New Item

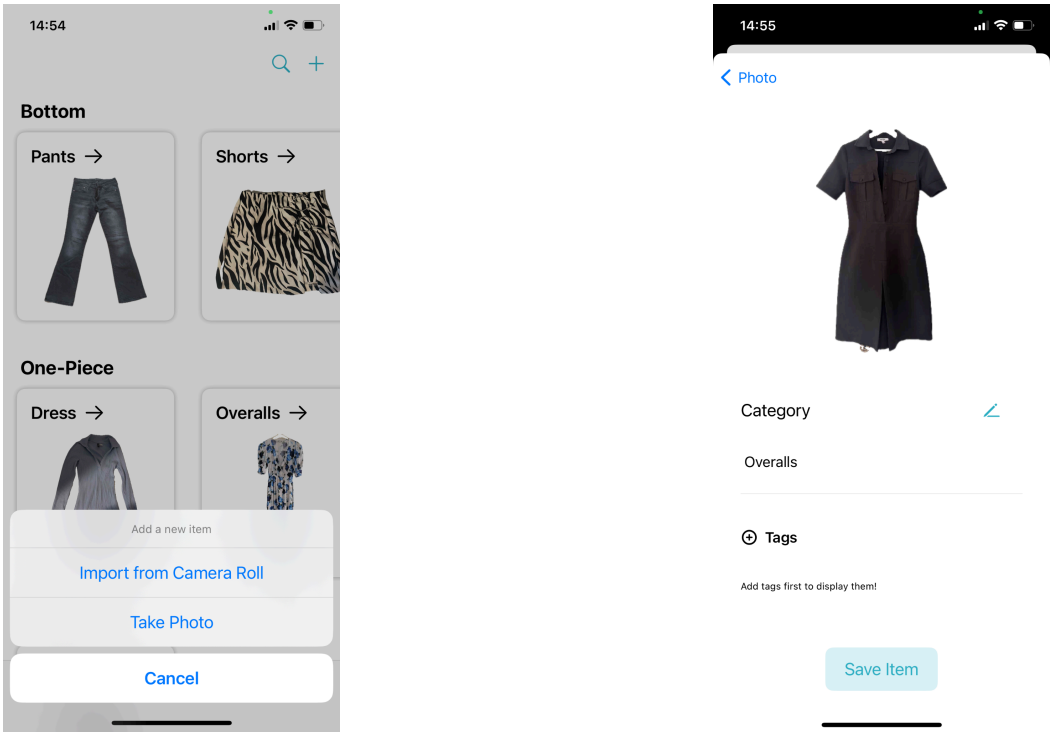

Figure 22: Outfits view and add new outfit page

To add a clothing item to the wardrobe, the user can navigate to the Wardrobe tab and click on the plus button on the right top corner of the screen. After taking a photo via camera or selecting one from the device gallery, selected image is processed to both segment the image, which removes the background and makes the cloth displayed beautifully on the UI, and extracts information, such as the category, subcategory, dominant color hex code and the color name.

After processing is completed, the user can change the detected subcategory if needed, add some tags, which are later used when recommending this new clothing item, and click on the “Save Item” button to add the item to the wardrobe. Upon completion, the item becomes available to use throughout the app, appearing in outfit logging, outfit recommendation, and wardrobe analytics, just to name a few.

9.6 Outfits Page

9.6.1 View Outfits

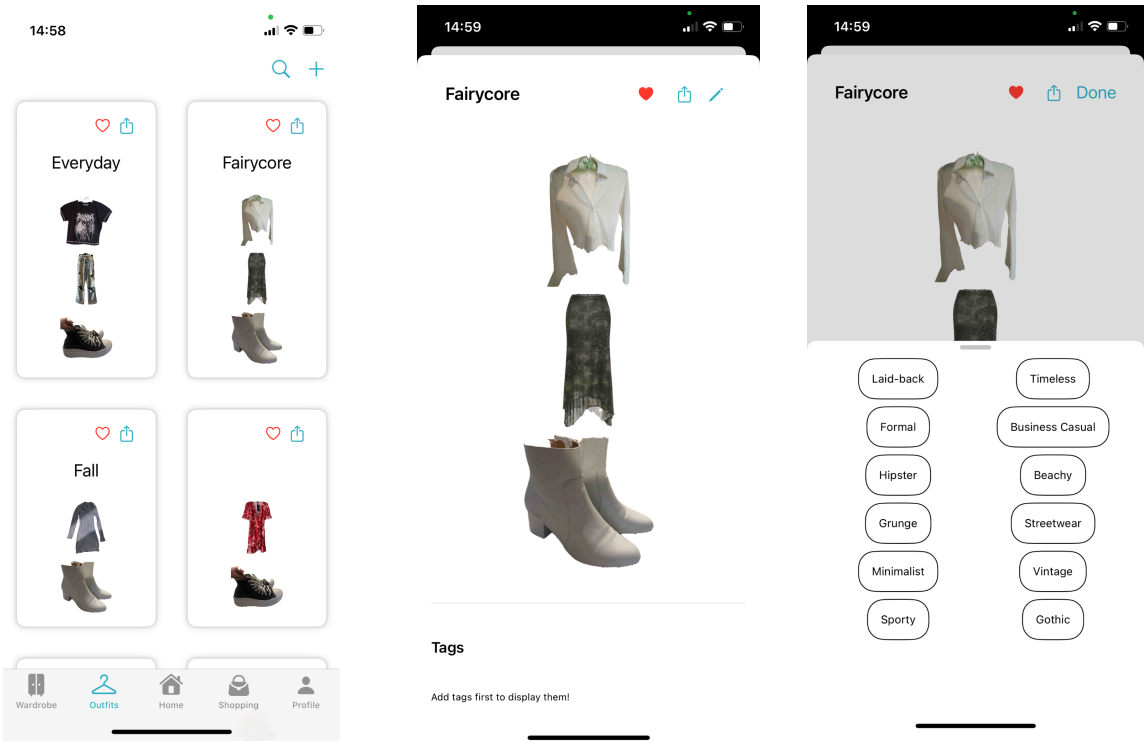

Figure 23: Outfits view and outfit viewing pages

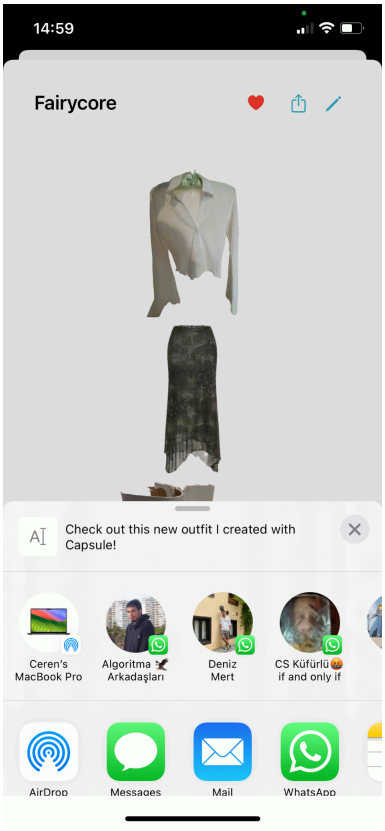

Figure 24: Outfit sharing page

In the Outfits tab, users can view their outfits in a card view by scrolling up and down. A detailed view is opened when the user taps on an outfit's card. The user can rename the outfit or edit its tags through this view. Also, the user can share the outfit with external apps by tapping the share icon on the top right of each card.

9.6.2 Add New Outfit Manually

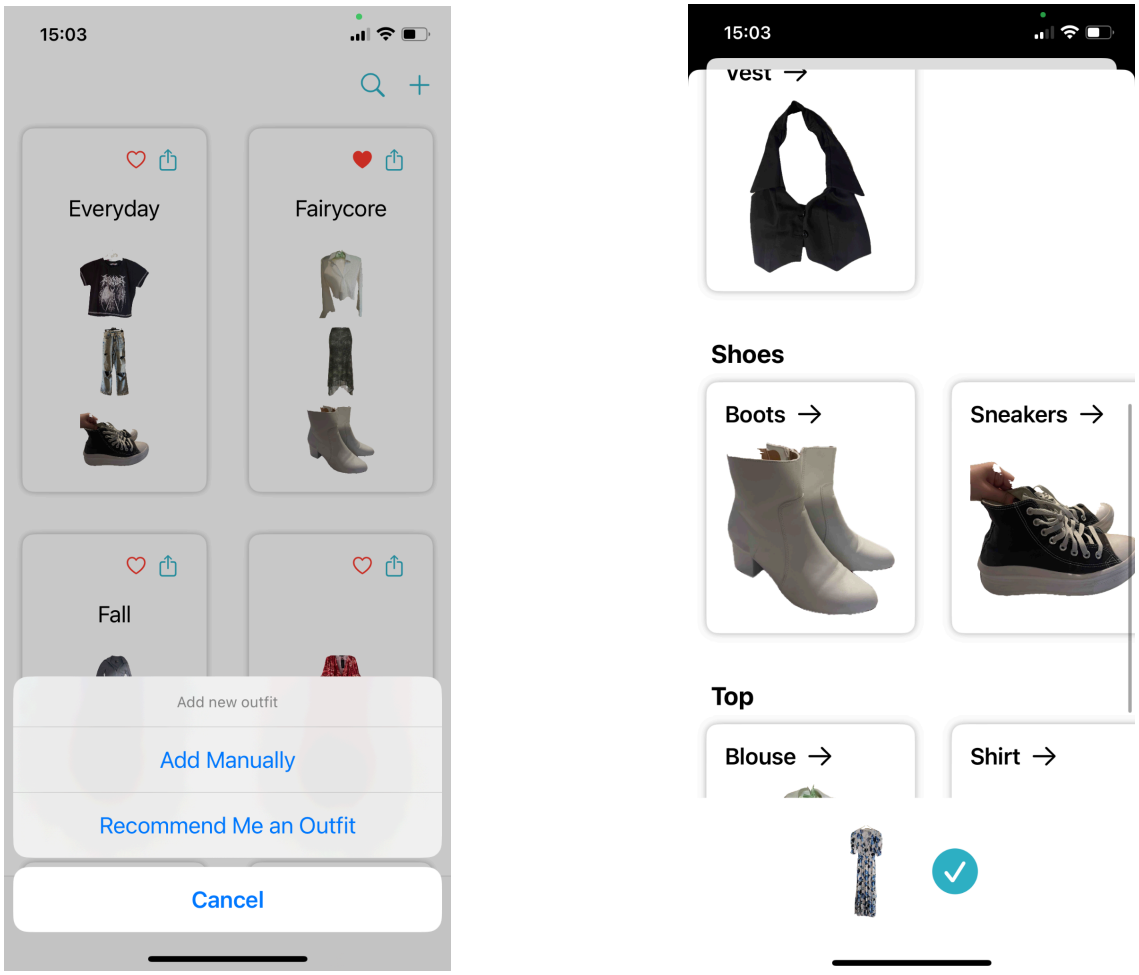

Figure 25: Manual outfit creation pages

When you have an outfit in mind and want to recreate it in Capsule, you just need to go to your outfits page and click the plus icon. A pop up will greet you. You should select “Add Manually” which will direct you to the manual outfit creation page. On this page, select the pieces of clothing you want to build your outfits with. After approving the items, give a name to your outfit and add tags to it.

9.6.3 Add New Outfit With Recommendation

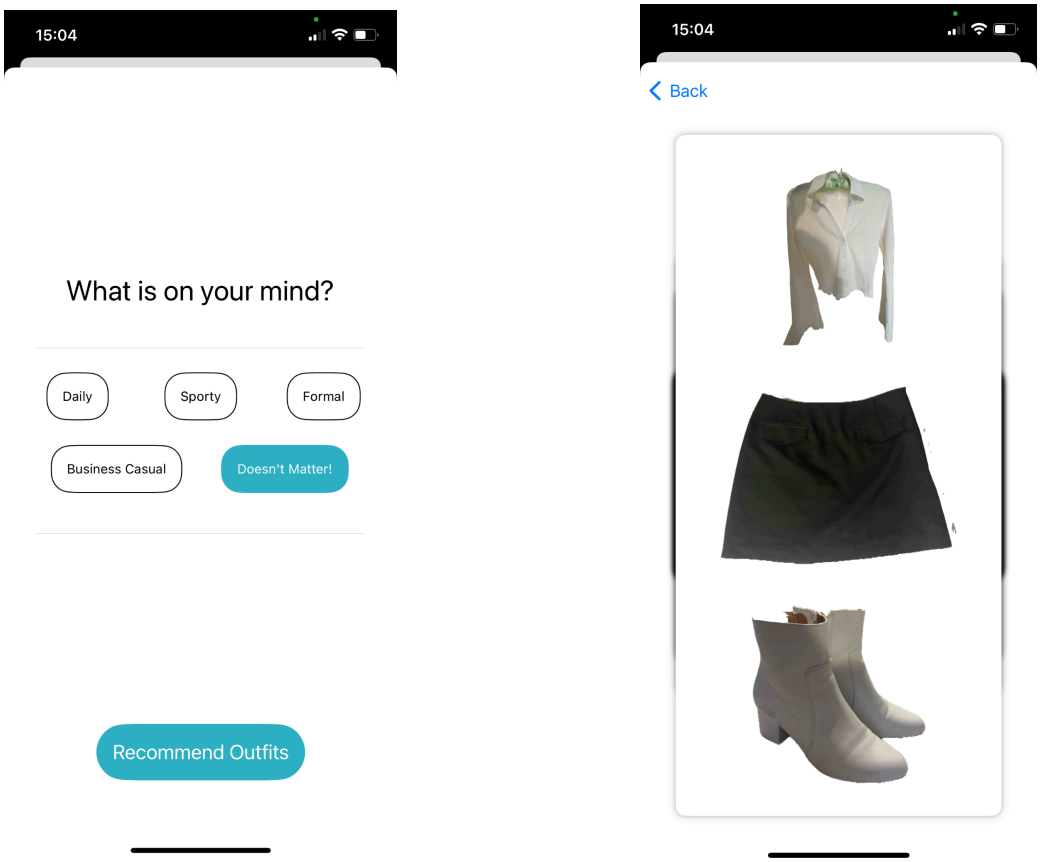

Figure 26: Outfit recommendation pages

In the recommendation page, the user has a choice to select between various occasions such as daily, sporty, formal, or business casual. Each choice corresponds to a text query which is used by the recommendation algorithm. When the user taps on the “Recommend Outfits” button, they are taken to a page where clothings best outfits are shown. The user then swipes right to like an outfit or left to dislike it. This data is collected to further improve the recommendation model.

Glossary

Capsule Wardrobe: A capsule wardrobe is a minimalist and versatile collection of essential clothing items designed to simplify and enhance one's personal style while promoting sustainable fashion. The goal is to create the maximum number of outfits with the minimum number of items.

GDPR: The EU General Data Protection Regulation

IEEE: The Institute of Electrical and Electronics Engineers

KVKK: Kişisel Verileri Koruma Kanunu

PEP8: Python Enhancement Proposal 8

PII: Personally Identifiable Information

References

[1] E. Newbery, “The Average American Spends This Much on Clothes Every Year,” the ascent, Jul. 26, 2022. [Online]. Available: https://www.fool.com/the-ascent/personal-finance/articles/the-average-american-spends-this-much-on-clothes-every-year/. [Accessed: Oct. 15, 2023].

[2] E. Penner, “The Ultimate Guide: How to Build a Capsule Wardrobe,” Modern Minimalism, 2023. [Online]. Available: https://modernminimalism.com/how-to-build-a-capsule-wardrobe/. [Accessed: Oct. 15, 2023].

[3] O. Tuna, “IOS MVC Mimarisi,” Medium, https://medium.com/mobil-dev/ios-mvc-mimarisi-bcda93d8c5a (accessed May 13, 2024).

[4] “Imaterialist (fashion) 2019 at FGVC6,” Kaggle, https://www.kaggle.com/c/imaterialist-fashion-2019-FGVC6/data?select=label_descriptions.json (accessed Mar. 14, 2024).

[5] Ternaus, “Ternaus/cloths_segmentation: Code for binary segmentation of cloths,” GitHub, https://github.com/ternaus/cloths_segmentation (accessed Mar. 14, 2024).

[6] Bearpaw, “Bearpaw/clothing-co-parsing: CCP dataset from "clothing co-parsing by Joint Image Segmentation and labeling " (CVPR 2014),” GitHub, https://github.com/bearpaw/clothing-co-parsing (accessed Mar. 14, 2024).

[7] A. Kirillov et al., ‘Segment Anything’, arXiv:2304. 02643, 2023.

[8] A. Radford et al., Learning Transferable Visual Models From Natural Language Supervision. 2021.

[9] X. Han, Z. Wu, Y.-G. Jiang, and L. S. Davis, “Learning Fashion Compatibility with Bidirectional LSTMs,” 2017.

[10] R. Sarkar et al., OutfitTransformer: Learning Outfit Representations for Fashion Recommendation. 2022.

[11] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, The Curious Case of Neural Text Degeneration. 2020.

[12] K.-H. Liu, T.-Y. Chen, and C.-S. Chen, "MVC: A Dataset for View-Invariant Clothing Retrieval and Attribute Prediction," in Proc. ACM Int. Conf. Multimedia Retrieval (ICMR), 2016.

[13] J. Mifsud, “Usability metrics - a guide to quantify the usability of any system,” Usability Geek, https://usabilitygeek.com/usability-metrics-a-guide-to-quantify-system-usability/ (accessed Mar. 14, 2024).