Cairo University
Faculty of Computers and Artificial intelligence
Department of Computer Sciences

Parfait Fit

Under Supervision of:

Dr. Cherry Ahmed

TA. Maya

Prepared by:

Alaa Mahmoud Ebrahim	20190105
Joseph Diaa Saied	20190155
Rana Ihab Ahmed	20190207
Mariam Amr Mohamed	20190520
Norhan Abdelkader Ali	20190600

Graduation Project
Academic Year 2022-2023
Final Documentation

Table of Contents

List of Illustrations

List Of Abbreviations

Abbreviation	Definition
AUC	Area under the curve metric
IDF	Inverse Document Frequency
MLP	Multi-layer Perceptron
NLP	Natural Language Processing
RN	Relational Network
TF	Term Frequency
VSE	Visual Semantic Embedding

Abstract

Fashion is an art form that allows individuals to express themselves through clothing and accessories. People are becoming more aware of fashion, and it is becoming a huge concern for them.

Fashion's increasing popularity resulted in an increase in the diversity of styles within the fashion industry. Therefore, with different styles existing, people are now more conscious of what they are wearing and seek out sustainable fashion alternatives.

They want to try different styles while looking nice and matching different clothes. However, this is a time-consuming daily occurring problem, wearing an outfit that is compatible, is not easy for some people. They would spend a lot of time finding a well-coordinated outfit or deciding whether an outfit is compatible or not.

Our purpose is to help these people construct a well ensembled outfit and decide whether an outfit is good or not. Parfait Fit would recommend clothes that are well coordinated with the given outfit by the user, and it will also give a score to the user's outfit that would represent the outfit compatibility.

In order to achieve our goal, we trained a relational network model that would output a score for a given outfit. We developed an application, using flutter and Django, that its main purpose to help people in wearing a well-coordinated outfit.

Chapter 1: Introduction

1.1. Motivation

With the extensive clothing options available from different brands, choosing the perfect outfit is challenging for most of us. Usually, people struggle to find pieces that go well together and complete their outfit.

Upon conducting a survey with people with ages ranging between 12 - 40+, We found that 71% of people face difficulties in picking an outfit that is compatible or finding a piece that matches an uncompleted outfit. Also, about 93% buy their clothes from different brands and about 95% of them would find it more convenient if they could use one application for all brands.

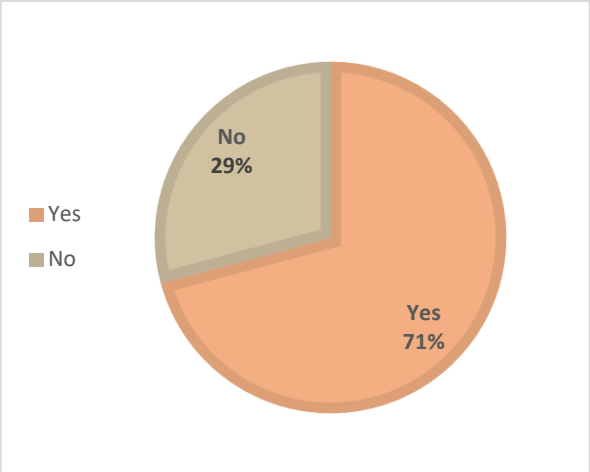


Figure 1-3 Difficulty in Picking an Outfit Chart.

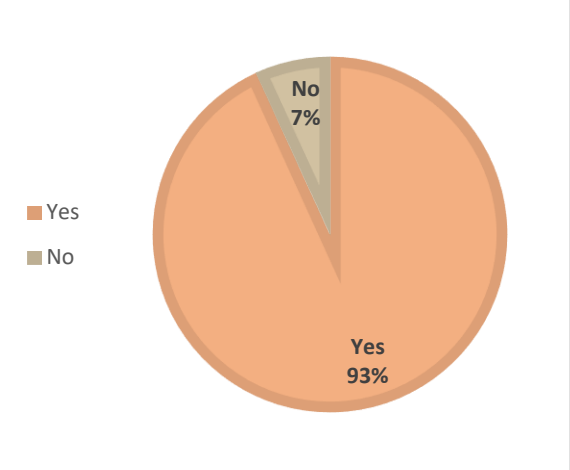


Figure 1-2 Users Buy from Different Brands Chart.

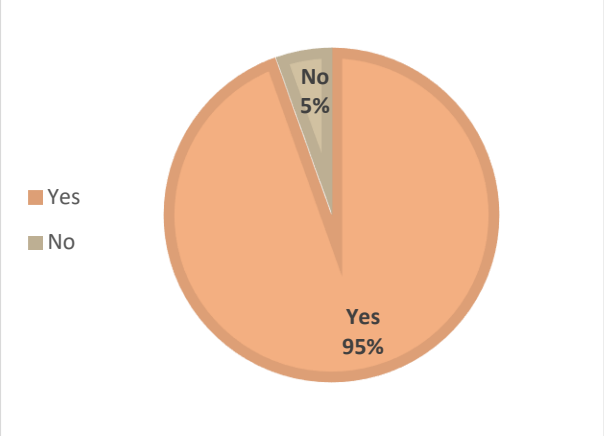


Figure 1-1 Finding Brand in on App Convenient Chart.

Our main goal is to enhance the overall fashion experience for all users and help customers with their outfit formulation process, providing outfit compatibility scoring for created outfits and a wide range of fashion choices where they get personalized recommendations and can search for pieces they want. Bringing together users and providers on a single platform.

1.2. Problem Definition

Based on the survey we conducted, we found that people find it challenging to decide what to wear and make sure that their outfit is compatible, while shopping online or offline, it is hard for people to determine if an outfit they want to buy is compatible or not, and search for certain pieces that will match and be compatible with an incomplete outfit they have.

These challenges stem from:

- **Lack of Outfit Compatibility Knowledge.**

  With the diverse clothing options and providers available, many users lack the necessary knowledge to determine if their outfits are compatible or not.

- **Time-consuming Selection Process.**

  Users go through the outfit selection process daily; They invest significant considering items whether items existing in their wardrobe or online.

- **Limited access to recommendation and outfit checking services.**

  Unfortunately, the available applications now are restricted to only checking the compatibility of the outfit or recommending clothes only from one brand. Also, these applications must be bought by brands in order to be used.

- **Diverse number of Providers.**

  With the great number of providers and brands available it’s hard for users to know where to search and find certain pieces they want.

  Also, providers struggle to reach and present their products to various users and potential customers.

So, we decided to build an application that will solve all these problems and gather all these features in one free application bringing together users and various providers.

1.3. Project Objective

The project aims to provide the users with an application that solves the challenges mentioned in the sections above by offering various features to customers and providers, saving the time and effort they had to exert daily.

Parfait fits offers:

- **Outfit Compatibility Scoring and Item Recommendation**

  Using the outfit compatibility model, we built using RN, Customers can check the compatibility score of the outfit they created through the application from different items from different brands. They can also get recommendations for items from a certain category that completes the outfit they created and is compatible with it.

- **Personalized Recommendations**

  Using the content-based recommendation model to offer customers product recommendations based on their preferences in their ‘for you’ page.

- **Various Clothing Choices**

  Since the application allows various and divers providers to present their products in one application, Customers will have access to a wide range clothing options to find items based on their preferences.

- **Search and Filtering**

  To enable users to find desired items quickly, we offer search and filtering functionalities allowing customers to find specific clothing items based on keywords, category, price, brand, and size they provide.

- **Platform for Providers**

  We offer providers a web application to reach and present their products to various users.

1.4. Project Time Plan

Table 1-1 Project Time Plan

Task	Task Title	Description	Start date	End date
1	Brainstorming	Decide the main idea of the project, collect and scrap data for the first idea. Many problems with the first idea led us to choose the current idea.	1-Aug-22	20-Oct-22
2	Search for resources	Find papers to help us with the project and understand the models	20-Oct-22	10-Nov-22
3	Search for related work	Find competitors and know the strengths and weaknesses of their features	20-Oct-22	10-Nov-22
4	Conduct Survey	Gather people's opinion about the idea from different age ranges, to know whether it will be helpful or not	20-Oct-22	4-Nov-22
5	Collect datasets	Find the needed datasets for the machine learning models	20-Oct-22	30-Nov-22
6	Study Technologies	CNN, Deep Learning, NLP, Flutter, Django REST	20-Oct-22	25-Feb-23
7	Diagrams	UML, ERD, Use case, and Sequence diagram	10-Nov-22	30-Jan-23
8	Data Cleaning	Drop unnecessary data and dealing with data with overlapping types	30-Nov-22	26-Feb-23
9	Prototype	Design the initial prototype for the application (web and mobile)	1-Dec-22	25-Feb-23
10	Mid-year documentation	Finish the document and show progress	20-Oct-22	1-Feb-23
11	Build Compatibility Classifier model	Implement the model which will take the outfit and return its compatibility score.	26-Feb-23	15-Apr-23
12	Recommend item compatible with a given outfit	Implement the feature which will return the top 50 items compatible with the given outfit	16-Apr-23	25-Apr-23
13	Build content-based recommendation model	Implement the personalized content-based recommendation	16-Apr-23	16-May-23
14	Front-end implementation	Finish the front-end of both web and mobile application	1-Feb-23	15-May-23
15	Back-end implementation	Finish the back-end with the rest of the features	26-Feb-23	20-Jun-23
16	Final documentation	Finish the final document required	20-Jun-23	7-Jul-23

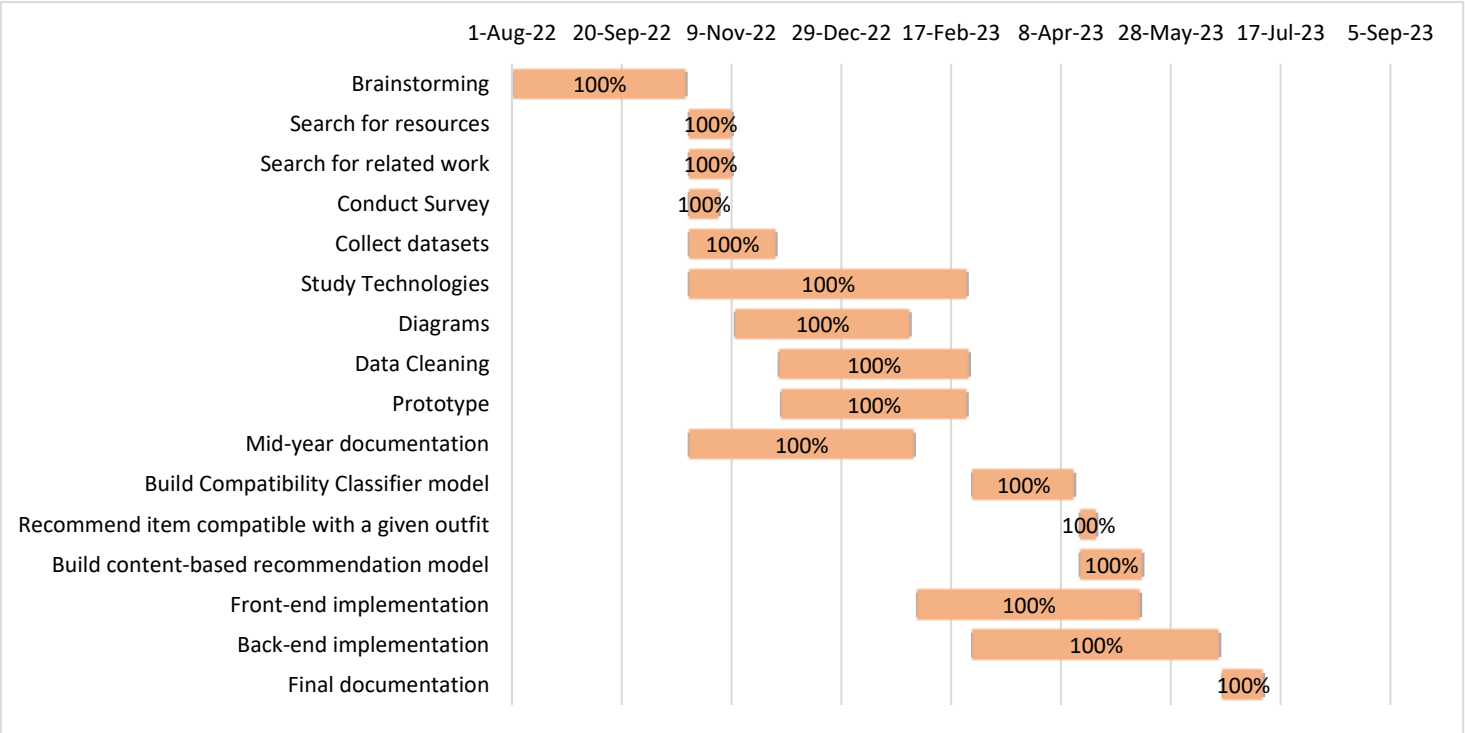


Figure 1-4 Gantt Chart

1.5. Project Development Methodology

For our project, we have chosen to follow the Iterative Waterfall methodology, which is an adaptation of the traditional waterfall methodology used in project management. It maintains the linear and sequential approach but incorporates modifications to make it suitable for practical software development.

A visual representation of the Iterative Waterfall methodology is shown in **Figure 1-5,** highlighting its sequential nature with iterative part in project management.

In this methodology, each phase must be fully completed before moving on to the next. However, it allows making necessary changes if errors are identified during later phases unlike classis waterfall methodology.

While the Iterative Waterfall approach provides benefits in terms of project understanding and management, especially when dealing with well-defined requirements, it is important to consider its potential drawbacks, such as difficulty to incorporate change requests and lack of support for incremental delivery.

The Iterative Waterfall methodology assures that each phase is completed before progressing, reduces the risk of overlooking requirements. It also provides feedback paths to the preceding phases to adjust any changes and to make the development more efficient.

Also, with clear requirements, accurate timelines and resource allocation can be achieved.

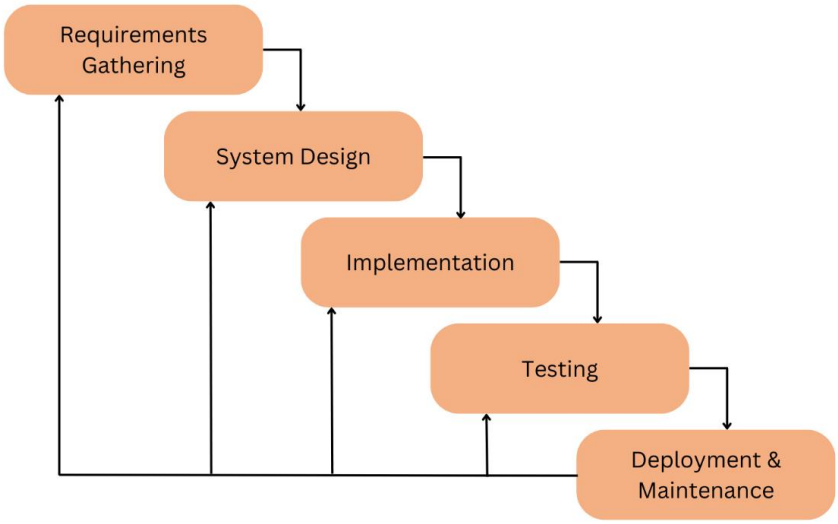


Figure 1-5 Iterative Waterfall Methodology

It consists of five phases: requirements gathering, design, implementation, testing and deployment. The next points illustrate the steps that we did in each phase.

1. Requirements Gathering:

- Starting with identifying the project requirements, objectives, and scope.
- Conduct a survey with potential users to gather their requirements and expectations and find out if the objectives of the project would be useful for them.
- Finally, define the functional and non-functional requirements of our project.

2. System Design:

- Starting with creating a detailed design for our system, including the System Architecture, Class Diagram, Use Case Diagram, ERD for the database, and Sequence Diagrams.
- Define the software components and their interactions.
- Finally, plan which techniques and technologies will be used to develop this system such as the main frameworks.

3. Implementation:

- In this phase, we start to code and implement the system, the models and implementation details we made will be discussed in chapter 5.

4. Testing:

- In this phase, we test the system by writing test cases that verify that the system is working correctly without any problem.

5. Deployment & Maintenance:

- Finally, in this phase the deployment is when the system will be launched in the respective environment to be used, and the maintenance is to address any reported issues or bugs in the system and if the system requires any code change.

1.6. The used Tools in the Project

1.6.1. Software Tools

1. Kaggle Notebook

   To ensure smooth training and prevent system crashes, during training, due to limited local resources, we used the cloud-based platform Kaggle Notebook.

   This platform provides high-performance hardware resources, including powerful CPUs, GPUs, and RAM, enabling us to train and run our deep learning model, Compatibility Classifier model, without any disruptions.

   The computational capabilities of this platform allow us to do more experiments and come up with better results, overcoming the limitations of our local hardware.

2. Flutter

   To satisfy the needs of different user groups (customers, admins, and providers), we used Flutter, powered by the Dart programming language, to build both mobile and web user interfaces (UIs)

   The mobile app for customers offering a personalized and optimized interface on their smartphones or tablets. On the other hand, the web interface was designed to satisfy admins and providers, providing them with a responsive and feature-rich experience accessible from their desktops or laptops.

   By making use of Flutter's flexibility, we successfully implemented the user interfaces, meeting the diverse needs of our users across both mobile and web platforms.

3. Django Rest Framework

   To develop a robust and efficient back-end with a database, we used Django Rest Framework (DRF), a robust framework for building RESTful APIs in Python which provided us a comprehensive toolkit that facilitated the development of efficient and scalable back-end services.

By the power of DRF's features, we implemented a secure API, enabling easy communication between the front-end and back-end components of our application.

With Its built-in serialization, authentication, and authorization capabilities, we ensured data integrity, user authentication, and proper access control. Also, we created a reliable and powerful back-end foundation, supporting the smooth flow of data and delivering a high-quality user experience.

4. PyTorch

   To develop our deep learning model, the compatibility classifier, and handle data loading efficiently, we used PyTorch, an open-source library known for its flexibility and performance in the field of machine learning, in addition to its built-in support for GPU acceleration significantly increased the model's training speed, enhancing overall performance.

   It has powerful data loading utilities, such as DataLoader and Dataset, the Dataset class allowed us to smoothly manage and organize our data, simplifying the data handling process, and be able to encapsulate our dataset, apply necessary transformations, and easily access individual data samples.

   With DataLoader, we efficiently preprocessed and batched our data, ensuring smooth and optimized training of the model which allowed us to handle our large dataset easily, enhancing the overall efficiency of the training process.

5. Redis

   To optimize performance and enhance the scalability of our application, we used Redis as a cache. It is a powerful in-memory data store that allowed us to store frequently accessed data in a fast and easily retrievable manner.

6. Postman for API

   To collaborate effectively on API development, we used Postman. Its rich features enabled us not only to test the API through a user-friendly interface but also to monitor its performance and availability.

7. Django Test

   To automate our testing process and ensure application quality, we used Python's Django Test framework.

8. PostgreSQL

   To ensure reliability and scalability, we used PostgreSQL which is a robust and feature-rich solution for data storage and management.

Its reliability, scalability, and advanced features enabled us to handle increasing data loads, concurrent user requests, and large volumes of data easily.

9. GitHub

   It offers efficient version control and collaborative development, allowing us to track changes, manage code revisions, and boost collaboration.

10. Docker

    It simplified our project by containerizing the back-end, front-end, database, and search engine (Meilisearch). With Docker, we packaged these components into portable containers with all dependencies, ensuring consistent deployments.

    Its scalability, isolation, and security benefits enhanced our application's performance and management. Docker Compose allowed us to manage multi-container applications without effort, improving development and deployment.

11. Meilisearch Engine

    To ensure accurate results and benefit from advanced indexing techniques, we used Meilisearch **[4]** as our search engine. With its fast search capabilities and customizable settings, it delivers precise search results.

    Its real-time indexing keeps search results up to date, ensuring a better search experience. Also, its easy integration with our backend and frontend systems simplified the implementation process.

12. VS Code

    With the ability to smoothly switch between languages and frameworks, a powerful integrated terminal, intelligent code completion, and easy integration with version control systems, VS Code provided us with a unified and efficient environment for coding and collaborating on both ends of our project.

13. Figma

    It helped us to design the UI for our web and mobile application. Its collaborative platform and reach features allowed real-time teamwork and consistent design, and the interactive prototyping ensured a smooth user experience.

14. Canva

    It helped us to create clear illustrations easily with its user-friendly interface, extensive design resources, and collaborative features.

15. Visual Paradigm

    It was essential in creating UML diagrams, including class, sequence, and use case diagram, as well as ERD. Its interface and collaborative features facilitated the visualization of our system structure.

16. Microsoft Excel

    It helps us to create a template for bulk upload products, charts, and more. The charting features facilitated Gantt chart creation for project management and progress tracking.

1.7. Report Organization

After providing an introduction about our project and its objectives. In the upcoming chapters, we are going to go over the details of the project's characteristics, features, design, and analysis.

The organization as follows:

- **Chapter 2: Related work:** The services and applications related to our project. We will discuss their features and the main differences between them and our project.
- **Chapter 3: System Analysis:** Describes the specification of the project, stating both the functional and non-functional requirements and the use case diagram.
- **Chapter 4: System Design:** Focuses on the design of the system shown by different diagrams, including component diagram, class diagram, sequence diagrams, ERD, and the GUI design of our system.
- **Chapter 5: Approach:** This chapter explains the models and approaches used to implement our functionalities. It will go in detail about relational network model used for outfit compatibility classifier and content-based filtering that is used for personalized recommendation stating how they are implemented and why we choose those two.
- **Chapter 6: Implementation and Testing:** Contains the details of the implementation with applied test cases to test our system.

By following this organizational structure, we aim to provide a systematic and comprehensive exploration of the project from various perspectives.

This organization ensures that readers can easily follow the logical flow of information, gaining a thorough understanding of the project's development and analysis process.

Chapter 2: Related work

In the domain of fashion and styling there are systems and applications that classify the compatibility of outfits.

In this chapter, we will discuss three of these systems, defining their features and comparing them with our own system, Parfait Fit.

The purpose of this chapter is to provide a snapshot of the selected systems, highlighting their unique characteristics and assessing how they compare to our own system.

By conducting this analysis, we aim to showcase the advantages of Parfait Fit, distinguishing it from the other existing outfit compatibility classification systems.

1. Wideesyes.ai **[9]**

Has multiple features including search by image, a recommender system based on similarity of the liking of the user, auto-tagging, and a style advisor which provides items that fit a certain product and provides the recommended look from a certain brand. This website requires requesting a demo to use.

2. Lookastic **[6]**

Recommends certain items that match a certain chosen item and recommends full outfits from pre-existing images of outfits that include that chosen item.

3. Vue.ai **[7]**

A website for vendors, has multiple features including product tagging and image moderation to ensure that the vendor images follow the guidelines, outfit recommendations based on certain themes and allow brands to add their own theme, advice on how to style certain items and an online fitting room that includes real-time styling by creating real realistic images of products on models. This website requires requesting a demo to use.

- The difference between our system and theirs:
    - Most of these websites are paid and not affordable for users.
    - Our system is not designed for certain brands to use, it allows multiple brands to upload their clothes. It allows users to explore more brands and find items that are similar to what they like over multiple brands providing personalized recommendation, considering the user's preference.
    - Customers can form their own outfit from different brands on the system; our system will help the user know whether their outfit is compatible or not.

- Customers can upload/take images of their own outfit from their wardrobe and find out whether the outfit is compatible or not through our system.

- Customers can upload/take image/s of item/s to form their outfit and specify the category of the wanted item to be compatible with their outfit, then the system will recommend items (with the same specified category) that go with their outfit and recommend items that is compatible with their outfit. (Recommend items compatible with a given outfit)

- Systems such as wideesyes.ai and vue.ai have product tagging feature which detects fashion-related tags from the images.

- Lookastic recommends from pre-existing images and vue.ai recommends based on pre-existing themes and added themes over time, while our system runs a compatibility algorithm to detect which items will be compatible with other items.

- To summarize the similarity and differences between the features of our system and the others

Table 2-1 Related Work Summary

Features	Our System	Wideesyes.ai	Vue.ai	Lookastic
Used by multiple brands/shops	✓	✗	✗	✗
Search by Image	✗	✓	✗	✗
Recommendation based on user's interests	✓	✓	✓	✓
Product tagging	✗	✓	✓	✗
User can form their own outfit of different items from different brands	✓	✗	✗	✗
Recommends items based on chosen product	✓	✓	✓	✓
Determine outfit compatibility for user	✓	✗	✗	✗
Online fitting room	✗	✗	✓	✗
Allow users (brands) to modify	✓	✗	✓	✗
Shoppers can upload their own items/outfits to get advice/recommendation	✓	✗	✗	✗
Free	✓	✗	✗	✓

Chapter 3: System Analysis

3.1. Project specification

3.1.1. Functional Requirements

- **For Users/Customers**
  1. Sign up for the system by email.
  2. Sign-in to the system by password and email.
  3. Edit his/her information and set a new password.
  4. View all products/clothes.
  5. View the details of a certain product.
  6. Upload an image of a clothing piece to add to an outfit.
  7. Create an outfit from items from different brands in the application gallery.
  8. Determine if a given outfit is compatible or not.
     - Return a score of the compatibility test.
  9. Get recommendation of an item compatible with a given outfit.
  10. View most liked products.
  11. Get personalized recommendation through their ”for you” page.
  12. Search by words for clothes.
  13. Filter the clothes by category, price, brand, and size.
  14. Have a favorite list and add clothes to it. (Add by pressing the heart icon)

- **For Providers**
  1. Sign up as a provider.
  2. Sign in.
  3. Add products. (Bulk Upload using excel sheet)
  4. Remove products.
  5. View all their products.
  6. View their status (Approved/Pending)
  7. Filter their products by category, price, and size.
  8. Search by words for clothes. (Within their products)

- **For Admin**
  1. Approve, suspend, or reject a provider.
  2. View provider.

3.1.2. Non-Functional Requirements

1. **Usability**
   Easy to navigate and use, clear features, and simple.
   (Make an instruction page for the user to understand how to use the application)

2. **Performance**
   Try to ensure that the response time does not exceed 5 seconds using Redis (cache) which will guarantee faster responses.
3. **Security**
   All users' information is secured by Django REST framework.
4. **Extensibility**
   Flexible to add and develop more features to keep up with the current users' needs or to fix a specific problem.
5. **Scalability**
   To accommodate the large number of users across web and mobile platforms. We will utilize Redis as a cache database to optimize response times and reduce strain on the primary data storage PostgreSQL.

3.2. Use-case Diagram

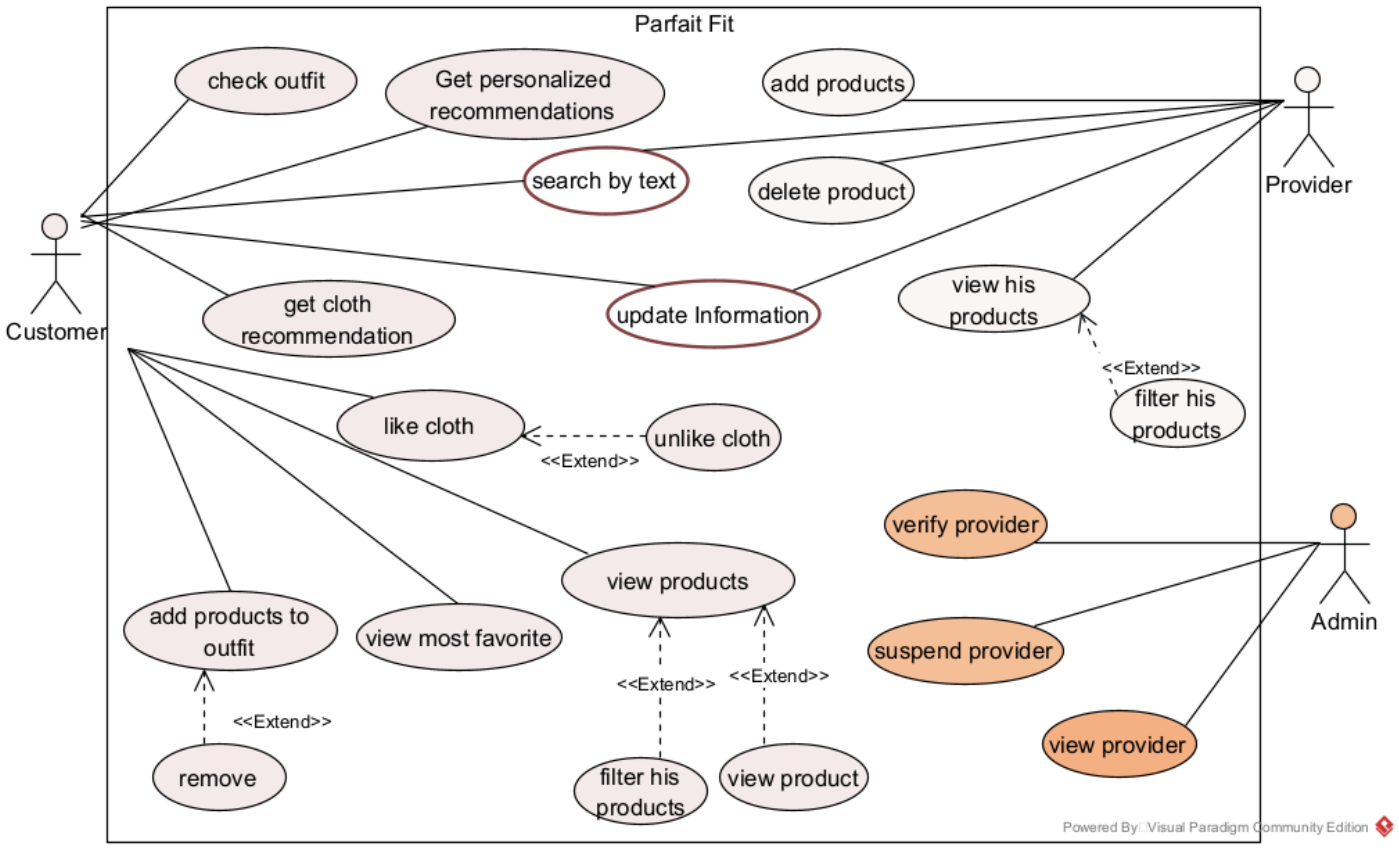


Figure 3-1 Use Case Diagram

Chapter 4: System Design

4.1. System Architecture

We will use layered architecture where each layer contacts the layer below it.

- ➔ Presentation layer: Responsible for the user interaction with the system.
- ➔ Business Layer: Handles requested coming from presentation layer and perform logic on them and fetch data from the data layer.
- ➔ Data Layer: Responsible for storing the data and connecting to the back end and retrieving the data.

  Consists of local database and Redis database responsible for caching.

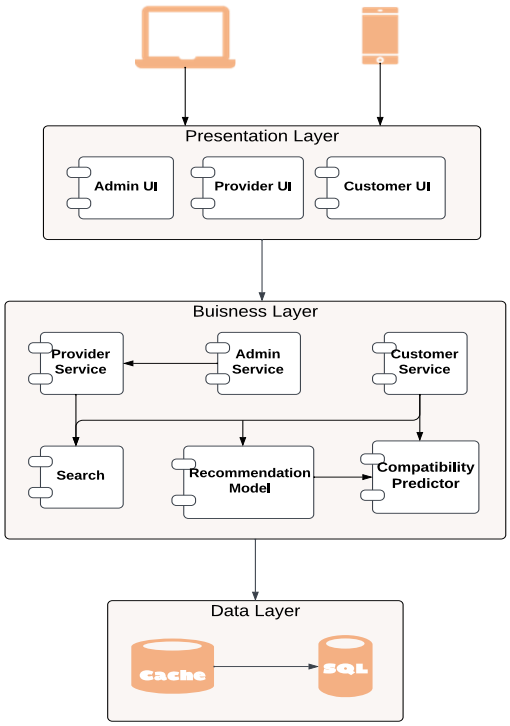


Figure 4-1 System Architecture

4.2. System Component Diagram

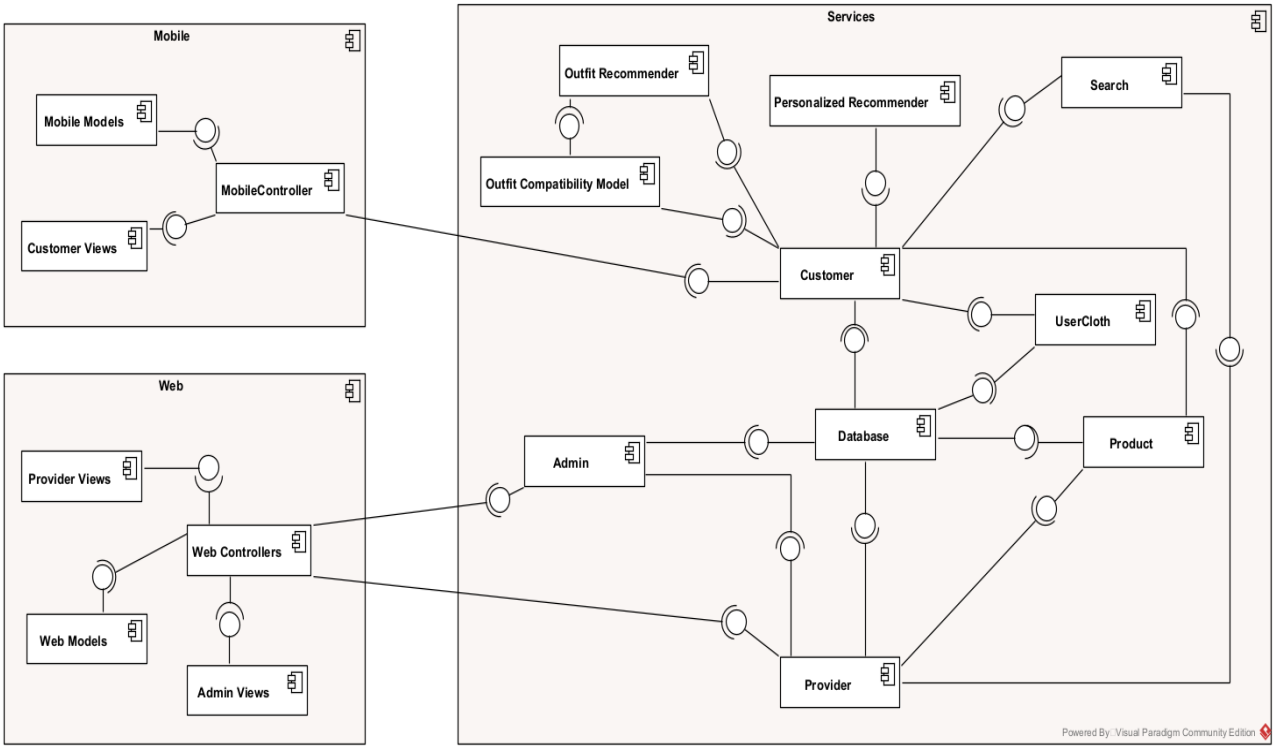


Figure 4-2 Component Diagram

4.3. System Class Diagram

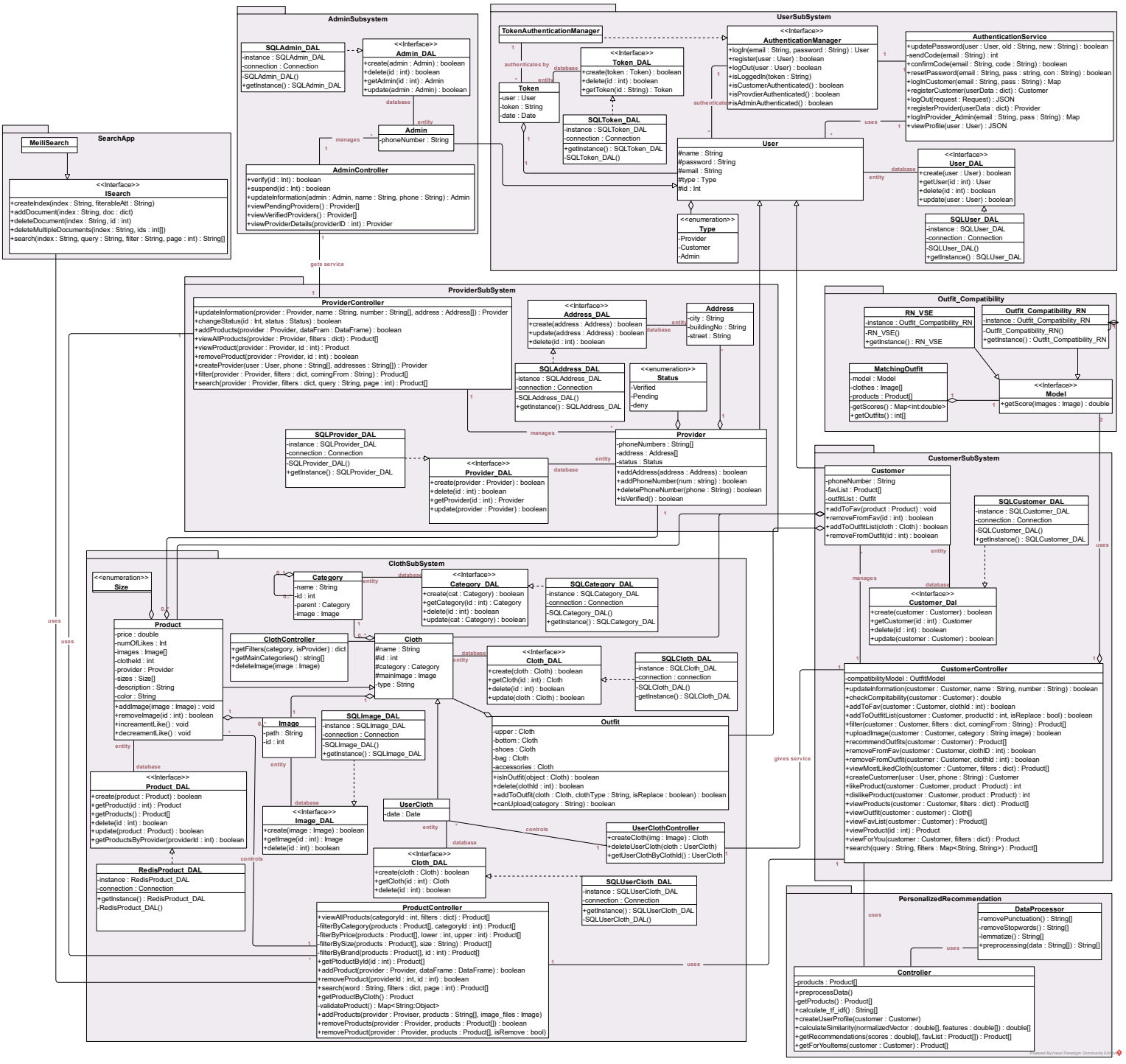


Figure 4-3 Class Diagram

4.4. Sequence Diagrams

1. Get Outfit Recommendation

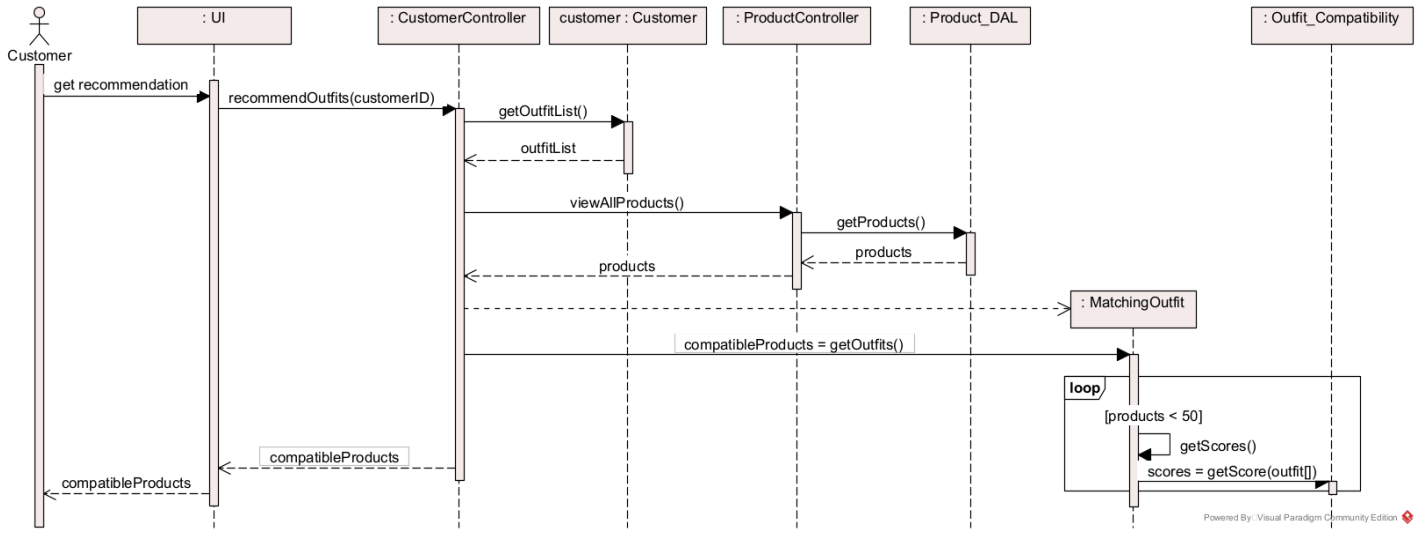


Figure 4-4 Get Outfit Recommendation Sequence Diagram

2. Check Compatibility

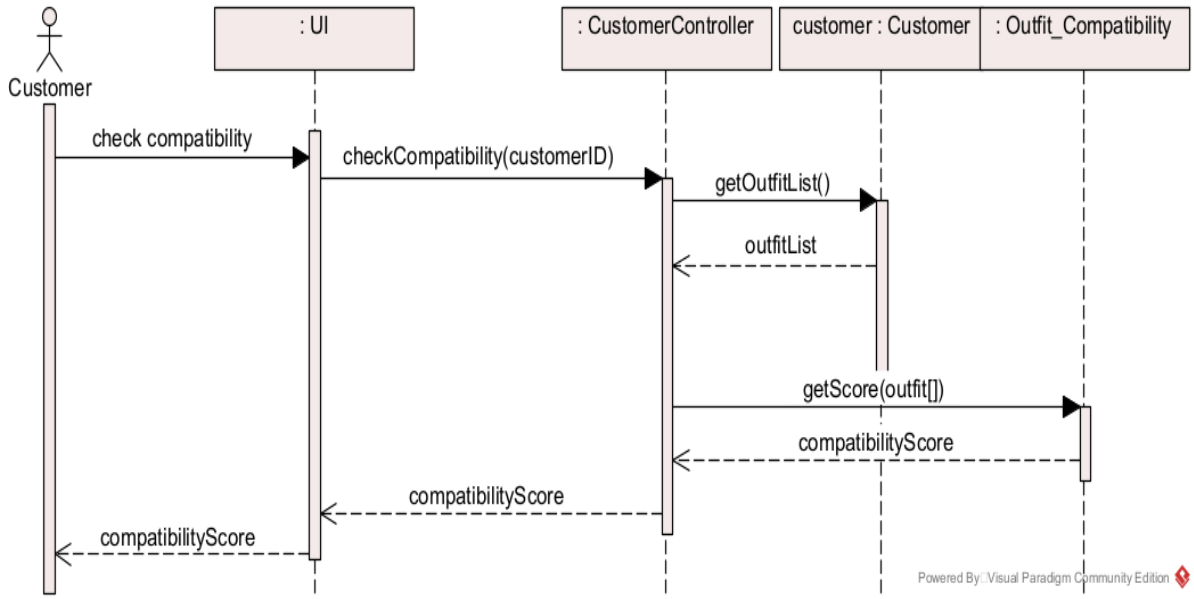


Figure 4-5 Check Compatibility Sequence Diagram

3. Add Products

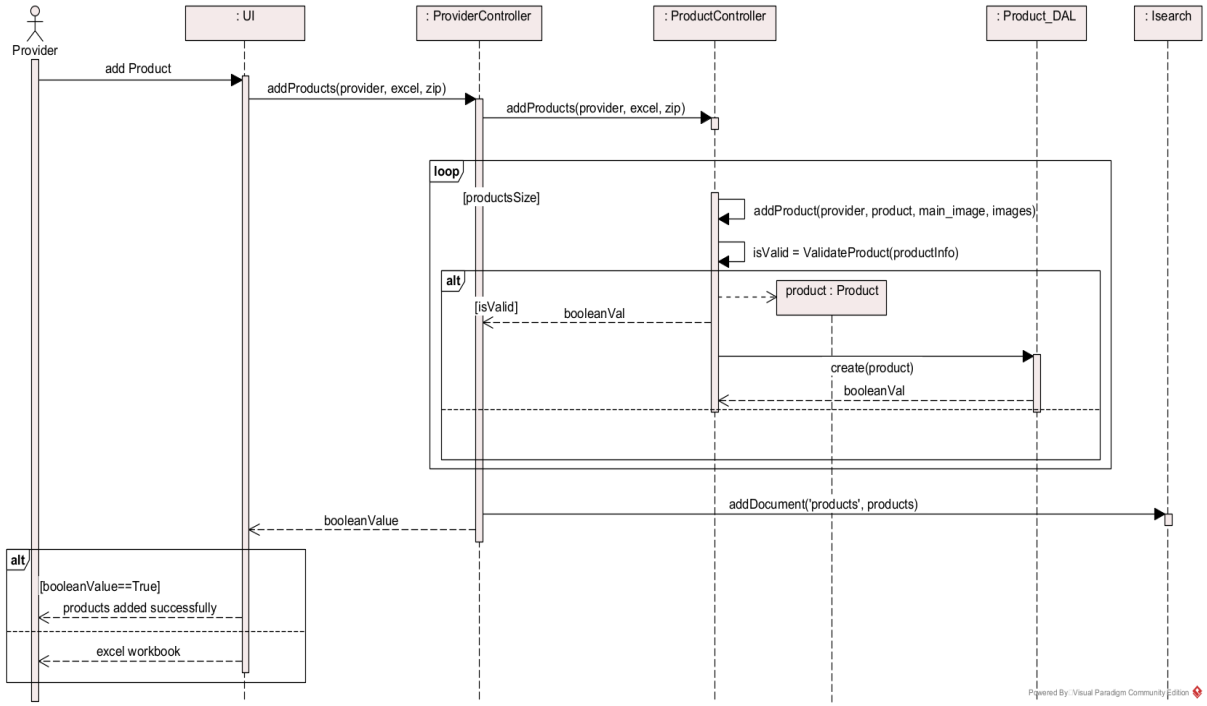


Figure 4-6 Add Products sequence Diagram.

4. Verify Provider

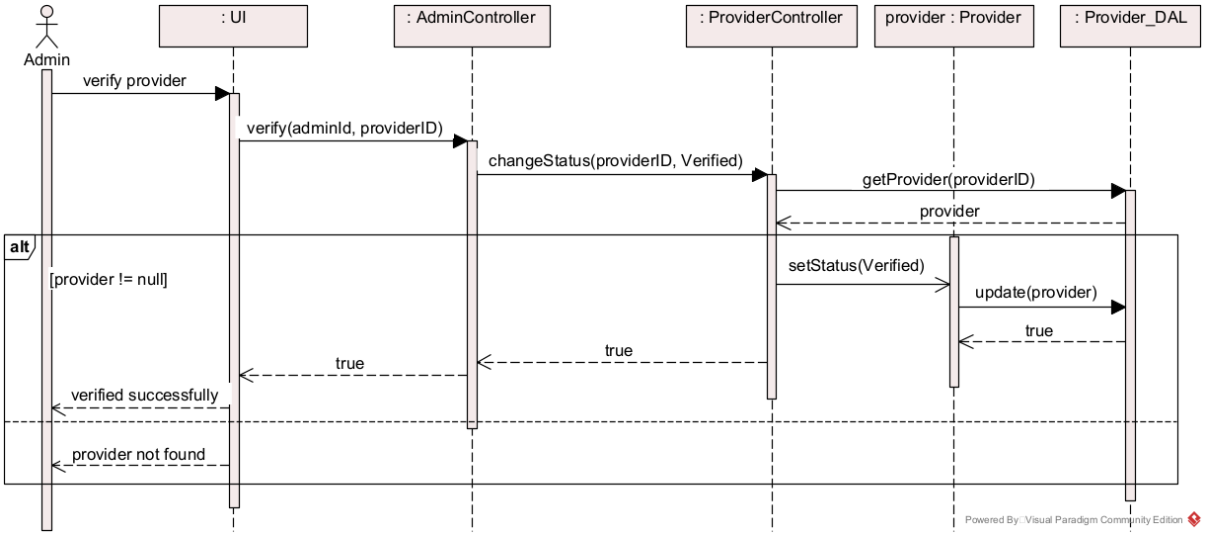


Figure 4-7 Verify Provider Sequence Diagram

5. Filter

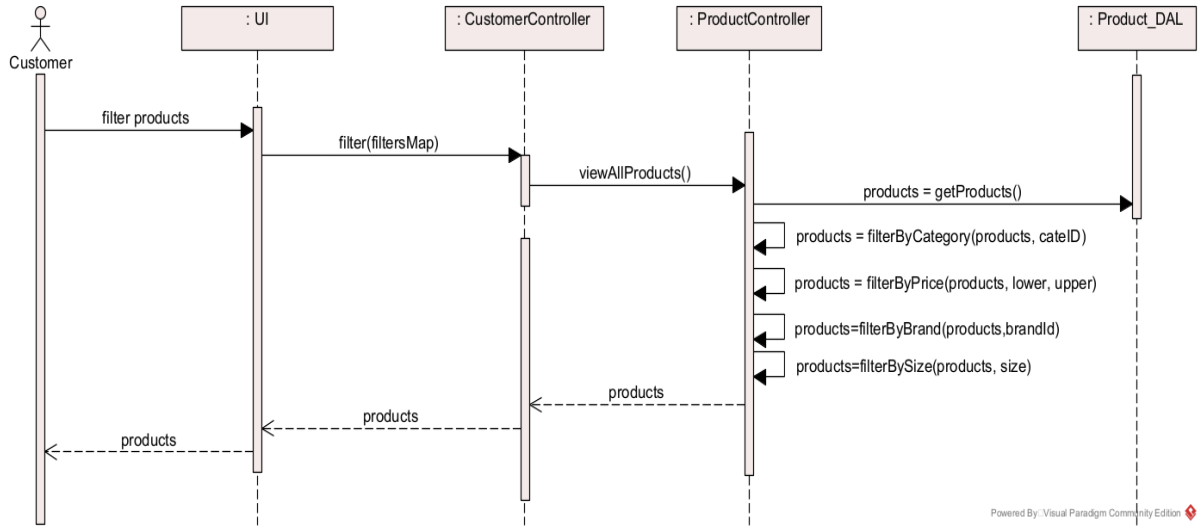


Figure 4-8 Filter Sequence Diagram

6. Remove From Outfit

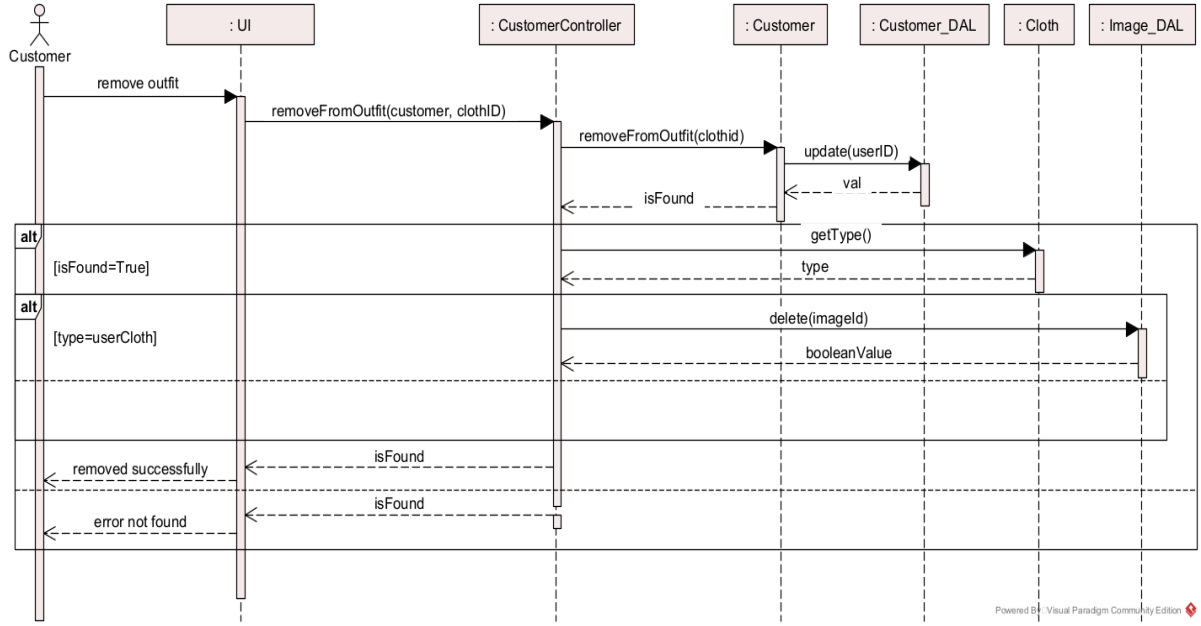


Figure 4-9 Remove from Outfit Sequence Diagram

7. Add to Favorite

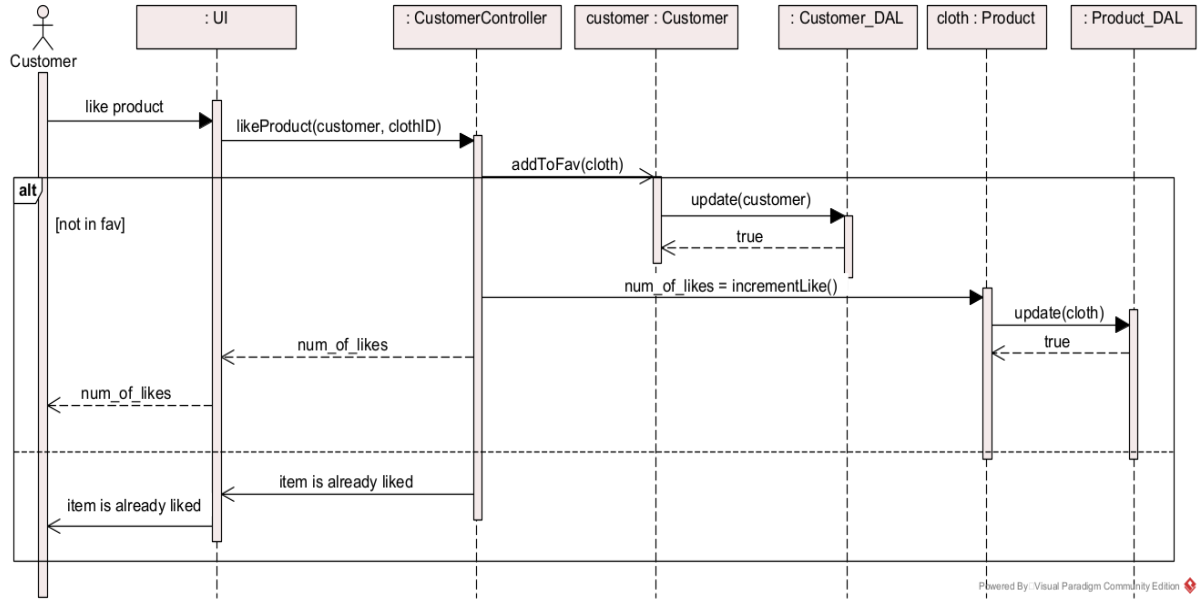


Figure 4-10 Add to Favorite Sequence Diagram

4.5. ERD

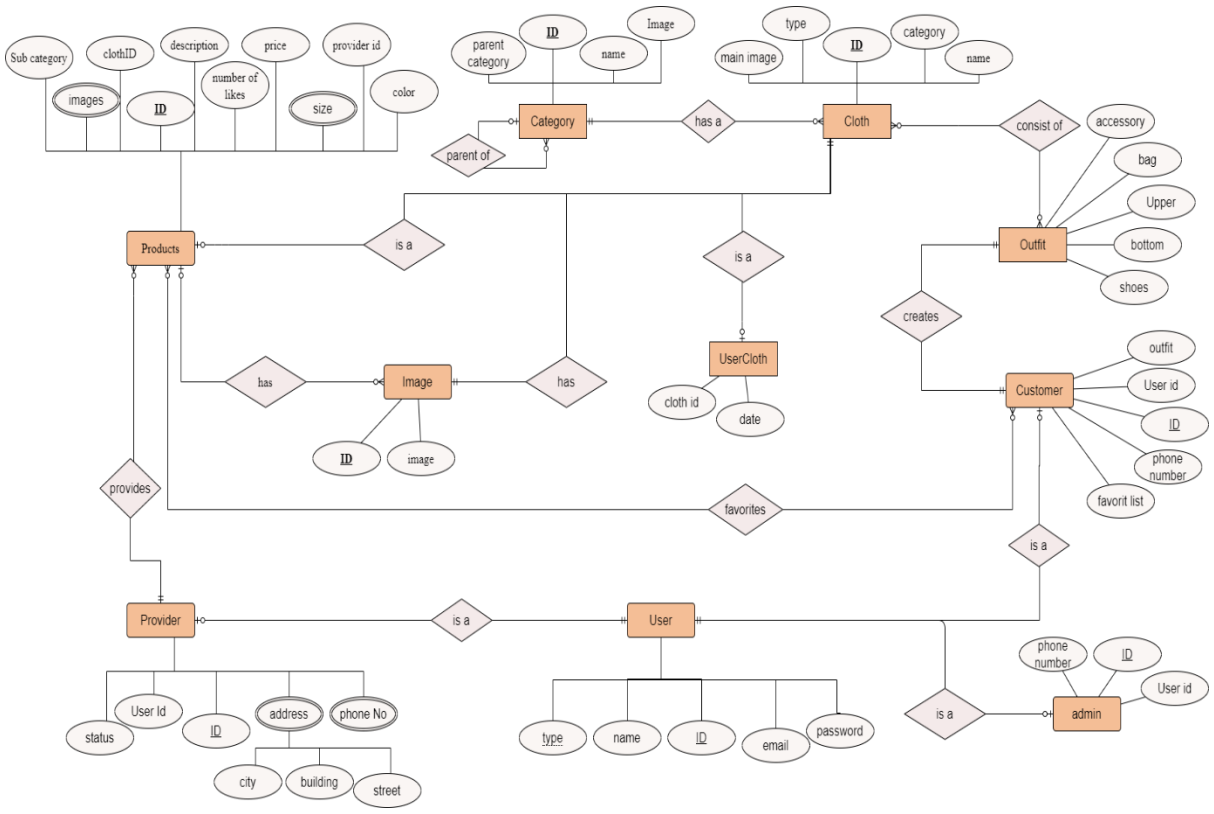


Figure 4-11 ERD

4.6. System GUI Design

4.6.1. Mobile UI

Figure 4-14 GUI: Login

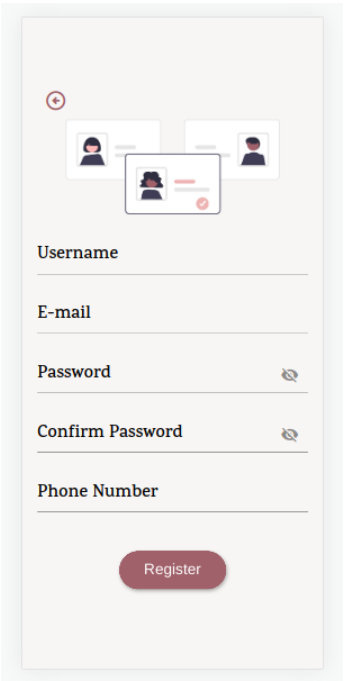


Figure 4-15 GUI: Register

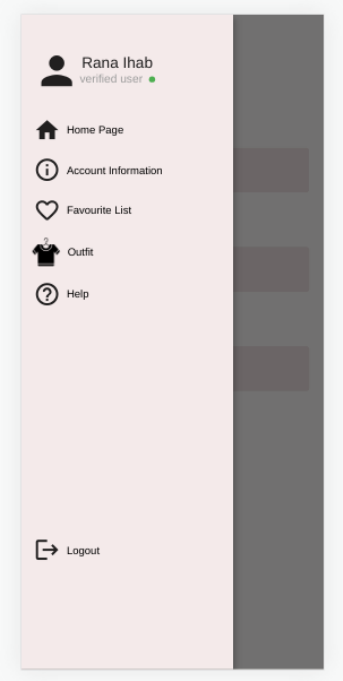


Figure 4-16 GUI: Sidebar

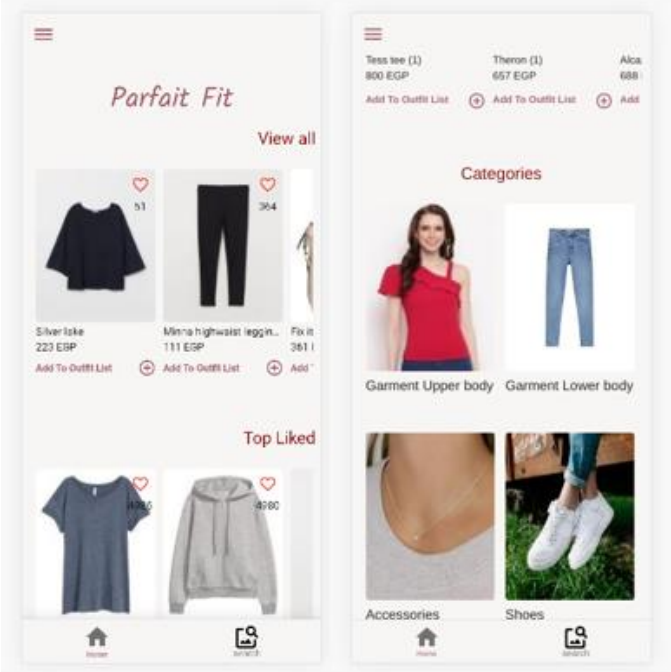


Figure 4-13 GUI: Homepage

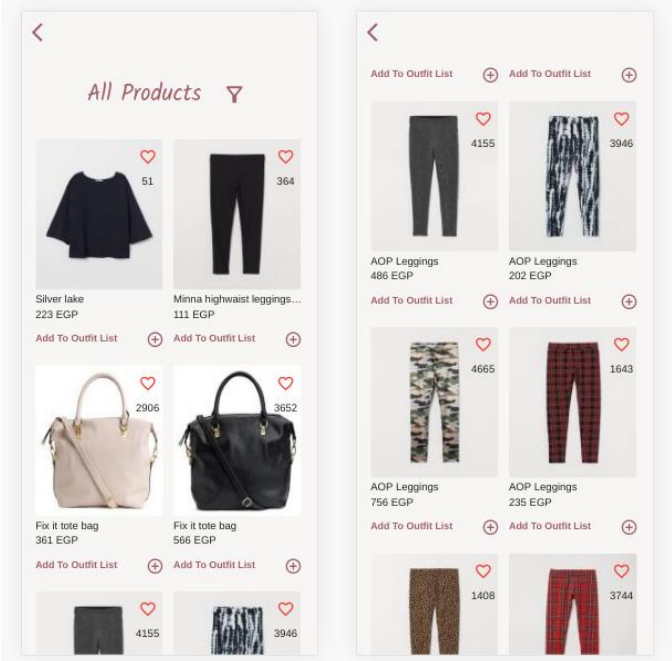


Figure 4-12 GUI: All products

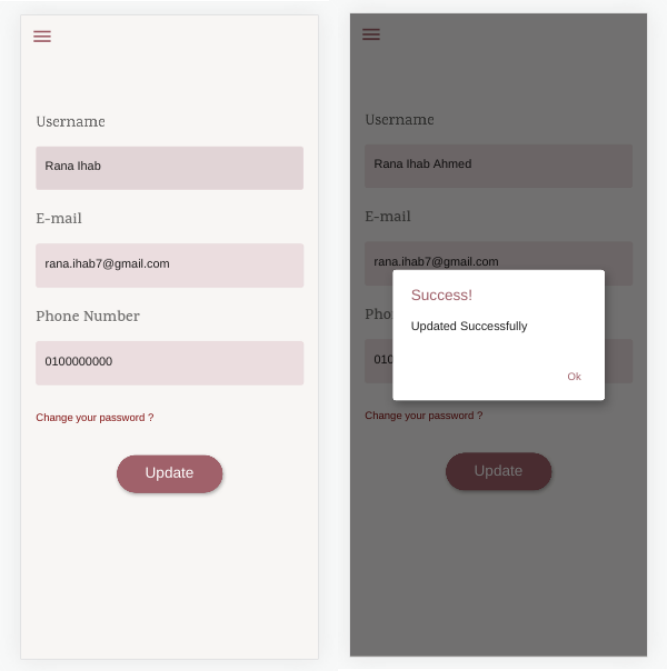


Figure 4-19 GUI: Update Profile

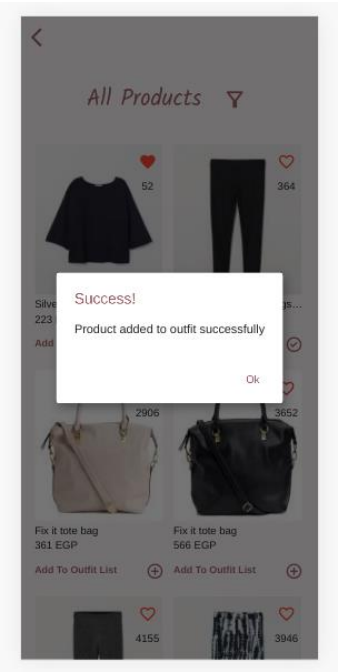


Figure 4-17 GUI: Add to Outfit List

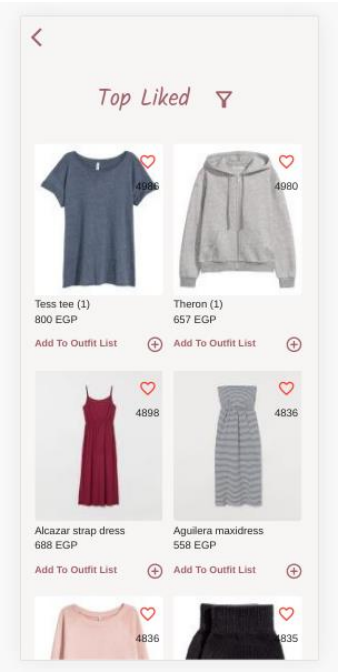


Figure 4-18 GUI: Top Liked

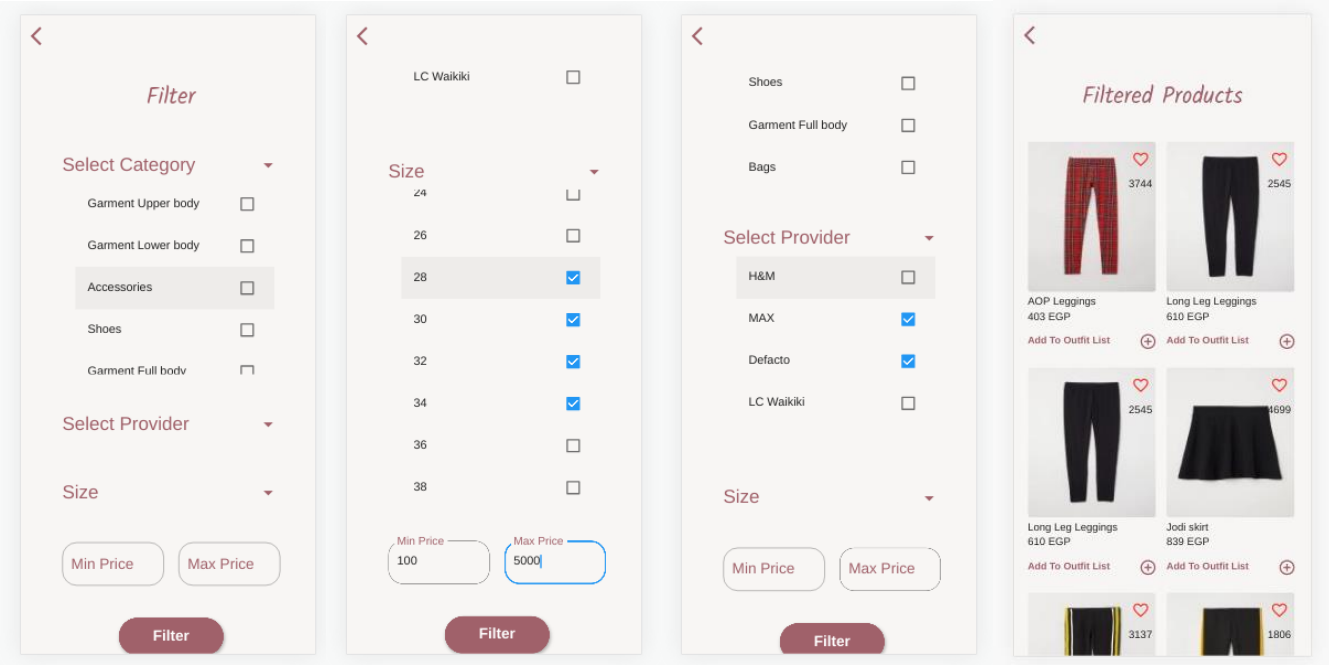


Figure 4-20 GUI: Filter Products

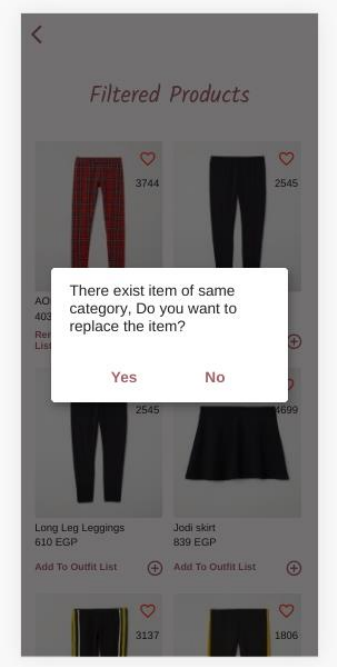


Figure 4-21 GUI: Add Product to Outfit List of Existing Category.

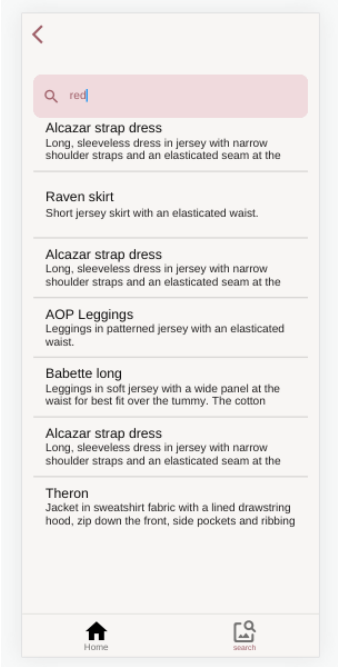


Figure 4-23 GUI: Search Products

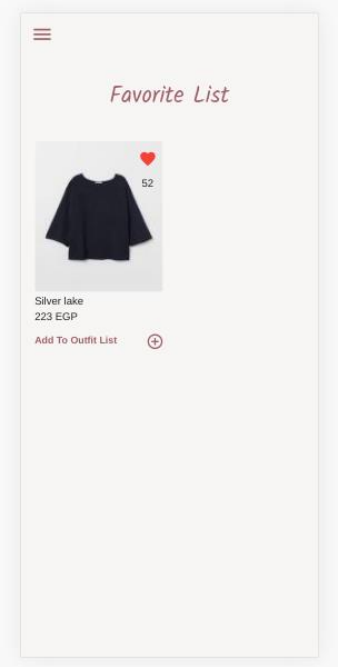


Figure 4-22 GUI: Favorite List

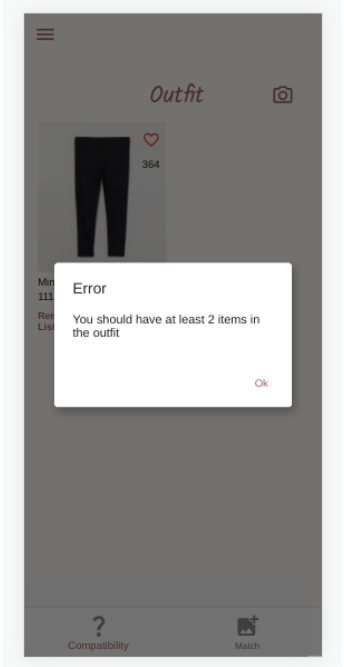
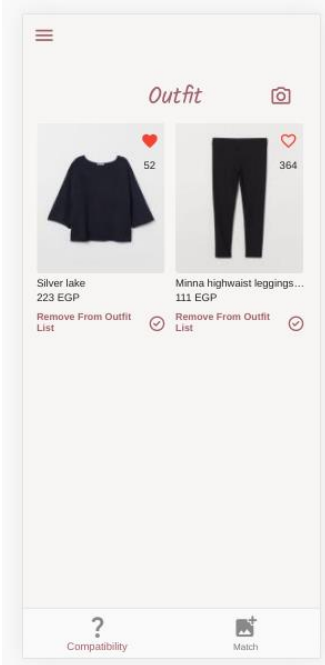


Figure 4-24 GUI: Check Compatibility invalid



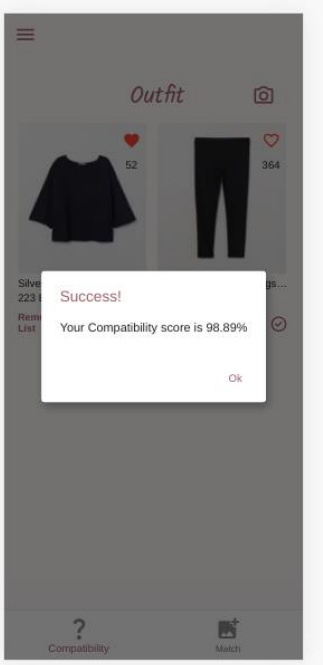


Figure 4-27 GUI: Check Compatibility Outfit Score

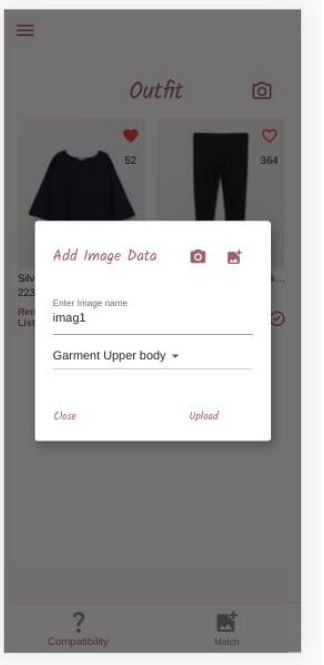


Figure 4-26 GUI: Upload Image to Outfit

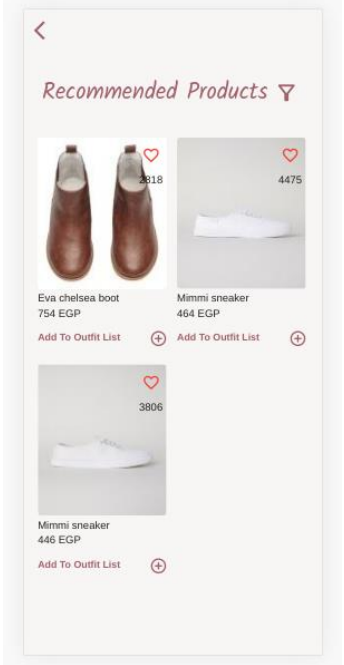


Figure 4-25 GUI: Matching item recommendation

4.6.2. Web UI

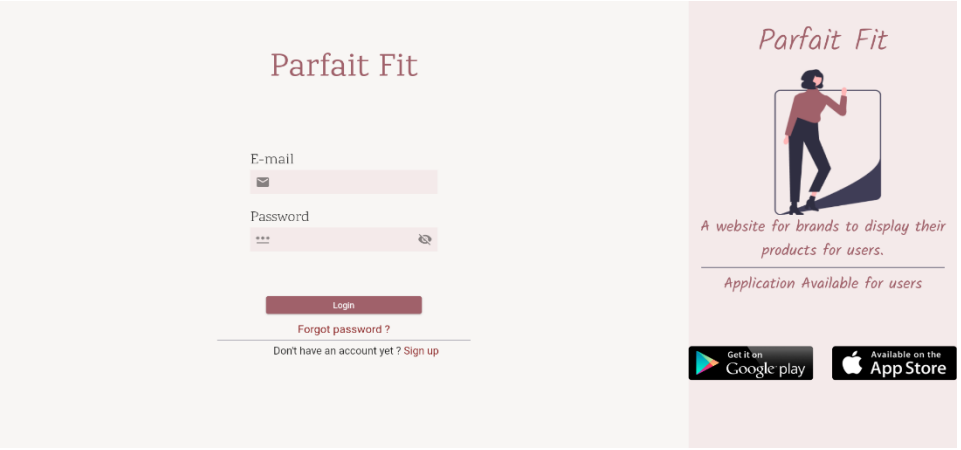


Figure 4-31 GUI: Log In

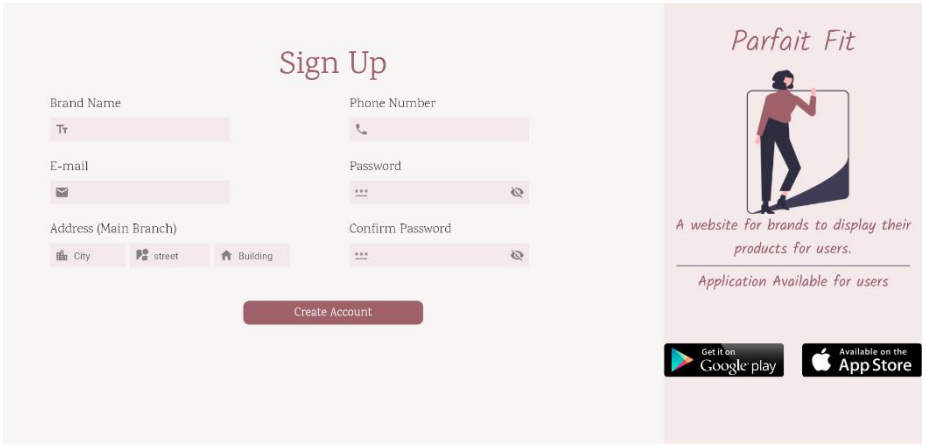


Figure 4-30 GUI: Provider Register

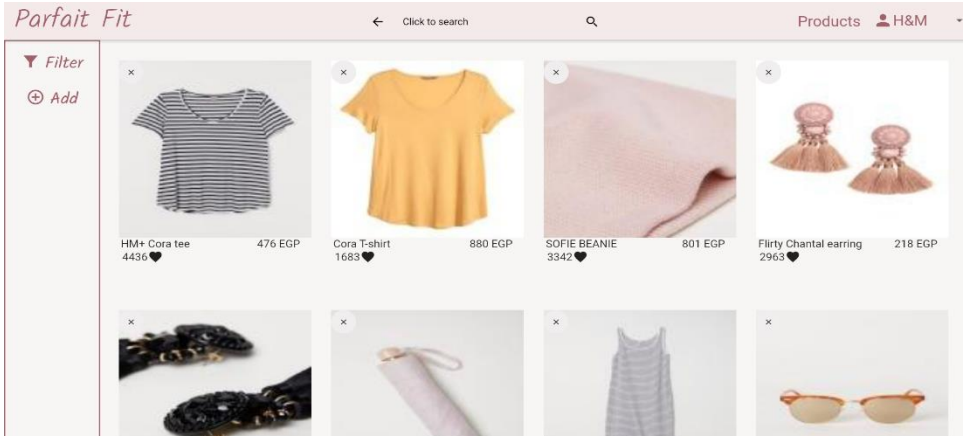


Figure 4-29 GUI: Provider Home

Figure 4-28 GUI: View Product Provider

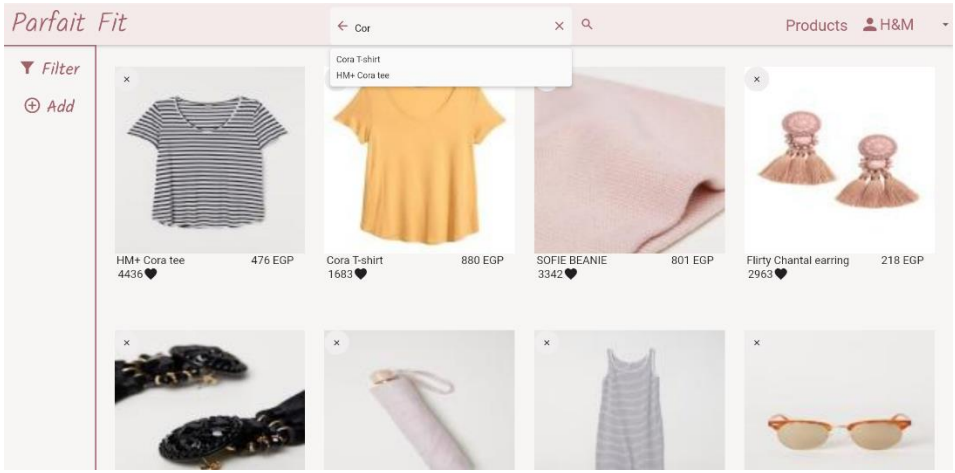


Figure 4-35 GUI: Search Web

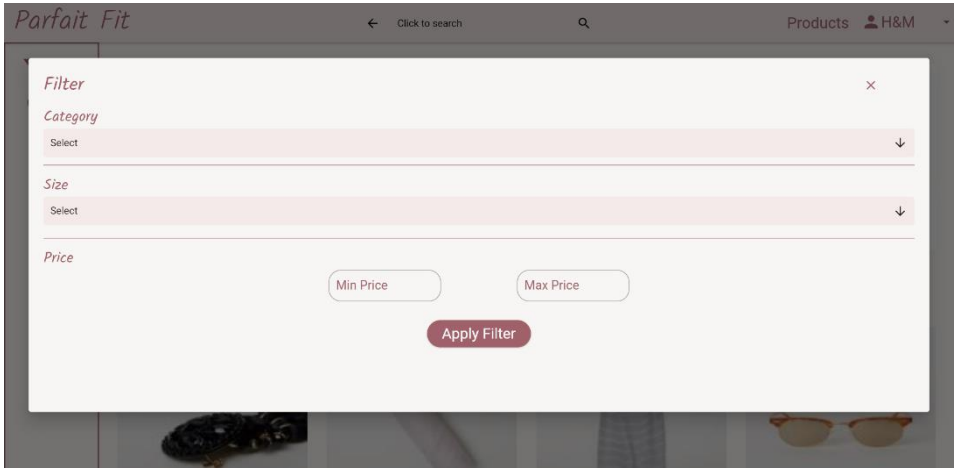


Figure 4-34 GUI: Filter Web Page

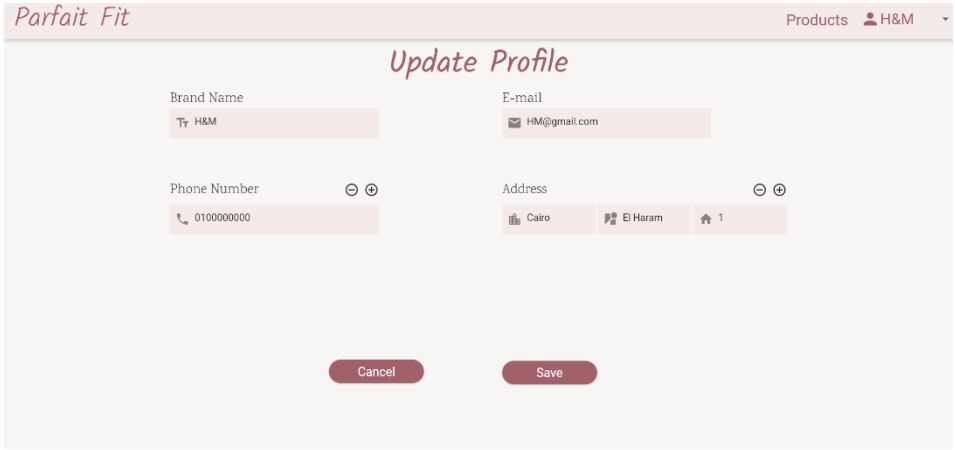


Figure 4-33 GUI: Provider Profile Page

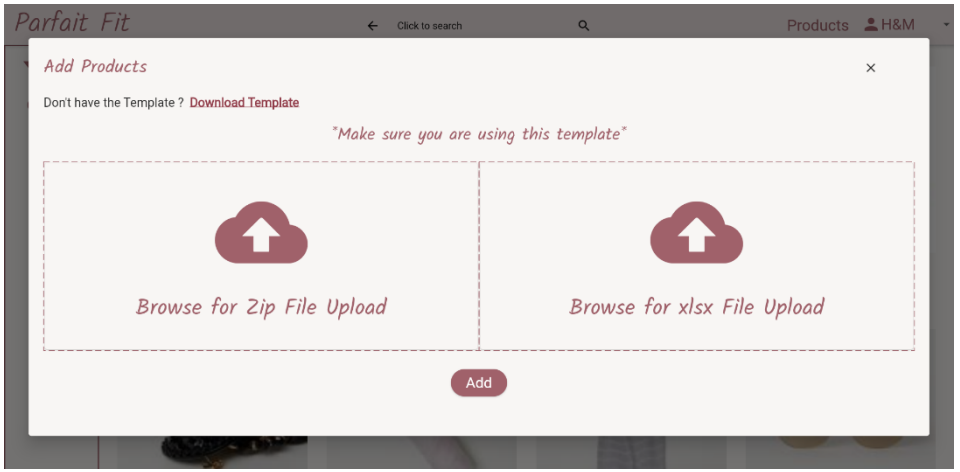


Figure 4-32 GUI: Add Products Web Page.

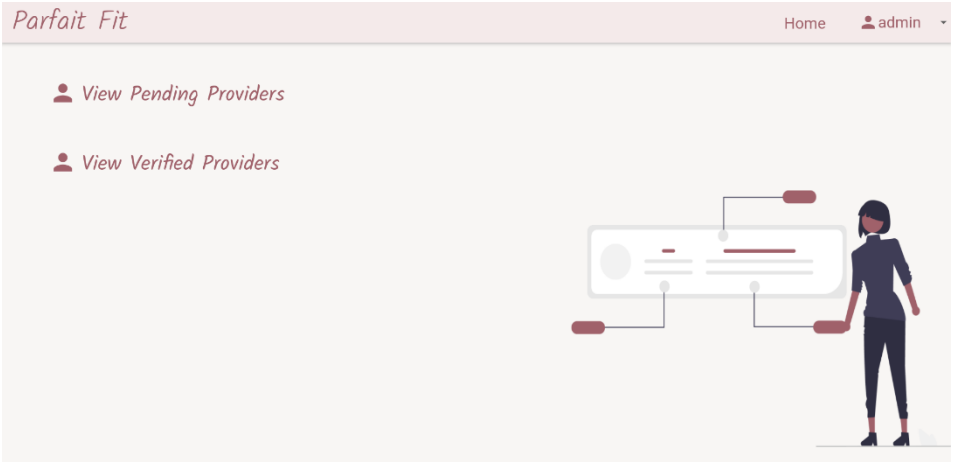


Figure 4-39 GUI: Admin Home Page

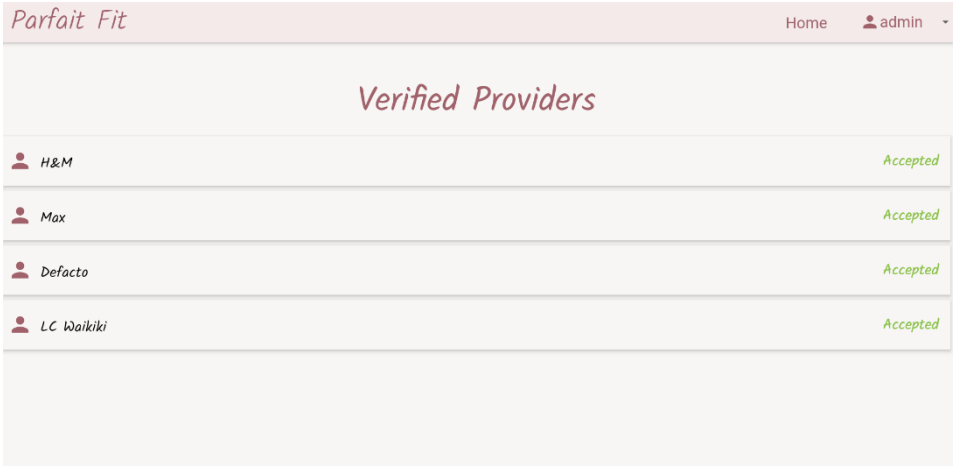


Figure 4-38 GUI: Verified Providers Page

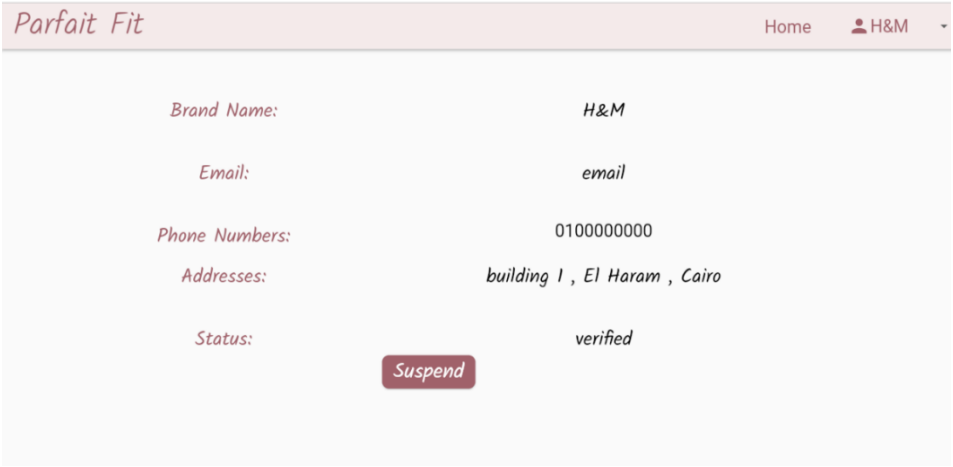


Figure 4-36 GUI: View Provider Information Page

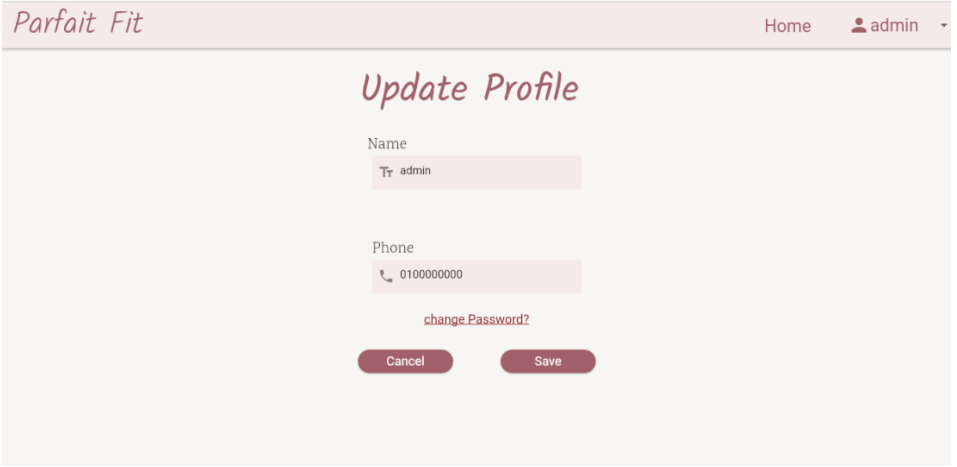


Figure 4-37 GUI: Admin Profile Page

Chapter 5: Approach

In this chapter we will cover the approaches used in implementing some of the functionalities in the system. We have a deep learning model, Outfit compatibility classifier, it is implemented by using relational network architecture. Also, we have personalized recommendation that we use content-based filtering for it. The next sections will cover these two approaches in detail, stating why we chose these approaches, the used dataset and evaluation.

5.1. Outfit Compatibility Classifier

Outfit compatibility classifier is a deep learning model used to classify an outfit into two classes: compatible or not compatible. It takes images of items in the outfit and predicts the compatibility score. We tried to architectures which are relational network (RN) and relational network with visual semantic embedding (RN-VSE).

RN-VSE is an enhancement on RN where it utilizes both visual and textual information. We chose to try these two models because they do not have a restriction on the order in which the items should be passed to the model (shoes then bag = bag then shoes) and the outfit does not have to be complete in order to determine its compatibility, we can check the compatibility of a shirt with trousers without shoes.

5.1.1. Dataset

We used polyvore dataset **[2]** which consists of 21,889 outfits from polyvore.com. Each outfit consists of items that make up this outfit and each item has an image, description, and category associated with it.

However, the dataset contains items other than clothing items therefore, we cleaned the dataset by removing items that are not clothing. The clothing items are grouped into bottom, upper, shoe, bag, and accessory and each outfit can contain only one item from each group. The outfits in polyvore are created by fashion experts so they are considered compatible outfits and to have incompatible outfits, we randomly generated outfits from the items existing in the dataset.

```
"214388560": {
    "upper": {
        "index": 1,
        "name": "shein sheinside pink knitted tassel trim asymmetrical long sleeve outerwear",
        "categoryid": 24
    },
    "shoe": {
        "index": 2,
        "name": "brown round toe faux leather chunky heel boots",
        "categoryid": 263
    },
    "bag": {
        "index": 3,
        "name": "shein sheinside grey contrast faux leather studded handbag strap",
        "categoryid": 37
    },
    "label": 1
},
```


Figure 5-1 Dataset Example

Table 5-1 Dataset Outfit Counts

Features	Train Data	Validation Data	Test Data
Compatible Outfits Number	16,176	1,197	2,463
Incompatible Outfits Number	16,176	1,197	2,463
Total	32,352	2,394	4,926

5.1.2. Relational Network Model

Following the architecture used in **[5]**, we implemented a deep learning model based on the concept of relational reasoning network architecture.

Relational network has the ability to reason about the relations between objects and their features. Its architecture focuses explicitly on relational reasoning. The input to the RN is a set of objects, where the objects form a scene, and it might take a relational question. For example, we have an image, which is considered as a scene, and it consists of multiple shapes and colors, and we want to know the size of a particular object relative to another object.

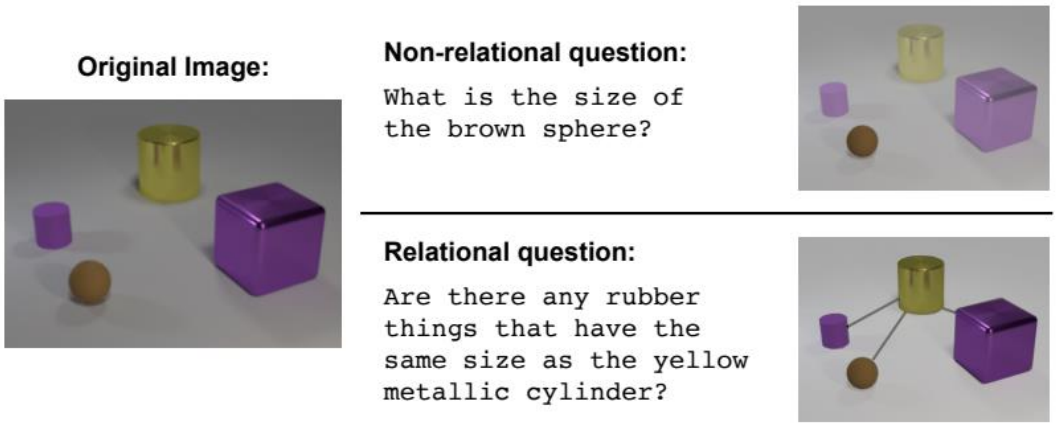


Figure 5-2 RN Example

In outfit compatibility, we can consider the outfit as the scene and the items of the outfit are objects of the scene thus unlike **[8]** that takes the scene as an image and extracts the objects, we take images of the items as objects. The model does not need to take a relational question as we are interested in learning a certain type of relation which is compatibility, and this relation is not question dependent.

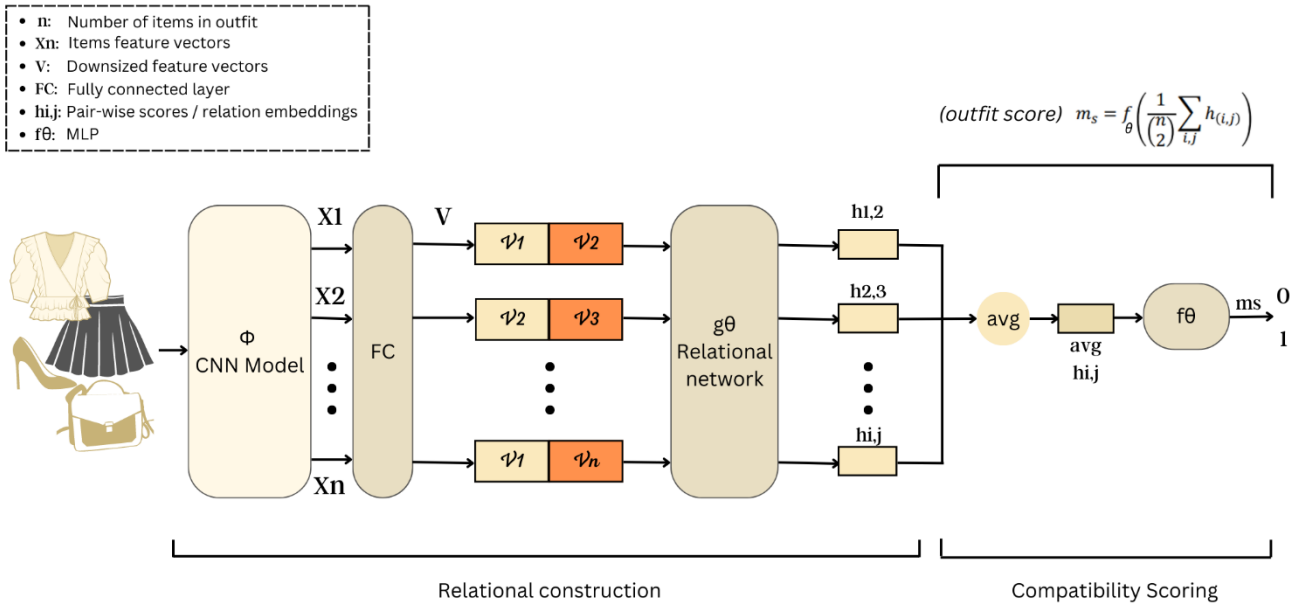


Figure 5-3 Outfit Compatibility RN Architecture

As shown in **Figure 5-3**, the model consists of two parts which are relation construction and compatibility scoring.

Relational construction: It is responsible for computing the pair wise score relation between each two items in the outfit. First, it will extract the features from the items' images by a CNN model *Φ* and then it will pass the extracted features *X* to a fully connected layer *FC* to lower the dimension of the vectors *X* to produce the downsized feature vectors V.

We then concatenate the feature vectors of each two items $[V_1|V_2]$ and pass them to the RN g_θ to explicitly extract the pair-wise scores, the relation embeddings $(h_{i,j})$. Where g_θ is a multi-layer perceptron (MLP).

Compatibility Scoring: It takes the relation embeddings and get their average then pass them to a MLP f_θ to get the compatibility score m_s of the outfit as a whole.

5.1.2.1. Experiment

We tried different architectures and performed hyperparameter tuning to try to improve the performance of the model as much as possible. We chose area under the curve (AUC) metric as the measure that we will use for evaluation:

AUC: Is the measure of the ability of a binary classifier to distinguish between classes.

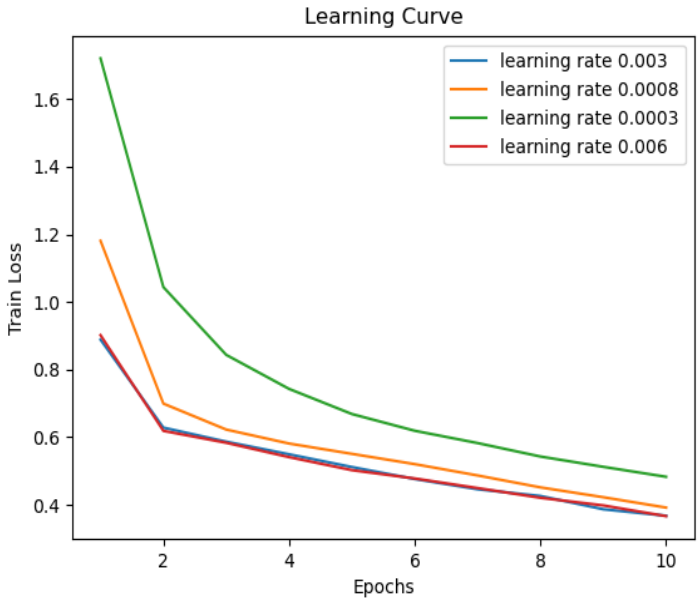


Figure 5-4 RN Learning Curves

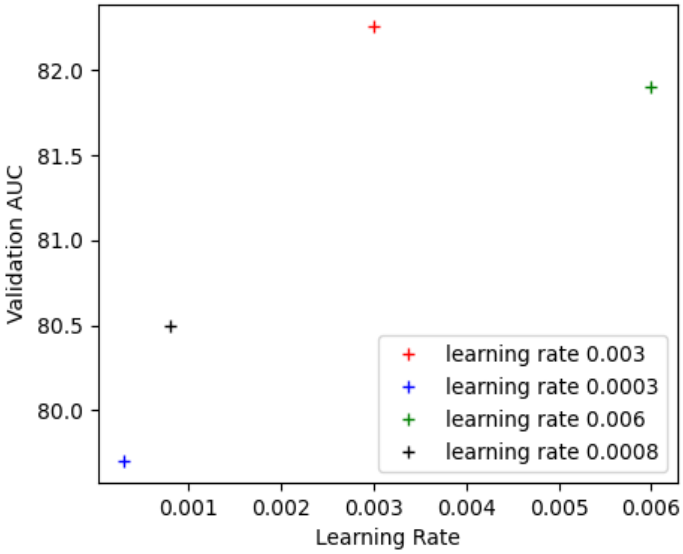


Figure 5-5 RN Learning Rata/Validation AUC

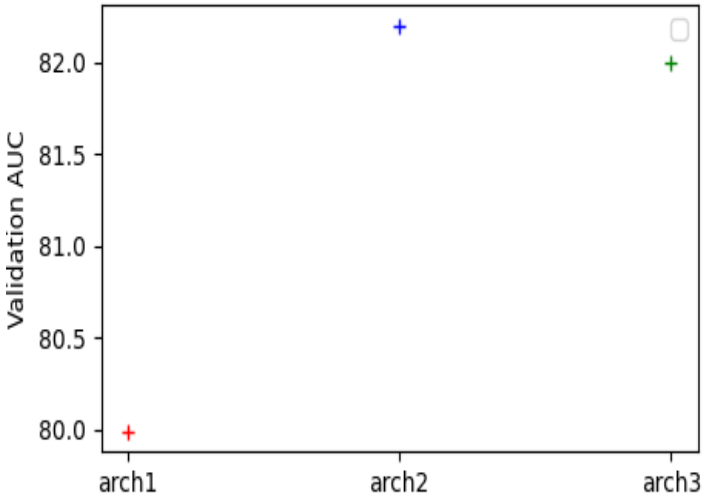


Figure 5-7 AUC for Architectures

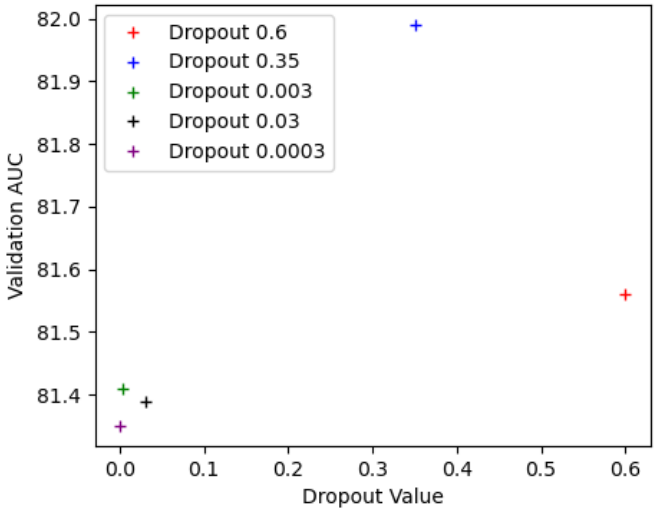


Figure 5-6 AUC for Dropouts

We trained three architectures for 10 epochs to choose the best one among them. When we evaluated the three models on the validation data, we found that architecture 2 has the best area under the curve (AUC) value **(Figure 5-7)** thus, we chose this architecture.

In architecture 2, we chose resnet50 as our CNN pretrained model Φ and we used transfer learning by training only the last layer in the resnet50, the *FC* layer has 1000 neurons. As mentioned, *f* and *g* are both MLP networks where in this architecture *g* has 4 layers with 512, 512, 256 and 256 neurons respectively. As for *f*, it contains 3 layers with 128, 128 and 1 neurons. For all the hidden layers in *f* and *g*, we add layer normalization and RELU activation function. However, we used sigmoid activation function for the last layer in *f* as it is a binary classification problem.

For architecture 1 and 3, they are mostly the same as architecture 2, but we made architecture 3 less complex by removing a layer from *f* and architecture 1 is more complex as we added a layer with 32 neurons. We tried adding and removing layers to see how the complexity of the model affects the results.

We then tuned the learning rate by trying different values and training architecture 2 for 10 epochs. We chose the learning rate 0.003 as **(Figure 5-4)** the learning curve of 0.003 and 0.006 converges faster than the others without oscillating and 0.003 has better AUC **(Figure 5-5)**.

Lastly, we tuned the dropout values and found that 0.35 has the best AUC **(Figure 5-6)**. To sum up our model, the architecture has 4 layers in *g* and 3 layers in *f* where we use RELU as the activation function for all the layers except the last in *f* which uses a sigmoid function. We trained the model for 20 epochs with a learning rate=0.003 and dropout value=0.35 for the layer in *f*. We used Adam optimizer to learn the parameters and binary cross entropy loss function since it is a binary classification problem.

One more thing is that we wanted to determine the optimal threshold for our classification problem. To achieve this, we have to balance between true positive (TPR) and false positive (FPR) rates by plotting the ROC curve which tries evaluates

the model on different threshold and plot their TPR and FPR. After getting the TPR and FPR and different thresholds, we can now calculate the optimal threshold by Youden's J statistic (J) equation that uses the sensitivity and specificity to find the threshold that balances between them.

Sensitivity is the true positive rates and recall. It determines the ability of a model classifying the positive class correctly. While Specificity is the inverse of the FPR which determines the model's ability in classifying the negative class correctly.

$$sensitivity = \text{True Positive} \,(\, \text{True Positive} + \text{False Negative}) = \text{TPR}$$

$$specificity = \text{True Negative} \,(\, \text{True Negative} + \text{False Positive}) = 1 - \ \text{FPR}$$

$$J = sensitivity + specificity - 1$$

After calculating J, we get the threshold 0.497018 that has the maximum value (**Figure 5-8** the black dot).

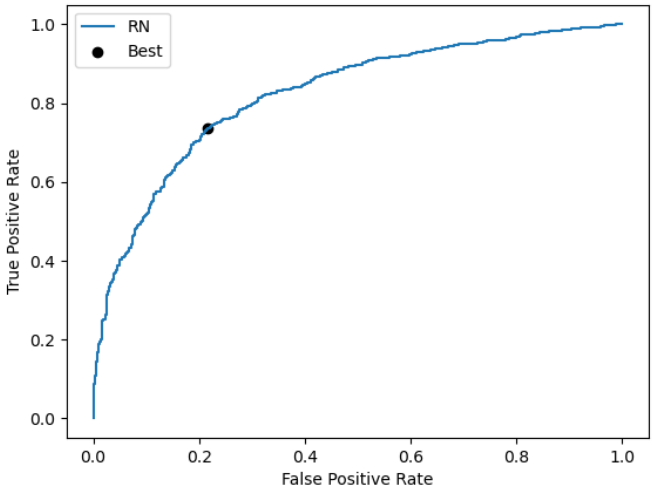


Figure 5-8 ROC Curve

5.1.3. RN-VSE

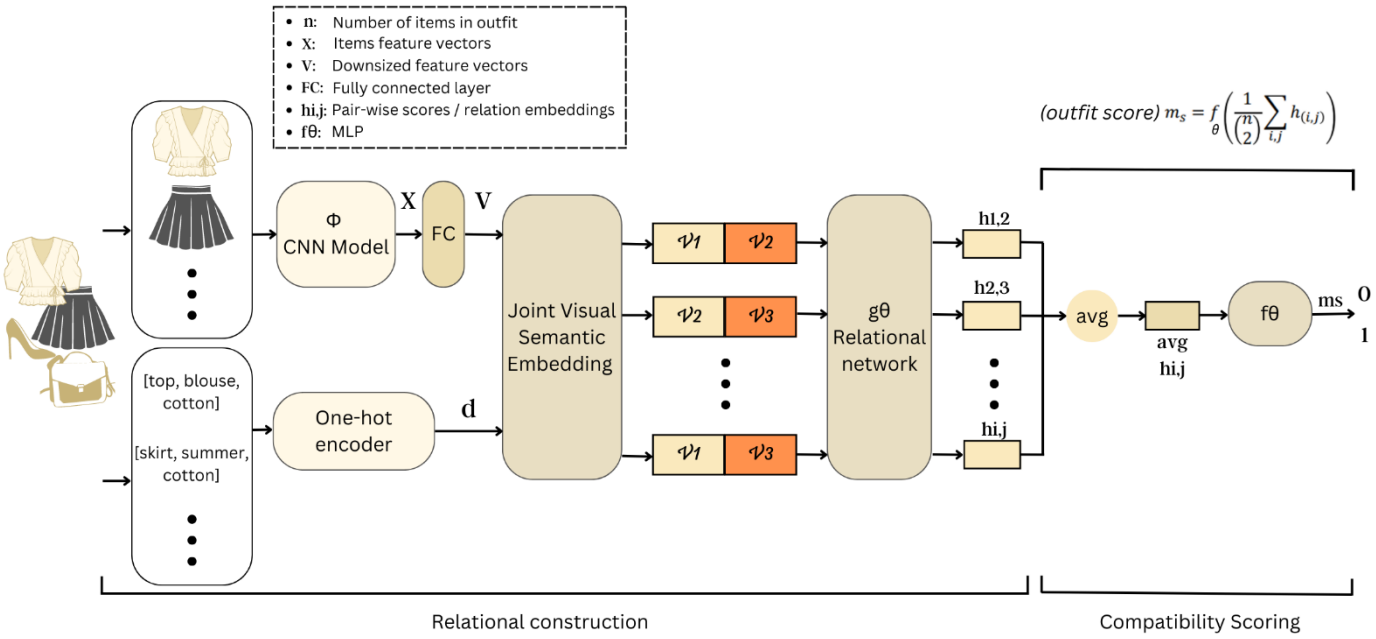


Figure 5-9 RN-VSE Architecture

The RN-VSE model is the same as the RN model but with an additional step that uses textual features with the visual features. Studies have found that utilizing both the visual and semantic features improves the performance in most of the models.

Therefore, VSE learns a multimodal embedding space of texts and images by using a small description for each item besides the image. The model first extracts the visual embeddings *V* as in RN by using a CNN model then passing it to *FC* layer and extracts semantic embeddings *D* from the textual data. It extracts *D* by representing each word with one-hot encoder, then getting its embedding encoding it by using bag of words which gets the average of the embeddings for each item's description.

After extracting the visual and semantic embeddings, we pass them to the joint visual semantic space where it estimates the similarity between the semantic and visual embedding for each item. The goal of the visual semantic space is to make the embeddings for the same item to be as similar as possible as they represent the same item. It is used to extract the visual and semantic features that better represent the item and best contribute to determining the compatibility.

VSE estimates the similarity between the visual and semantic embeddings by using cosine similarity *x(v,d)*. The images and descriptions become in the joint space by minimizing the following loss function:

$$E = \sum_{v}\sum_{d} \max\left(0, m - x(v,d) + x\left(v,d_f\right)\right) + \sum_{d}\sum_{v} \max\left(0, m - x(d,v) + x\left(d,v_f\right)\right)$$

In the above equation, $x(v,d)$ and $x(d,v)$ are the similarity between the visual and semantic embeddings of the same item, $x\left(v,d_f\right)$ donates the similarity between the visual embeddings with semantic embeddings of other items, $x\left(d,v_f\right)$ donates the similarity between the semantic embeddings with visual embeddings of other items and m is a margin for error.

Description and image for the same item should be as similar as possible and descriptions and images for different items should be different therefore, the above equation tries to learn the weights for the embeddings matrix to achieve this.

Semantic embeddings are only used to compute the loss but are not passed to the multi-layer perceptron. Its loss function will be added to the binary cross entropy function to get the total loss.

5.1.3.2. Experiment

For RN-VSE architecture, we tried different learning rates, dropouts, l2 regularization, architectures, and CNN models. We also tried experimenting making the CNN model have more trainable layers as opposed to RN where only the last layer is trainable however, all the trials resulted in a model that overfits. Nevertheless, the best architecture is the same as the one used for the RN model except for making the layers in the last resnet50 block trainable.

5.1.4. Evaluation

As mentioned before, the main metric that we use is the AUC, but we also evaluated the performance on test data using different metrics; accuracy, recall, precision, and F1-score.

Table 5-2 Performance of Models on Test Data

Metrics	RN	RN-VSE
AUC	83.56	85.8
Accuracy	75.6	78.8
Recall	73.6	85.5
Precision	83.7	80.4
F1-Score	78.3	82.9

Even though RN-VSE has better performance, since it overfits we decided to use RN model.

5.2. Personalized Recommendation

In personalized recommendation, we recommend products to the user taking into consideration the user's preference hence, personalized. Among the different available approaches, we chose content-based filtering which focuses on the content of the items so that we would recommend similar items in terms of the content.

The recommendation process is TF-IDF based Product Similarity, where we recommend items existing on the system that are similar to the items the customer liked in terms of description, category, color and price. We chose to calculate the similarity by the TF-IDF approach because TF-IDF puts more focus on frequent rare terms so, it would be more able to determine which aspects the user is more interested in.

5.2.1. Preprocessing Data

To apply this, we first start by preprocessing the data existing on the system by applying NLP techniques as removing punctuation, removing stop words, applying lemmatization and tokenization on each product's description, name, and color then this preprocessed data is saved in the database to be used later in the recommendation.

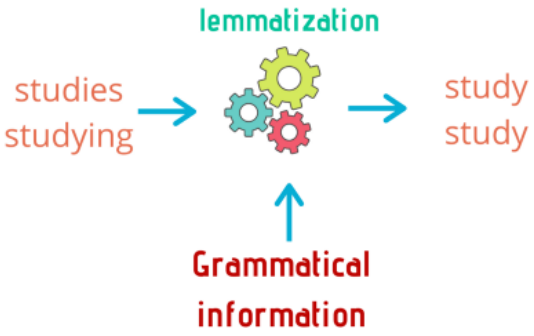


Figure 5-11 Lemmatization

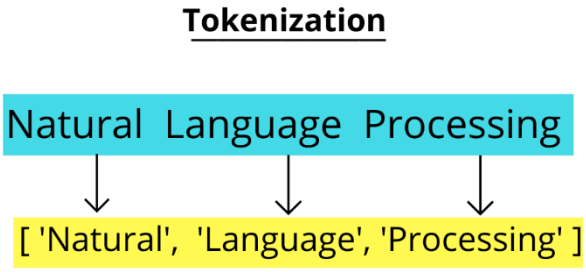


Figure 5-10 Tokenization

5.2.2. Calculate TF-IDF Vector

We then create the feature vector for the products in the system that will be used in the similarity calculation, by calculating the TF-IDF (as in **[1]**) for the saved preprocessed data for each product and concatenating the resulting vector with the normalized price vector of the products on the system.

The normalization technique used to normalize the price vector is $(prices - min\_val) / (max\_val - min\_val)$.

Now we have feature vector of the products representing the description, category, color, and price.

Table 5-3 Example for TF-IDF

Products / Preprocessed data terms	Term 1	Term 2	Term 3
Product 1	0.70701	0.00000	0.33105
Product 2	0.12510	0.81509	0.41287

Assuming the price of product 1 is 100 and the price of product 2 is 400, The resulting normalized price vector will be [0,1] and the resulting features vector will be $[\,[\,0.70701, 0.00000, 0.33105, 0\,], [\,0.12510, 0.81509, 0.41287, 1\,]]$

5.2.3. Create User Profile

We then create the user profile (as in **[10]**) to get the vector that will be used in similarity calculation. This step is done for each user registered on the system.

To create the user profile, we get the list of products the user has previously liked then we get the features of these products from the features vector calculated above and append them in an empty list to represent the features vector of the products liked by the user.

We then aggregate the features by adding the values of the feature vector of each product and normalize the resulting vector using L2 normalization to have a vector representing the user's preference.

Example:

If the user liked products were product 1 and product 2:

Table 5-4 User Profile Creation 1

Products / Features	Term 1	Term 2	Term 3	Normalized priced
Product 1	0.70701	0.00000	0.33105	0
Product 2	0.12510	0.81509	0.41287	1

We first aggregate using summation:

Table 5-5 User Profile Creation 2

	Term 1	Term 2	Term 3	Normalized priced
User Profile	0.83211	0.81509	0.74392	1

Then we perform l2 normalization:

Table 5-6 User Profile Example

	Term 1	Term 2	Term 3	Normalized priced
User Profile	0.48778	0.47780	0.43608	0.58619

We get the feature vector representing the user profile

$[\,0.48778, 0.47780, 0.43608, 0.58619]$

5.2.4. Applying Cosine Similarity

We then apply cosine similarity between the resulting feature vector for each user profile and the feature vector of the products in the system, we set the similarity threshold to 0.6 to recommend items based on the users' preferences above that threshold.

5.2.5. Dataset

We used the H&M dataset provided by an old competition **[3]** to build this model. It contains articles.csv which has detailed metadata for each product available for purchase on their system, transactions_train.csv which has the transactions of the users on their system, …, and the images of the products.

We filtered this data, by keeping only the products related to females, to test and do experiments on our model, the total number after filtration is 44804 products.

Regarding the transactions.csv file, it originally contains 22M records each record represents the customer id, the id of the product purchased, and the price of the product. So, we grouped the customers with their purchased products (discarding the duplicate ones) to make the customers' profiles and know their preferences.

Although the dataset contains the transactions of the customers of the H&M system, it was a good choice for us to test our model on it, we prepared the data to work as if it had liked items not purchased items to match our system by discarding the duplicated purchased items.

5.2.6. Example for the content-based results.

To illustrate the result of the contend-based recommendation, we've plotted the recommendation results for a few users based on the used H&M dataset mentioned above.

- User 1

```
plot_old_products(all_users_list_items[0])
```

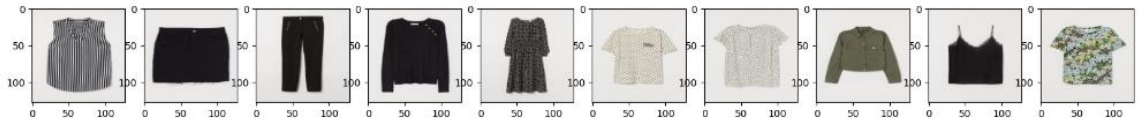


Figure 5-12 User 1 Previously Purchased Items (Content-Based)

```
plot_recommendation_items(all_recommendation[0],10)
```

Figure 5-13 User 1 Top 10 Recommended Items (Content-Based)

- User 2

```
plot_old_products(all_users_list_items[50])
```

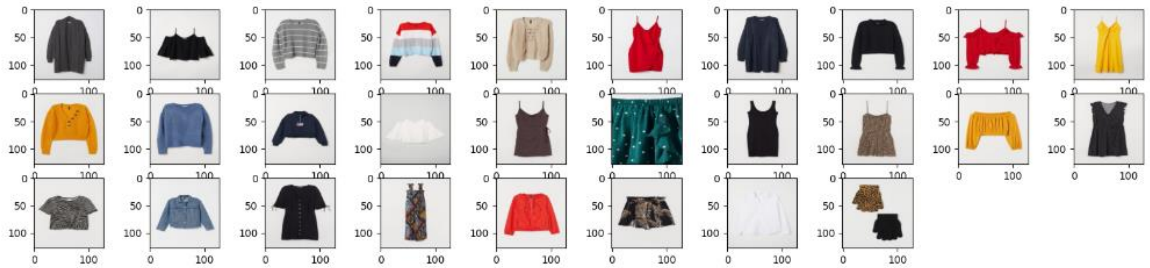


Figure 5-14 User 2 Previously Purchased Items (Content-Based)

```
plot_recommendation_items(all_recommendation[50],10)
```

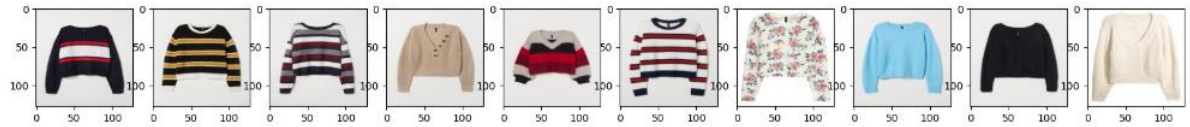


Figure 5-15 User 2 Top 10 Recommended Items (Content-Based)

Chapter 6: Implementation and Testing

6.1. Implementation

After finishing the analysis and design, we now have enough details to start the implementation. The implementation consists of 3 parts which are the frontend, backend, and models. We implemented the system so that the frontend and backend could be developed in parallel, and they communicate by the frontend calling the backends' APIs and use the standard status codes to determine the response.

We have already talked about the models in the previous chapter. Therefore, in this section, we will discuss how we implemented the frontend and backend; going into the details of some of the functions; check outfit compatibility, compatible items recommendations, bulk upload products, search, and personalized recommendations. Lastly, we will end the chapter by explaining the technical obstacles that we faced and how did we decide to solve them.

6.1.1. Front-end

We used flutter to implement the frontend of the application. We implemented two views, one is mobile view which the customer will use and the other is web, both the admin and provider will use the web to interact with our application.

We decided on this as it would be easier and more suitable for the customer to use mobile application since he might need to take pictures of his clothes' items while the provider will need to upload a zip file and excel file when adding his products and this is easier done using a PC or laptop. The web and mobile frontend are in two separate projects.

We divided our dart files into 4 groups: models, pages, components, and controllers to make our code more readable, reusability, and for the singular responsibility. The models are like structs that are used to hold data that are related to a concept so that we could easily send it from a page to another.

The pages folder contains the actual design and view of the mobile and web application pages. The components folder contains the cards or part of the components that will be used in multiple pages so instead of rewriting this part in each page, we will reuse this component. Lastly, the controllers are responsible for the validations on the inputs and for sending the requests to the backend and handling the responses with the status codes.

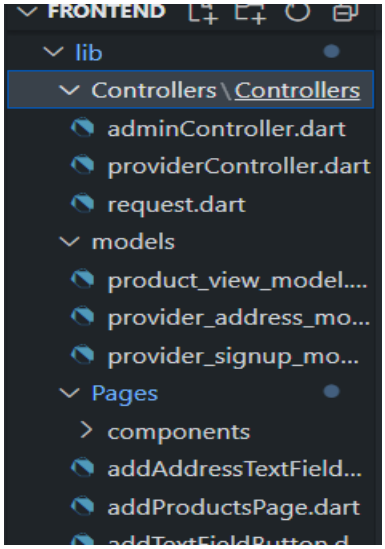


Figure 6-1 Frontend Structure

6.1.2. Back-end

As stated before, we used Django rest-framework to develop our APIs and the backend. Just as in the class diagram, we divided our system into packages, each package contains the entities/models needed, serializers and for each major model, there is a controller that handles the functions related to it. The controllers communicate with each other to perform the overall service. The serializers transform the object into a json format so that we could send this json to the frontend to use the data.

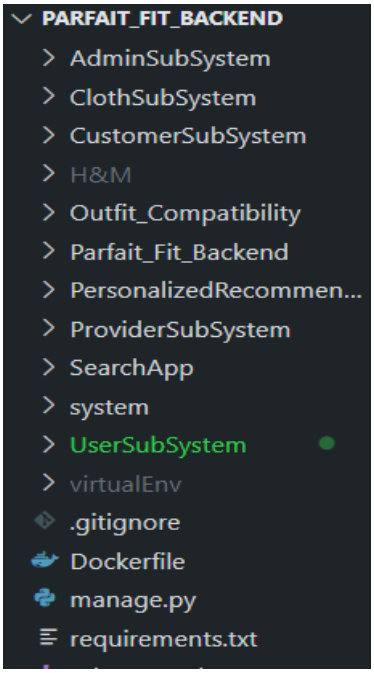


Figure 6-2 Backend Structure

We wanted to make our application as efficient as possible. Therefore, we minimized the database hits by using the Django decorator transactions.atomic and using select_related that gets the objects and their related objects in one query. Furthermore, we applied pagination on the data sent to the frontend to minimize the amount of data sent to speed up the transaction.

Also, to make our application secure, we used JWT authentication which minimizes the possibility of other people accessing the information or account of others. We used access-token, which is used for accessing the functions, it has a short expiration time so even if another person was able to use it, it will be for a short amount of time. Then we use the refresh-token to renew the access-token when it expires. The refresh-token has a longer expiration time.

6.1.3. Check Outfit Compatibility

In this feature, we allow the customer to check if an outfit is compatible or not. The data in the outfit page should be previously added by using functions uploadImage (images uploaded by customer) and addProductToOutfit (products in the system). The outfit should consist of at least 2 items, each item is in a separate image (example **Figure 6-3**). When the customer requests checking the compatibility of the outfit, a score will be shown. This score is the percentage of the outfit compatibility, where scores above 0.497018 are compatible and below this value, will be considered incompatible.

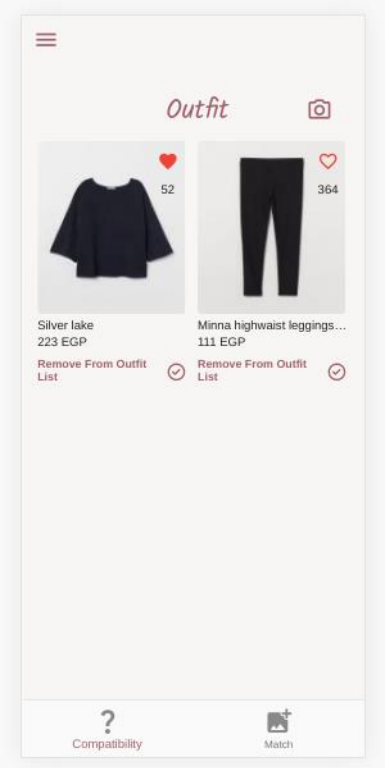


Figure 6-3 Customer Outfit Example

6.1.4. Compatible Items Recommendation

In this feature, we provide the customers with the option to get recommendations for products from certain categories compatible with their outfit list. The customer should have at least 1 item in the outfit to request this functionality.

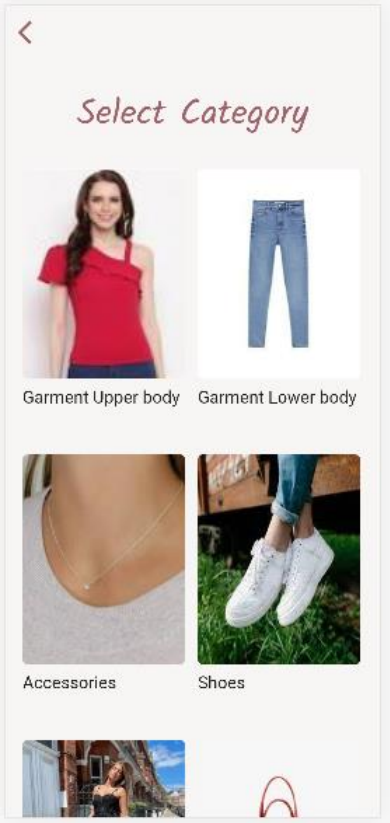


Figure 6-5 Select Category from Application

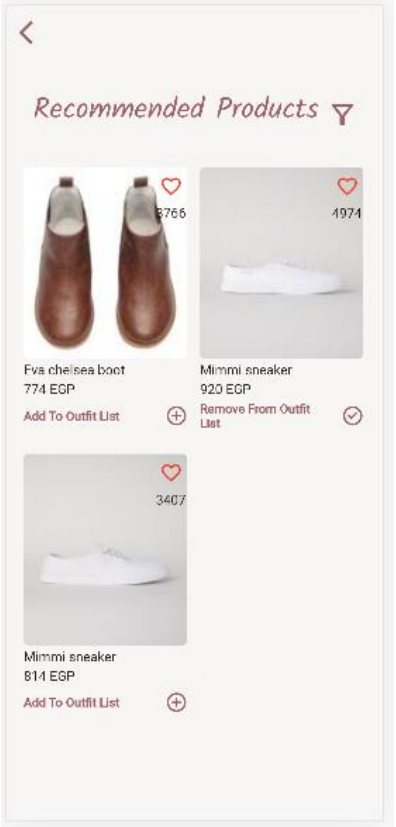


Figure 6-4 Recommended Products Example from Application

After choosing a certain category, we generate a random subset of size 50, from the products of this category existing on the system.

We then generate an outfit for each item in the list with the items existing in the customer's outfit.

The generated outfits are then passed to the compatibility model mentioned above to get the outfits' compatibility score, we've set a threshold of 50% for the compatibility scores to insure recommending compatible items, as a result we will return to the user the items that generate a compatibility score above 50%. We then recommend the top 50 items that resulted in the highest outfit scores.

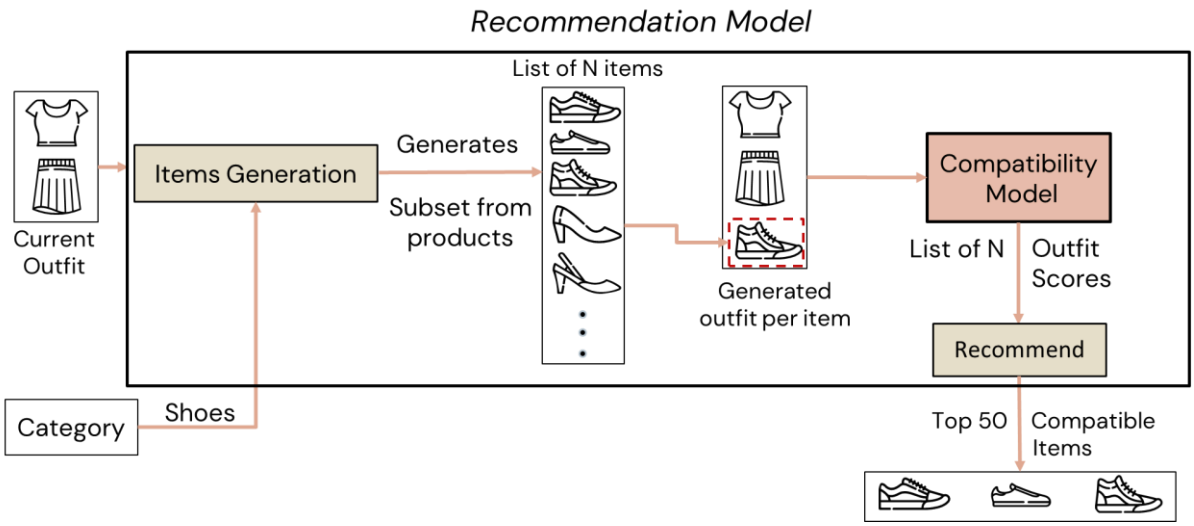


Figure 6-6 Compatible Items Recommendation Illustration

6.1.5. Bulk Upload Products

In this feature, we allow the provider to upload multiple products at once instead of uploading product by product to not waste his time.

We provide the provider with an excel template which he needs to fill up with his products by following it. We add notes and restrictions about the attributes of the product so that the provider can know which values are allowed.

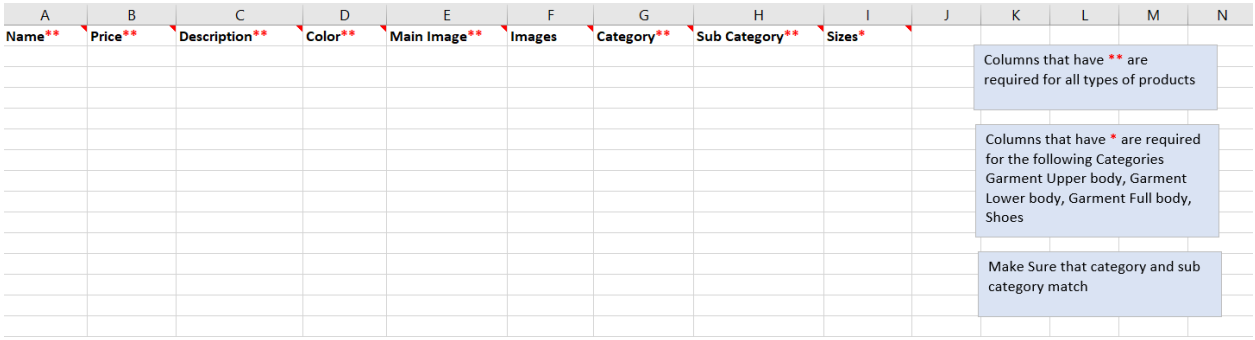


Figure 6-7 Template Example

After the provider fills up the template with his products, he should upload the excel file and a zip file containing the images of the products to the server so that we could add these products. We then loop on each row and extract the information from it, validating the data, if the data is valid, the product will be added. If it is not valid, we create an excel file containing only the invalid products with the error message for each product and return it to the user so that he can download it, fix the errors then resubmit it. Hence, the provider won't waste time reuploading all the products, he will only need to upload the ones that were invalid.

This function requires many database hits, so it is very inefficient. To improve its performance, we used a decorator that Django provides which is transactions.atomic.

This decorator ensures that all database operations within the block are treated as a single unit. Which means, that all the database hits will happen at once at the end of the function.

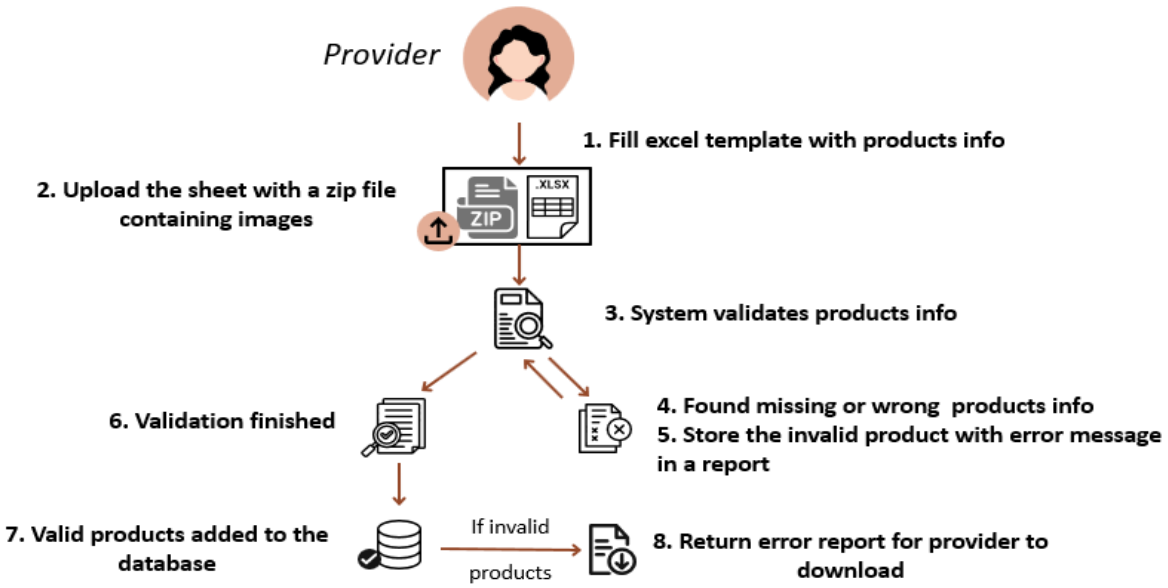


Figure 6-8 Bulk Upload Product Flowchart

6.1.6. Search

To implement the search functionality, we used Meilisearch API. Meilisearch is an open-source search engine that provides fast and relevant search experience for search. It is suitable for small to medium-sized applications. Also, from its advantages is that it handles bad queries such as misspelled words.

To use Meilisearch, we first open a connection with them. Then, when a provider adds products, we send a post request with these products to Meilisearch so that it can index them and build the inverted index. When a provider deletes a product, we send a request to Meilisearch so that it would remove the product from the index.

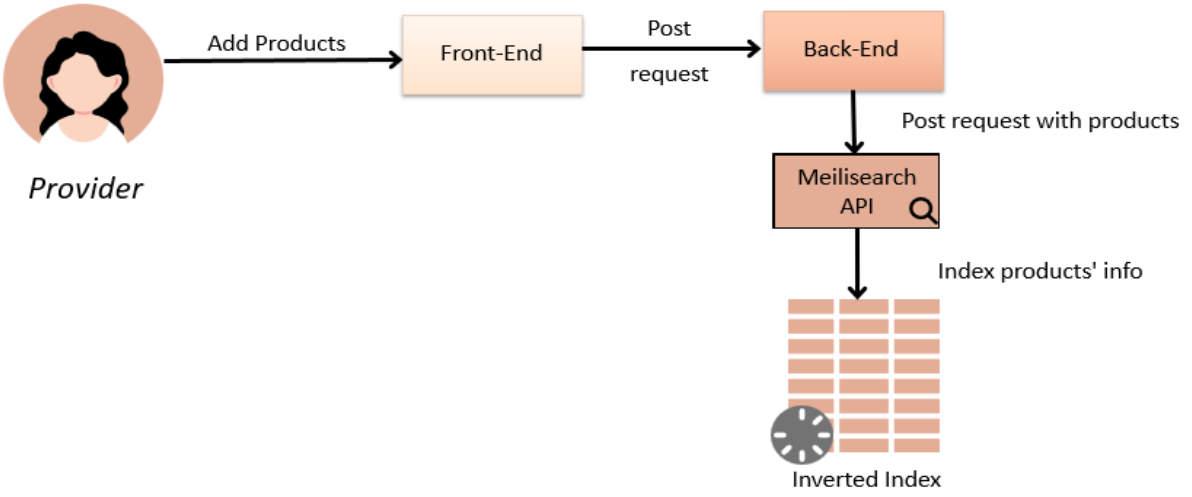


Figure 6-9 Search Indexing

Since there is an index built, we can now send the users' queries to Meilisearch so that it can quickly return the ids of the relevant products, then get those products from the database and send the results back to the user.

Meilisearch also provides filtering on the products, so we do not need to implement a filter function on the results of the search products, we only need to tell Meilisearch when building an index which attributes are filterable so that it can index it in a way that would make it more efficient when applying filtering for a query.

Both the frontend and backend call the meilisearch API to reduce the request on the backend. The frontend only has access to the search function, while the backend can access the functions that add, delete, update, and search the index. While the user is typing in the search box, we keep presenting him with the titles of some of the products that match the query. However, if we send these requests to the backend, it might slow down the performance of the system. So, we only send the search request to the backend, when the user finishes his query as then we would need more information about the products and this information only exists in the backend.

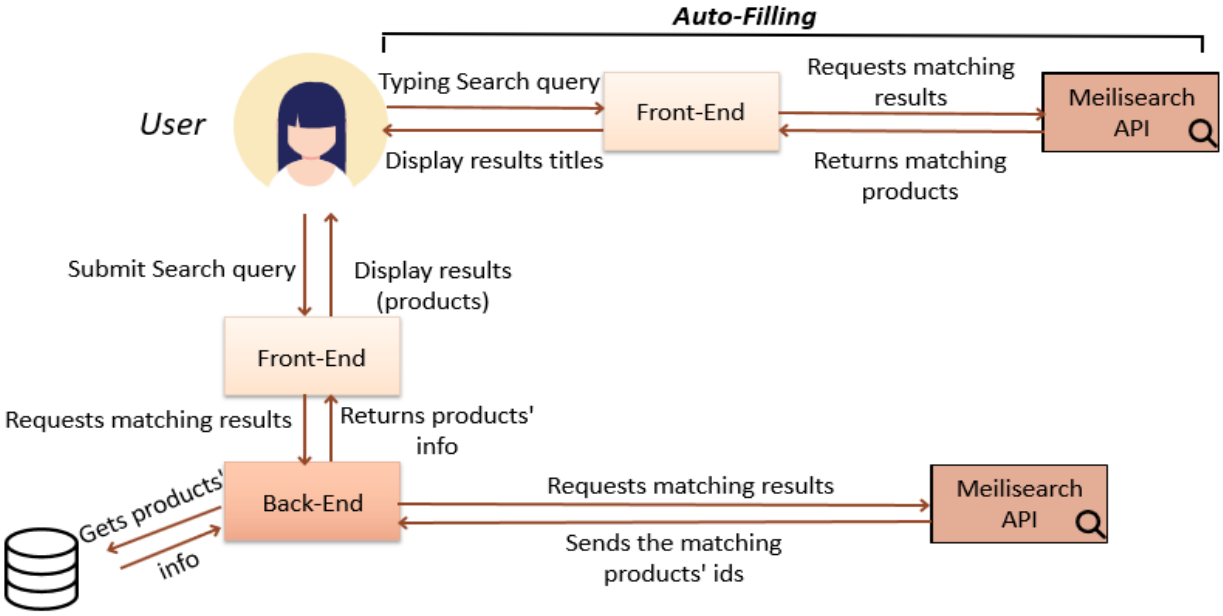


Figure 6-10 Search Flowchart

6.1.7. Personalized Recommendation

In this feature, we've applied content-based recommendation technique for the for you page for the customer.

We recommend items to the customers based on their profile where we take into consideration their preferences by the items, they previously liked.

To create the vector representation for the process, we need to preprocess the data. Therefore, we use a cron job (task scheduler) that runs each day at 12am, it

preprocesses the information of the new products and saves it in the database so that we would not need to process it each time to save computation.

6.1.8. Technical Obstacles

While developing our system, we faced some obstacles that we tried to solve as best as we could.

First, to train the Outfit Compatibility classifier, the only dataset that we could find was polyvore dataset however this dataset is not labeled. However, as we stated in **5.1.1**, the outfits in polyvore are created by fashion experts therefore, we found that we could consider these outfits to be compatible outfits.

For the other class, we decided to generate random outfits from the items in the dataset. Since they are randomly generated, they will have a high probability of being incompatible and even if they were compatible, they would not be as good as outfits created by experts.

Second, in recommending matching items with the outfit, we found that it took a lot of time to predict the compatibility score for all the items in the given category. We solved this by taking a subset and then predicting the scores of the products in this subset only. If we got 50 items that have compatibility score above 50% then we would return these items to the user and stop.

However, if we do not have 50 items that have a compatibility score above the threshold, we then use the next subset from the products and repeat the process while appending the scores to the already calculated scores.

Lastly, we could not find a dataset that would be suitable for evaluating the personalized recommendation as H&M dataset is based on users' transactions not users' likings. So, we evaluated the system by plotting the recommendations and products of some of the users and comparing the results, just to make sure that the recommendations are similar.

When the system is deployed and used by users, we will plan to ask for their feedbacks about the given recommendations and also evaluate by the data that would be available in the system later on.

6.2. Testing

We tested our system from postman to test the APIs, the frontend and through automated tests for the backend to make sure that if the code is changed, the system stills run correctly.

6.2.1. Test Register Customer

Table 6-1 Register Customer Test Cases

Test Scenario	Prerequisites	Steps	Expected Result	Status
1. Customer register with valid data	---	1. Enter username. 2. Enter non-existed email. 3. Enter password. 4. Enter confirm password. 5. Enter phone number.	Customer's data are saved in database and the registration is done successfully	passed
2. Customer register with email already exists.	---	1. Enter username. 2. Enter used email. 3. Enter password. 4. Enter confirm password. 5. Enter phone number.	error message: "Email already exists." the registration failed.	passed
3. Customer register with incomplete data.	---	1. Enter username. 2. Leave email empty. 3. Enter password. 4. Enter confirm password. 5. Enter phone number.	error message: “Email should be provided.” the registration failed.	passed

1. Customer Register with Valid Data:

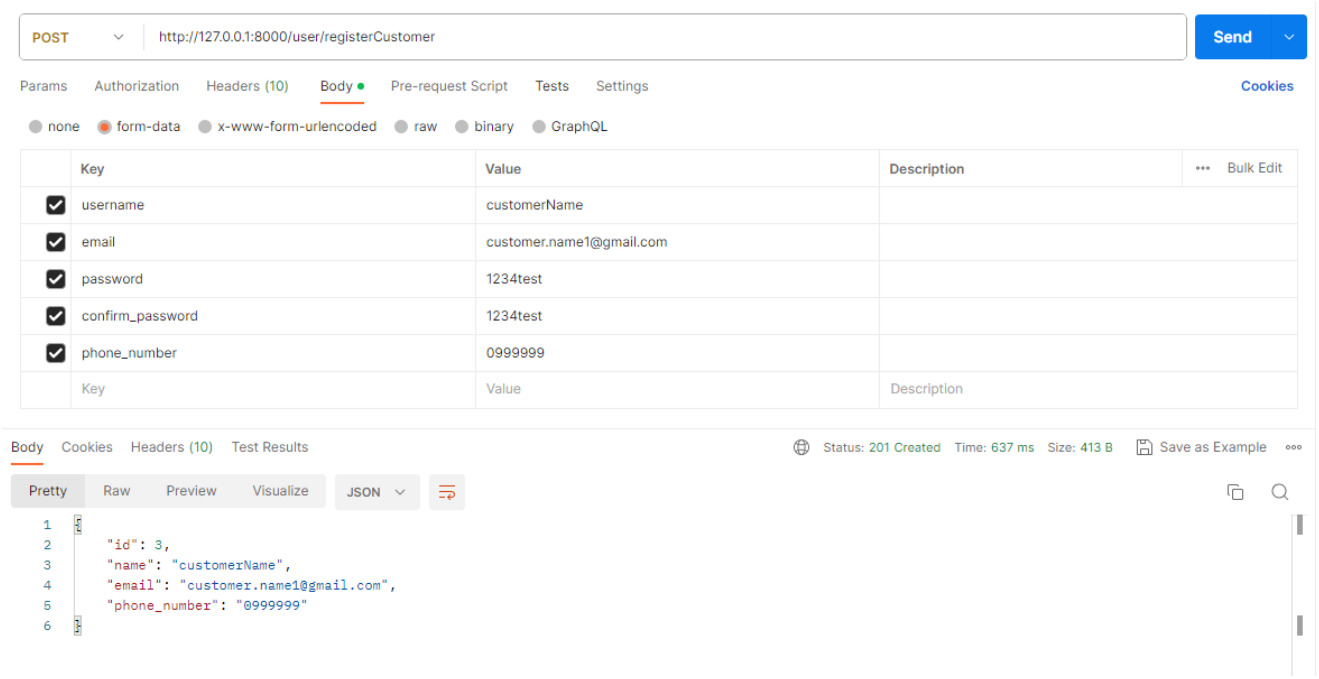


Figure 6-11 Postman Test Customer Register with Valid Data

2. Customer Register with Email already Exists:

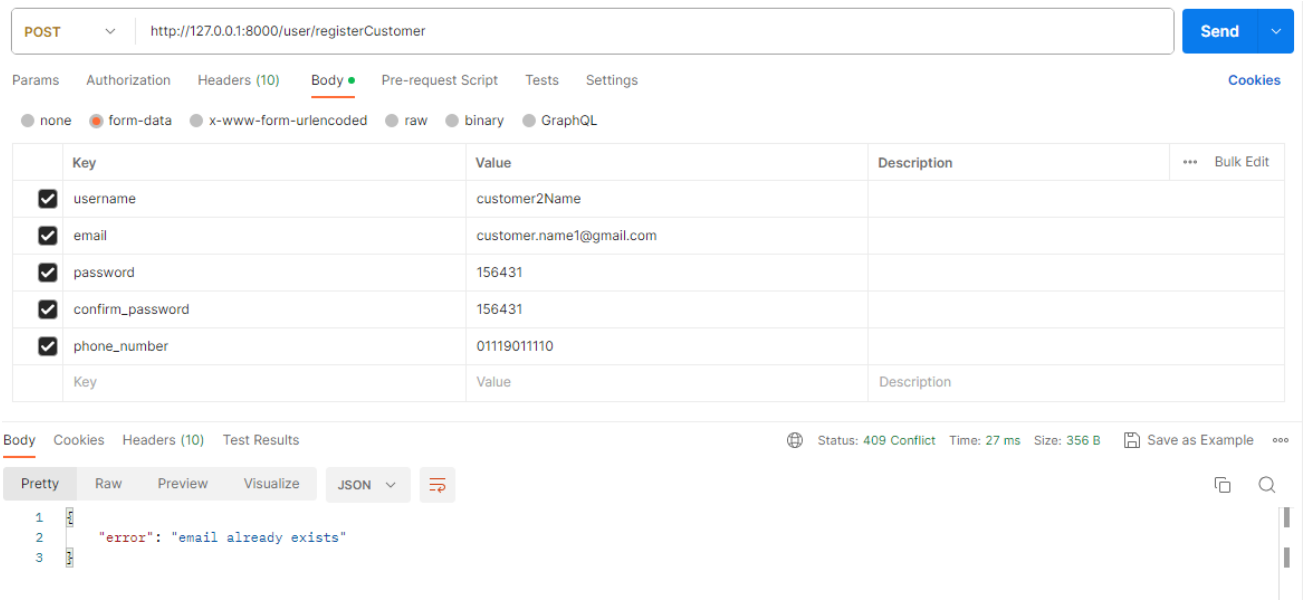


Figure 6-12 Postman Test for Customer Register with Email already Exists.

3. Customer Register with Incomplete Data:

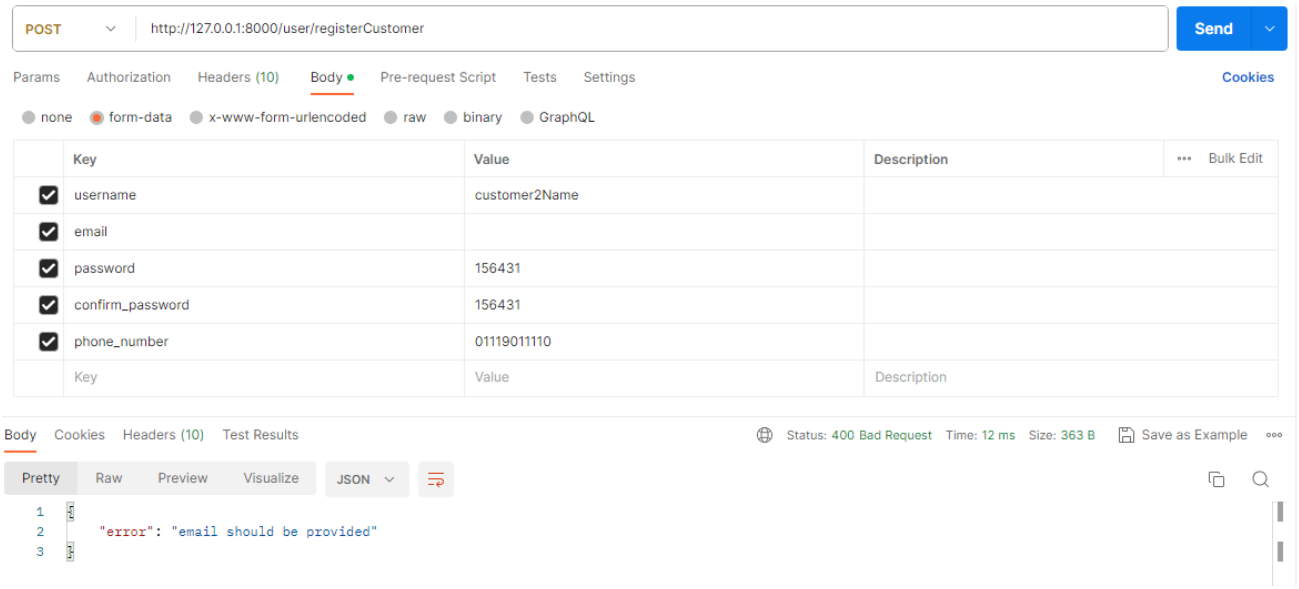


Figure 6-13 Postman Test for Customer Register with Incomplete Data.

6.2.2. Test Login Customer

Table 6-2 Login Customer Test Cases

Test Scenario	Prerequisites	Steps	Expected Result	Status
1. Customer login with valid data.	---	1. Enter email. 2. Enter password.	Customer logged in successfully.	passed
2. Customer login with wrong password.	---	1. Enter email. 2. Enter wrong password.	error message: "Password is incorrect." the Login failed.	passed
3. Customer login without registration.	---	1. Enter invalid email. 2. Enter password.	error message: “user does not exist.” the Login failed.	passed

1. Customer Login with Valid Data:

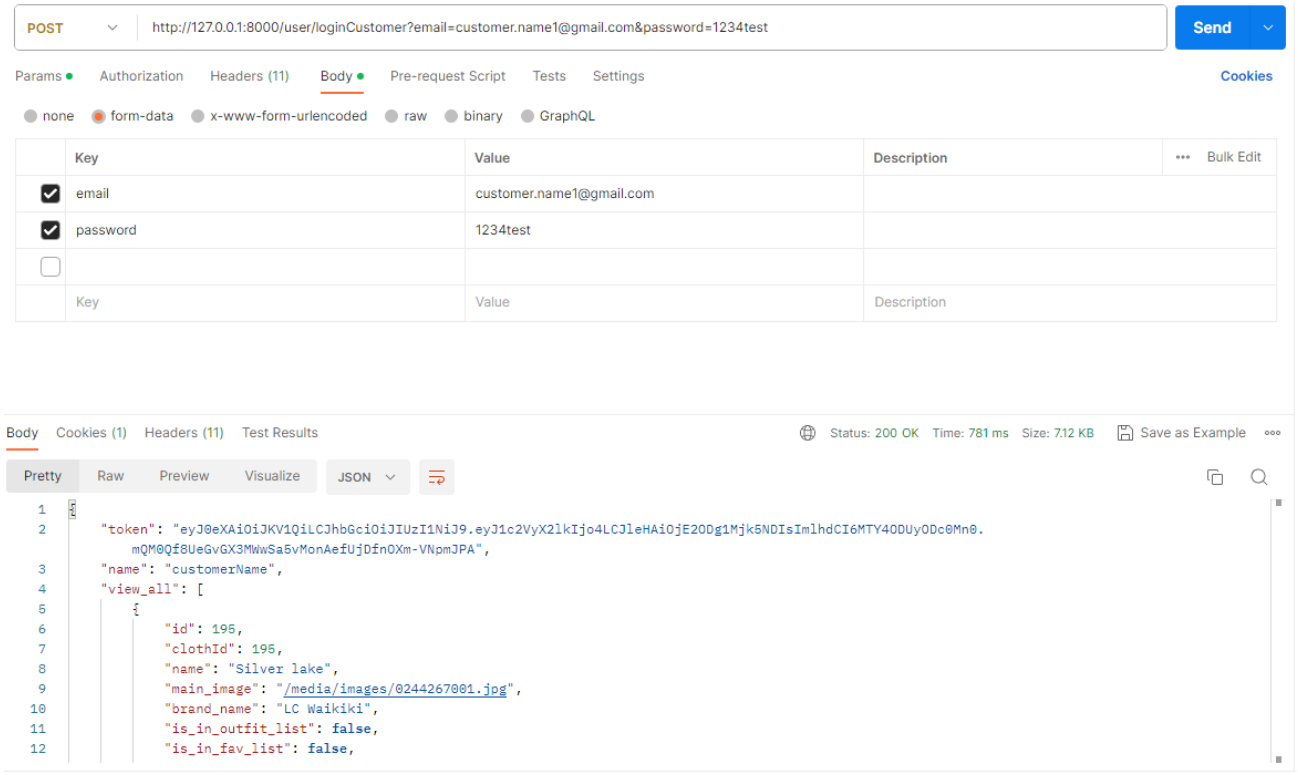


Figure 6-14 Postman Test for Customer Login with Valid Data

2. Customer Login with Wrong Password:

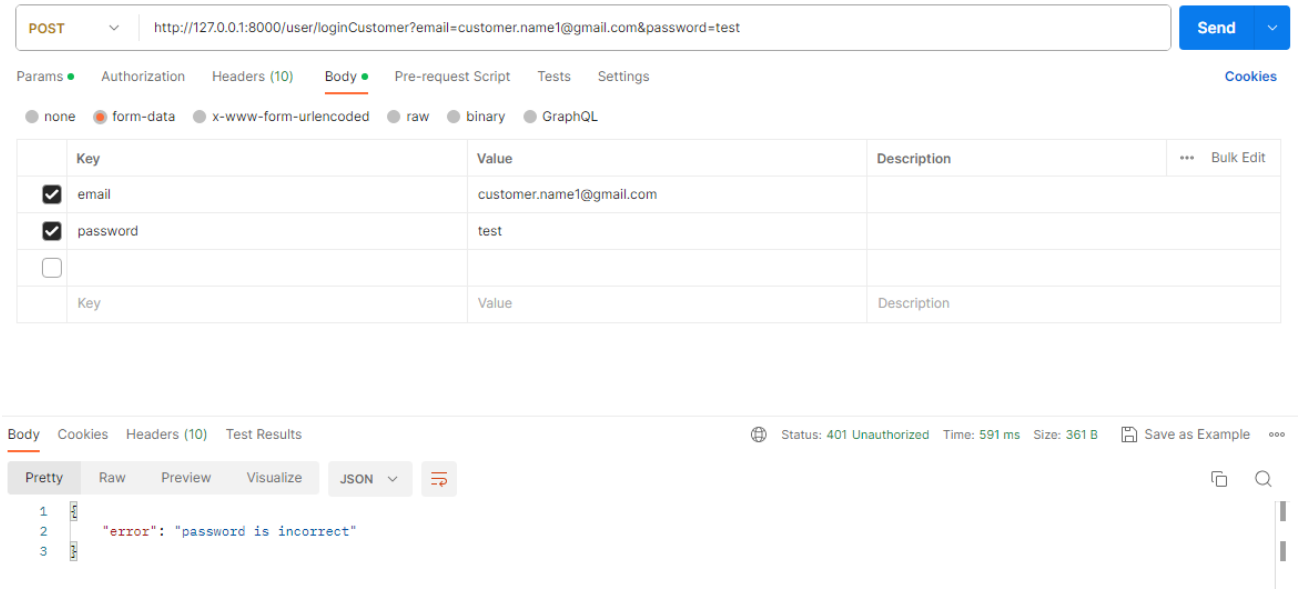


Figure 6-15 Postman Test for Customer Login with Wrong Password

3. Customer Login Without Registration:

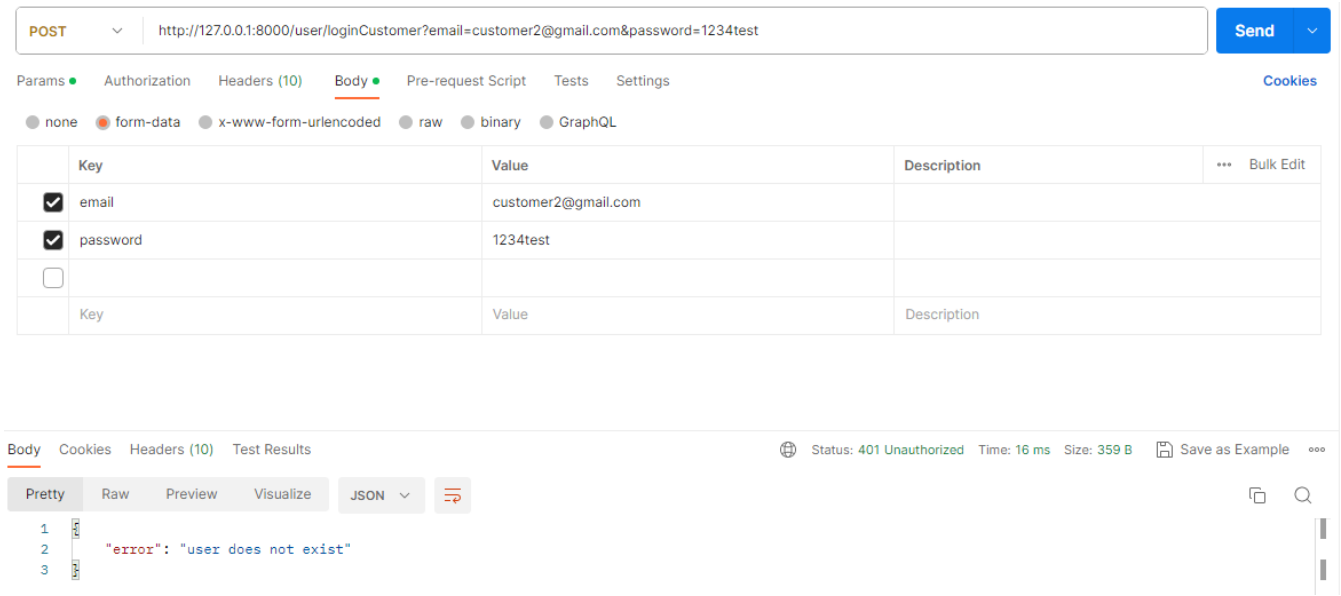


Figure 6-16 Postman Test for Customer Login Without Registration

6.2.3. Test Check Compatibility

Table 6-3 Check Compatibility Test Cases

Test Scenario	Prerequisites	Steps	Expected Result	Status
1. Get outfit compatibility score.	User logged in as customer	1. Add at least two items to the outfit. 2. Click button.	Compatibility score appear	passed
2. Check outfit compatibility no enough items.	User logged in as customer	1. Remove items. 2. Click button.	"You should have at least 2 items in the outfit" error message	passed

1. Get Outfit Compatibility Score (Front-End test)

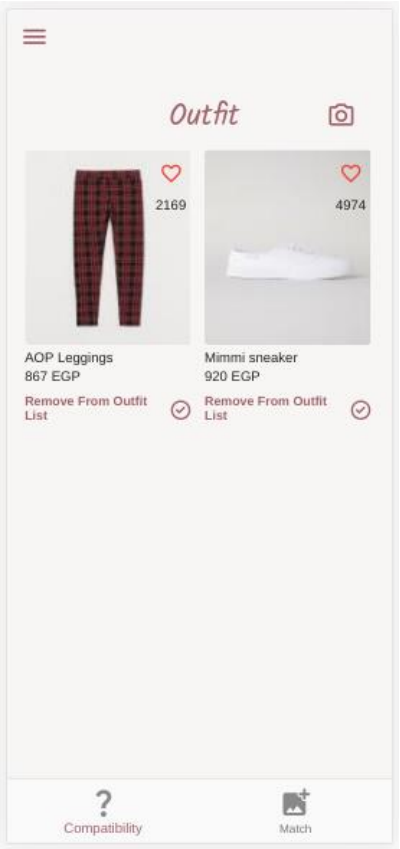


Figure 6-18 Outfit List for Get Outfit Compatibility Score Test.

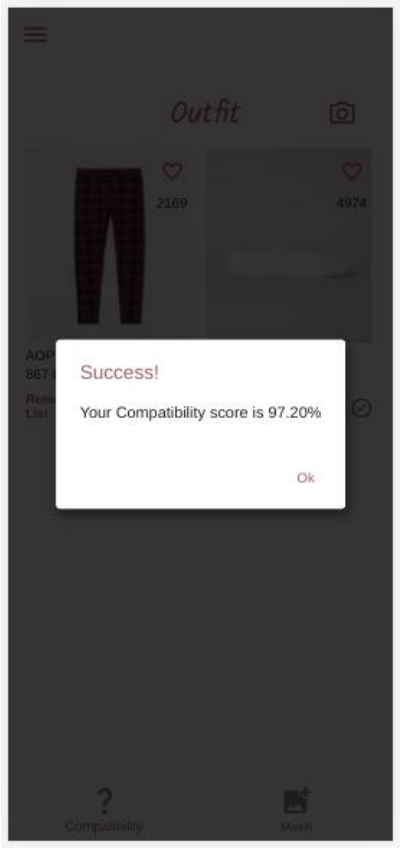


Figure 6-17 Outfit Score Output for Get Outfit Compatibility Score Test.

2. Check Outfit Compatibility no Enough Items Test:

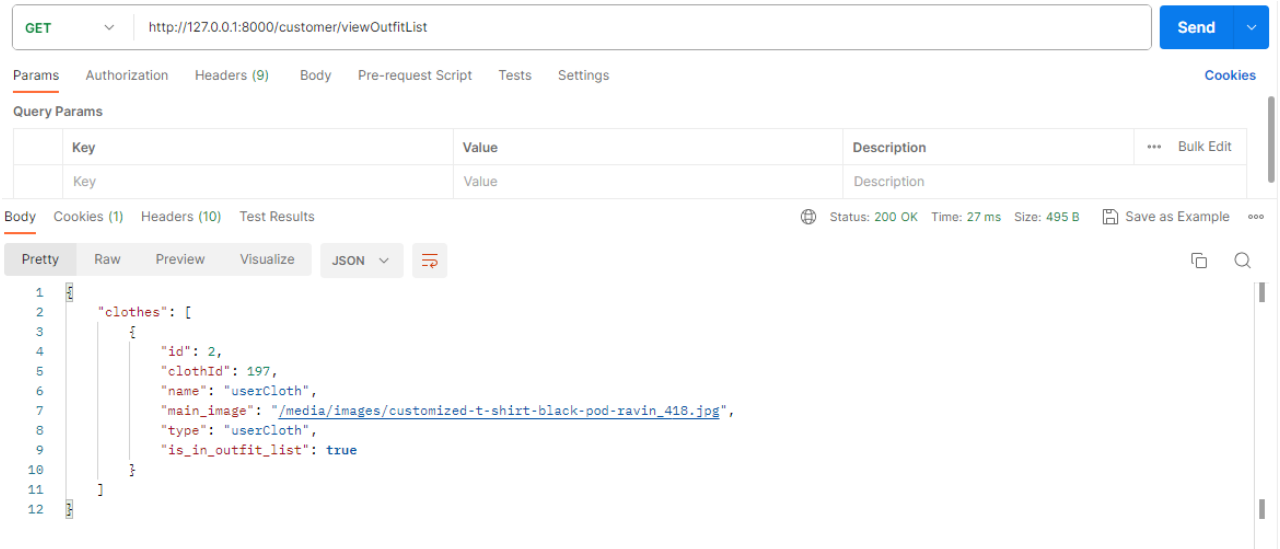


Figure 6-19 Outfit List for Check Outfit Compatibility no Enough Items Test.

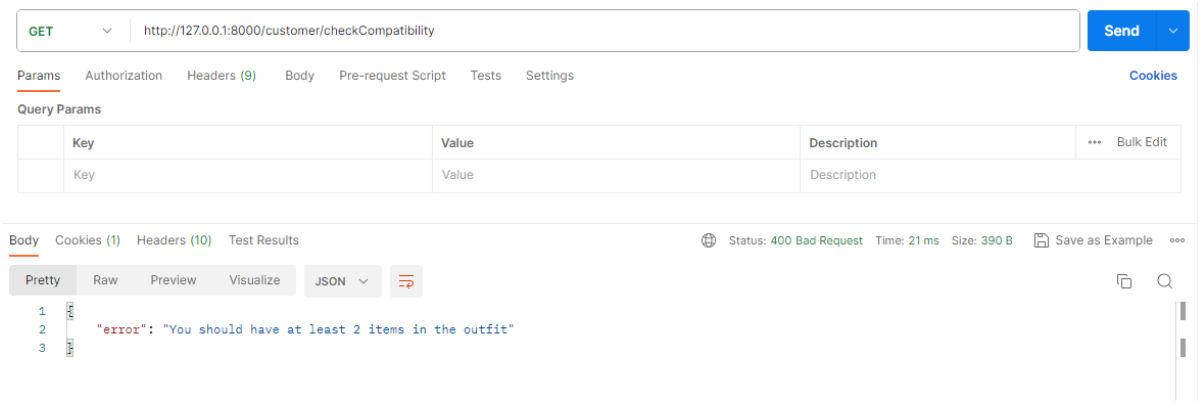


Figure 6-20 Output for Check Outfit Compatibility no Enough Items Test.

6.2.4. Test Bulk Upload

Table 6-4 Bulk Upload Test Cases

Test Scenario	Prerequisites	Steps	Expected Result	Status
1. Upload Data Valid.	User logged in as provider.	1. Upload zip file with the images of the products. 2. Upload the excel sheet	"All products added successfully" message	passed
2. Upload invalid product missing color.	User logged in as provider	1. Upload zip file with the images of the products. 2. Upload the excel sheet with the color info missing for a product	Valid products are added and an excel sheet with the product that has the problem is returned to the provider stating what the error.	passed
3. Upload no excel sheet.	User logged in as provider	1. Upload zip file with the images of the products.	“Please upload an excel sheet" error message	passed
4. Upload invalid products missing main image.	User logged in as provider	1. Upload zip file with the images of the products. 2. Upload the excel sheet with the main image missing for certain products	Valid products are added and an excel sheet with the products that has the problem is returned to the provider stating what the error.	passed

1. Upload Data Valid.

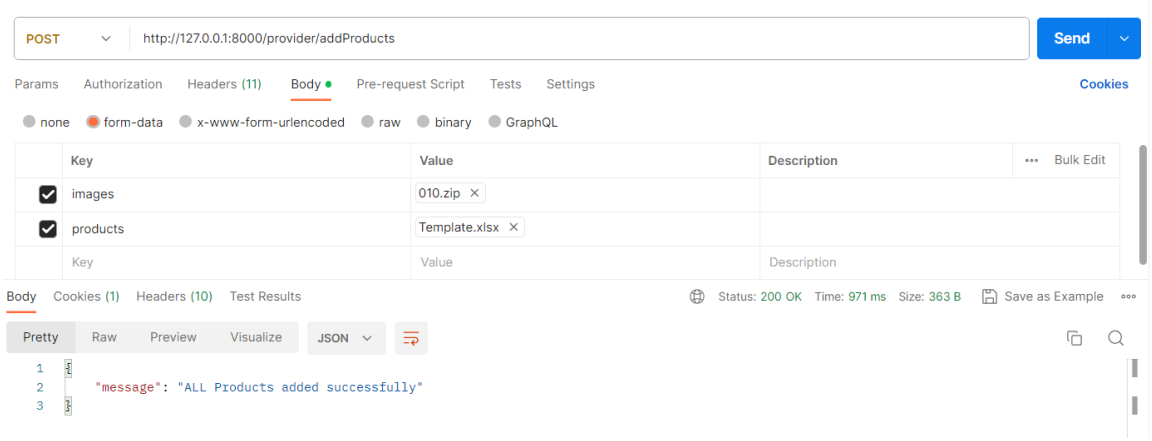


Figure 6-21 Postman test for Upload Valid Data

2. Upload Invalid Product Missing Color.(Front-End test)

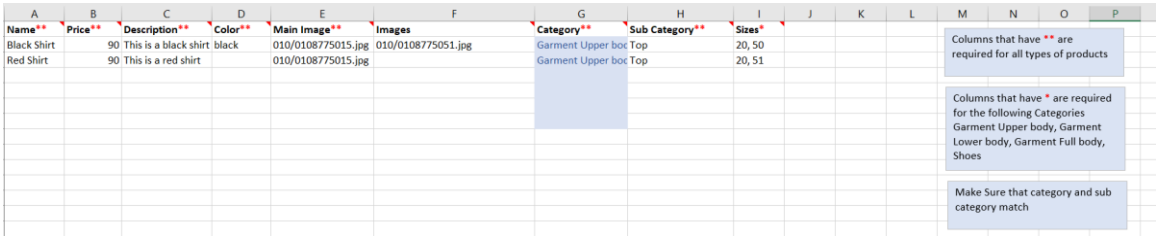


Figure 6-22 Input Excel Sheet for Upload Invalid Product Missing Color

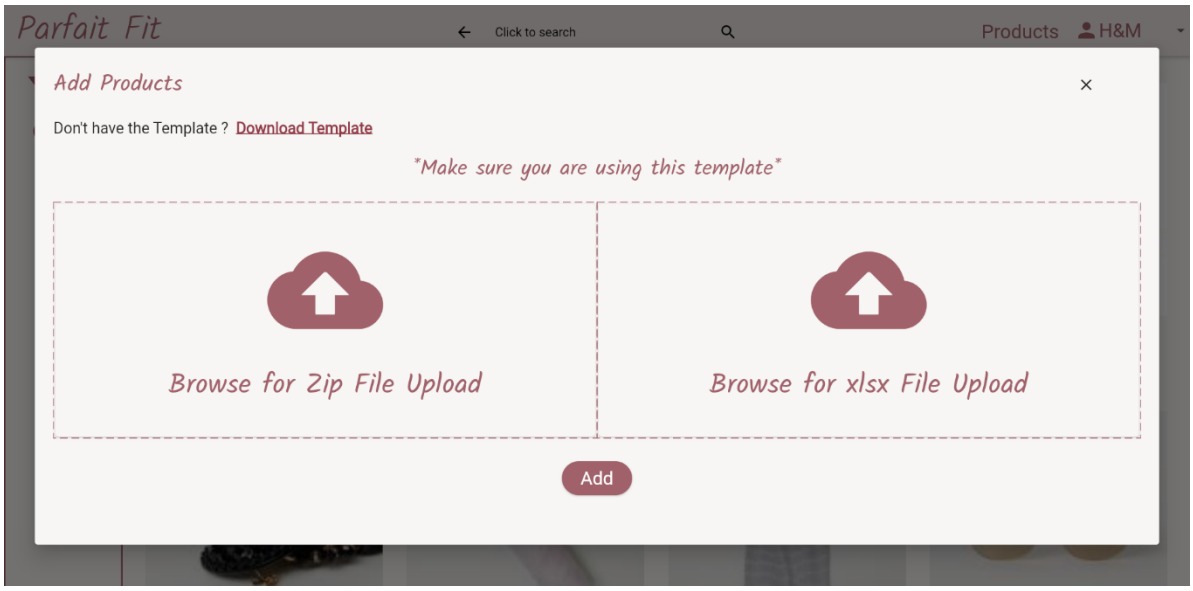


Figure 6-23 Upload Invalid Product Missing Color

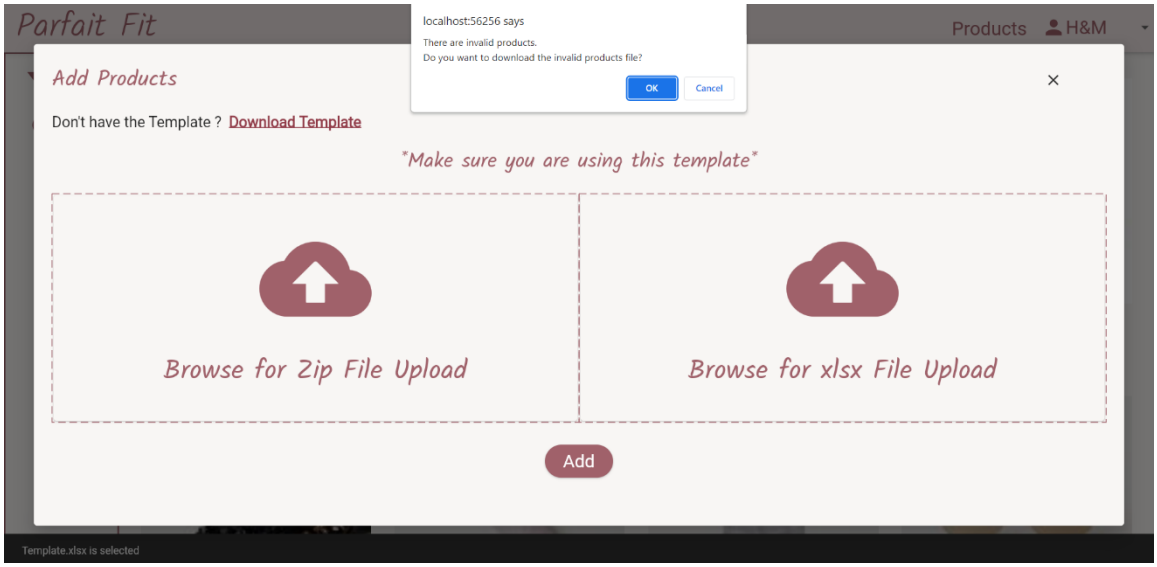


Figure 6-24 Pop up to Download the Error Report

Name**	Price**	Description**	Color**	Main Image**	Images	Category**	Sub Category**	Sizes*	Errors
Red Shirt	90	This is a red shirt		010/0108775015.jpg		Garment Upper body	Top	20, 51	Please enter color

Figure 6-25 Output Error Excel Sheet for Upload Invalid Product Missing Color.

3. Upload no Excel Sheet.

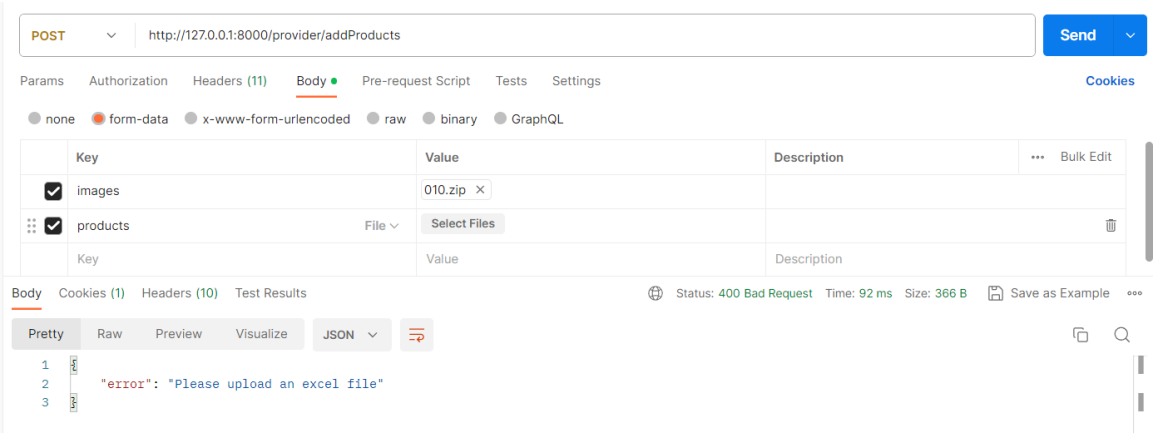


Figure 6-26 Postman Test for Upload no Excel Sheet.

4. Upload Invalid Products Missing Main Image.

Name**	Price**	Description**	Color**	Main Image**	Images	Category**	Sub Category**	Sizes*
Black Shirt	90	This is a black shirt	black	011/0108775015.jpg	011/0108775051.jpg	Garment Upper boc	Top	20, 50
Red Shirt	90	This is a red shirt		011/0108775015.jpg		Garment Upper boc	Top	20, 51

Columns that have ** are required for all types of products

Columns that have * are required for the following Categories Garment Upper body, Garment Lower body, Garment Full body, Shoes

Make Sure that category and sub category match

Figure 6-27 Input Excel Sheet for Upload Invalid Products Missing Main Image.

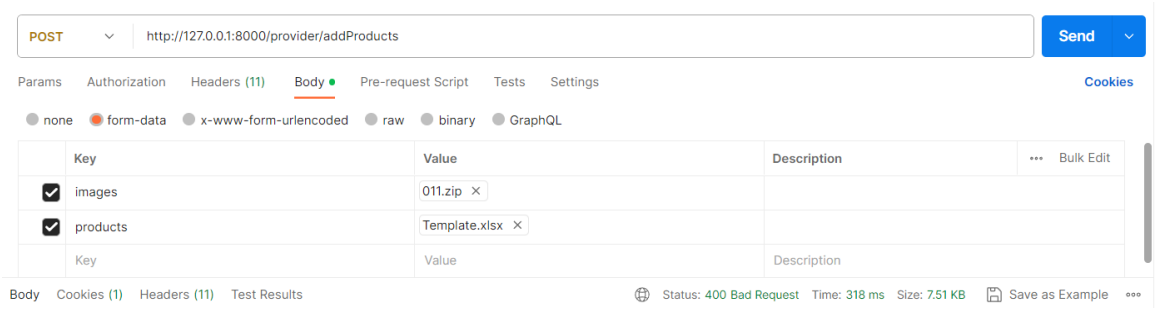
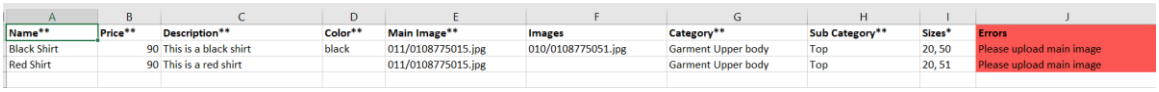


Figure 6-28 Postman Test for Upload Invalid Products Missing Main Image.



Name**	Price**	Description**	Color**	Main Image**	Images	Category**	Sub Category**	Sizes*	Errors
Black Shirt	90	This is a black shirt	black	011/0108775015.jpg	010/0108775051.jpg	Garment Upper body	Top	20, 50	Please upload main image
Red Shirt	90	This is a red shirt		011/0108775015.jpg		Garment Upper body	Top	20, 51	Please upload main image

Figure 6-29 Output Error Excel Sheet for Upload Invalid Products Missing Main Image.

6.2.5. Test Upload Item to Outfit

Table 6-5 Upload Image to Outfit Test Cases

Test Scenario	Prerequisites	Steps	Expected Result	Status
1. Upload Item with category doesn't exist in the outfit.	User logged in as customer.	1. Upload image. 2. Choose the category of the image.	Item is successfully added to the outfit list.	passed
2. Upload Item with category exists in the outfit (with isReplace = true).	User logged in as customer.	1. Upload image. 2. Choose the category of the image.	Item replaces the one with the same category and added to the outfit list.	passed
3. Upload Item with category exists in the outfit (with isReplace = false).	User logged in as customer.	1. Upload image. 2. Choose the category of the image.	"There exists item of same category" error message	passed

1. Upload Item with Category doesn't Exist in the Outfit.


```
"clothes": [
    {
        "id": 113,
        "clothId": 113,
        "name": "Beyonce cropped tank",
        "main_image": "/media/images/0218354047.jpg",
        "brand_name": "H&M",
        "is_in_outfit_list": true,
        "is_in_fav_list": false,
        "price": 595.0,
        "description": "Cropped vest top in slub jersey.",
        "color": "Off White",
        "num_of_likes": 3886,
        "type": "product"
    },
    {
        "id": 42,
        "clothId": 242,
        "name": "userCloth",
        "main_image": "/media/images/white_pants.jpg",
        "type": "userCloth",
        "is_in_outfit_list": true
    }
]
```


Figure 6-31 Outfit List for Upload Item with Category doesn't Exist in the Outfit. (Before)


```
"clothes": [
    {
        "id": 113,
        "clothId": 113,
        "name": "Beyonce cropped tank",
        "main_image": "/media/images/0218354047.jpg",
        "brand_name": "H&M",
        "is_in_outfit_list": true,
        "is_in_fav_list": false,
        "price": 595.0,
        "description": "Cropped vest top in slub jersey.",
        "color": "Off White",
        "num_of_likes": 3886,
        "type": "product"
    },
    {
        "id": 42,
        "clothId": 242,
        "name": "userCloth",
        "main_image": "/media/images/white_pants.jpg",
        "type": "userCloth",
        "is_in_outfit_list": true
    },
    {
        "id": 44,
        "clothId": 244,
        "name": "userCloth",
        "main_image": "/media/images/heels.jpg",
        "type": "userCloth",
        "is_in_outfit_list": true
    }
```


Figure 6-30 Outfit List for Upload Item with Category doesn't Exist in the Outfit. (After)

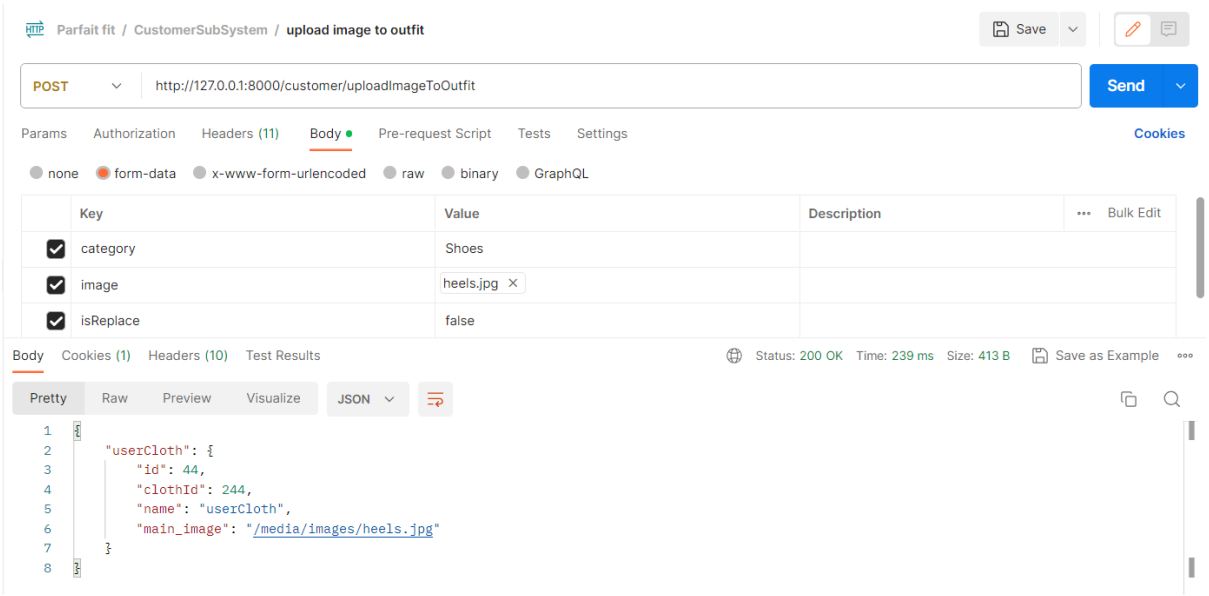


Figure 6-32 Postman test for Upload Item with Category doesn't Exist in the Outfit.

2. Upload Item with Category Exists in the Outfit, Customer chooses to replace item. (Front-End test)

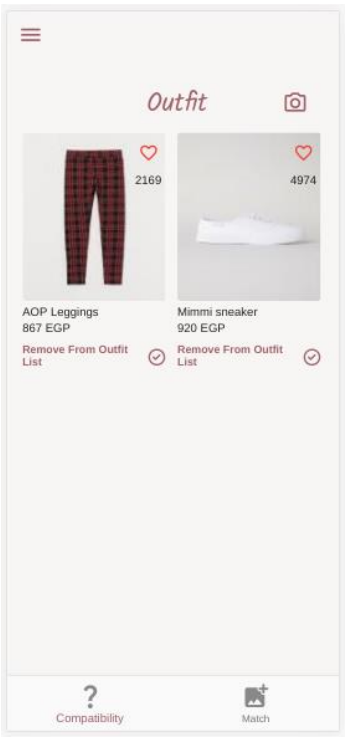


Figure 6-33 Test Case 2 - Outfit List (Before)

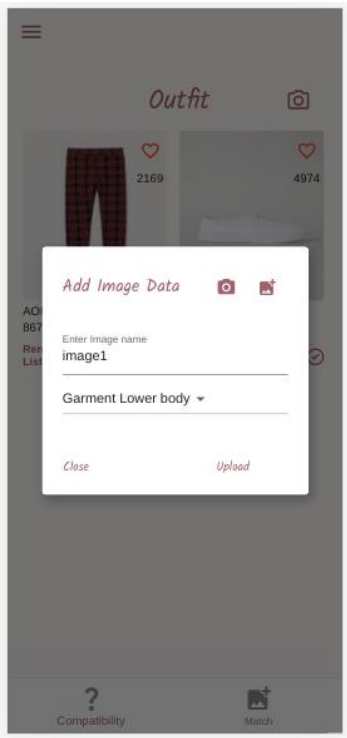


Figure 6-34 Uploading Item with Category Exists in the Outfit.

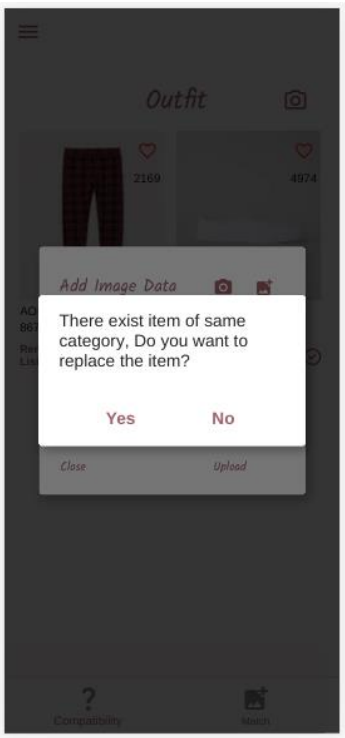


Figure 6-36 Choose to Replace Item. (Click Yes)

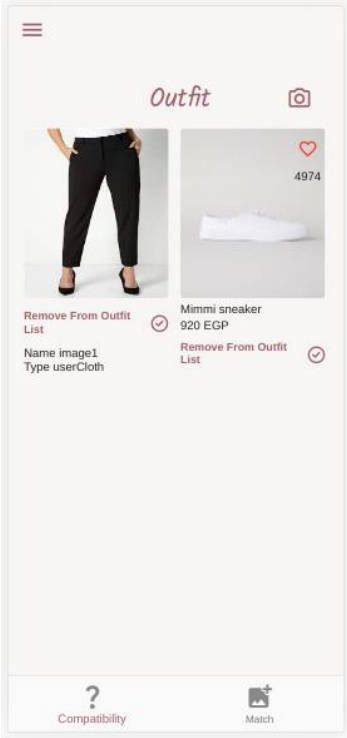


Figure 6-35 Test Case 2 - Outfit List (After)

3. Upload Item with Category Exists in the Outfit (with isReplace = false).

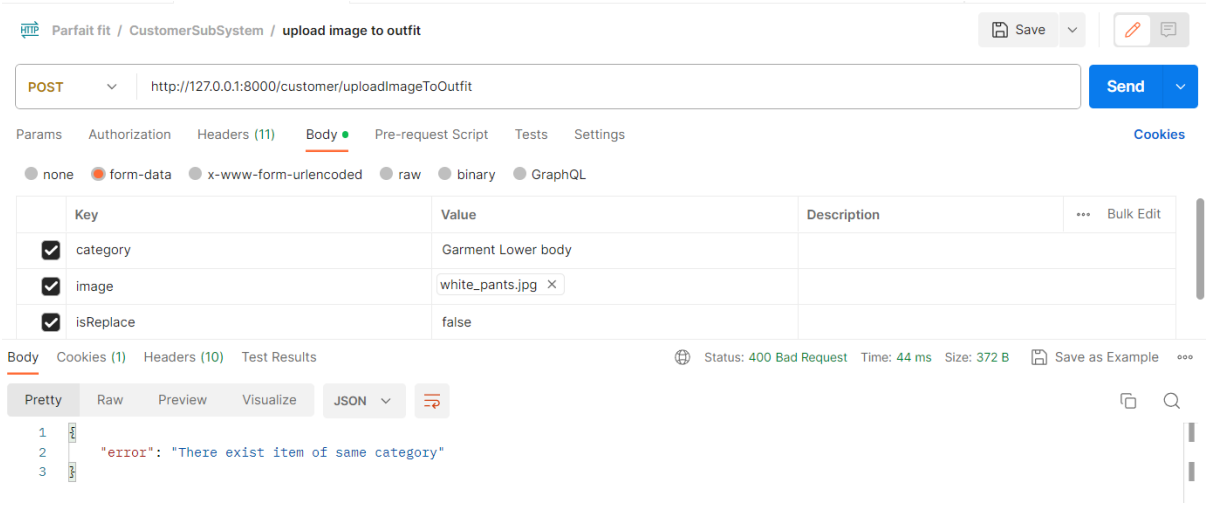


Figure 6-37 Postman Test for Upload Item with Category Exists in the Outfit (with isReplace = false).

Future Works

We plan to extend our application and add more features in the future, making it more helpful to people regarding the fashion industry, by taking into consideration the outfit's suitability on them, using characteristics such as body shape, skin color, height, and weight to detect whether it will fit the user or not.

We would also like to support a more diverse type of users; men, children, and hijabis in our system. Our aim is to facilitate the whole shopping process by not just telling them the outfit's compatibility but by also allowing them to buy clothes and try it on them. Therefore, we would like to add features such as shopping and virtual try-on as well as visual search.

Lastly, we would like to add auto tagging on the products uploaded by the providers and images uploaded by the users to make sure that they are suitable and classify the users' images into categories.

Conclusion

The way we dress, wearing a stylish and well ensembled outfit, has a significant role in boosting confidence. However, a stylish and nice look is difficult to achieve for some people, they would need other people's opinion and help, and this is not always an available solution. Parfait fit will be an on-hand, pocket-size stylist that is always available and easy to access. People can consult parfait fit regarding their outfit, getting feedback about how compatible it is and getting recommendation of items that will match their outfit.

Just as it helps ordinary people, it also helps clothing providers and brands to gain more profit and exposure. By allowing them to upload their products on the system so that users will be able to look through them. Parfait fit will also recommend their products to the users which will help them have more exposure and profit.

References

[1] Das, S. (2021, July 26). Amazon Fashion Discovery Engine (content-based recommendation). Medium. https://medium.com/mlearning-ai/amazon-fashion-discovery-engine-content-based-recommendation-19d63563ef01

[2] Han, X., Wu, Z., Jiang, Y.-G., & Davis, L. S. (2017). Learning fashion compatibility with bidirectional lstms. Proceedings of the 25th ACM International Conference on Multimedia. https://doi.org/10.1145/3123266.3123394

[3] Hasija, S. (2022, February 8). H&M-(128X128) dataset. Kaggle. https://www.kaggle.com/datasets/odins0n/handm-dataset-128x128

[4] Meilisearch Documentation. Retrieved from www.meilisearch.com

[5] Moosaei, M., Lin, Y., & Yang, H. (2020). Fashion Recommendation and Compatibility Prediction Using Relational Network. arXiv preprint arXiv:2005.06584.

[6] Personal outfit recommendations. Lookastic. (n.d.). Retrieved from https://lookastic.com/

[7] Personalized outfit recommendation solution for eCommerce. Vue.ai. (n.d.). Retrieved January 30, 2023, from https://vue.ai/products/outfit-recommendations/

[8] Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., & Lillicrap, T. (2017). A simple neural network module for relational reasoning. Advances in neural information processing systems, 30.

[9] Style advisor - ai-driven outfit recommendation by Wide Eyes. WIDE EYES TECHNOLOGIES. (2019, April 5). Retrieved from https://wideeyes.ai/style-advisor/

[10] Toprak, M. (2020a, April 26). Content-based Recommender System. Medium. https://medium.com/@toprak.mhmt/content-based-recommender-system-bdfc60b1bee8