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

Parfait Fit

Supervised by
Dr. Cherry Ahmed

TA. Maya

Implemented by

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

Graduation Project
Academic Year 2022-2023
Midyear Short Documentation

Table of Contents

List Of Figures

1. Abstract

Choosing the perfect outfit is not the easiest thing to do for most of us. Usually, people struggle to find items that complete their outfits for them and to find pieces of clothes that are compatible. Our main goal is to help people with searching for items that they want and matching suitable, elegant, and simple outfits depending on any piece of clothes they have or displayed by any brand.

For building this system, we will use a large dataset of clothes to build a model using CNN and neural network. This model will extract features of any added piece of clothes then predict the compatibility between the clothes, taking into consideration many factors such as fabrics and colors, etc.

This prediction and matching will be based on real items from stores, that are displayed on the application, and clothes uploaded by the user.

2. Background

2.1. Main Area

A place for brands and customers. Brands get to display their products for users. Users view the items uploaded by brands, search for the items and get recommendations. They can also form outfits either by uploading images of their clothes or from the items displayed by brands. They can then check the compatibility of the outfit or get recommendations for an item that matches the outfit.

The main area of the application is convolutional neural network (CNN) and deep learning. CNN will be used to extract the features from an image, while deep learning will be used to build a compatibility predictor model that will be used in checking compatibility of the outfit and recommendation features.

2.2. Motivation and beneficiary

Upon conducting a survey of people with ages ranging between 12 - 40+, we found that 71% of people face a problem 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.

Our main goal is to build an application to facilitate all these difficulties, improve convenience for the users and help them style better. Also, we would like to help brands by letting them advertise their products (upload them) on the app and giving the users recommendations from these items. This will help brands increase their sales/profit.

2.3. Main Techniques, Technologies, and Application

The technologies and techniques and their respective applications that are used to build the system are:

1. **Django Rest Framework**

   It is a toolkit that provides features that enable us to build Restful API. It provides serialization, authentication, authorization as well as security.
   Application: Backend of the system

2. **Flutter**

   An open-source framework developed by Google. It is used for building multi-platform applications from the same code. Therefore, we could use it to develop an application that runs on both android and IOS with the same code. It can also be used for websites' user interface development.
   Application: Frontend for both mobile and web.

3. **PostgreSQL**

   It is an open-source object relational database management system. It supports application features such as JSON, XML as well as user-defined types. It is suitable for large systems or systems that have large dataset also, it is secure, reliable and is fast for both read and write operations.
   Application: Database management and queries.

4. **Redis**

   It is an open-source in-memory data store platform and can be used as a cache or database. It is known for being simple and its ease of use.
   Application: Cache.

5. **Deep Learning**

   It is a type of machine learning and neural networks that consist of 3 or more layers. It simulates the human brain in thinking and could be either supervised or unsupervised. However, we will use a supervised technique where the dataset is labelled.
   Application: Compatibility predictor

6. **Convolutional neural network (CNN)**

   CNN is a deep learning network that takes images as its input. It consists of convolution layers, pooling, and then a fully connected neural network where the decision is made.
   Application: Feature extractor

2.4. Models

Machine learning models we will build for the system:

1. **Compatibility Prediction**

1.1. Feature Extractor

A CNN model which contains multiple convolutional layers takes images either uploaded by brands to the system or by customers as input and extracts features from them including color, fabric, etc. The images uploaded by the customer are images of separate pieces of clothes to form an outfit.

The extracted features are saved in a feature vector which is passed to a layer that performs pairwise compatibility measurement for every pair combination of the outfit, returning pairwise compatibility scores.

1.2. Full Outfit Compatibility Predictor

A fully connected neural network which takes the output of the compatibility layer that computed pairwise compatibility scores and computes the compatibility score of the whole outfit.

The computed compatibility is displayed for users as the score for the created outfit.

2. Recommendation Model

❖ Non-Personalized recommendation:

Customers will get recommendations:

- The most liked items are displayed on their homepage.
- The top K compatible items, of a category chosen by the user and that are compatible with the outfit (consists of one or more items) which is formed by them.

  This recommendation will be based on generating a random subset of the items existing in the system and of the chosen category. The random subset is of size N.

  The N outfits are formed with the items in this subset, then the compatibility predictor will run on them. The items that get a score higher than a certain threshold are chosen. This algorithm is repeated till we get top K recommended items or reach certain number of iterations.

➔ **Dataset**

We will use a clean version of Polyvore [5] dataset, The original dataset has 21K expertise selected outfits where each item in the outfit has a category, price, and name.

Upon viewing the results of the survey, we conducted, it was apparent that females are slightly more interested in the app, so we've decided to make the application targeting females and to avoid high computation we've decided to drop males' outfits.

In the clean version the categories will be filtered by dropping non-fashion related categories, grouping overlapping categories and classifying them into 5-6 types (top, bottom, shoes, bag, accessory, etc.). We will ensure that each outfit in the dataset will have 3 – 5 items of different types.

3. Problem definition

Based on the survey we conducted, we found that people face a daily problem which is deciding what to wear and making sure that their outfit is compatible and find this problem time-consuming.

When shopping online or offline, it is hard for people to determine if an outfit they want to buy is compatible or not, if a certain piece will be compatible with an old incomplete outfit they have or match a piece with their outfit.

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 have to be bought by brands in order to be used. So, we decided to build an application that will solve all these problems and gather all these features in one free application.

4. Related work

❖ There are multiple websites with similar features as

1. Wideesyes.ai [1]: Requires requesting a demo to use, 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.
2. Lookastic [2]: 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 [3]: Requires requesting a demo to use, 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.

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

Table 4.1 – 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	✓	✗	✗	✓

❖ Main Differences between our system and theirs:

It's free unlike most of them that require payment for demo use, and it is not designed for a certain shop to use, it allows multiple brands to upload their products and allows users to explore more brands and find items that are like what they want over multiple brands.

Unlike other systems that either recommend outfits based on pre-existing themes, we recommend pieces that go with the formed outfit in terms of compatibility, the outfit is either formed of a combination of items of different brands or outfits uploaded by users from their wardrobe.

5. Project Specifications

5.1. 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.

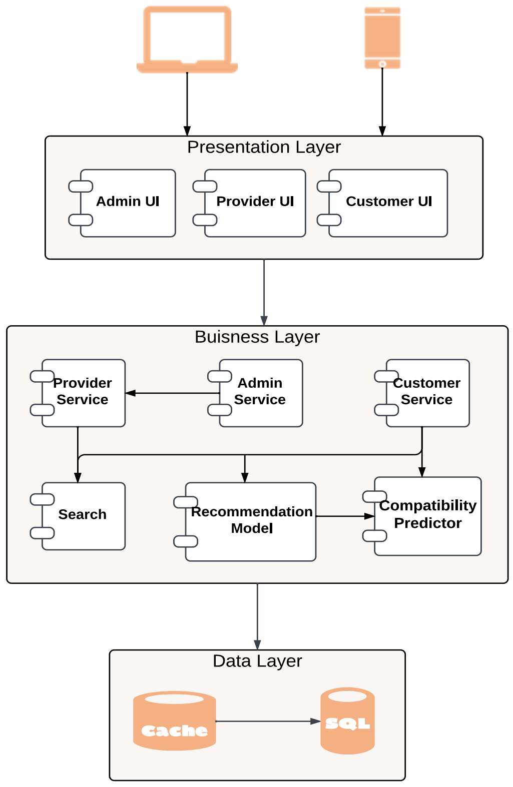


Figure 5.1 - System Architecture

5.2. Stakeholders

1. **End users**
   1. Females who need fashion advice to know if an outfit is compatible or not before going out or when buying it at the store or exploring online, match their old clothes with a new one to create a compatible outfit or get recommended items.
   2. Shop owner /provider who wants to show his products (clothes) to the user to buy them.

   3. Admins who verify and suspend users.

2. **Developers**
   Collect the data needed for the system, design, build, deploy, and test the system to be ready for users.

5.3. 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. Upload an image of a clothing piece to add to an outfit.
6. Create an outfit from items from different brands in the application gallery.
7. Determine if a given outfit is compatible or not.
    - Return a score of the compatibility test.
8. Get recommendation of an item compatible with a given outfit.
9. Recommend clothes. (Most favorable clothes)
10. Search by words for clothes.
11. Filter the clothes by category, price, brand, and size.
12. 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. Upload images of his products. (Add image)
4. Edit or delete images. (His 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.

5.4. 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.

5.5. Use-case Diagram

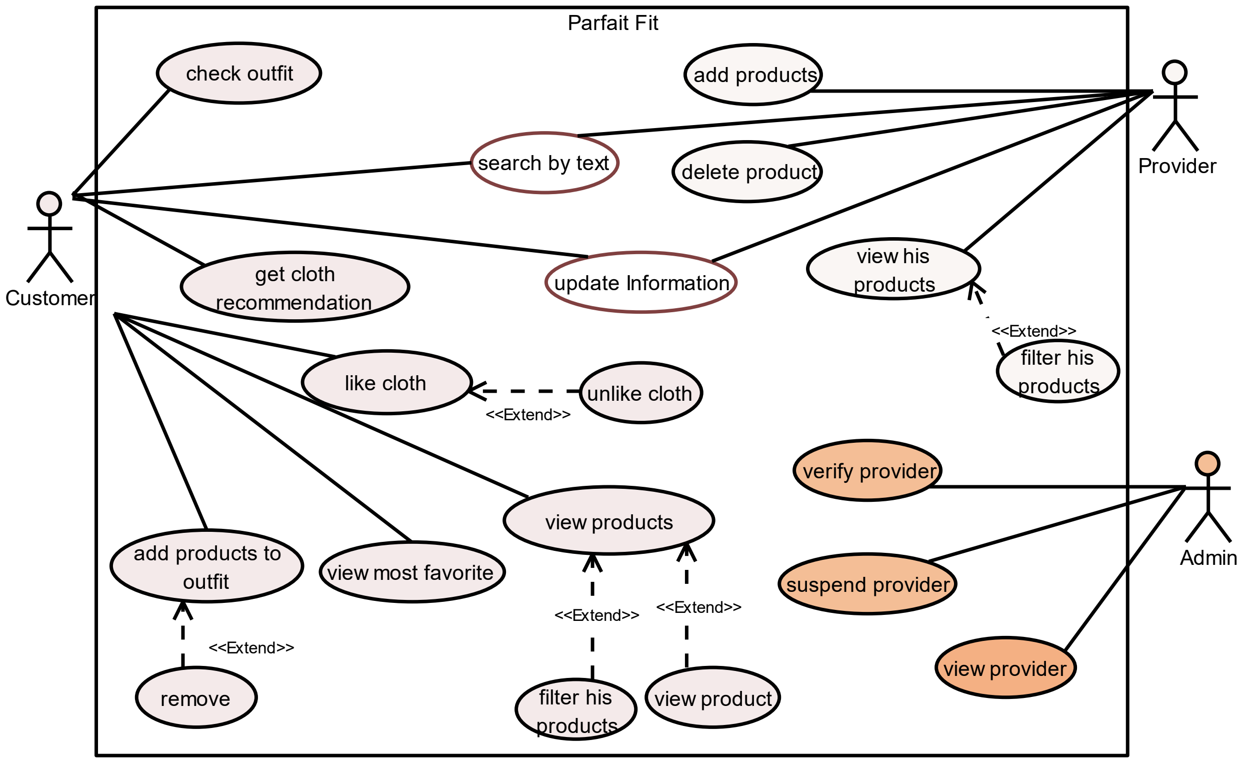


Figure 5.2 – Use Case Diagram

5.6. Entity Relationship Diagram (ERD)

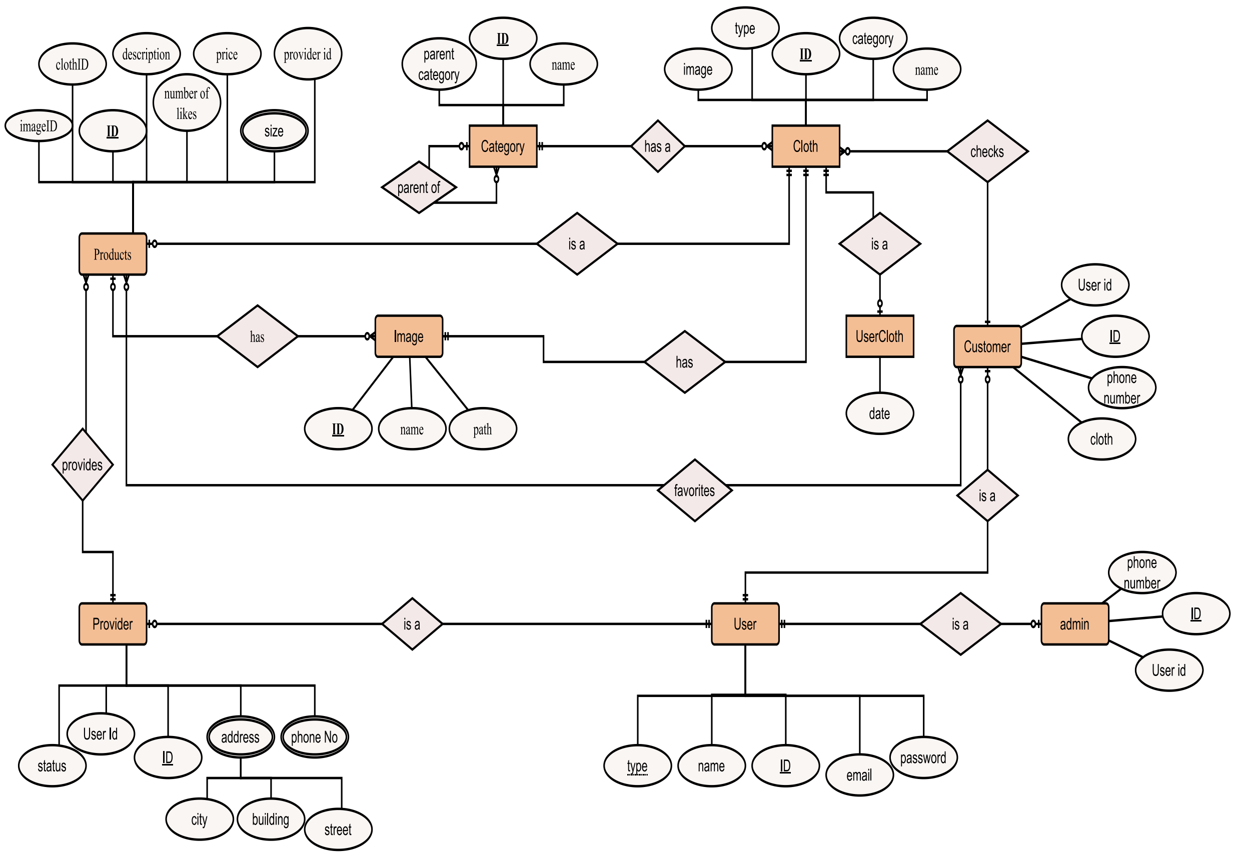


Figure 5.3 – ERD

5.7. Class Diagram

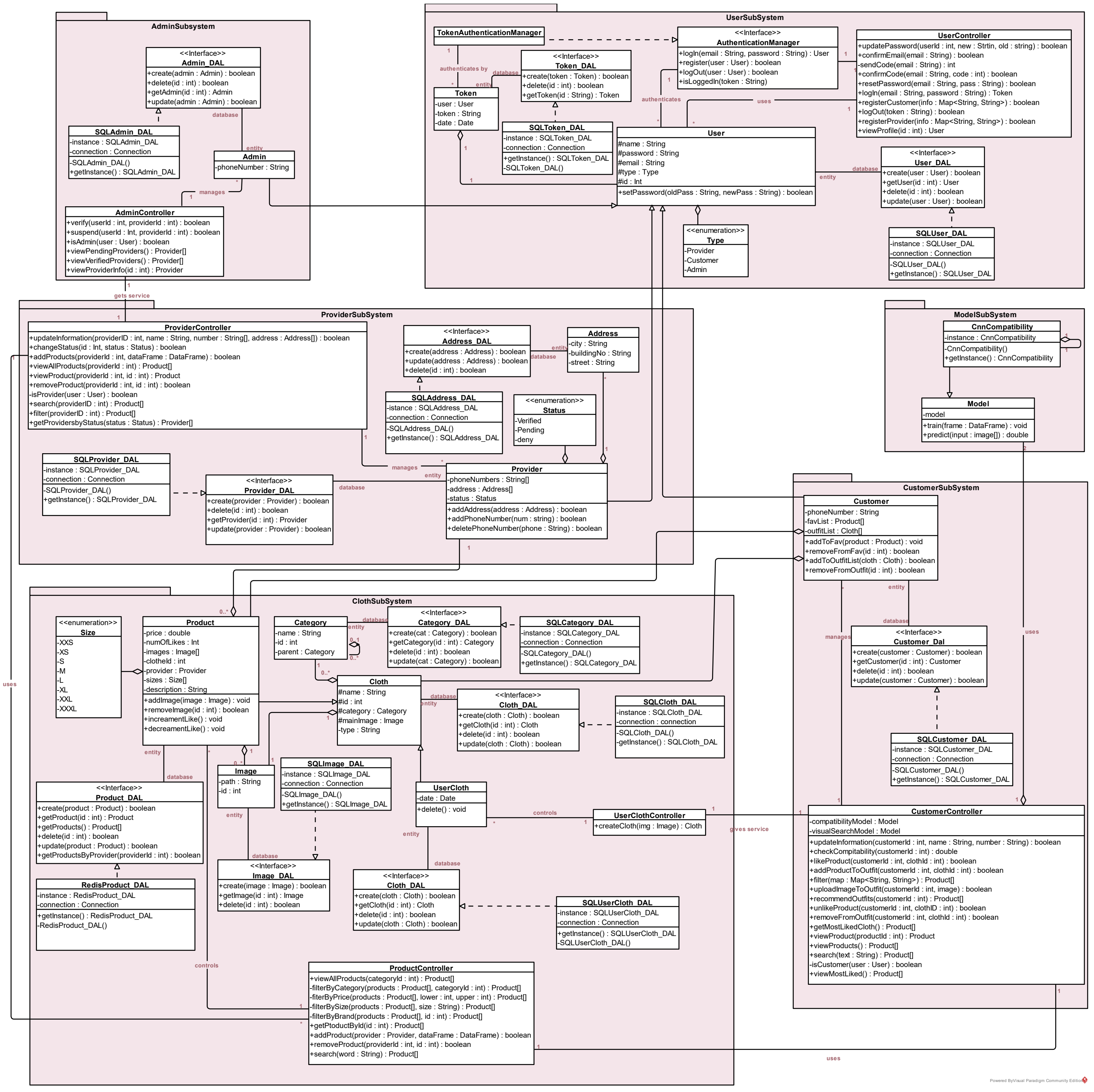

AdminSubsystem
<<Interface>> Admin_DAL
+create(admin : Admin) : boolean
+delete(id : int) : boolean
+getAdmin(id : int) : Admin
+update(admin : Admin) : boolean
SQLAdmin_DAL
-instance : SQLAdmin_DAL
-connection : Connection
-SQLAdmin_DAL()
+getInstance() : SQLAdmin_DAL
Admin
-phoneNumber : String
database
entity
manages
AdminController
+verify(userId : int, providerId : int) : boolean
+suspend(userId : Int, providerId : int) : boolean
+isAdmin(user : User) : boolean
+viewPendingProviders() : Provider[]
+viewVerifiedProviders() : Provider[]
+viewProviderInfo(id : int) : Provider
gets service
UserSubSystem
TokenAuthenticationManager
authenticates by
<<Interface>> Token_DAL
+create(token : Token) : boolean
+delete(id : int) : boolean
+getToken(id : String) : Token
Token
-user : User
-token : String
-date : Date
SQLToken_DAL
-instance : SQLToken_DAL
-connection : Connection
+getInstance() : SQLToken_DAL
-SQLToken_DAL()
<<Interface>> AuthenticationManager
+logIn(email : String, password : String) : User
+register(user : User) : boolean
+logOut(user : User) : boolean
+isLoggedIn(token : String)
authenticates
uses
UserController
+updatePassword(userId : int, new : Strtin, old : string) : boolean
+confirmEmail(email : String) : boolean
-sendCode(email : String) : int
+confirmCode(email : String, code : int) : boolean
+resetPassword(email : String, pass : String) : boolean
+logIn(email : String, password : String) : Token
+registerCustomer(info : Map<String, String>) : boolean
+logOut(token : String) : boolean
+registerProvider(info : Map<String, String>) : boolean
+viewProfile(id : int) : User
User
#name : String
#password : String
#email : String
#type : Type
#id : Int
+setPassword(oldPass : String, newPass : String) : boolean
<<Interface>> User_DAL
+create(user : User) : boolean
+getUser(id : int) : User
+delete(id : int) : boolean
+update(user : User) : boolean
<<enumeration>> Type
-Provider
-Customer
-Admin
SQLUser_DAL
-instance : SQLUser_DAL
-connection : Connection
-SQLUser_DAL()
+getInstance() : SQLUser_DAL
ProviderSubSystem
ProviderController
+updateInformation(providerID : int, name : String, number : String[], address : Address[]) : boolean
+changeStatus(id : Int, status : Status) : boolean
+addProducts(providerId : int, dataFrame : DataFrame) : boolean
+viewAllProducts(providerId : int) : Product[]
+viewProduct(providerId : int, id : int) : Product
+removeProduct(providerId : int, id : int) : boolean
-isProvider(user : User) : boolean
+search(providerID : int) : Product[]
+filter(providerID : int) : Product[]
+getProvidersbyStatus(status : Status) : Provider[]
<<Interface>> Address_DAL
+create(address : Address) : boolean
+update(address : Address) : boolean
+delete(id : int) : boolean
Address
-city : String
-buildingNo : String
-street : String
SQLAddress_DAL
-istance : SQLAddress_DAL
-connection : Connection
-SQLAddress_DAL()
+getInstance() : SQLAddress_DAL
<<enumeration>> Status
-Verified
-Pending
-deny
SQLProvider_DAL
-instance : SQLProvider_DAL
-connection : Connection
-SQLProvider_DAL()
+getInstance() : SQLProvider_DAL
<<Interface>> Provider_DAL
+create(provider : Provider) : boolean
+delete(id : int) : boolean
+getProvider(id : int) : Provider
+update(provider : Provider) : boolean
Provider
-phoneNumbers : String[]
-address : Address[]
-status : Status
+addAddress(address : Address) : boolean
+addPhoneNumber(num : string) : boolean
+deletePhoneNumber(phone : String) : boolean
ModelSubSystem
CnnCompatibility
-instance : CnnCompatibility
-CnnCompatibility()
+getInstance() : CnnCompatibility
Model
-model
+train(frame : DataFrame) : void
+predict(input : image[]) : double
CustomerSubSystem
Customer
-phoneNumber : String
-favList : Product[]
-outfitList : Cloth[]
+addToFav(product : Product) : void
+removeFromFav(id : int) : boolean
+addToOutfitList(cloth : Cloth) : boolean
+removeFromOutfit(id : int) : boolean
<<Interface>> Customer_Dal
+create(customer : Customer) : boolean
+getCustomer(id : int) : Customer
+delete(id : int) : boolean
+update(customer : Customer) : boolean
SQLCustomer_DAL
-instance : SQLCustomer_DAL
-connection : Connection
-SQLCustomer_DAL()
+getInstance() : SQLCustomer_DAL
CustomerController
-compatibilityModel : Model
-visualSearchModel : Model
+updateInformation(customerId : int, name : String, number : String) : boolean
+checkCompitability(customerId : int) : double
+likeProduct(customerId : int, id : int) : boolean
+addProductToOutfit(customerId : int, clothId : int) : boolean
+filter(map : Map<String, String>) : Product[]
+uploadImageToOutfit(customerId : int, image) : boolean
+recommendOutfits(customerId : int) : Product[]
+unlikeProduct(customerId : int, clothID : int) : boolean
+removeFromOutfit(customerId : int, clothId : int) : boolean
+getMostLikedCloth() : Product[]
+viewProduct(productId : int) : Product
+viewProducts() : Product[]
+search(text : String) : Product[]
-isCustomer(user : User) : boolean
+viewMostLiked() : Product[]
gives service
uses
ClothSubSystem
<<enumeration>> Size
-XXS
-XS
-S
-M
-L
-XL
-XXL
-XXXL
Product
-price : double
-numOfLikes : Int
-images : Image[]
-clothId : int
-provider : Provider
-sizes : Size[]
-description : String
+addImage(image : Image) : void
+removeImage(id : int) : boolean
+increamentLike() : void
+decreamentLike() : void
Category
-name : String
-id : int
-parent : Category
<<Interface>> Category_DAL
+create(cat : Category) : boolean
+getCategory(id : int) : Category
+delete(id : int) : boolean
+update(cat : Category) : boolean
SQLCategory_DAL
-instance : SQLCategory_DAL
-connection : Connection
-SQLCategory_DAL()
+getInstance() : SQLCategory_DAL
Cloth
#name : String
#id : int
#category : Category
#mainImage : Image
-type : String
<<Interface>> Cloth_DAL
+create(cloth : Cloth) : boolean
+getCloth(id : int) : Cloth
+delete(id : int) : boolean
+update(cloth : Cloth) : boolean
SQLCloth_DAL
-instance : SQLCloth_DAL
-connection : connection
-SQLCloth_DAL()
-getInstance() : SQLCloth_DAL
<<Interface>> Product_DAL
+create(product : Product) : boolean
+getProduct(id : int) : Product
+getProducts() : Product[]
+delete(id : int) : boolean
+update(product : Product) : boolean
+getProductsByProvider(providerId : int) : boolean
Image
-path : String
-id : int
SQLImage_DAL
-instance : SQLImage_DAL
-connection : Connection
-SQLImage_DAL()
+getInstance() : SQLImage_DAL
UserCloth
-date : Date
+delete() : void
UserClothController
+createCloth(img : Image) : Cloth
controls
RedisProduct_DAL
-instance : RedisProduct_DAL
-connection : Connection
+getInstance() : RedisProduct_DAL
-RedisProduct_DAL()
<<Interface>> Image_DAL
+create(image : Image) : boolean
+getImage(id : int) : Image
+delete(id : int) : boolean
<<Interface>> Cloth_DAL
+create(cloth : Cloth) : boolean
+getCloth(id : int) : Cloth
+delete(id : int) : boolean
+update(cloth : Cloth) : boolean
SQLUserCloth_DAL
-instance : SQLUserCloth_DAL
-connection : Connection
+getInstance() : SQLUserCloth_DAL
-SQLUserCloth_DAL()
ProductController
+viewAllProducts(categoryId : int) : Product[]
-filterByCategory(products : Product[], categoryId : int) : Product[]
-fiterByPrice(products : Product[], lower : int, upper : int) : Product[]
-filterBySize(products : Product[], size : String) : Product[]
-filterByBrand(products : Product[], id : int) : Product[]
+getPtoductById(id : int) : Product[]
+addProduct(provider : Provider, dataFrame : DataFrame) : boolean
+removeProduct(providerId : int, id : int) : boolean
+search(word : String) : Product[]
controls
uses
Powered By Visual Paradigm Community Edition

Figure 5.4 – Class Diagram

5.8. Sequence Diagrams

5.8.1. Get Recommendation

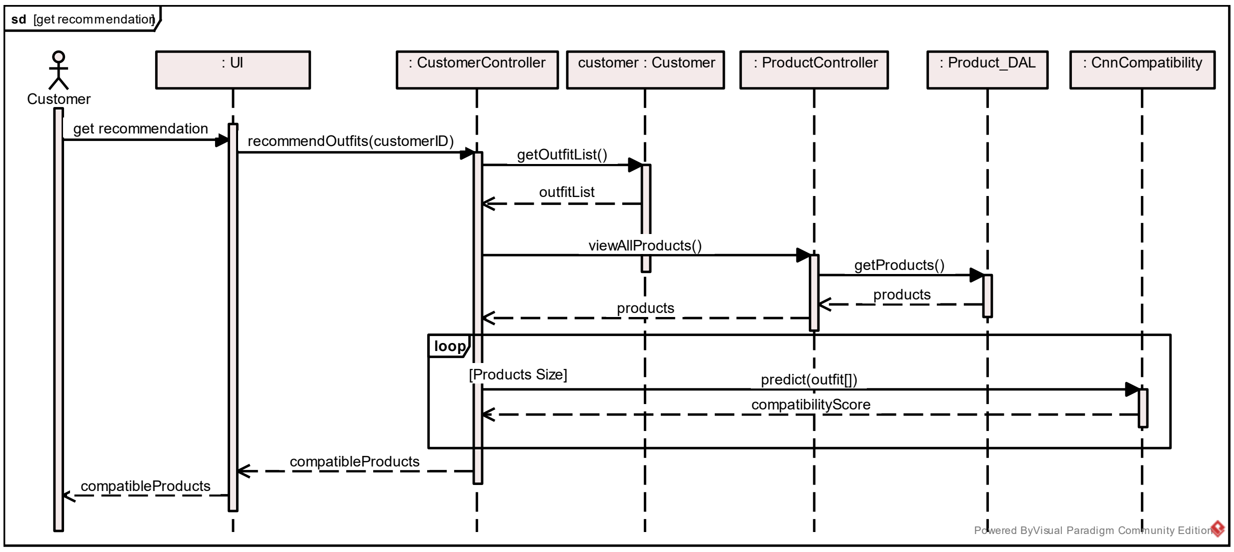


Figure 5.5 – Get Recommendation Diagram

5.8.2. Check Compatibility

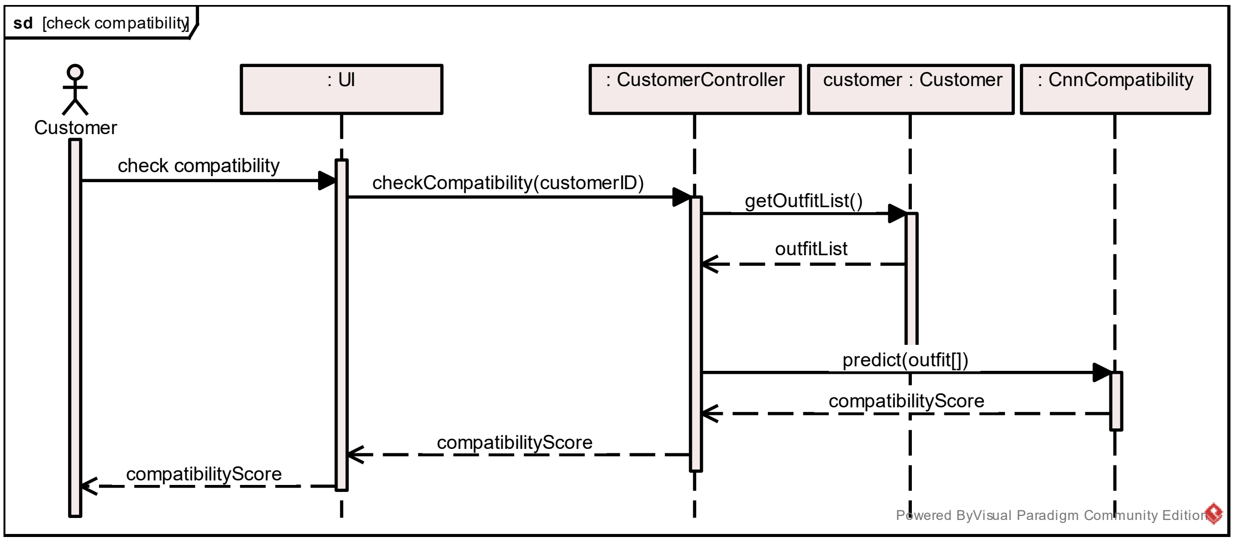


Figure 5.6 – Check Compatibility Diagram

5.8.3. Add Products

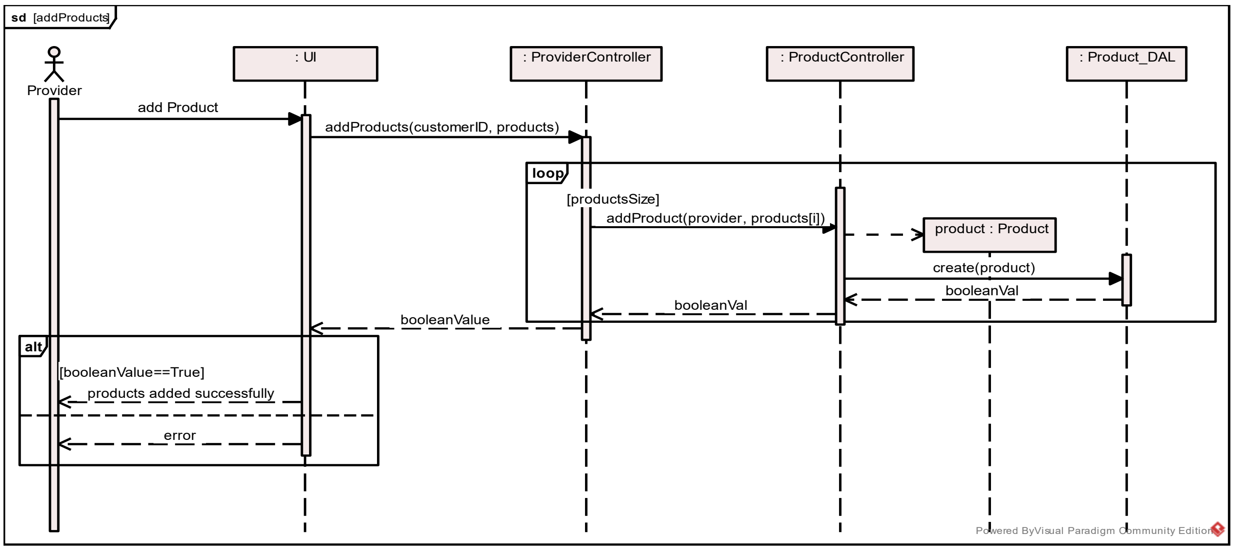


Figure 5.7 – Add Products Diagram

5.8.4. Verify Provider

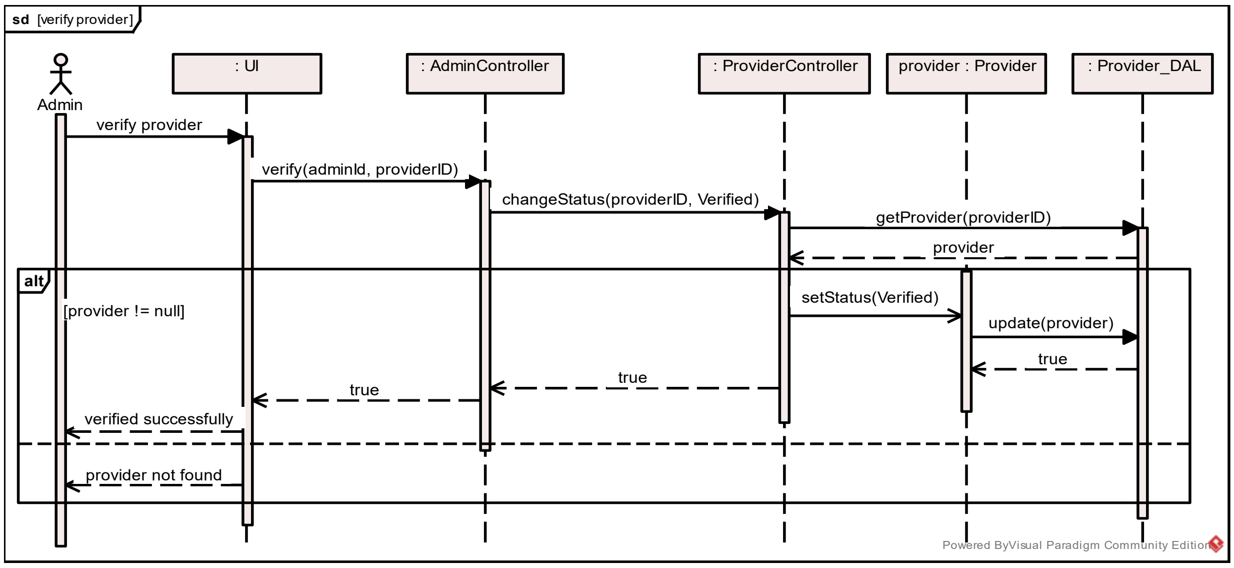
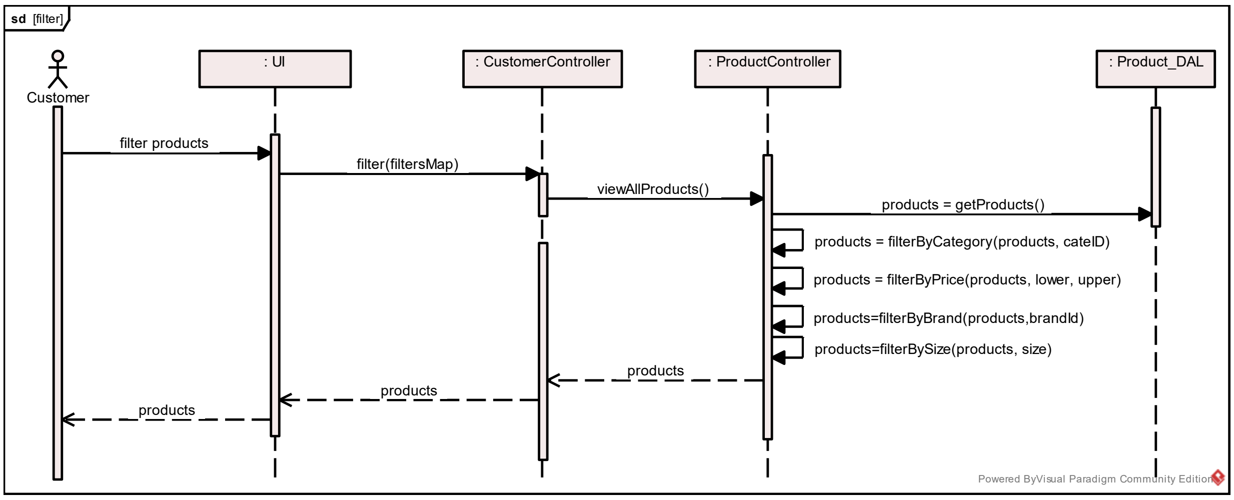


Figure 5.8 – Verify Provider Diagram

5.8.5. Filter



sd [filter]
Customer
: UI
: CustomerController
: ProductController
: Product_DAL
filter products
filter(filtersMap)
viewAllProducts()
products = getProducts()
products = filterByCategory(products, cateID)
products = filterByPrice(products, lower, upper)
products=filterByBrand(products,brandId)
products=filterBySize(products, size)
products
products
products
Powered ByVisual Paradigm Community Edition

Figure 5.9 – Filter Diagram

5.8.6. Remove From Outfit

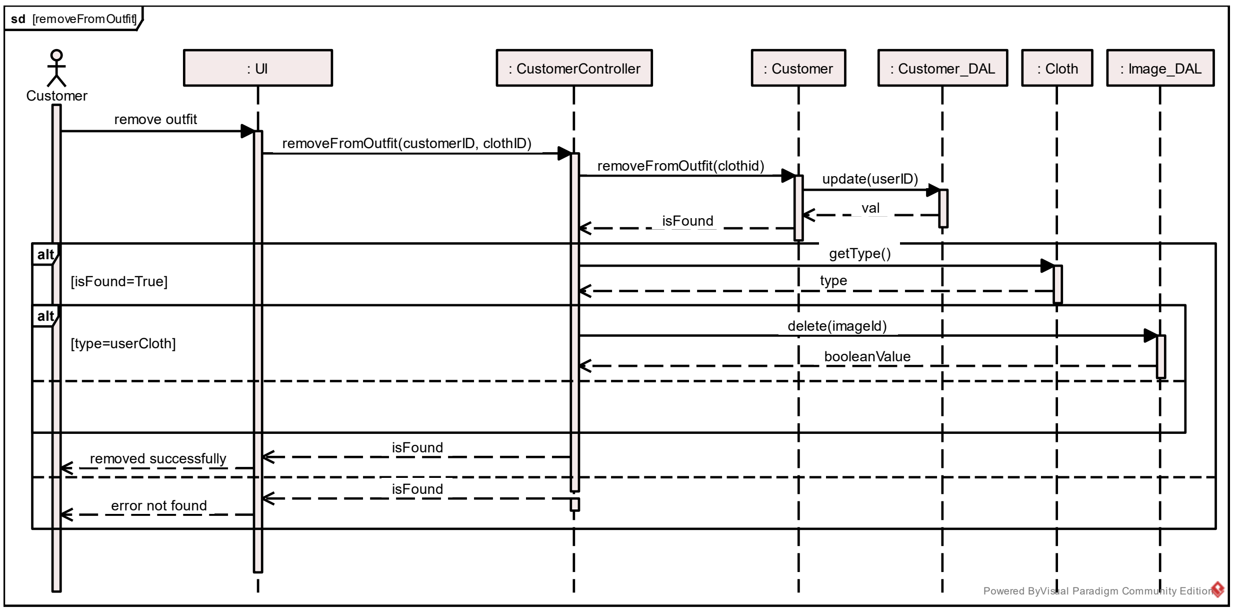


Figure 5.10 – Remove from Outfit Diagram

5.8.7. Add To Favorite

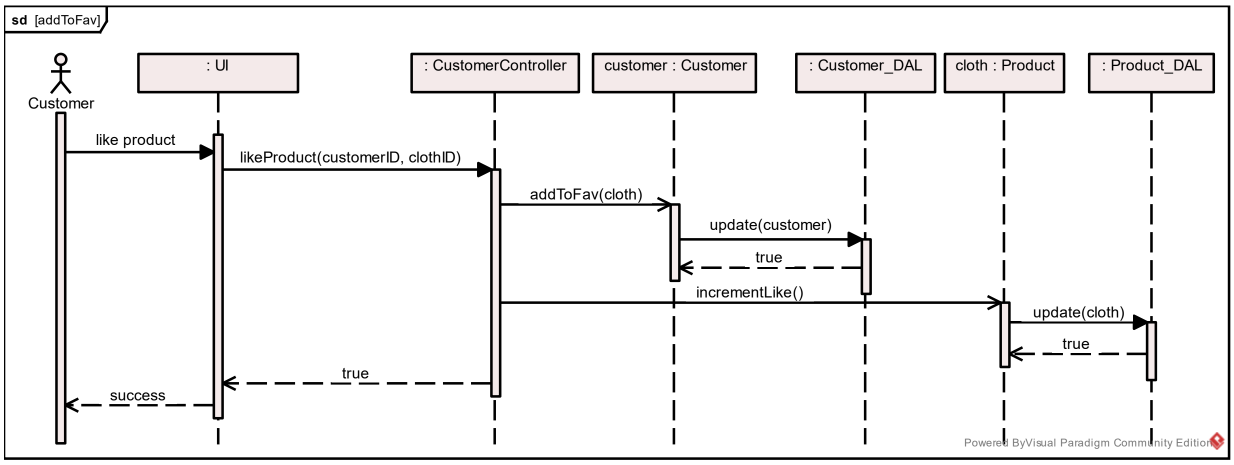


Figure 5.11 – Add to Favorite Diagram

6. Work Plan

Table 6.1 - Work Plan

Task	Task Title	Description	Task status
1	Brainstorming	Decide what is the main idea of the project and collect and scrap data for the first idea. Many problems with the first idea led us to choose the current idea.	Completed
2	Search for resources	Find research and papers to help us with the project and understand the models	Completed
3	Search for related work	Find competitors and know the strengths and weaknesses of their features	Completed
4	Conduct Survey	Gather people's opinion about the idea from different age ranges, to know whether it will be helpful or not	Completed
5	Collect datasets	Find the needed datasets for the machine learning models	Completed
6	Study Technologies	CNN, Deep Learning, Flutter, Django REST	In progress
7	Diagrams	UML, ERD, Use case, and Sequence diagram	Completed
8	Data Cleaning	Drop unnecessary data and dealing with data with overlapping types	In progress
9	Prototype	Design the initial prototype for the application (web and mobile)	In progress
10	Mid-year documentation	Finish the document and show progress	Completed
11	Build Compatibility Prediction model	Implement the model which will take the outfit and return its compatibility score.	Planned
13	Build Recommendation model	Implement the model for recommending items to the user	Planned
14	Front-end implementation	Finish the front-end of both web and mobile application	Planned
15	Back-end implementation	Finish the back-end with the rest of the features	Planned
16	Final documentation	Finish the final document required	Planned

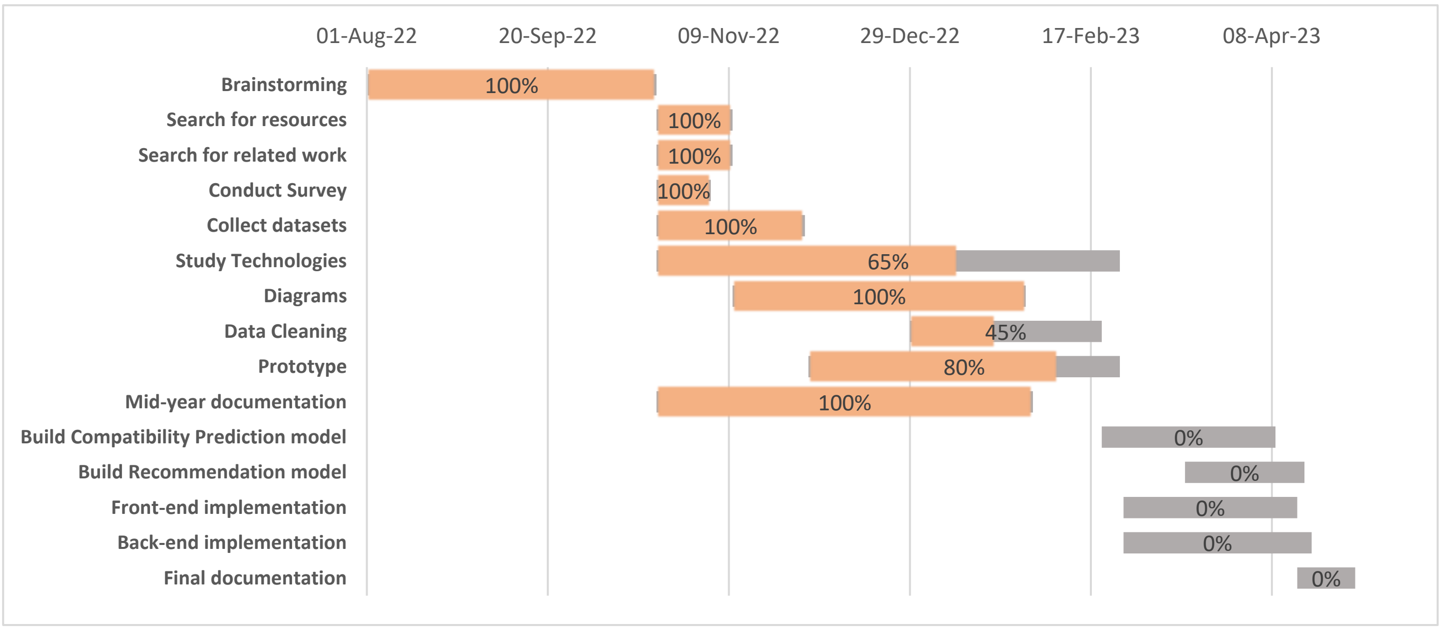


Figure 6.1 - Gantt Chart

7. References

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

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

[3] *Personalized outfit recommendation solution for eCommerce*. Vue.ai. (n.d.). Retrieved from https://vue.ai/products/outfit-recommendations/.

[4] Wang, X., Wu, B., & Zhong, Y. (2019). Outfit compatibility prediction and diagnosis with multi-layered comparison network. *Proceedings of the 27th ACM International Conference on Multimedia*. https://doi.org/10.1145/3343031.3350909

[5] 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

[6] Richards, M. (n.d.). *Software architecture patterns*. O'Reilly Online Learning. Retrieved from https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html.

[7] *Create rest api using django rest framework: Django rest framework tutorial - javatpoint*. www.javatpoint.com. (n.d.). Retrieved from https://www.javatpoint.com/create-rest-api-using-django-rest-framework.

[8] *Build apps for any screen*. Flutter. (n.d.). Retrieved from https://flutter.dev/

[9] Redis. (n.d.). Retrieved February 1, 2023, from https://redis.io/

[10] Nath, B. (2022, October 29). *PostgreSQL vs mysql*: *Differences and similarities*. Geekflare. Retrieved from https://geekflare.com/postgresql-vs-mysql/