Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: Outletter

High Level Design Report

Muhammad Bilal Bin Khalid, Muhammad Saboor, Mian Usman Naeem Kakakhel, Daniyal Khalil, Muhammad Arham Khan

Supervisor: Hamdi Dibeklioğlu

Jury Members: Çigdem Gündüz-Demir and Can Alkan

Progress Report

Dec 27, 2020

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

Contents

1 Introduction

In today's capitalistic age, shopping is one of the most time-consuming activities that people from all walks of life participate in on a regular basis. From shopping for some clothing articles, to a new pair of shoes to even buying the latest electronic gadget, people spend hours trying to validate their shopping decisions and confirming that they're getting the best possible bargain. But considering the inherent information gap in the manual store-by-store checking process [1], we spend many hours only to buy what we got the best deal on at the three stores we visited.

So seeing the potential for improvement in the process, we decided to introduce Outletter, A one-stop shopping companion mobile application that allows users to point their mobile phones towards items like fashion articles, electronics etc. Our app would automatically recognize the product in realtime and render metadata like price comparison with nearby stores, product reviews and other available variants in nearby stores, right in the users field of view using Augmented Reality. This way, whenever a user is out for shopping, they can rely on Outletter to help them make the most reliable decision about a product and be sure to get the best deal in a very easy and fast way. Considering that Outletter will be able to identify almost any product that is available online and that there aren't any steps with manual data entry or button clicks (the user can access all information just by pointing the mobile phone at a product), the application would revolutionize the shopping process for the mass user-base including the elderly.

Apart from this, Outletter will also provide the user with the latest deals and offers going on in a particular store as soon as they are near that store. This way, users will never regret missing out a good deal from a store just because they didn't watch an ad campaign and, as promised by our app, will always be making the most well-informed shopping decision possible.

1.1 Purpose of the System

Outletter is a cross-platform mobile application that will act as the perfect shopping companion. The main purpose of this application is to help the user make informed choices about the article they're trying to buy. Outletter will provide the users with a convenient way to find the best price for the product they are interested in by using a real-time object recognition engine to identify

the objects the device is being pointed at and render metadata about the product like the best price and other pricing options at different stores.

1.2 Design Goals

For our application to function as planned and be used commercially, the following properties need to be met.

1.1.1 Availability

- **Service:** The server downtime should not be more than 1% on a monthly regulation cycle otherwise the server should be available for requests 24 hours a day, 7 days a week.
- **Location:** The Application will only be available to users in Turkey, due to the limited scope of websites that can be parsed.
- **Operating System:** The Application should be available in both Androids 7.0 or higher and iOS 11.0 or higher operating systems.

1.1.2 Response Time

The Application Response Time for user queries and searches should be less than 3 seconds. The Response Time for User Recommendations should also be less than 3 second.

1.1.3 Scalability

The Application usage is dependent upon the user demand. Hence the application should be scalable to meet with increasing user demands by increasing the number of users and database storage. For now, due to not having proper production servers, the number of concurrent users that would be able to use the application would not be more than 100.

1.1.4 Accuracy

The product results given by the application should have at least have an Accuracy of 90% and the recommendations should all be based on the personalized tastes of the user.

1.1.5 Extensibility

The application should be extendible to different areas and categories to meet the user demand if needed. Currently, as mentioned before, the application is available only in one location which is Turkey.

1.1.6 License

All the datasets used in training the models, and all the libraries used in the development should be properly licensed and accredited.

1.1.7 Privacy

The application should ask the user before using their data in improving the recommendation system to more suit their tastes.

1.1.8 Maintainability

The application should adopt a low coupling modular design to allow the modules to not affect each other during alterations.

1.1.9 Network & Server

The Application should run smoothly on 1mbps internet smoothly without any issues. The server should be able to handle 100 concurrent users concurrently. The server should be running on a minimum 2.2 GHz.

1.1.10 Usability

The Application should be user-friendly and have an intuitive design to help promote user-functionalities.

1.3 Definitions, Acronyms, and Abbreviations

React-Native: An open-source mobile application framework. It is used to develop applications for Android, iOS and more.

Django: Django is a Python-based free and open-source web framework that follows the model-template-views architectural pattern.

PostgreSQL: An open-source relational database management system emphasizing extensibility and SQL compliance.

ARCore: A software development kit developed by Google that allows for augmented reality applications to be built.

OpenCV: OpenCV is a library of programming functions mainly aimed at real-time computer vision.

Docker: Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.

API: Is the abbreviation for the term Application Programming Interface. The term can be defined as an interface to an application component that is designed to ease the usage of such components by other developers than the owner.

Training: Is the term for the process of presenting collected data to a model/network for it to find relationships/patterns between the inputs and expected outputs. The output of this process is a machine learning model.

Machine Learning Model: It can be defined as a black box system that embodies a mathematical representation of a real-world process.

Image Segmentation: In digital image processing and computer vision, image segmentation is the process of partitioning a digital image into multiple segments.

1.4 Overview

Outletter will be a shopping companion that is developed for Android and IOS platform. It will help its users make informed choices about the article they're trying to buy. The app will utilise AR and a cloud-based, real-time object recognition engine to identify the objects the device is being pointed at and render metadata about the product like the best price and other pricing options at different stores. Additionally, Outletter will also provide the user with the latest deals and offers going on in a particular store as soon as they are near that store so they do not miss out on a good deal.

There will be two types of users, Customer and Shop Owner. After downloading and installing the application, the user will be asked to sign up and create an account either as a Customer or a shop owner for the application. After creating the account and signing in, the customer user type can start taking pictures of products to scan them and find their best price. In the camera screen of the application, when a user points to the product and presses the scan button, a list of top products sorted by price and similarity shows up along with another list of recommended items which may go well with the scanned product based on the user's choice. At any moment the Customer can access a menu to view their reviews and wishlist or they can access settings or log out from the application.

The Shop Owner user can make a profile of their company and place ads in AR. This can be done by choosing an image of an ad from their phone's gallery and using Outletter's ad placement screen to position it in the real world. These ads placed in AR can be seen by the Customer user type from their screen and they can choose to interact with them. Similar to Customers, the Shop Owners can access a menu to go to settings or log out from the application.

Outletter will provide convenience for the people shopping as well as help shops promote their products by placing ads for their deals. This combination will promote a better and more enjoyable shopping experience.

2 Current System

There are various mobile applications present on the market that help their users to find a product online by image. There are other applications that provide you with reverse image search results from Google or other search engines. Some example of such systems are:

1. Google Lens
2. Pictpicks
3. Search by Image

These applications find different attributes in the image and then find similar images present on the Internet. However, Outletter is different from such applications because it also provides useful insight to the user about the product he is searching. This includes price comparison of the product on different stores and the public opinion about the products.

3 Proposed System

3.1 Overview

This part of the report will explain the software architectural design of the proposed Outletter application. First, the system decomposition of the application is shown. Outletter's system decomposition aims to present the structures of its systems and subsystems in great detail. The subsystem decomposition showcases how each of the chosen subsystems (explained in greater detail in the following sections) interacts with each other. From the diagrams, it is possible to see the responsibilities of each subsystem, which information sends, receives or computes, and how the transition between different subsystems takes place. Getting more accurate details for the subsystems before starting the implementation of the application will result in fewer errors during coding and less revision required.

3.2 Subsystem Decomposition

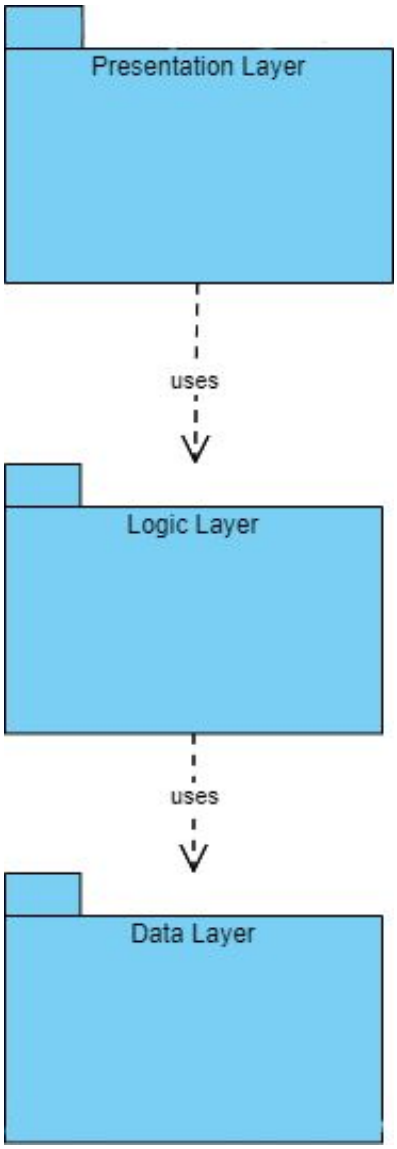

Figure 1: Subsystem Decomposition

Since Outletter is going to be a dynamic mobile application, we wanted to base our structure on maximum modularity possible, hence we divided all the components into non highly dependent packages that have high coherence and low coupling amongst each other. Hence we implemented a three tier architecture to make developing and integrating new functionalities within the application easier and understandable.

Our proposed structure contains 3 layers. Presentation, Logic and Data layer. All the layers are being developed independently of each other to low coupling between them, and all the three layers happen to have hierarchical relations among them.

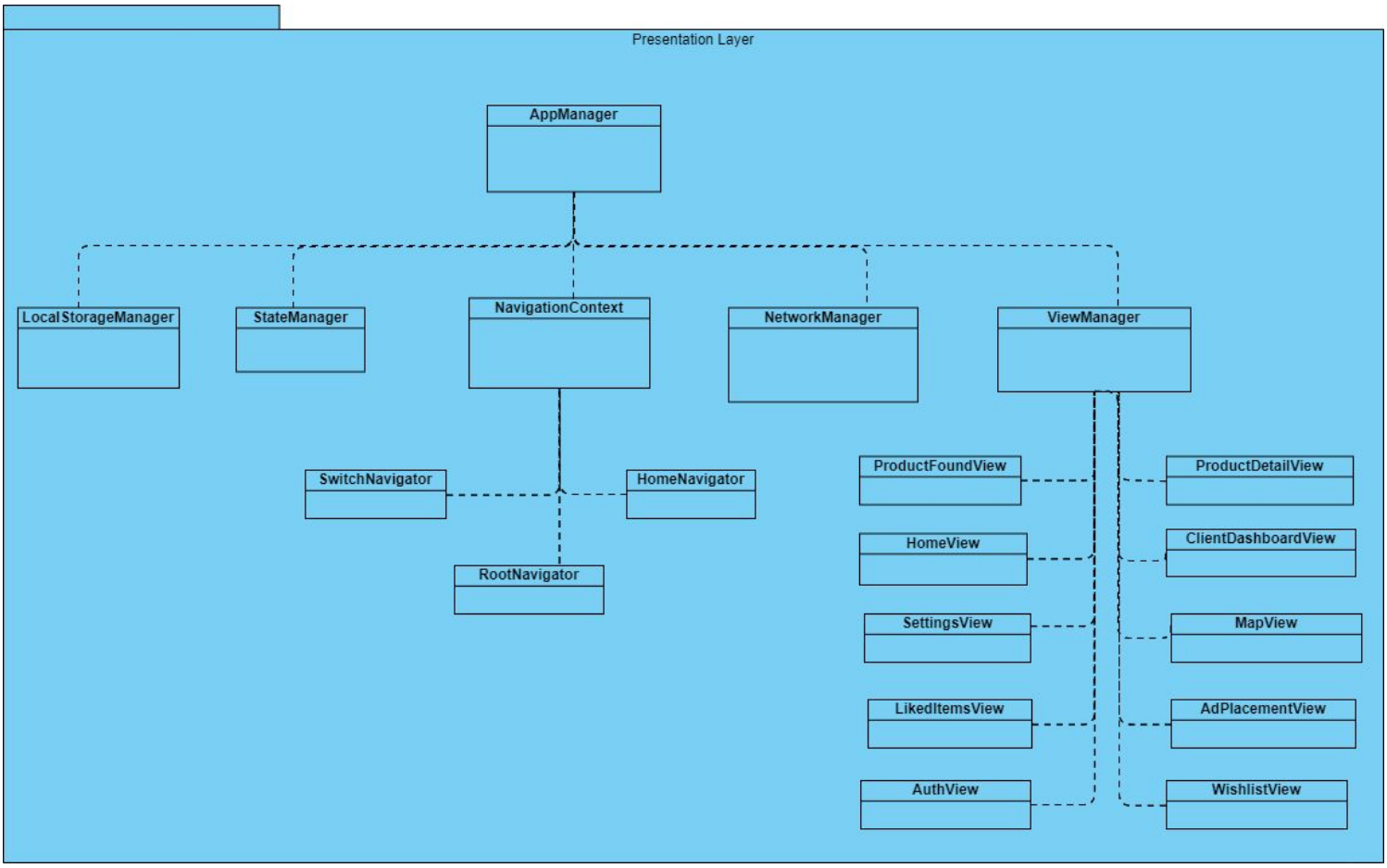

Figure 2: Presentation Layer

Presentation Layer:
The Presentation Layer is the UI (User Interface) Layer that we have implemented. It is the foremost layer in Outletter. All the interactions with the user are performed through the Presentation Layer. This layer is an amalgamation of user screens and handlers. The only data that this layer stores are the user login information and credentials.

All the user requests are stored temporarily and then sent to the logic layer through API calls, the response from the API calls is then rendered on the screens for the users. This layer does very trivial functionalities and is mostly used as a user interface only.

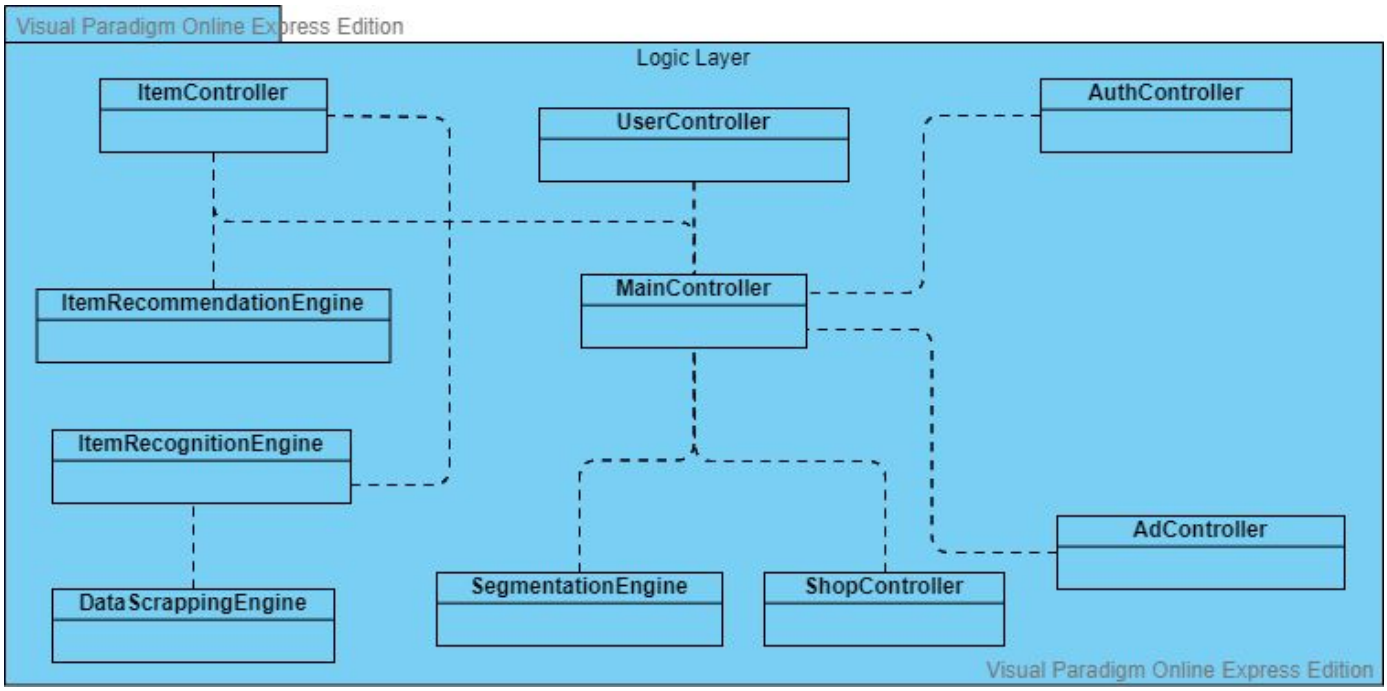

Figure 3: Logic Layer

Logic Layer:
This is the core of Outletter, the main Layer of the application or in other words the Application Layer, this layer contains all the main functionalities of Outletter. This layer is connected with the Presentation Layer through API endpoints.

This layer contains our machine learning models for item similarity and recommendations for the users, and also the web scrapers for item finding as well. It also contains all the backend functionalities for all the user screens in the Presentation Layer. This layer does not store any data either, and all the data used in the processing is temporary, it stores all the data through the DataBase Controller in the Data Layer.

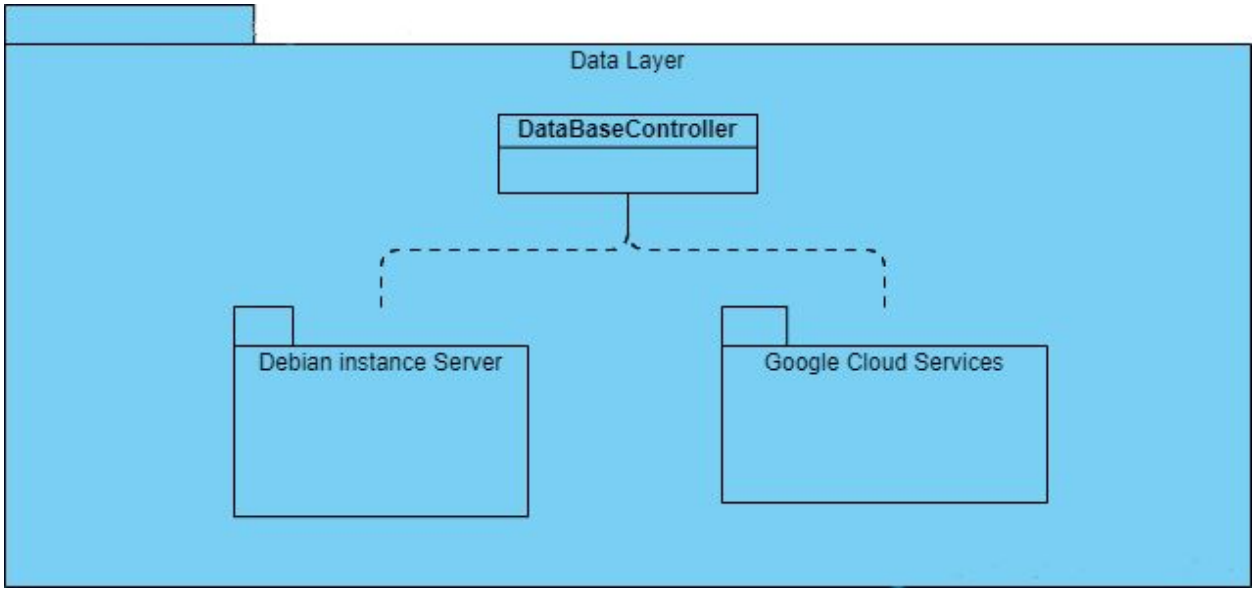

Figure 4: Data Layer

Data Layer:

This is literally the Database of our application, all the user information and data are stored in this layer, along with our server instance. The server instance makes function calls to the Database Controller and then the controller saves the data in our Cloud Database.

3.3 Software/Hardware Mapping

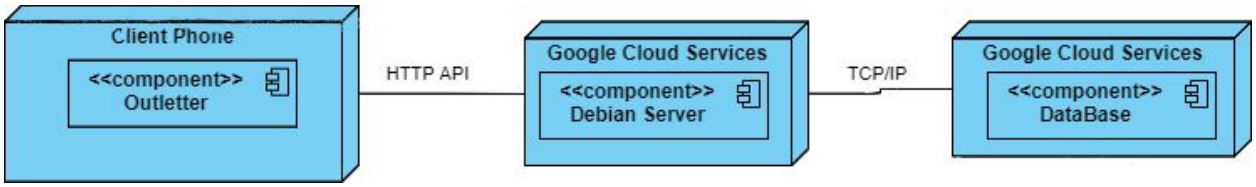

Figure 5: Software/Hardware Mapping

This deployment diagram shows how the three layers of our architecture will be deployed and how they will interact with one and another. The Presentation Layer will be a ReactNative application on the Client's Phone, the Logic Layer will be a Debian Instance Server on Google Cloud Hosting, and last the Data Layer will also be hosted on Google Cloud Services but it will be independent from our Debian Instance Server.

The tasks for each node will be as follows, the Outletter application will interact with the user and will find out what the user wants and will convey that to the Debian Server through API Calls. The server will receive the user requests and then process them and respond to them accordingly. If there is data that needs to be saved or used, the Server will request the Data Layer from retrieval and save.

3.4 Persistent Data Management

Since most of our data is stored and the computations are done on the cloud, our application does not require crucial data storage. However, since our application follows an authentication protocol we will store data regarding that in the device such as authentication token and current user credentials. Outletter will allow for partial offline availability which includes user data such past searches, wishlist items, reviews etc.

3.5 Access Control and Security

In a multi-user system, different actors must have different access rights over the data and functionality [2]. This makes access control a necessity to ensure the users only access information they are permitted to. In Outletter, users will need to provide basic personal information during sign up, which users will prefer to keep private, so we will need to ensure the user data is kept private.

Outletter will have two types of users, Customer and Shop Owner. As the features and permissions of these types of users are different, when signing up for the application, the user will choose which type user they are and the account created will remain as that type. The users will register with a valid email and password. An email address can only be linked with one account, this is done to limit the number of accounts a user can make.

Data privacy is a significant societal issue that we should certainly safeguard. Outletter is designed to conform to the General Data Privacy Regulation (GDPR) [3]. The application should ask the user before using their data in improving the recommendation system to more suit their tastes.

3.6 Global Software Control

In this section, we describe how the overall system is controlled on a global scale. We give a discussion on how requests are initiated and how subsystems synchronize. Finally, we address certain concurrency issues.

We will consider the client and server part of our system. Our system works in the following way, the client sends the server requests, these requests are then processed and response is sent back to the client. This can be divided in three parts:

Request: This past initiates when the request is sent from the client side. When the user takes the picture of the product, it triggers a request which sends the captured picture the user took to the server as an input to be processed by multiple models. Here, a common problem that may occur is that the server may be overwhelmed by a large number of requests by a large influx of new users or a malicious Denial of Service (DoS) attack [4].

Processing: When the backend server receives the image in the request, it first sends the image to the segmentation engine which segments the image to get rid of the background. Then the segmented image is sent to the tagging engine to tag the image with relevant tags. These tags along with other data from the request (gender and shop) is used to search the similar items online using the data scraping engine. These items are sent to the similarity engine to sort the items in decreasing order of similarity with the query image. Furthermore, the query image is also sent to the recommendation engine which generates dynamic recommendations based on the search history of the user. The backend server should be able to process multiple requests asynchronously.

Response: The output from the similarity engine and recommendation engine is sent back to the client and it is displayed on their screen.

3.7 Boundary Conditions

Outletter will have three boundary conditions. These include: Starting the application, terminating the application and facing failure in the application. These conditions are discussed in detail below.

Initialization

The user needs to download the application from the Google Play Store or Apple App Store. To register for the application, users will need a valid email address and password. After signing up, the user can use the email and password to log in to the app. For processing the scanned image to find its best price as well as find similar products, a connection with the server must be established. Thus, internet connection is needed to initialize the application.

Termination

The users can use the log out button in the collapsable menu. After closing, the application does not run in the background, all functionality of the application stops which includes the camera functionality in the main screen for Customer users. If the user quits from the application by closing it, when the application is opened again the standard procedure for initiating the navigation will be followed.

Failure

There are a few cases which could cause failure in the application.

- In the case where internet connection is lost while scanning the product, the application waits for the device to reconnect to the internet and then resumes scanning. During this process the user is notified with a message, informing the user that internet connection is lost and they should reconnect to the network.
- In the case where the device's camera functionality stops working due to low battery, the camera screen freezes and a message pops up that informs the user to charge the phone in order to use the application.
- In the case where one of our processing modules stops working due to a bug or server side problem. The user is informed that there has been a problem in processing and they should try again later.

4 Subsystem Services

4.1 Presentation Tier

AppManager: The default entry point for the application where all root components and navigators are initialized and native/ JS bundles for the React-native application are loaded. The AppManager also enables necessary capabilities like app state persistence and app life cycle events.

LocalStorageManager: This stores data objects and other persistent values like user details, login tokens, wishlist items etcetera on the device's local storage. This is necessary to identify

the user and device in network calls and also to enable some limited offline capabilities like seeing liked items and wishlist items.

StateManager: The StateManager basically uses Redux and Redux-thunk middleware to enable app-wide state management. This way, a homogeneous user experience is made possible with components being able to communicate their current state with other components and actions being dispatched outside the component hierarchy. Hence allowing for flexible and effective application development.

NavigationContext: NavigationContext stores the universal navigation and component state of the application and allows inter-navigator navigation inside the application. This way, the application doesn't have to rely on one navigator to handle all navigation and can have flexible navigation. For example, the DrawerNavigator could trigger a tab change in TabNavigator.

SwitchNavigator: This will be the navigator containing views like Login, Register and then the home navigator for the app. This enables fluent navigation between the application and disables any erroneous navigation using back or life cycle events.

HomeNavigator: This will be the navigator containing the actual functionality of the application and other child navigators like TabNavigator, DrawerNavigator and StackNavigator. This will enable path navigation functionalities like back navigation, breadcrumb navigation etcetera.

RootNavigator: This will be the root navigator component loaded inside the NavigationContext containing both, the SwitchNavigator and the HomeNavigator. This is used to create a universal parent to all navigators and wrap it in the navigation context to enable app-wide navigation state persistence and state accessibility.

NetworkManager: This is an extended class object from Axios volley library that adds relevant headers like API keys, Authorization headers and ClientKeys to each request and also implements a universal error handling mechanism for failed request calls.

ProductFoundView: The component for the page that shows up when the user requests for an item to be found on Outletter, the page shows the best results along with similar items that were found through our application.

ProductDetailView: The component for the page that opens up when the user selects an item from the ProductFoundView, the page shows all the details of the product along with its price.

HomeView: The component for the Home for the user, this page is a basic camera view, through which a picture can be taken, which will be the query image from the user to Outletter.

ClientDashboardView: The component for the Home for the shopOwner, this page contains the details regarding the shop along with a list view of the advertisements placed by the shop.

SettingsView: This is the settings view for both user and the client, the settings contain privacy settings, along with the basic notification and volume settings in applications.

MapView: This is the map view that the user will be taken to from the ProductDetailView, when the user wants to check where the item is available, if available physically.

LikedItemsView: This is the list view for users to see which items they liked previously, this list view is also used to store the list with which user preferences are built upon in the backend.

AdPlacementView: This is an exclusive view only available for shop owners, in this the owners can place their advertisement in the AR mode near their shops physically.

AuthView: This is the login and signup view for users and shop owners, through this users and shop owners can sign up and login separately.

WishlistView: This view is available for users. When the user wishlists an item from the product details page, the item is added in this view for users to view at another time.

4.2 Logic Tier

MainController: This is the main application layer controller in the backend, this controller controls all the other backend controllers by creating the Django routes for all the API endpoints.

AuthController: This controller holds all the API calls related to user login/signups. This controller checks the API parameters received against the data in the Data Layer and then sends the response to the Presentation Layer accordingly.

UserController: This controller controls all the user based API calls received from the Presentation Layer, such as the call for user wishlist data, user liked items data, user settings, user login details and previous search history and all.

ItemController: This controller controls all the API calls regarding products and searches regarding them, all the images that the user searches are sent to this controller, instead of responding back from itself. The item controller calls the SegmentationEngine, DataScrappingEngine and the ItemRecognitionEngine to find the item, and then sorts it based on the similarity of it.

ItemRecommendationEngine: This module uses the user's liked items from the Data Layer and produces a list of recommended items for the user. The module works through a trained machine learning model made for item recommendation.

ItemRecognitionEngine: This module is called by the item controller to recognize the item and sort all the items given to it in an order of similarity. This module works through a trained machine learning model made for item recognition.

DataScrappingEngine: This module is called by the item controller as well to find items similar to the query item sent by the user. This module uses the google search api to find similar items on certain web pages.

SegmentationEngine: This module is called by the item controller as well to segment the query image sent by the user to remove all the irrelevant details from the image, and to produce an image that only contains the item.

ShopController: This is a shop controller that receives all the API calls made by shopOwners with respect to their shops and then send the responses accordingly, to the Presentation Layer and the Data Layer depending upon the call.

AdController: This is the controller made for placing and showing the advertisements in the AR mode near the shop physically. All the advertisement related API calls are directed to this controller.

4.3 Data Tier

DatabaseController: Database Controller is never called directly by the Presentation Layer, instead the Presentation Layer makes API calls to Logic Layer and Logic Layer Controllers makes calls to the Database Controller when they need to extract or store data in the Database.

Debian Instance Server: The Calls to the DatabaseController are made from the server, and then according to those calls, the controller either responds with the required data or just stores it.

CloudDatabase: This is where all the data that the application Outletter uses is stored, it has all the user data, along with the application data stored inside it.

5 Consideration of Various Factors in Engineering Design

- **Public Health:** Although our application doesn't expose the user to any major health concerns, we did consider the possibility of public health concerns like eye stress and headaches from prolonged exposure to screens. Hence, we implemented dark mode and shade overlays on cameraView in the application to ensure that the user had the option available to reduce strain on their eyes.
- **Safety:** During the design process of Outletter, we considered the usage of our application in public places and the possibility of users getting distracted and getting exposed to the possibility of minor accidents. Hence, instead of utilizing a usage where the user takes a picture and waits for the results of the product while looking at their phones, we consider using AR to project the results in the customer's field-of-view. So this way, we ensure that the customers are in touch with the real world and can avoid any possible accidents.
- **Welfare:** We didn't consider any welfare factors in the development of our application.
- **Global Factors:** Since our application is targeting the Turkish audience only, we didn't consider many global factors in the development. But since even in Turkey, the brands and audiences we were targeting were international, we included multi-language support in the application (Turkish and English).
- **Cultural Factors:** During the implementation of our application, we will consider the cultural background of the Turkish public (our target audience) in all our decisions including store selection, indexable item type selection and item appropriateness. This can include factors like only using culturally relevant data to train our similarity models and retrieving only relevant and appropriate items in the listings.
- **Social Factors:** The application considers various social factors in the engineering design process including ensuring the most common and relevant stores to retrieve similar items and providing a diverse variety of possible stores and item types in the application for users with different socio-economic backgrounds. Furthermore, the application also allows users to share the identified products on multiple platforms using a deep-link to let other users see what they found as well, hence enabling socialization. Finally, instead of focusing the application on an isolated shopping experience like conventional online shopping, the application promotes users to interact with the real world and shop for products with the flexibility and knowledge access of online shopping while actually being in a real social setting: outlets.
- **Environmental Factors:** This was a major factor considered in the development of this application's ad placement feature. As it replaces the necessity for paper flyers and print media to market campaigns and products, this contributed to reducing the carbon footprint of the outlets in general.
- **Economic Factors:** Outletter enables users to find the best possible price for a relevant product and hence, we are taking economic factors into account for its development. One of the major factors we consider is being brand-standard agnostic in selection of the brands so users from all economic brands and shopping preferences can use the application to find their products, regardless of their price range.

Factors	Effect Level	Effect
Public Health	4	Eye stress and headaches
Safety	3	Minor Accidents
Welfare	0	No effect
Global Factors	4	Language Barrier and diverse target audience
Cultural Factors	5	Store selection and item appropriateness
Social Factors	8	Sharing items, shopping in real social setting
Environmental Factors	6	Removing print media
Economic Factors	9	Price suggestion, brand selection and target user

Table 1: Factors and Effects

6 Teamwork Details

6.1 Contributing and Functioning Effectively on the Team

It is important to note that all members of the team worked on all the reports equally. Here is our summary of distribution of the tasks among our team members:

Member Name	Contribution in Work Package
Daniyal Khalil	Specification Report Analysis Report High Level Design Report Segmentation Engine Backend Development
Mian Usman Naeem Kakakhel	Specification Report Analysis Report High Level Design Report

	Similarity Engine Development Operations (Devops) Backend Development
Muhammad Bilal Bin Khalid	Specification Report Analysis Report High Level Design Report User Interface Mobile Application Development Data Scraping Engine
Muhammad Arham Khan	Specification Report Analysis Report High Level Design Report Tagging Engine Data Scraping Engine Mobile Application Development
Muhammad Saboor	Specification Report Analysis Report High Level Design Report Integration Backend Development Similarity Engine Data Scraping Engine

Table 2: Distribution of Tasks

Muhammad Bilal Bin Khalid worked on designing the User Interface as well as the development of the mobile application in React-Native. He first brainstormed and designed the Screens of the application in Adobe Photoshop Creative Cloud 2020. Then he translated these pages into actual working screens using React-Native. He also Integrated the backend API calls with the frontend mobile application.

Muhammad Arham Khan implemented the tagging engine user Google Cloud Vision API and refined the results of the API. Then using the refined results, he implemented the data scraping engine which would provide a list of similar items. He also helped in the integration of backend API calls with the frontend mobile application.

Mian Usman Naeem Kakakhel developed the machine learning model for the classification of clothing items. He also developed the similarity engine to sort a set of images in order of most similarity with the query image. He created an initial Docker environment for easy setup and development for all the members. He also helped the backend team with their testing of the application.

Muhammad Saboor developed the backend application using Django framework with Daniyal to create the model classes and connect them to the database. He integrated all the backend engines to implement the API calls. He also helped Usman in developing the machine learning model for classification and similarity engine. He also improved the data scraping engine by fixing some bugs.

Daniyal Khalil implemented the segmentation engine by developing a computer vision model to segment the query image to filter out the background in the image. He also developed the backend application with Saboor to create the model classes and helped him in the integration of backend engines with the application.

6.2 Helping Creating a Collaborative and Inclusive Environment

We set up a Docker environment for smooth development experience for each of our team members. After setting up the environment, this environment was uploaded on Github along with the Readme file which contained the instructions on how to use this environment. After that, the members made multiple branches of the Github repository where they would work on their respective tasks. After all the members completed their individual tasks, the backend development and integration team worked with the mobile development team to come up with the format of request and response of the API calls. In this meeting, dummy JSONs were created which could be used by both teams later on for testing purposes. After this, both teams worked together to complete their respective group tasks. Finally, all of the members worked on the testing and debugging of the entire application and made fixes or improvements where necessary.

To organise ourselves better and keep track of our progress, we used the Trello task management program [5]. We created multiple section lists to track the progress of each task. The lists were Pending, In Progress, Control and Completed. Furthermore, we also assigned a color label to

each task card so that we could track which member of the team worked on which task. This way we were able to complete unit tasks in an Agile-like development process easily.

6.3 Taking Lead Role and Sharing Leadership on the Team

Before starting the project, we divided the task under different leaderships, to reduce the workload on each member as well as ensure everyone works equally. One member would be completely dedicated to a few Work Packages while helping other teams with their tasks. We have divided the leadership of work packages as following:

Member Name	Leader of Work Package
Daniyal Khalil	Segmentation Engine Analysis Report High Level Design Report
Mian Usman Naeem Kakakhel	Similarity Engine Development Operations (Devops)
Muhammad Bilal Bin Khalid	User Interface Mobile Application Development
Muhammad Arham Khan	Tagging Engine Data Scraping Engine
Muhammad Saboor	Integration Backend Development Specification Report

Table 3: Leadership of Work Packages

7 References

1. Mohsin, Maryam. “10 Online Shopping Statistics You Need to Know in 2020 [Infographic].” [Infographic]. Oberlo, September 29, 2020. https://www.oberlo.com/blog/online-shopping-statistics

2. R. S. Sandhu and P. Samarati, "Access control: principle and practice," in *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40-48, Sept. 1994, doi: 10.1109/35.312842.

3. “Code of Ethics.” Accessed October 12, 2020. https://www.nspe.org/resources/ethics/code-ethics

4. Overill, R.E. (1999), "Denial of Service Attacks: Threats and Methodologies", *Journal of Financial Crime*, Vol. 6 No. 4, pp. 351-351. https://doi.org/10.1108/eb025906

5. Trello. (n.d.). Retrieved December 27, 2020, from https://trello.com/en