Bilkent University

Department of Computer Engineering

Senior Design Project

Project Name: ShopCart

Final Report

Group Members:

- Ahmet Kaan Uğuralp
- Ahmet Işık
- Furkan Ahi
- Mehmet Yaylacı
- Ravan Aliyev

Supervisor: Asst. Prof. Dr. A. Ercüment Çiçek

Jury Members: Asst. Prof. Dr. A. Ercüment Çiçek, Asst. Prof. Dr. Shervin Arashloo, Asst. Prof. Dr. Hamdi Dibeklioğlu

Website: https://ahmetisk.github.io/CS-Senior-Project---ShopCart/Website/

Final Report

May 6th, 2022

Table of Contents

Abstract

ShopCart is a mobile application to help users to improve their shopping habits. People run out of products eventually they use these products and forget to buy them before they run out. Another problem is finding appropriate prices for their budget. Our application helps to solve this problem by allowing them to create shopping lists and add products to the list easily in 4 different ways: scanning barcode, scanning image, voice search and typing product name. Users can share these lists with their communities and the application notifies them when they leave home to remind them that they have to buy products which they are about to run out of. This mobile application makes people’s life easier by improving their smart shopping practices.

Final Report

ShopCart

1. Introduction

People have been going shopping in order to meet their needs. We all need to purchase food and beverages to survive. Storing the purchased food in refrigerators, cooking and ordering meals are all common activities. When ingredients are used for these purposes, they run out eventually. People make the shopping list when the refrigerator is nearly empty. However, they may forget some of the products and this causes them to go shopping again. Since food prices keep rising, smart shopping becomes more challenging. Smart shopping means knowing what to buy and when. [1] Key point here is knowing the basics before going shopping, like what you need, the amount of each food needed and where you can find these products with the lowest price.

After the COVID-19 pandemic, businesses have taken strict measures to properly enforce social distancing standards. They tried to keep customer satisfaction at the highest level while minimizing face-to-face interaction. The pandemic conditions we were in at the beginning of the project also pushed us to think similarly and we thought about how our daily grocery shopping could be done better. That is why we planned to develop a mobile application for this purpose.

Existing home order and grocery store applications need to be improved in certain aspects. The ShopCart project should be easier, reusable and more effective for the customers themselves to fill in and present because this conventional service is missing.

ShopCart is a mobile application which helps users to keep track of the products that run out and purchase reminders. The information about these products includes their monthly consumption frequency, the number of the days that it has been run out, and so on. So, this application helps consumers to be a smart shopper by showing when each product is out of stock, amount of the needed product, which market offers the lowest price for each product and calculating the total spending on food. Before throwing away the depleted product, users scan its barcode, enter its name manually or scan a photo in order to create a shopping list. After the product is added to the shopping list, its numeric data depletion time and other information will be preserved as a result of these methods. Users can create a "community" on the app if they live together. By the help of this feature users living together can create a shopping list of the community and each member can add and remove products from this list. Users can also edit the list after purchasing it to choose not to purchase it again. The goal of this app is to make people's lives easier by improving their smart shopping practices. Our shopping practices have big implications for our health, our society and the environment and this app helps to raise living standards by improving consumption practices [2].

What distinguishes ShopCart from other grocery shopping applications is that it is more user-oriented and suitable for common use compared to its counterparts. Many shopping applications give results according to user searches and list them according to the results. In ShopCart, on the other hand, in addition to all these, the user can use the application both as the needs of his/her household arise, and when s/he comes to his/her mind and enters own needs into the application.

The system of the ShopCart is decomposed into three subsystems using three-tier architecture as Client tier, View tier, and Server tier. The view tier will be built using React-Native framework, the Django environment will be used in the client and server tier, and The MongoDB, PostgreSQL database program, used in the back-end. These tiers deploye to the static file storage using Amazon Web Services (AWS S3).

2. Requirements Details

2.1.1. Functional Requirements

The system of the *ShopCart* should provide the following activities to the business owners and their customers.

2.1.1.1. Login Screen

- This should be the first screen the user encounters when logging into the application.
- The user must be prompted for a username and password.
- "Forgot your password?" even though the user has forgotten his password. There should be a button and this button should direct the user to the required page.
- For users who have no account or want to open a new account, there should be a "Create New Account" button and this button should direct the user to the required page.

2.1.1.2. Register Screen

- The user must enter the required information for registration.
- Users should enter the "Community Code" on this screen if they intend to use a shared shopping cart.

2.1.1.3. Add Product to Cart

In this area, the user can add products to his cart in many ways. Depending on the support provided by the technology used in the product, the user can choose the method. This preference may vary depending on the condition of the product and environmental conditions. The 5 types used in adding products are:

- Adding the product by scanning its packaging:
- Must be able to read the brand and product name on the packaging
- Must have the ability to add weight/unit
- Adding by reading Barcode/QR Code
- Adding with voice detection
- Manual Typing

2.1.1.4. My/Our Cart

- There should be a list of products listed inside this car display.
- The list should include the properties of the products (product name, product quantity).
- There should be buttons for manually removing products from the list.
- There should be a button series where the purchase amount of the products can

be increased or decreased.
  - The first of these is if the user no longer wants to buy the product: the product quantity should be reduced and the product data should be removed from the list.
  - Second, if the user has already purchased: the product quantity will be reduced and the product data will be kept in the database as the day and quantity received.
- The user should be able to have more than one shopping cart.
- The location data of the selected place (home/office) for the user group or individual user will be retrieved. When users leave home with location services turned on, they will receive shopping list notifications.

2.1.1.5. Pricing

- After obtaining the necessary data from APIs with product information and prices, the user should be offered the most appropriate pricing.
- Price comparison: The difference between the average market price of the products and the best price we provide should be shown, and the user should be shown how much he has saved.

2.1.1.6. Statistics

- Individual and community data on the consumption habits of users should be shown.
- Data should be visualized in charts and pie charts with weekly and monthly periods.

2.1.1.7. Credits

- This section should include the names and pictures of the app developers

2.1.1.8. Quit

- It must allow the user to exit the game.

2.1.2. Nonfunctional Requirements

2.1.2.1. User Interface and Human Factors

In the mobile application to be developed:

- An interface with easy-to-use components and vivid colors that will not disturb the user should be used.
- The number of components on the main screen and on the screens directed after it should not exceed seven to make the application more understandable and user-friendly.
- Labels of tappable components, such as button names and screen labels, should be self-explanatory.
- It should have an interface that users can easily use without the need for an extra learning process. Potential users should be able to use it with their current application usage information.

2.1.2.2. Application Content

- The application to be made should solve a problem in daily life or suggest a much more practical solution than the ongoing habits.
- The **usability** of the application should be increased to provide a better experience with the sounds and routers to be used in its content.

2.1.2.3. Reliability

Users can make some mistakes while using mobile applications:

- To avoid these, confirmation pop-ups should appear on many transition screens and confirmation processes.
- Operations other than the user's request should not occur, except where otherwise permitted.

2.1.2.4. Supportability

- Must support both mainstream mobile platforms (iOS and Android)
- Must be able to read in accordance with various barcode systems.
- Must be able to integrate various cards into the system successfully.
- Should contain all possible order materials in the household.

2.1.2.5. Efficiency

- Users should be able to order the products they want in a much shorter time than normal purchases.
- Users should be able to control the deficiencies in the household automatically, rather than using traditional methods.
- The application should predict orders according to user habits and shorten the order and thinking process.

2.1.2.6. Extendibility

- By using the data entered by the users, an automatic system based on user habits can be created with the help of data training and artificial intelligence.
- Our application, which is mostly aimed at grocery shopping, can be easily adapted to many sectors where we can obtain data.
- There may be an updated version where the special offers and discounts are notified in real time and the shopping cart is arranged accordingly.

2.1.3. Pseudo Requirements

- The application will be for a mobile app.
- It will be developed by using React/React Native framework.
- In the backend part, we consider using the Python language and related libraries.
- The app will have a database for user and product datasets.
- The app will have a client-server architecture.
- We will use Git for version controlling and tracking.

3. Final Architecture and Design Details

3.1. Overview

Below, we explain System Decomposition, Controller Subsystem, Server Subsystems and Hardware/Software Mapping.

We had the following goals in our project:

- Making people's lives easier by making them remember what they needed to buy.
- Making people save money by seeing the minimum price of products from all markets
- Enabling users to share the contents of their shopping carts with other communities
- Making people reduce the market product waste by enabling them to keep track of what they bought
- Having an easy-to-use interface.

3.2. System Decomposition

We have used 2-tier architectural style for the design of our system. We have used Client-Server model in our system-architecture. This model is popular for Android-IOS development.

The Client tier is largely responsible for user interfaces and user experience. The Client layer also has dynamic responses for user gestures. This layer also has the Controller subsystem so that our Server layer can have a better understanding of the users' interactions to our system. The Server layer oversees communication between the application and the data sources. Main work of this layer is to manipulate the database schemes upon the changes of the entities. This layer also takes care of database configurations and database connections. Another main concern of this layer is to process the data sent by the users via the Controller subsystem.

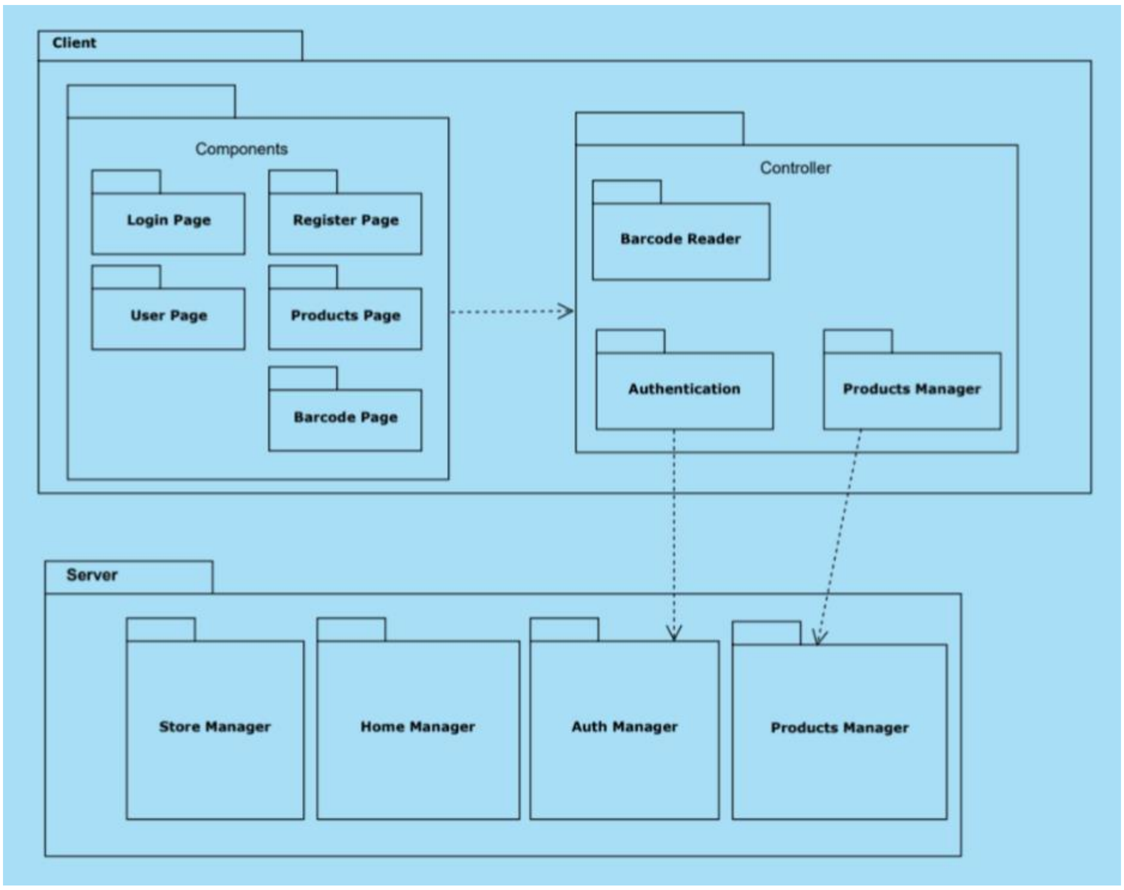


Figure 1: *Subsystem decomposition diagram*

3.2.1. Controller Subsystem

This subsystem is concerned with the connection of the frontend components and the server subsystems. This subsystem will take information from the user and will send this information to the server. And this subsystem will also ask for data from the backend to use on frontend. Authentication subsystem has the functionality of login, signup and logout for a user. *BarcodeReader* subsystem can ask the server to search for given barcodes. These barcodes will be taken from the camera of the app. User subsystem has the functionality of manipulating lists, adding products to user lists and the communities. *UserManager* class has the connection to backend to add products manually.

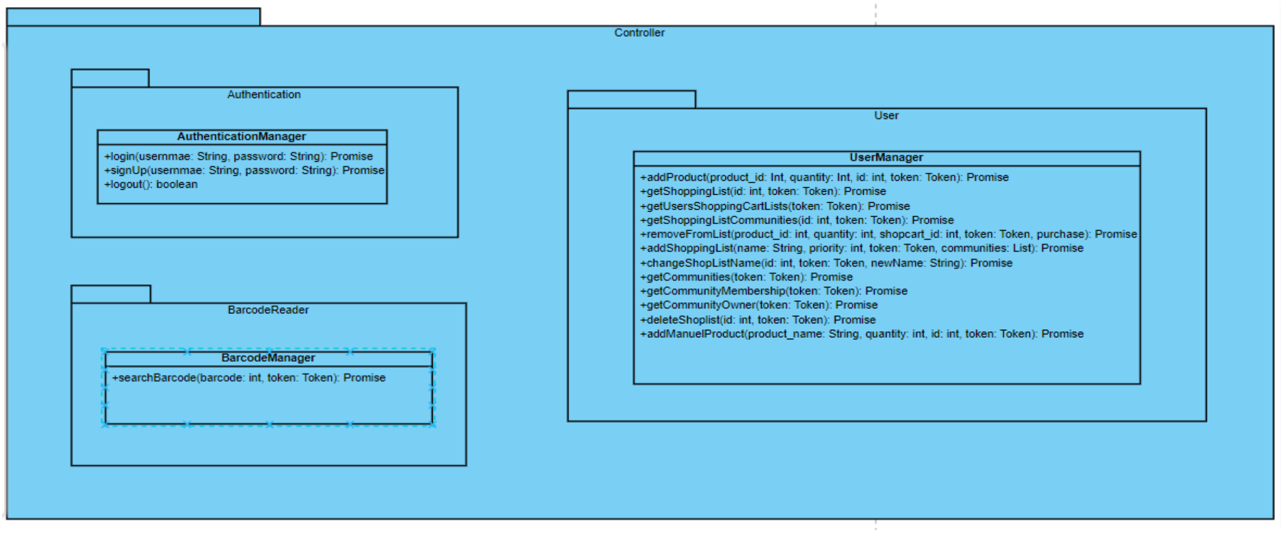

Controller
Authentication
AuthenticationManager
+login(usernmae: String, password: String): Promise
+signUp(usernmae: String, password: String): Promise
+logout(): boolean
BarcodeReader
BarcodeManager
+searchBarcode(barcode: int, token: Token): Promise
User
UserManager
+addProduct(product_id: Int, quantity: Int, id: int, token: Token): Promise
+getShoppingList(id: int, token: Token): Promise
+getUsersShoppingCartLists(token: Token): Promise
+getShoppingListCommunities(id: int, token: Token): Promise
+removeFromList(product_id: int, quantity: int, shopcart_id: int, token: Token, purchase): Promise
+addShoppingList(name: String, priority: int, token: Token, communities: List): Promise
+changeShopListName(id: int, token: Token, newName: String): Promise
+getCommunities(token: Token): Promise
+getCommunityMembership(token: Token): Promise
+getCommunityOwner(token: Token): Promise
+deleteShoplist(id: int, token: Token): Promise
+addManuelProduct(product_name: String, quantity: int, id: int, token: Token): Promise

Figure 2: Controller subsystem diagram

3.2.2. Server Subsystems

3.2.2.1. AuthManager Subsystem

This subsystem is concerned with the users and their authentication. *MetaUser* class has the necessary data for a given user. When we register *AuthManager* class will create a new object for the given user. When we try to login *AuthManager* will check its list of registered users. *AuthManager* class also has the capability of keeping statistical information about a given user. This class also has different serializers which are helpful for the login and registration processes.

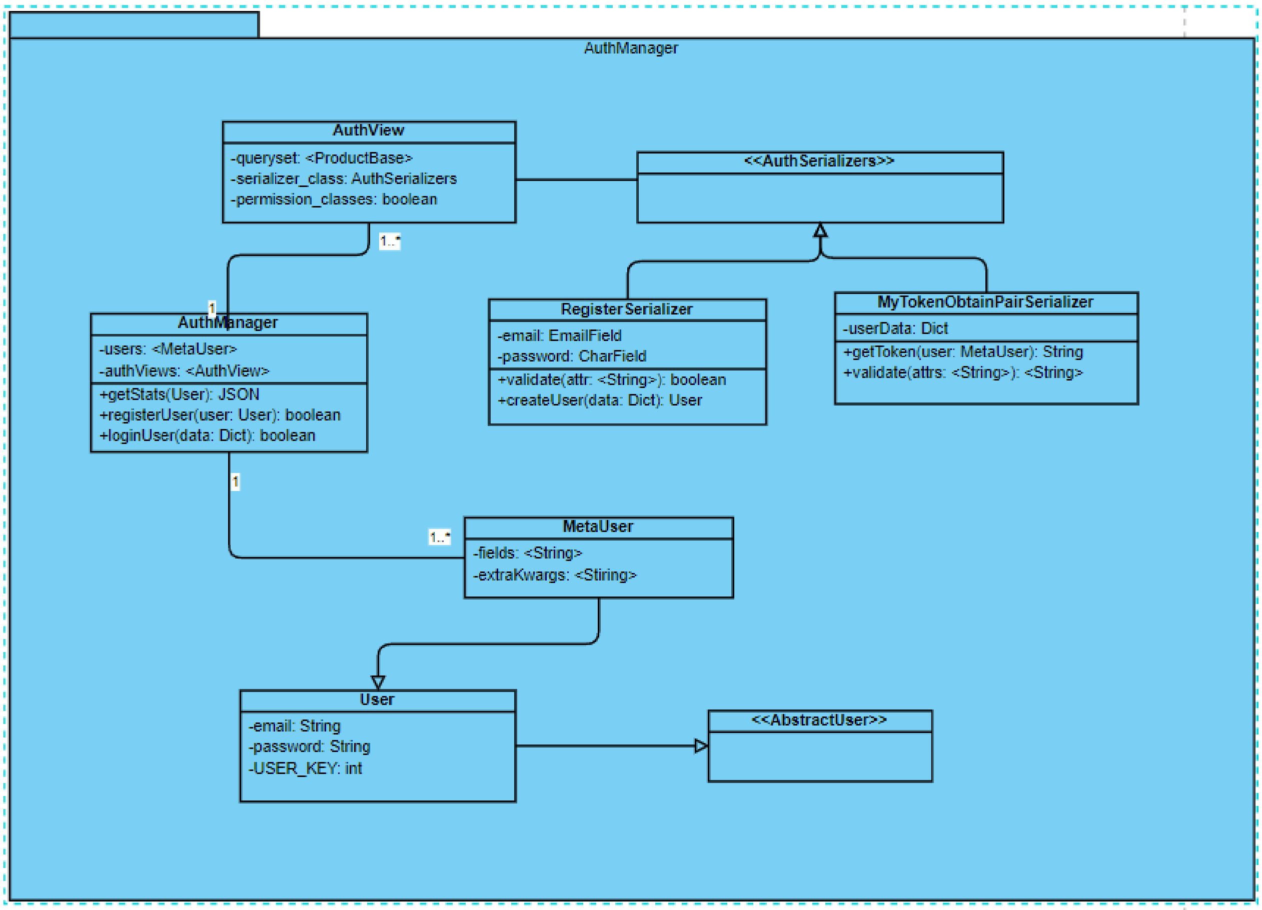


Figure 3: *AuthManager subsystem diagram*

3.2.2.2. BarcodeReader Subsystem

BarcodeReader subsystem has the capability of searching barcodes on the internet so that our system can find out information about the products the user has submitted. This information is the price, name, photo and the stores which sell the product. This information is returned using a Python dictionary in the *scrape(*) functions that have been overridden by different classes. There are five different classes that override the *scrape()* function. These classes scrape different sites. For example, *GoogleScrape* scrapes the Google Images site and the *CimriSoup* scrapes *cimri.com*. These classes use Python's *requests* and *BeautifulSoup* libraries. Scraper class holds the necessary amount of these scraper classes in a given environment.

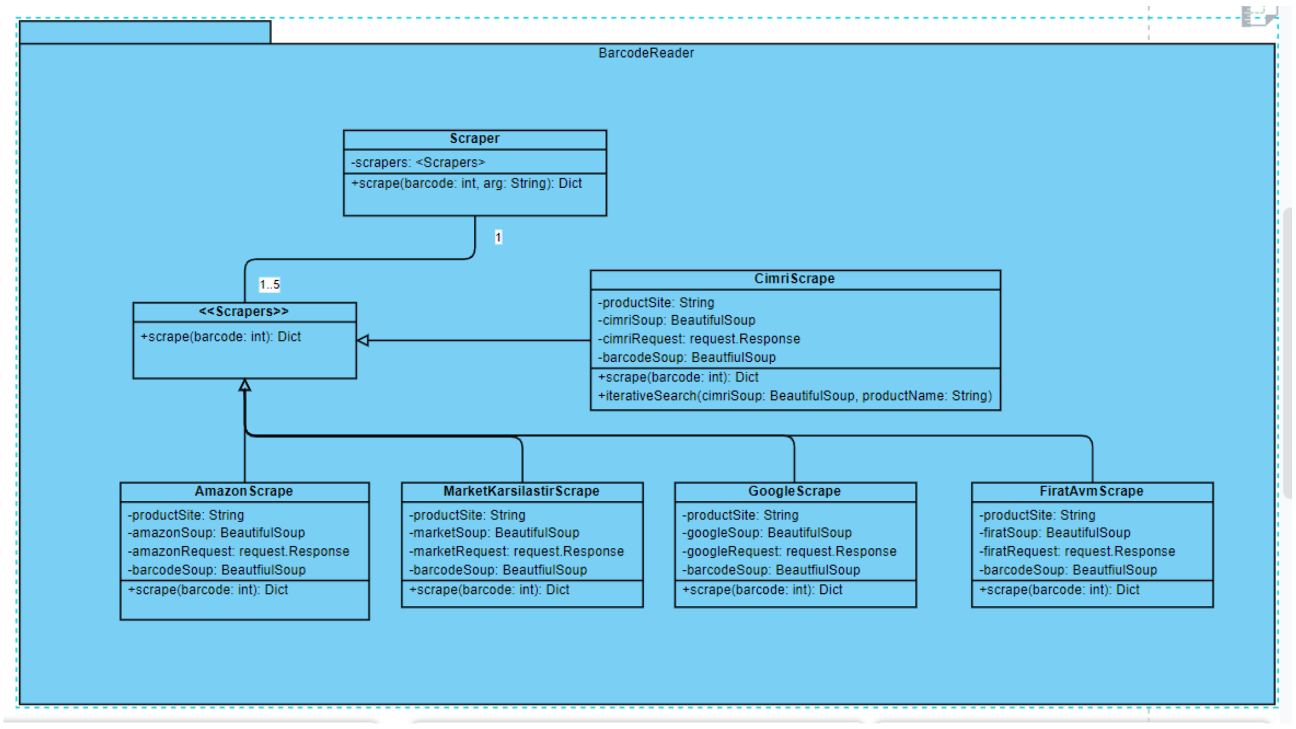

BarcodeReader
Scraper
-scrapers: <Scrapers>
+scrape(barcode: int, arg: String): Dict
1
1..5
<<Scrapers>>
+scrape(barcode: int): Dict
CimriScrape
-productSite: String
-cimriSoup: BeautifulSoup
-cimriRequest: request.Response
-barcodeSoup: BeautfiulSoup
+scrape(barcode: int): Dict
+iterativeSearch(cimriSoup: BeautifulSoup, productName: String)
AmazonScrape
-productSite: String
-amazonSoup: BeautifulSoup
-amazonRequest: request.Response
-barcodeSoup: BeautfiulSoup
+scrape(barcode: int): Dict
MarketKarsilastirScrape
-productSite: String
-marketSoup: BeautifulSoup
-marketRequest: request.Response
-barcodeSoup: BeautfiulSoup
+scrape(barcode: int): Dict
GoogleScrape
-productSite: String
-googleSoup: BeautifulSoup
-googleRequest: request.Response
-barcodeSoup: BeautfiulSoup
+scrape(barcode: int): Dict
FiratAvmScrape
-productSite: String
-firatSoup: BeautifulSoup
-firatRequest: request.Response
-barcodeSoup: BeautfiulSoup
+scrape(barcode: int): Dict

Figure 4: *BarcodeReader subsystem diagram*

3.2.2.3. **ProductManager Subsystem**

This package is concerned with the products that have been added by the users. *ProductBase* stores the information of the products. *PriceInCart* and *PriceInStore* are classes which are used when products are added to carts or are linked to the stores which sell them. *ProductManager* keeps a Scraper object which will be initialized with *initializeScraper(). ProductManager* takes the barcodes given by users and creates new *ProductBase* objects. *ProductManager* class also has the capability of changing the prices of the products. *ProductManager* also keeps the products that are in the cart.

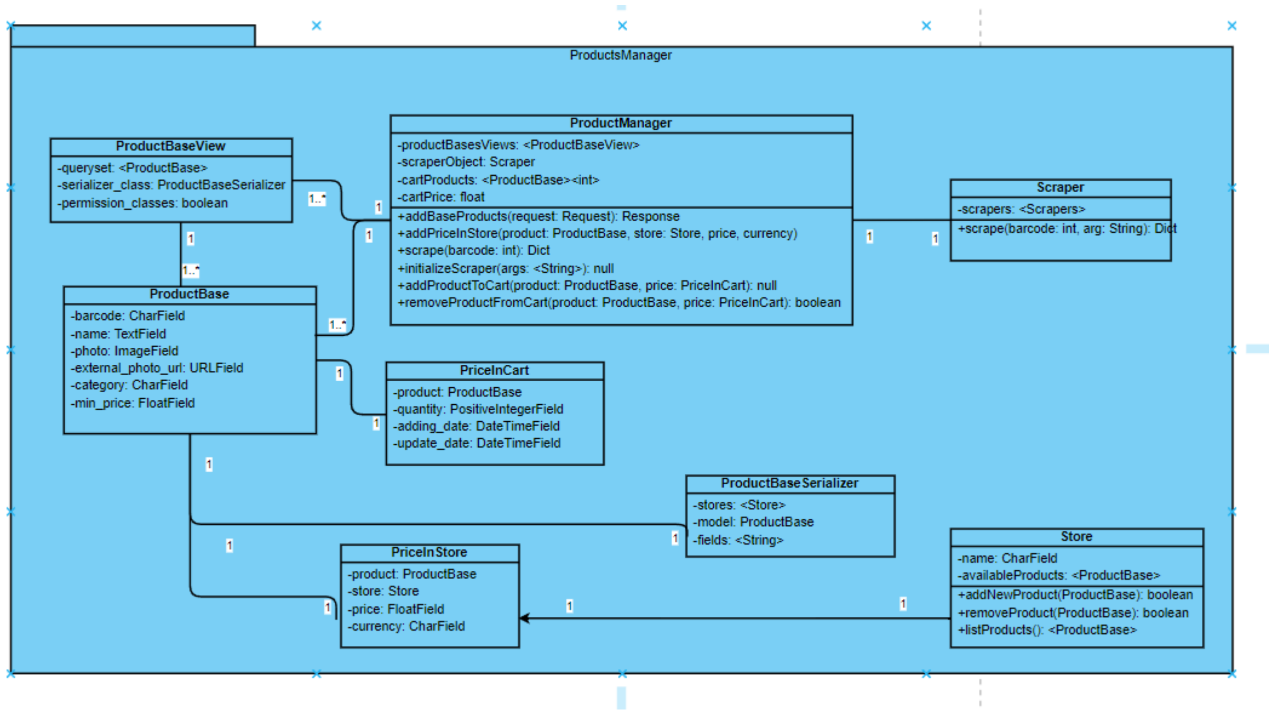

ProductsManager
ProductBaseView
-queryset: <ProductBase>
-serializer_class: ProductBaseSerializer
-permission_classes: boolean
ProductManager
-productBasesViews: <ProductBaseView>
-scraperObject: Scraper
-cartProducts: <ProductBase><int>
-cartPrice: float
+addBaseProducts(request: Request): Response
+addPriceInStore(product: ProductBase, store: Store, price, currency)
+scrape(barcode: int): Dict
+initializeScraper(args: <String>): null
+addProductToCart(product: ProductBase, price: PriceInCart): null
+removeProductFromCart(product: ProductBase, price: PriceInCart): boolean
Scraper
-scrapers: <Scrapers>
+scrape(barcode: int, arg: String): Dict
ProductBase
-barcode: CharField
-name: TextField
-photo: ImageField
-external_photo_url: URLField
-category: CharField
-min_price: FloatField
PriceInCart
-product: ProductBase
-quantity: PositiveIntegerField
-adding_date: DateTimeField
-update_date: DateTimeField
ProductBaseSerializer
-stores: <Store>
-model: ProductBase
-fields: <String>
PriceInStore
-product: ProductBase
-store: Store
-price: FloatField
-currency: CharField
Store
-name: CharField
-availableProducts: <ProductBase>
+addNewProduct(ProductBase): boolean
+removeProduct(ProductBase): boolean
+listProducts(): <ProductBase>
1..*
1
1..*
1

Figure 5: *ProductManager subsystem diagram*

3.2.2.4. CommunityManager Subsystem

This subsystem is concerned with the Community requirement declared in earlier reports. The *CommunityManager* class gives a user named the *communityOwner* the capability of adding and removing users from a community. *CommunityManager* holds different Communities and the users in these Communities can add and remove products. *CommunityManager* class has the functionality of listing the products in a given community.

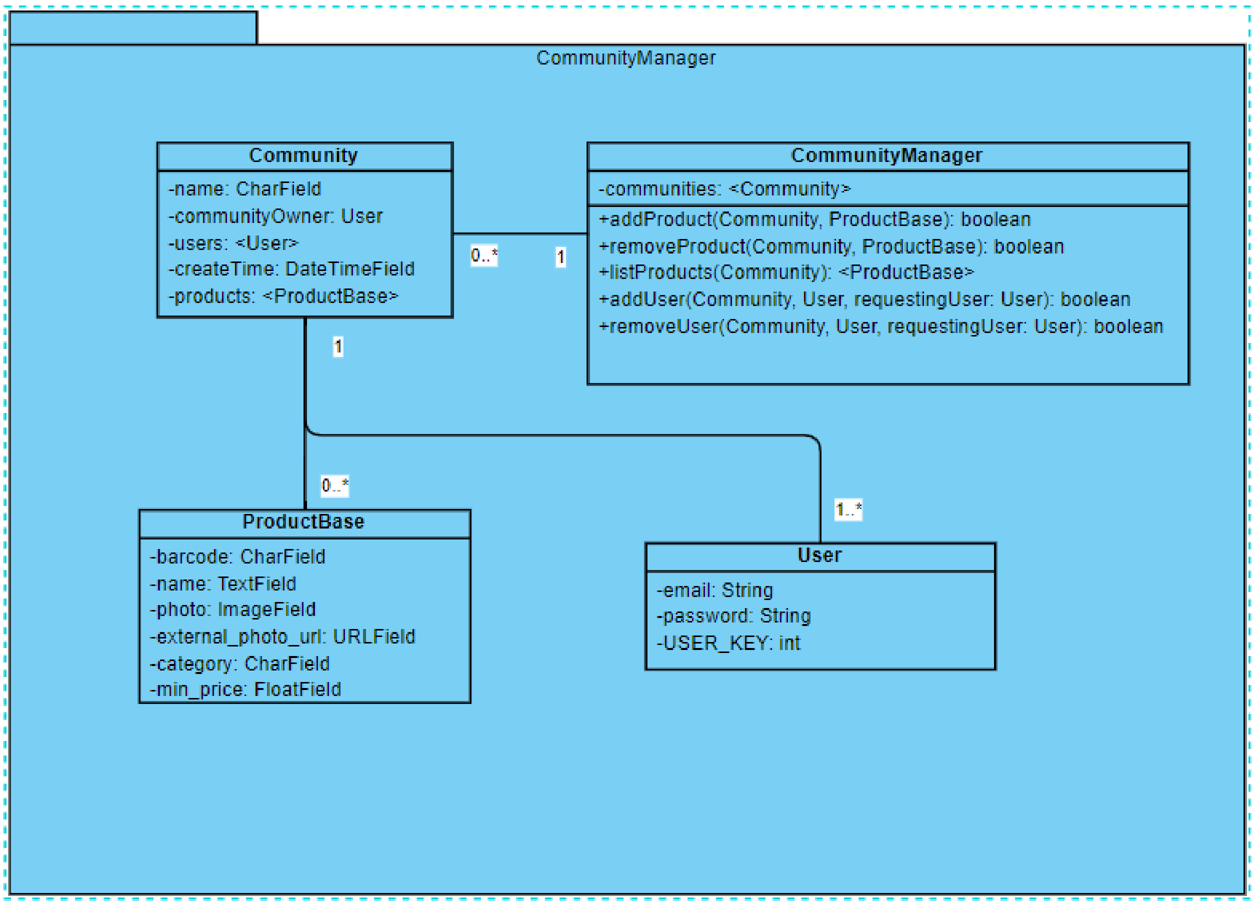

CommunityManager
Community
-name: CharField
-communityOwner: User
-users: <User>
-createTime: DateTimeField
-products: <ProductBase>
CommunityManager
-communities: <Community>
+addProduct(Community, ProductBase): boolean
+removeProduct(Community, ProductBase): boolean
+listProducts(Community): <ProductBase>
+addUser(Community, User, requestingUser: User): boolean
+removeUser(Community, User, requestingUser: User): boolean
0..*
1
1
0..*
1..*
ProductBase
-barcode: CharField
-name: TextField
-photo: ImageField
-external_photo_url: URLField
-category: CharField
-min_price: FloatField
User
-email: String
-password: String
-USER_KEY: int

Figure 6: *CommunityManager subsystem diagram*

3.2.2.5. StoreManager Subsystem

The *StoreManager* subsystem keeps the stores and the products sold by these stores. *StoreManager* class can add and remove Stores. This class can also list the products in a given Store. Stores have the products that are sold by these Stores. Stores have unique names. Stores can also list the products sold by these Stores. We can add and remove products in a Store. *PriceInStore* class has a connection with the Store class and the *ProductBase* class.

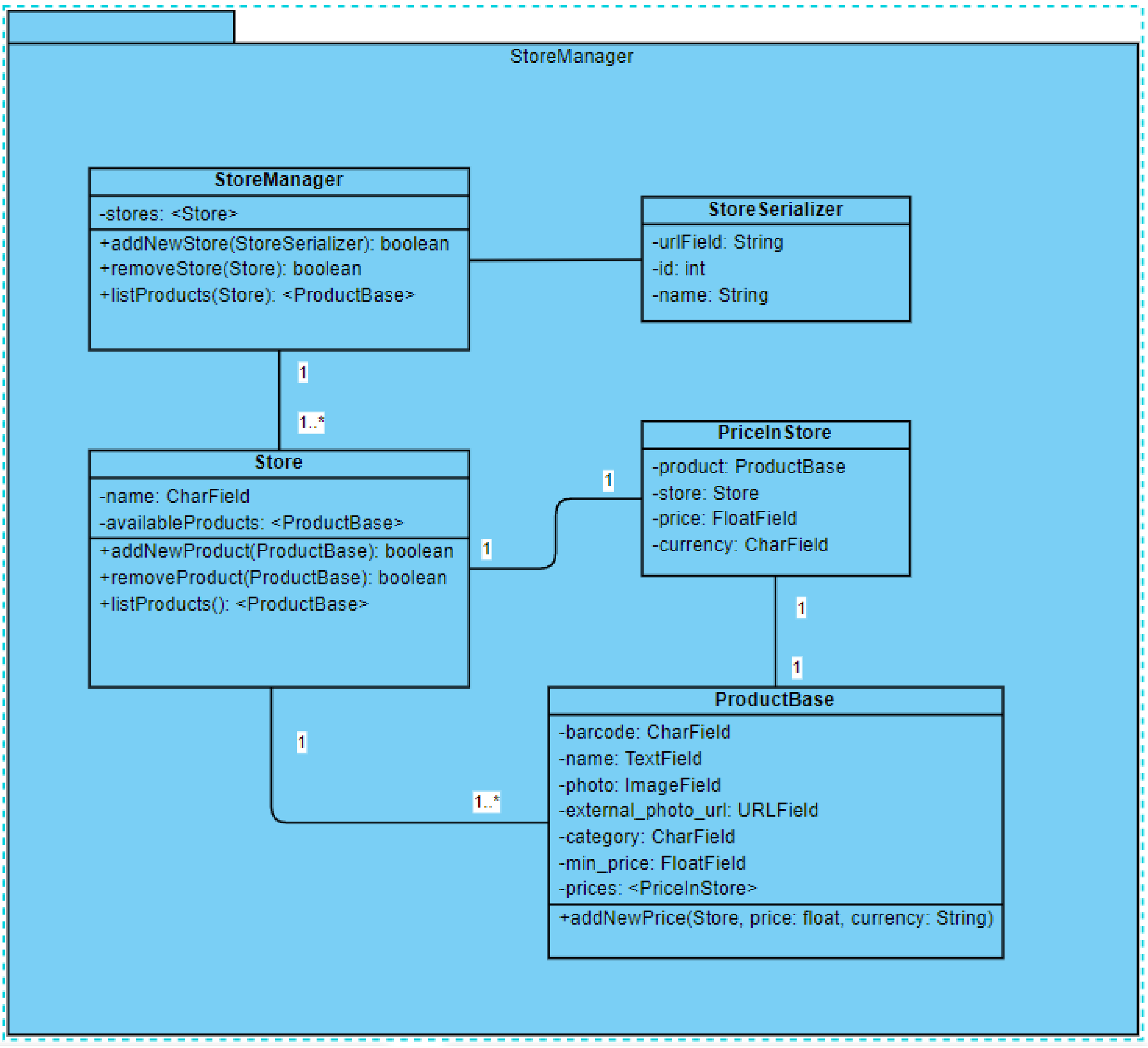


Figure 7: *StoreManager subsystem diagram*

3.3. Hardware/Software Mapping

ShopCart will be a mobile application developed with React Native. The user will not need an extra device other than a mobile phone to use the application. The application, which will be multiplatform, will support both Android and iOS platforms. The application will be able to use the phone's camera so that it will be able to scan images from the environment. Since the data coming from the camera will be processed either in the form of reading barcode / QR Code or scanning the product image, it is planned to send the data obtained to the necessary servers and receive their answers. In the application, mobile internet networks will be used and data exchange will be made over the internet. Thanks to the data we obtain through various APIs, comparisons to the products determined by the user will be made on the backend of the application, the results will be shared with the user and the necessary information will be stored in the database. The application, which also supports the shared use of multiple people, will use a synchronized database and common network for this.

4. Development/Implementation Details

4.1. Front-End

For the frontend of Shopcart, we chose React-Native [3], which is an open-source, front end, JavaScript library for building user interfaces or UI components [3]. We chose React because it provides ready to use mobile components which are responsive across the both android and iOS platforms. Additionally, React-Native has encouraged reusability of UI components by enabling us to build front-end components that are easier to modularize compared to other technologies. The fact that some of our friends had prior knowledge on this subject was an important factor in our preference for React-Native.

The front-end consists of the Client tier’s view components. In this tier, the classes are responsible for creating the UI components to build the front-end of the application. The Client tier has multiple pages. We have initially started writing the User's pages then we have added the Community pages and their functionalities. After that we have added the List pages and their functionalities. Finally, we have added the Barcode scanner page so that we have implemented a use-case declared in earlier reports.

The necessary screen-shots of the front-end of the application can be found in the User Manual section of the report.

4.2. Back-End

Django Rest Framework was used in the back-end of ShopCart. There are many factors in making this decision, a few of which can be briefly explained. First of all, Django was developed with Python, which is relatively easy to learn and write. Also, Django has extensive documentation and community support since it is an open-source web framework. Django’s ORM support has been very beneficial for the integration of the project and the database, the migration of the project can be realized with a few lines of code, if needed. Besides, great community support and its security standard were instrumental in our choice of Django. Finally, the built-in web-browsable admin panel has provided great convenience in many stages of the project.

Our backend consists of the subsystems mentioned in the section 3. These subsystems are supported by the Serializers that are useful at converting the query sets to native Python objects which can be then rendered to specific content types such as JSON and *ViewSets* which is the controller where the incoming data gets processed and the request gets formed by making necessary query calls. We have created Serializers and *Viewsets* for all models to manage and modify the data easily and created custom endpoints for more specific use-cases such as adding or removing a product to a shopping cart.

4.3. Persistent data management

The data and other user inputs from the camera are the main drivers of the application. The data obtained from Expo-barcode-scanner framework ShopCart also receives data from the APIs of famous e-commerce sites for price tracking, such as *cimri.com* and compares the data of the markets and returns the most appropriate output to the user. [4] We also use SQL injection protection; CSS protection features of the Django framework so that data can be stored securely. We use the JWT token for authentication and periodically refresh these tokens. [5] The application will also have a permanent data store because the users' usernames, passwords, product baskets, group information etc. will be stored in this warehouse. On the contrary, we found it pointless to keep the pictures and barcode/QR Codes taken by the users while using the application in any storage, and we do not intend to store this data in any way. As a result, we will store the above-mentioned information in a database using PostgreSQL, using the Django framework.

5. Testing Details

In order to develop a quality mobile application, it is very important to get feedback from users and improve deficiencies accordingly. In an ideal system, although test automation would be very useful, we prioritized application development with the goal of producing a useful and functional application. While developing, we tried to detect errors by doing manual testing with the utmost care.

5.1. User Interface

Since our application will be used by normal phone users, we have taken care to make our application interface as clear and simple as possible. Thus, every user would be able to use the application easily without the need for prior notification. That's why we designed the interface components self-explanatory and simple. While testing the designed interfaces, we used people who do not have much experience with the phone. We have also improved the voice input feature as well as image identification for easier use.

5.2. Usability

Usability is another important aspect for ShopCart. Since our goal is to make it easy for users to use, we tried our app on many devices and in many possible situations (low charge, bad network conditions, low RAM). As a result, we have verified that our app provides acceptable use.

By keeping the logic of the application as well as the interface very simple, we have made it easy for every user to use it. Our manual tests have produced results that confirm this.

5.3. Security

Although our application does not keep very critical information about the user, this may vary and it is still the application's duty to protect personal information. Django's security system was quite adequate to protect this information, and the fact that it was free gave us a lot of relief.

6. Maintenance Plan and Details

6.1. Maintenance and Optimization of the Servers

The monitoring dashboard component relies on certain versions of the React UI component libraries. Thus, we needed to check and update the external libraries we are using to maintain the code base dependencies. We have used the AWS services to rent servers, we rented and wrote configuration files to deploy our software. Based on the number of customers we will deploy our product; we can easily scale using the AWS services. Additionally, we monitor the performance of our software such as latency and throughput to make sure we fulfil our non-functional requirements.

6.2. Issue Tracking and Bug Triaging

Even though we as project members strongly stand for Open-source software (OSS), our business model is B2B which requires us to work closely with the business partners. Thus, we only track issues published by the project members. Additionally, we manually open issues based on the feedback we receive from our customers and testing groups. The issues assigned to the project members who are responsible for specific features of the product which issues are identified. Since we just started our user tests and finished the development of the minimum viable product. We aim to continue issue tracking and bug triaging.

6.3. Maintenance of the Deployed Software

We conducted user experience interviews in the field location Alanya. We chose Alanya due to its convenience of access for us and the profile of restaurants in the region. Most restaurant owners in Alanya are familiar with the English language due to the high tourist population of the region. We will write legal documents for the user agreements as we deploy our product to the restaurants to protect us and the restaurant owners/customers from the legal obligations of the copyrighted software. We will continuously collect user feedback on our product hopefully updating our software for a better user experience.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

7.1.1. Sustainability

Sustainability is the motivation of our project. Our resources are limited, and the total population of the world is ever-growing. This means fewer resources for each individual. Our project aims that the users buy what they need and do not waste grocery resources.

7.1.2. Public Safety

Since the continuing epidemic, it is critical for public safety to remain isolated. Our team is unable to meet in the same setting as frequently as we would want as a result of this element, making the development process more difficult in terms of cooperation and teamwork.

7.1.3. Technological Factors

Technology is always changing, and this is something we need to keep in mind for our project. So that our work does not become obsolete, we must maintain track of this evolution and incorporate it into our project. We must also keep an eye out for comparable items and modify our application accordingly for our product to be useful.

7.1.4. Cultural Factors

Cuisine may differ from culture to culture. These differences can affect some predetermined cuisines and recipes added to the system. People from different cultures might view these recipes as unfamiliar.

7.1.5. Social Factors

Users must communicate with the program in a language that they understand. We need to explore translating our application into multiple languages to make it available worldwide. Our application will be in English by default, which might be a problem given that English is spoken by around 20% of the world's population.

	Effect level	Effect
Sustainability	10	As our resources are getting scarce, sustainability helps reduce the negative effects.
Public safety	5	Making cooperation and teamwork harder for development due to isolation.
Technological factors	10	Change in design for developments of technology we utilize and the existence of similar applications.
Cultural factors	6	Change in the accuracy of image processing because of cultural differences in appearance.
Social factors	6	The language barrier between users and the application.

Table 1: Factors that can affect analysis and design

7.2. Ethics and Professional Responsibilities

We are aware of the expectations and responsibilities regarding ethics and professionalism, so we have developed our practice in line with these contexts.

7.2.1. Ethical Responsibilities

- We will adhere to the General Data Protection Regulation [6] and the Code of Ethics [7][8].
- The application should not collect customers' personal information without their consent.
- After the customer leaves the application, the customer sessions will be closed. This aims to prevent possible misuse.
- Reducing the interaction between the customer and the seller and the fact that the customer can find cheap products much faster provides the customer to get rid of many

cultural and ethical concerns. Shopping from many different points can raise some concerns for both the customer and the seller.

7.2.2. Professional Responsibilities

- The source code of the project keeps in a private repository.
- We believe that the use of open-source or free services in our application will not pose a professional problem.
- Our team shares the tasks right after the project announcement times and meets close to the project delivery (usually 1 week before) and holds a meeting about the state of the flow.
- The work will be divided accordingly among the team members based on their technical expertise and interest.

7.3. Judgements and Impacts to Various Contexts

The following table explains the project impact within various dimensions (impact in a global context, impact in an economic context, impact in environmental context, impact in societal context):

	Impact Level	Impact
Impact in Global Context	5	The app will help customers easier and more accessible market items.
Impact in Economic Context	8	The app will be free and can be sponsored by various potential companies. The app will provide more efficient shopping experience to its customers
Impact in Sustainable Context	6	The application aims to make repetitive shopping easier by following user habits and to easily add out-of-stock items to the cart.
Impact in Environmental Context	1	The app will provide less transportation cost and fuel use.

Table 2. Evaluation of impact based on the judgement

7.4. Teamwork Details

This section explains how we share tasks among group members and the strategies we apply to balance the workload among team members.

7.4.1. Contributing and functioning effectively on the team

We used a variety of communication channels, as discussed in the following sections. We generally used verbal and written communication channels such as WhatsApp, Discord, and held face-to-face meetings to assign tasks to group members. We gave priority to the skills and interests of the team members while distributing the work. Below is a brief description of each group member who contributed to the project.

Ahmet Kaan Uğuralp:

Took part in the deployment of the system to the cloud and both front-end and back-end work; contributed mainly to the server part. Also created *FiratAvmScrape*r and the *MarketKarsilastirScraper*. Additionally, configured, operated and performed maintenance on software to ensure the requirements of the project met. He also took responsibility in the development of the data model for the system.

Ahmet Işık:

Took part in the front-end work. Created all the screens, components, API services, stylings in frontend. He also helped back-end architecture design phase. He performed maintenance on software to ensure the requirements of the project met. He took an active role in connecting front-end views with the back-end of the application. He also took responsibility in the development of the data model for the system.

Furkan Ahi:

Took part in the architecting and designing phase. Primarily responsible for the reports and presentations. He also created an app logo. He had some contribution about data modelling and high-level design.

Mehmet Yaylacı:

Took part in the scraping part of the back-end. Created *CimriScraper*, *AmazonScraper*, *GoogleScraper* classes. Primarily contributed to the development of the ShopCart API. He had some contribution about data modelling and high-level design. He also met with *cimri.com* engineers and sharing ideas about our project. He is the creator of project video.

Ravan Aliyev:

Took responsibility in the development of the data model for the system. He had some contribution about data modelling and high-level design. He designed most part of the UI/UX. He also created the project poster.

Note: All members took part in the report preparation process and helped with the bug solving, code reviewing, and software development. Although each member of the group held accountable for certain features and parts of the software, everyone helped each other and shared tasks with each other to ensure the project delivered according to the specified deadlines.

7.4.2. Helping creating a collaborative and inclusive environment

The project was conducted in a collaborative and inclusive working environment. Each member was asked to make improvements on the project in their spare time from other courses. Task sharing regarding areas of interest was made close to the project announcements and deadlines. We tried to listen to each other and give tasks that align with the learning goals of our group members. In our meetings, we tried to get each other more by sharing our duties in other courses and our views on the weight of the workload. We worked together from time to time apart from time to time to save time.

7.4.3. Taking lead role and sharing leadership on the team

Although we usually make decisions together, we chose Ahmet Işık as the leader of our team. Ahmet made the general assignment and assignments. Kaan was given full authority in the back-end part and it was ensured to work as healthy as possible. In the reporting part, Furkan took on the tasks of planning, writing and finalizing the majority of it by himself. Revan was responsible for UI/UX design, Mehmet front-end and various other services. In short, although we determined a leader, we shared the leadership among ourselves, in the sub-sections, and we proceeded on this path in a healthy way.

7.4.4. Meeting objectives

At the analysis phase, we devised a project plan to keep track of the requirements and deadlines we need to meet in order to implement the project and get feedback from our users before the final project deadline.

We removed some of the features that we initially planned after getting feedback from our supervisors and mentors. Specifically, the features related to AI and complex UI. Although it is good, we removed this feature due to the concern of not making it to the deadline and improved the UI. During the development phase, we realized that the timeframes we envisioned in the analysis phase were too optimistic.

For collaboration and communication on the project, we made use of online mediums such as Zoom, Discord and WhatsApp. Also, we kept a shared folder on Google Drive and OneDrive which made it accessible for all of the project members.

7.4.4.1. Functional Objectives

In the following section, we are listing the functional requirements that met for the project.

7.4.4.2. Login Screen

- The user must be prompted for a username and password.
- “Forgot your password?” even though the user has forgotten his password. There should be a button and this button should direct the user to the required page.
- For users who have no account or want to open a new account, there should be a

"Create New Account" button and this button should direct the user to the required page.

7.4.4.3. Register Screen

- The user must enter the required information for registration.
- Users should enter the "Community Code" on this screen if they intend to use a shared shopping cart.

7.4.4.4. Add Product to Cart

- Adding the product by scanning its packaging:
  - Must be able to read the brand and product name on the packaging
  - Must have the ability to add weight/unit
- Adding by reading Barcode/QR Code
- Adding with voice detection
- Manual Typing

7.4.4.5. My/Our Cart

- There should be a list of products listed inside this car display.
- The list should include the properties of the products (product name, product quantity).
- There should be buttons for manually removing products from the list.
- There should be a button series where the purchase amount of the products can be increased or decreased.
- The first of these is if the user no longer wants to buy the product: the product quantity should be reduced and the product data should be removed from the list.
- Second, if the user has already purchased: the product quantity will be reduced and the product data will be kept in the database as the day and quantity received.
- The user should be able to have more than one shopping cart.
- The location data of the selected place (home/office) for the user group or individual user will be retrieved. When users leave home with location services turned on, they will receive shopping list notifications.

7.4.4.6. Pricing

- After obtaining the necessary data from APIs with product information and prices, the user should be offered the most appropriate pricing.
- Price comparison: The difference between the average market price of the products and the best price we provide should be shown, and the user should be shown how much he has saved.

7.4.4.7. Statistics

- Individual and community data on the consumption habits of users should be shown.

- Data should be visualized in charts and pie charts with weekly and monthly periods.

7.4.5. Non-Functional Objectives

In this section, non-functional requirements that are met will be discussed. To make them good, whole members have put a great effort.

7.4.5.1. User Interface and Human Factors

In the mobile application to be developed:

- An interface with easy-to-use components and vivid colors that will not disturb the user should be used.
- The number of components on the main screen and on the screens directed after it should not exceed seven to make the application more understandable and user-friendly.
- Labels of tappable components, such as button names and screen labels, should be self-explanatory.
- It should have an interface that users can easily use without the need for an extra learning process. Potential users should be able to use it with their current application usage information.

7.4.5.2. Application Content

- The application to be made should solve a problem in daily life or suggest a much more practical solution than the ongoing habits.
- The **usability** of the application should be increased to provide a better experience with the sounds and routers to be used in its content.

7.4.5.3. Reliability

Users can make some mistakes while using mobile applications:

- To avoid these, confirmation pop-ups should appear on many transition screens and confirmation processes.
- Operations other than the user's request should not occur, except where otherwise permitted.

7.4.5.4. Supportability

- Must support both mainstream mobile platforms (iOS and Android)
- Must be able to read in accordance with various barcode systems.
- Must be able to integrate various cards into the system successfully.
- Should contain all possible order materials in the household.

7.4.5.5. Efficiency

- Users should be able to order the products they want in a much shorter time than normal purchases.
- Users should be able to control the deficiencies in the household automatically, rather than using traditional methods.

- The application should predict orders according to user habits and shorten the order and thinking process.

7.4.5.6. Extendibility

- By using the data entered by the users, an automatic system based on user habits can be created with the help of data training and artificial intelligence.
- Our application, which is mostly aimed at grocery shopping, can be easily adapted to many sectors where we can obtain data.
- There may be an updated version where the special offers and discounts are notified in real time and the shopping cart is arranged accordingly.

7.5. New Knowledge Acquired and Applied

We learnt a lot about the development of large-scale cloud systems throughout the development of *ShopCart*. Additionally, we applied and integrated good software engineering practices to write maintainable, and modularized code in alignment with the best industry practices. Following are the topics in which we developed our applied technical knowledge.

- AWS Cloud Services
- React-Native Development
- Django and Networking
- Software Development Planning
- Software Architecture Planning
- User Experience Research
- Cloud Applications
- Mobile App Development
- UI/UX Design
- Logo Design
- Video Editing
- Website Creation & Design

Since we wanted to implement a good application, we needed in-depth knowledge about mobile development. Developing a useful grocery/market shopping app required us to understand how a market app operates. Additionally, we learnt a lot about the cloud service providers such as AWS S3 and how to use their services to develop applications that can serve numerous users, which did not be covered in Bilkent's courses. In order to develop user experience, we look through the UI design. Following through our choice of deployment platform, we gained a broad perspective on mobile development and maintainable application design. Each member held responsible for specific part of the app and assigned with a role. Some methods of learning we used to learn the topics discussed above

- Literature Review
- User Experience Research
- Online Learning
- Hands-on Experience
- Self-Learning

We used a literature review to understand common architectural design and pattern choices made for a mobile application. We followed through online documentation and web tutorials to learn technologies such as React Native for the front-end and Django/Python for the backend. The tutorials and open-source software available accelerated the development phase. Self-learning and hands-on experience were the most crucial part of our learning process since we got the opportunity to experiment with the software and were able to learn from our own mistakes.

8. Conclusion and Future Work

8.1.1. Conclusions

We managed to create a system that we planned starting from the beginning of our Senior Year. Now *ShopCart* is nearly ready to be released on both *Google Play Market* and *App Store*. We are happy with our progress and the user experience that are provided by us. We would like to be able to take more customer feedback and act accordingly but unfortunately, time limits and COVID-19 restrictions have blocked us. Throughout the project, we learnt to become a better team player and improved our software engineering skills. We welcome any feedback on our product as we strive to improve the user experience and get better.

8.1.2. Future Work

Our priority may be to release new versions that are improved in line with our functional and non-functional requirements in line with the feedback we receive from our users.

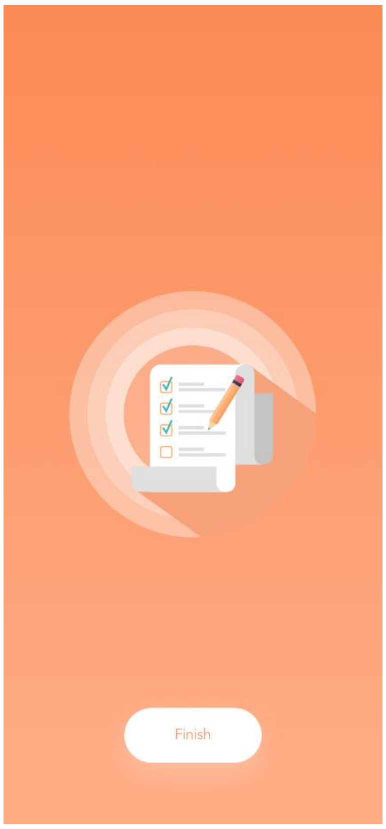
As the use of ShopCart increases, more APIs can be accessed, business agreements with more companies can be made, and campaigns can be made in which both parties will be profitable. In addition, customers can make profitable purchases for themselves.

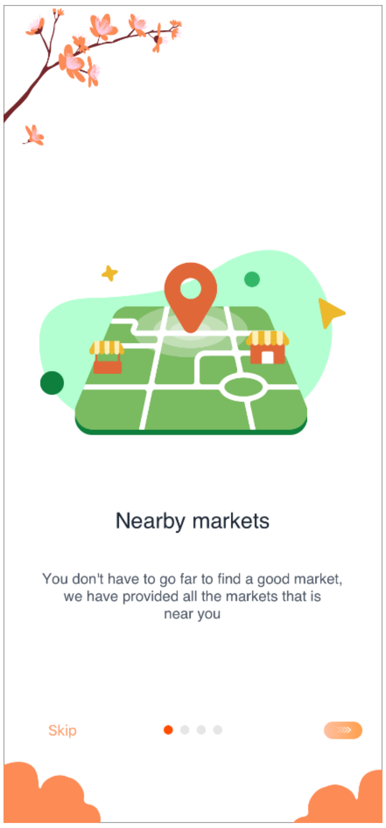
The feature we most want to add in our dreams is an artificial intelligence supported shopping assistant. By collecting various user data and following user habits, we plan to do their daily shopping with the help of an assistant, without forcing them to think. However, developing these versions before reaching sufficient competence in concepts such as data privacy, artificial intelligence, and deep learning is waiting as a future job for now.

9. User Manual

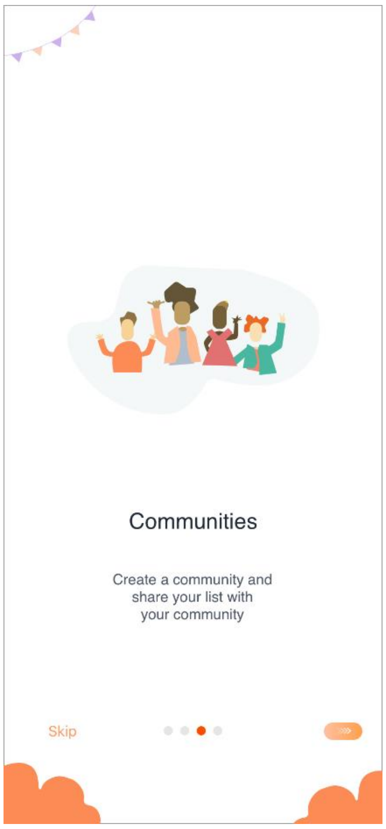
In this part of the report, we are providing a user manual for the ShopCart. We are providing screenshots from our app with their explanations.

9.1. Tutorial Screens

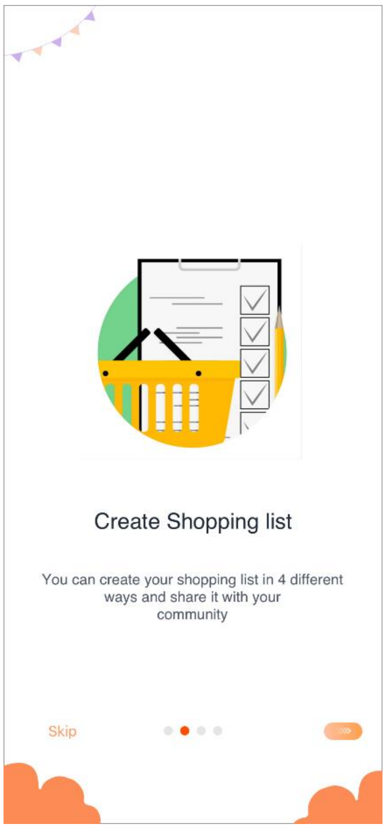



Nearby markets

You don't have to go far to find a good market, we have provided all the markets that is near you


Communities

Create a community and share your list with your community

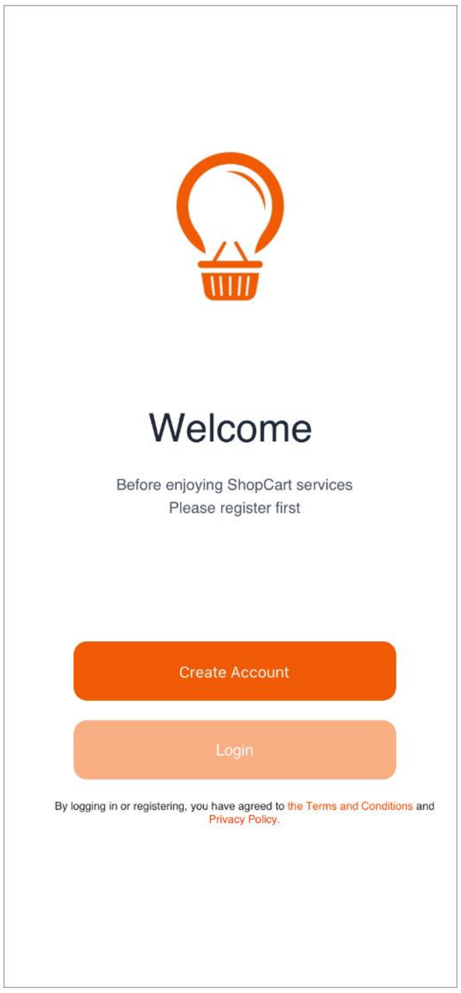

Create Shopping list

You can create your shopping list in 4 different ways and share it with your community

Screenshot 1-2-3-4: Tutorial Screens

These screens consist of summary information created for the user to easily learn the flow and usage at the first login to the application.

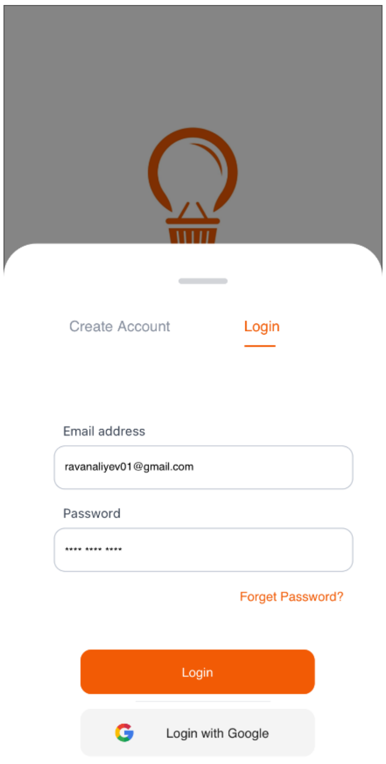
9.2. *ShopCart* Welcome Screen


Welcome
Before enjoying ShopCart services
Please register first
Create Account
Login
By logging in or registering, you have agreed to the Terms and Conditions and Privacy Policy.

Screenshot 5: Welcome Screen

This screen is the user welcome screen. Here the user can login if registered, can create an account if not.

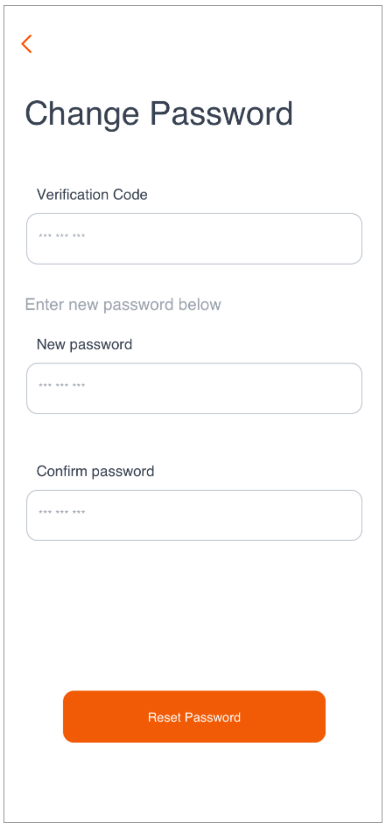
9.3. Login Screen and Forgot Password Screen



Screenshot 6: Login Screen

On this screen, you are directed to the usual login or password renewal options.

The buttons are not activated until the required information is entered, and if a new password is created successfully, a success message is displayed.

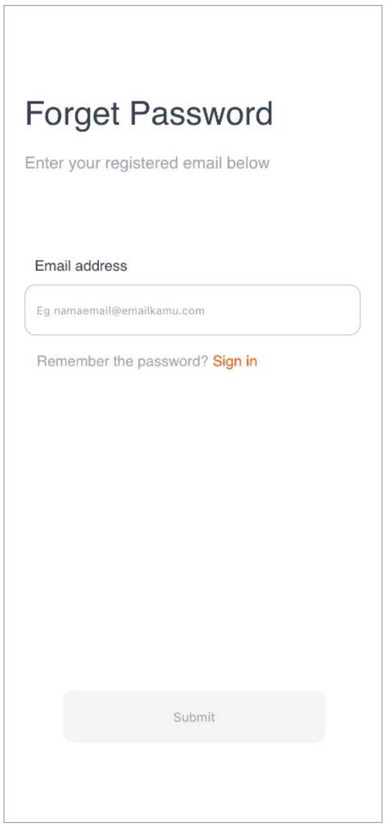

Change Password

Verification Code

Enter new password below

New password

Confirm password

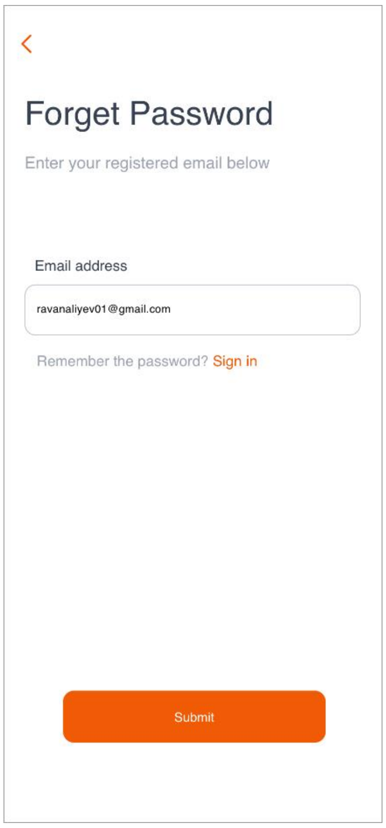

Forget Password

Enter your registered email below

Email address

Eg namaemail@emailkamu.com

Remember the password? Sign in

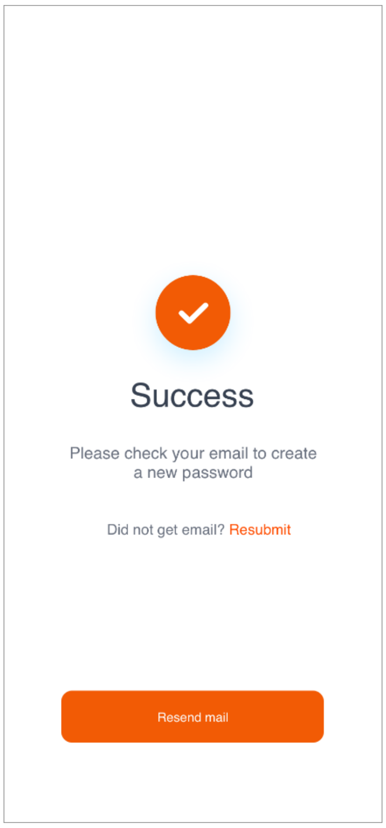

Forget Password

Enter your registered email below

Email address

ravanaliyev01@gmail.com

Remember the password? Sign in

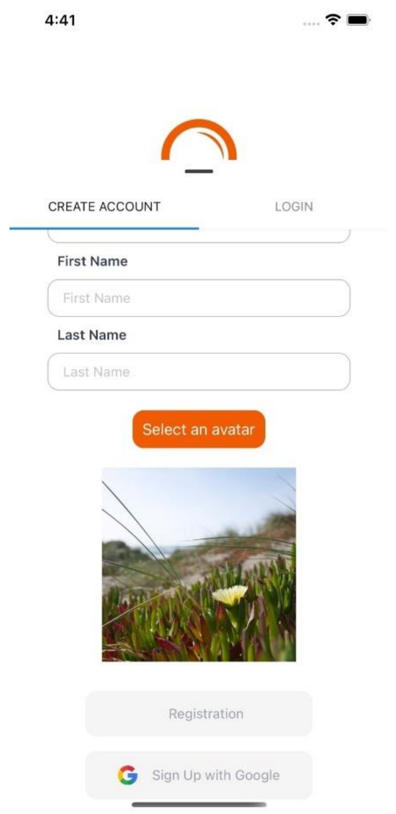

Success

Please check your email to create a new password

Did not get email? Resubmit

Screenshot 7-8-9-10: Forgot & Change Password Screens

9.4. Register Screen

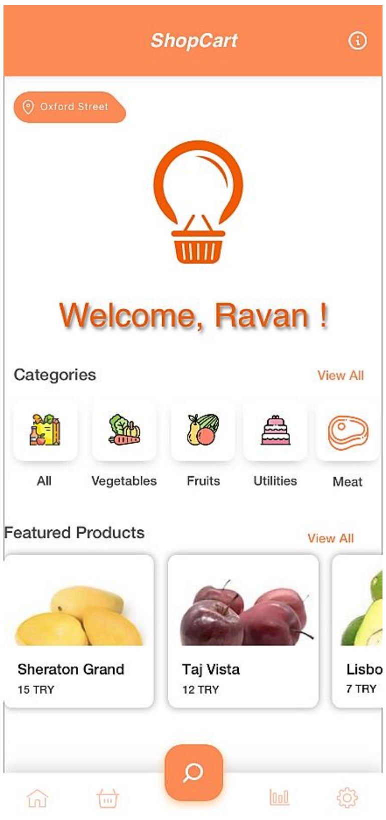


Screenshot 11: Register Screen

On this screen, you are directed to the usual registration screen.

You can also sign in with a Google account.

9.5. Home Screen



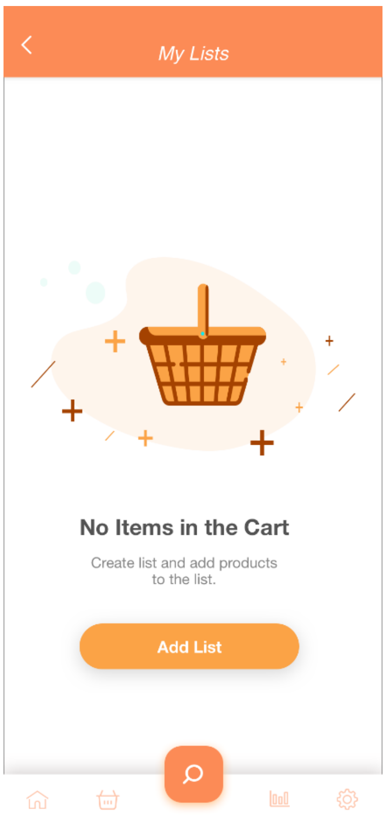
Screenshot 12: Home Screen

On this screen,

- main categories of the market,
- profile button,
- address bar and
- featured products
- search button
- home button
- my lists button (basket icon)
- statistics button
- settings button (community section is also in here) are displayed.

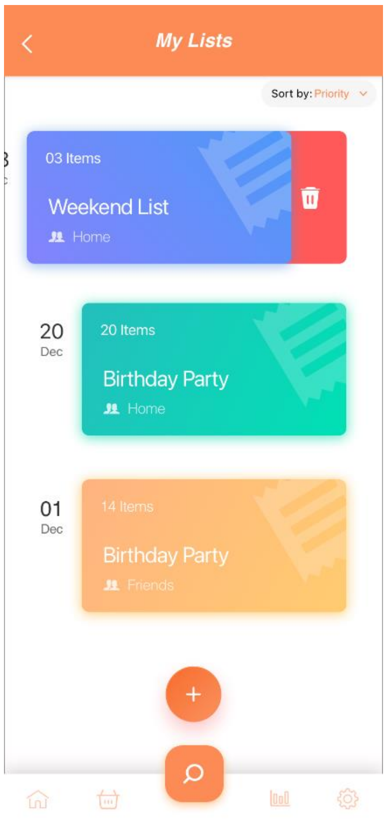
You are directed to the necessary screens with these buttons.

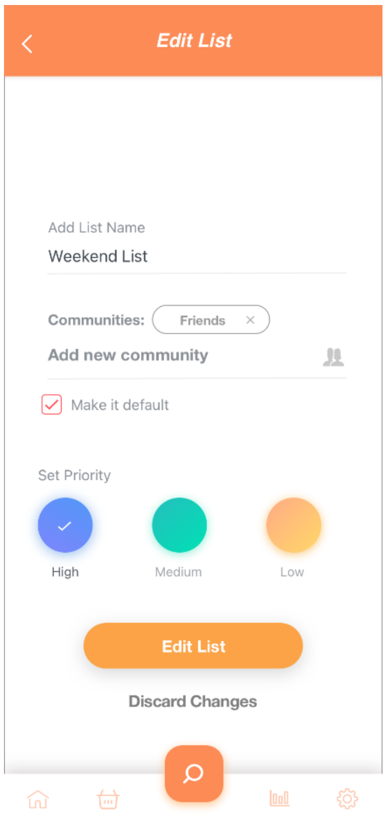
9.6. My Lists Screens



Screenshot 13: My List Screen when a user has no list

On this screen, a user can create a list from *Add List* button.





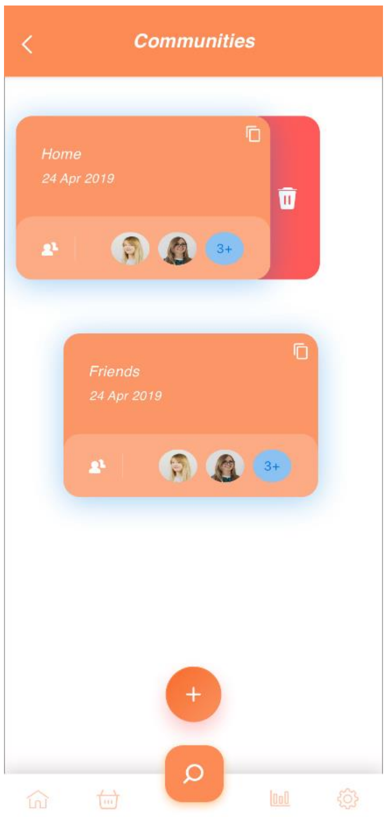
Screenshot 14: My List Screen with Add/Delete Options

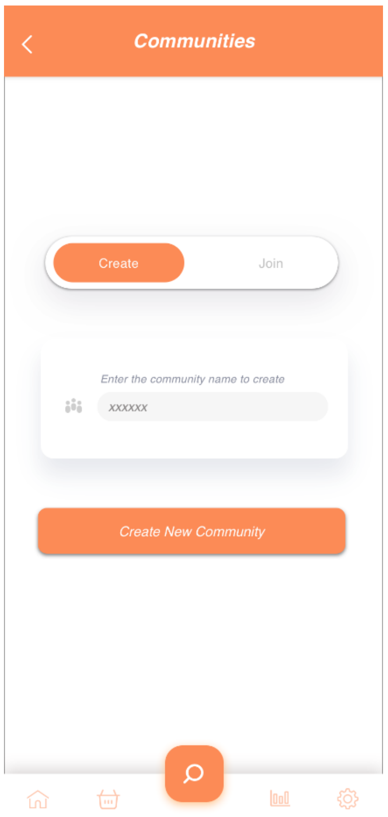
Screenshot 15: Edit List Screen

A User can

- Add,
- Delete,
- Edit, and
- Sort his/her list(s).

9.7. Community Screens



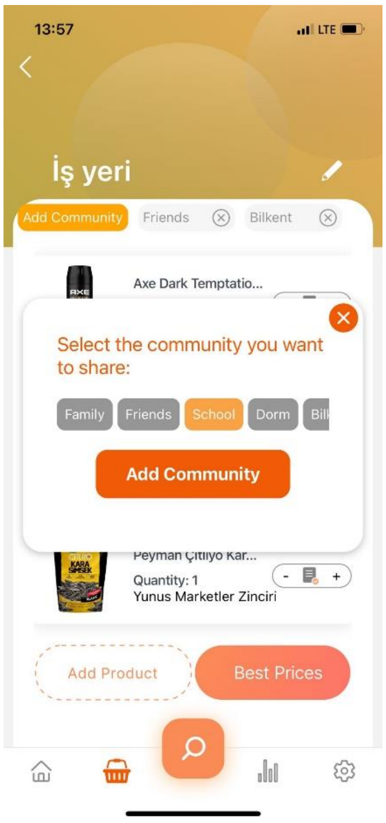


Screenshot 16-17: Community main and Create/Join Screens

On these screens, a user can

- Create a community
- Join a community
- Display the communities joined
- Leave from the communities.

9.8. A List Screen (Detail)

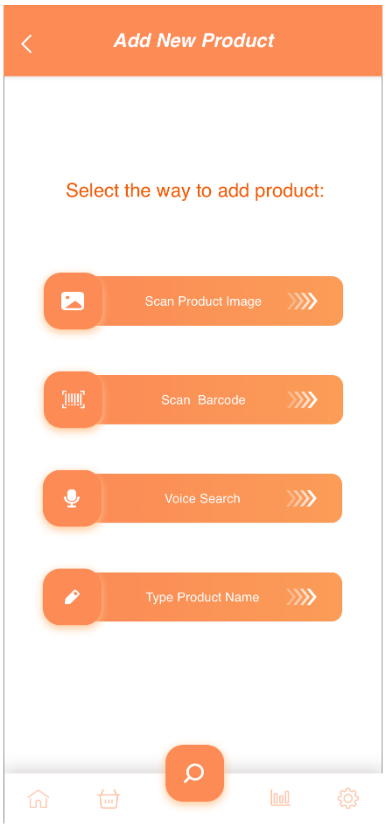


Screenshot 18-19: Detail Screens of a List

On this screen, you can

- Edit the name of the list
- Display the items that is added to the list
- Remove the items from the list
- Mark an item as bought
- Empty the list
- Add item to the list (you can add a product to multiple shopping communities) and
- Examine the offered best prices for a product

9.9. Add New Product Screen

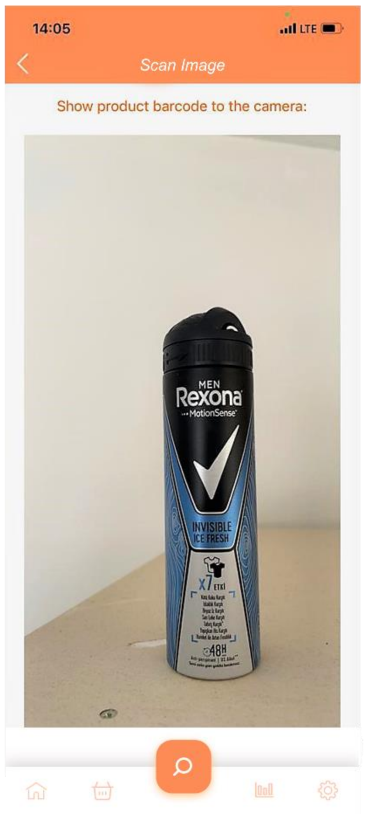


Screenshot 20: Add a New Product to a List Screen

While adding a new product, we offer several ways:

- Scanning product image
- Scan the barcode of a product
- Voice Search
- Manual product name typing

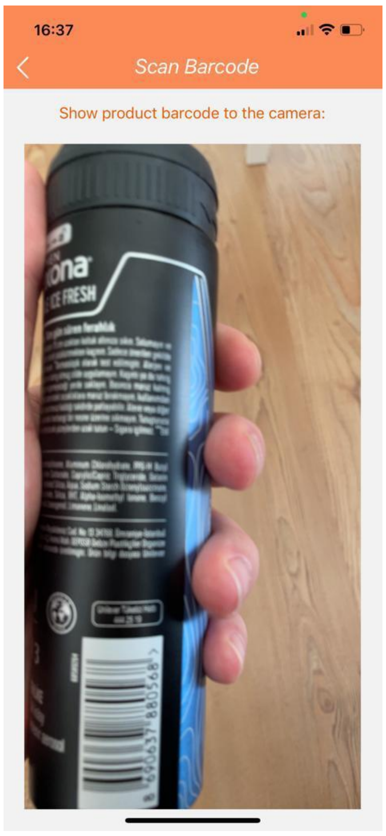
9.9.1. Scan Product Image Screen



Screenshot 21: Scanning Image Screen

A user can scan an item, if it is in the API's we used, it automatically is added to the list.

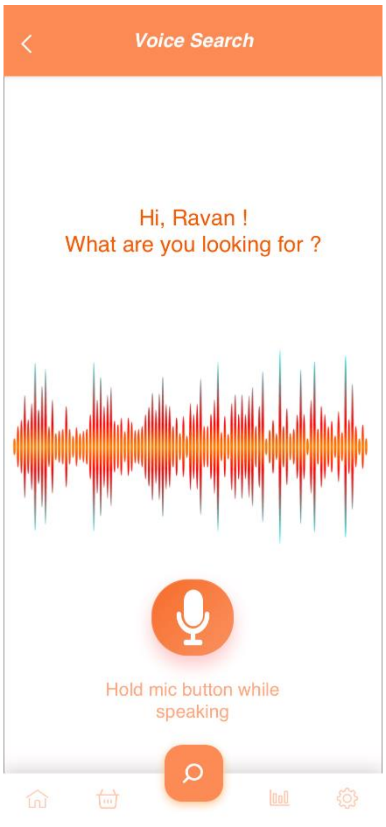
9.9.2. Scan Barcode Screen

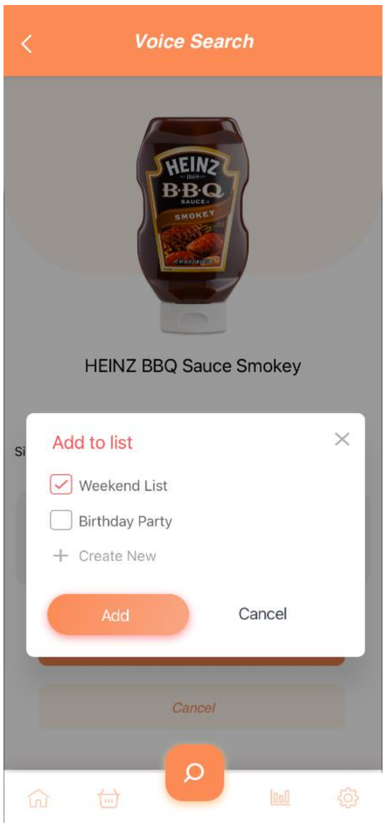


Screenshot 22: Scanning Barcode Screen

A user can scan the barcode of an item, if it is in the API's we used, it automatically is added to the list.

9.9.3. Voice Searching Screen

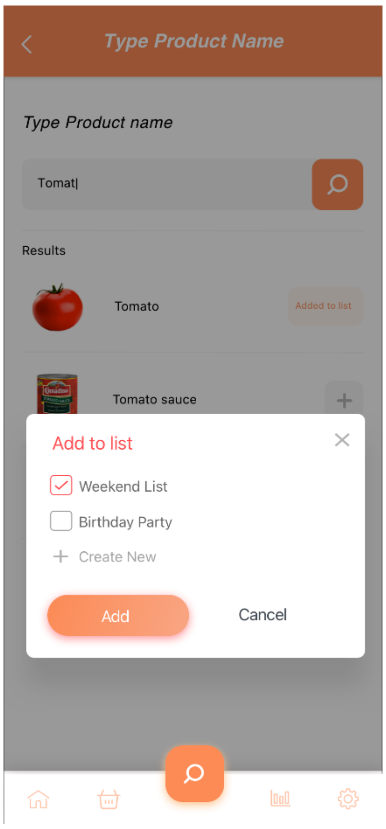




Screenshot 23-24: Voice Search Screens

Similarly, a user can add an item by voice searching, if it is in the API's we used, it automatically is added to the list.

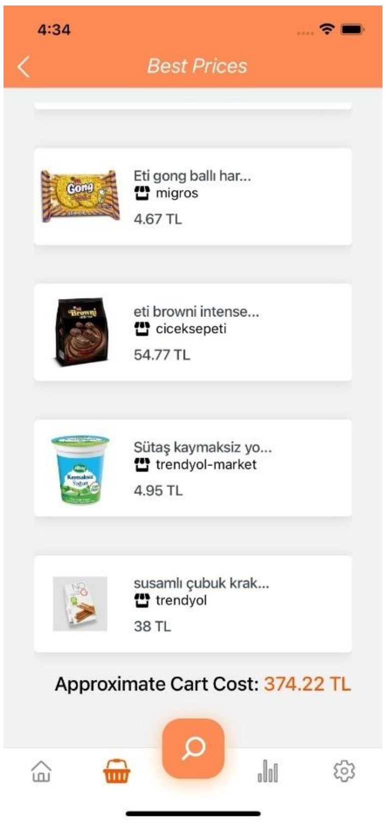
9.9.4. Manual Product Name Typing Screen



Screenshot 25: Manual Typing Screen

If previous methods do not work in some reason or the user prefers in that way, the classic typing method also works for a product adding.

9.10. Best Prices Screen


4:34

Best Prices

Eti gong ballı har...
migros
4.67 TL

eti browni intense...
ciceksepeti
54.77 TL

Sütaş kaymaksiz yo...
trendyol-market
4.95 TL

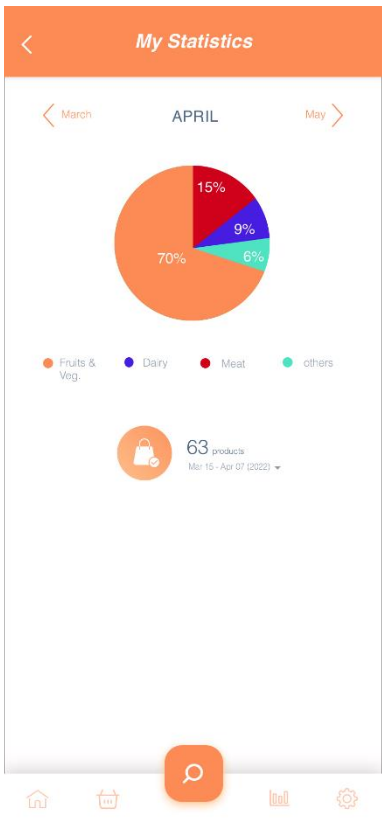
susamlı çubuk krak...
trendyol
38 TL

Approximate Cart Cost: 374.22 TL

Screenshot 26: Best Prices Screen

On this screen, users can see the offered prices from various markets.

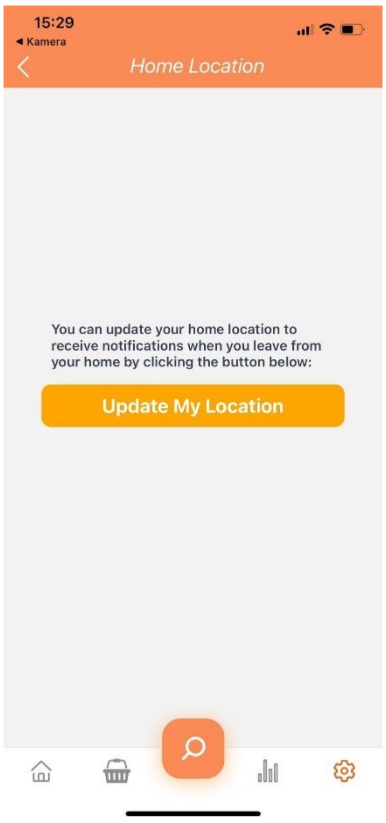
9.11. Statistics Screen



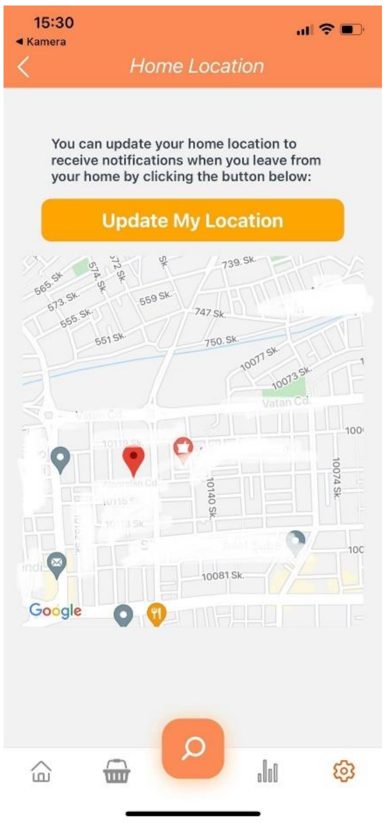
Screenshot 27: Statistics Screen

This screen helps the user to categorize the products purchased in the past and examine them in the form of a pie chart.

9.12. Map Screen



You can update your home location to receive notifications when you leave from your home by clicking the button below:

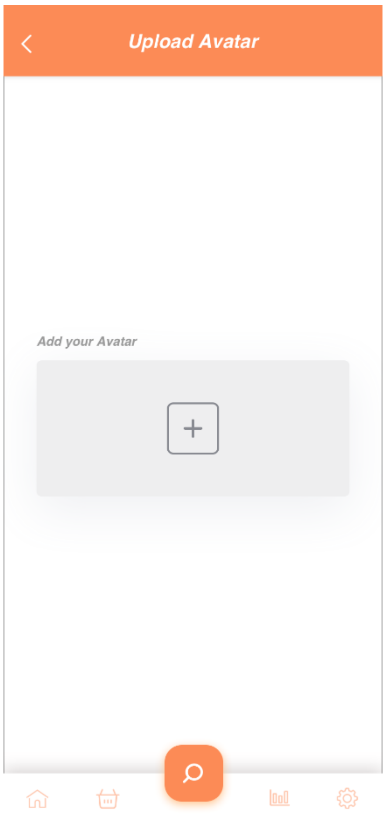


You can update your home location to receive notifications when you leave from your home by clicking the button below:

Screenshot 28-29: Map Screen

This screen is for address locating, users can select your current location or somewhere else from the map. User can reach the map from the settings.

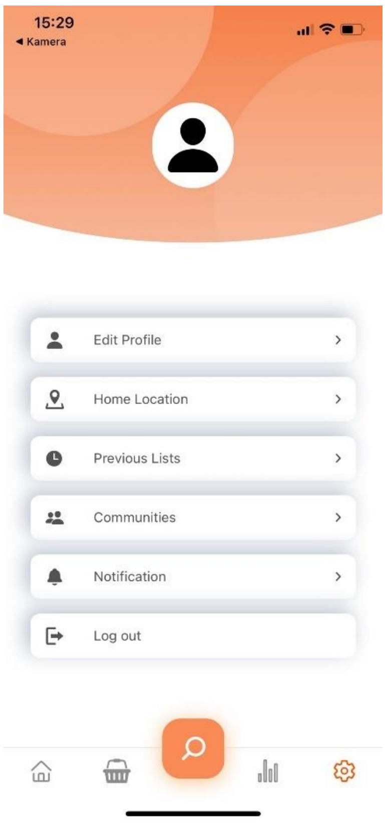
9.13. Avatar Screen



Screenshot 30: Avatar Screen

Users can upload their avatar to the application from this screen.

9.14. Settings Screen

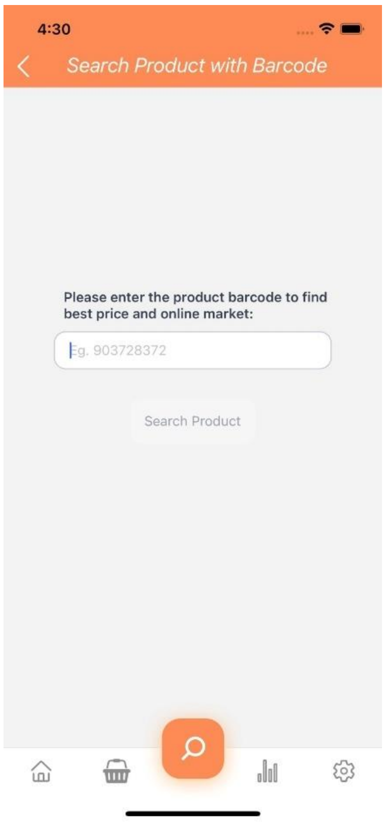


, Screenshot 31: Edit Profile Screen

Users can

- Edit their profile
- Edit their home location
- See their previous lists
- See their communities and edit them
- Log out from this screen.

9.15. Search with Barcode Screen

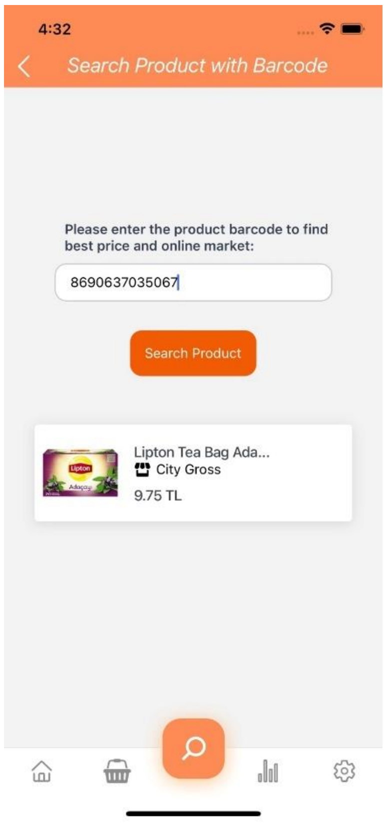

4:30

Search Product with Barcode

Please enter the product barcode to find best price and online market:

Eg. 903728372

Search Product


4:32

Search Product with Barcode

Please enter the product barcode to find best price and online market:

8690637035067

Search Product

Lipton Tea Bag Ada...
City Gross
9.75 TL

Screenshot 32-33: Search with Barcode Screen

By clicking the search button at the bottom navigator, user can search an item by typing its barcode and get product data that contains the best price, the store where it is sold, and other information about it. In this way, user can reach the cheapest product price.

10. Glossary

- **Machine Learning:** "Study of computer algorithms that can improve automatically through experience and by the use of data [9]"
- **Computer Vision:** "a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images [10]"
- API: Application Programming Interface.

Third Party Applications

- **TensorFlow:** E2E open-source Machine Learning (ML) framework, used to architect and deploy ML applications [17].
- **MySQL:** World's most popular open-source database [11].
- **DLib:** C++ library containing machine learning algorithms and tools [12].
- **Django:** High-level Python web framework used for both frontend and backend development [13].
- **React Native:** A popular JavaScript-based mobile app framework that allows you to build natively-rendered mobile apps for iOS and Android. The framework lets you create an application for various platforms by using the same codebase.
- **SQLite:** Claims to be the "most used database engine in the world" [14].
- **GitHub:** A code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.
- **OpenCV:** Open-source library used for computer vision applications.
- **Agile Development:** software development methodologies centered round the idea of iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.
- **External API services:** (hepsiburada.com/api and trendyol.com/api)
- **AWS:** Amazon Web Services. Claimed to be the world's most comprehensive and broadly adopted cloud platform [15].
- **S3 Bucket:** Simple storage service available in AWS.
- **Expo-barcode-scanner:** An advanced barcode-scanner written in JavaScript and TypeScript [16]

References

[1] "Grocery Guide", American Heart Association, 2020, [Online] Available: https://www.heart.org/-/media/aha/recipe/pdf-files/grocery-guide-english-shopping-budget.pdf?la=e. [Accessed: 06-May-2022].

[2] "Food consumption in UK", Rand Corporation, 2020, [Online] Available: https://www.rand.org/content/dam/rand/pubs/research_reports/RR4300/RR4379/RAND_RR4379.pdf. [Accessed: 06-May-2022].

[7] "React – A JavaScript library for building user interfaces", ReactJS. [Online]. Available: https://reactjs.org/. [Accessed: 06-May-2022].

[4] "CIMRI engineering," Cimri Engineering. [Online]. Available: https://engineering.cimri.com/. [Accessed: 06-May-2022].

[5] "What is SQL Injection (SQLi) and How to Prevent Attacks." Acunetix, https://www.acunetix.com/websitesecurity/sql-injection/. [Accessed: 06-May-2022]

[6] "Official Legal Text," General Data Protection Regulation (GDPR), 2019. [Online]. Available: https://gdpr-info.eu/. [Accessed: 06-May-2022].

[7] "node-sass", npm. [Online]. Available: https://www.npmjs.com/package/node-sass. [Accessed: 06-May-2022].

[8] "Moment.js", Moment.js | Home. [Online]. Available: https://momentjs.com [Accessed: 06-May-2022].

[9] "Machine learning," Wikipedia, 04-Nov-2021. [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning. [Accessed: 06-May-2022].

[10] "What is Computer Vision?" IBM. [Online]. Available: https://www.ibm.com/topics/computer-vision. [Accessed: 06-May-2022].

[11] "Tensorflow," TensorFlow. [Online]. Available: https://www.tensorflow.org/. [Accessed: 06-May-2022].

[12] "MySQL documentation," MySQL. [Online]. Available: https://dev.mysql.com/doc/. [Accessed: 06-May-2022].

[13] Django. [Online]. Available: https://www.djangoproject.com/. [Accessed: 06-May-2022].

[14] Sqlite. [Online]. Available: https://www.sqlite.org/index.html. [Accessed: 06-May-2022].

[15] "What is AWS". [Online]. Available: https://aws.amazon.com/what-is-aws/. [Accessed: 06-May-2022].

[16] "BarCodeScanner." Expo Documentation, https://docs.expo.dev/versions/latest/sdk/bar-code-scanner/. [Accessed: 06-May-2022].