Bilkent University

Department of Computer Engineering

Senior Design Project

Farket

Low Level Design Report

Deniz Alkışlar | H. Buğra Aydın | M. Erim Erdal | M. Enes Keleş | Hakan Türkmenoğlu

Supervisor: Eray Tüzün

Jury Members: Mustafa Özdal and Özcan Öztürk

Innovation Expert: Barış Misman

Progress/Final Report

February 18, 2019

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.

1. Introduction

With the economic recession problem Turkey is facing these months, it has become a necessity for medium and low income families to shop in a careful and calculated manner. When dollar had wild swings up and down last summer, the grocery prices in markets have started to differentiate from each other drastically. We are in a time that one can find the same grocery items in a market with double the price in another market. We are trying to solve this problem by creating an application which will track the prices in several stores simultaneously and save the customers from the trouble of searching for the cheapest prices in every store.

There are several applications trying to solve this problem currently. However, when they are analysed it can be seen that they lack some important functionalities. In

example, www.cimri.com currently being competitors in terms of functionality, is able to show where the cheapest product is from a wide range of market options but lacks the option to show the user how to gather these cheapest products from each market in the shortest travel path possible which is a functionality we offer to our customers. It also lacks the ability to create a shopping list in the website which our services offer and order these products, it only navigates users to those online stores or local stores, which is also a functionality we consider adding into later stages of the program. Another functionality of our program will be a receipt scanning system where customers will get different rewards for scanning their receipts using a simple camera.

We will use Python 3[1] for backend implementation and crawler for the websites for gathering the price data from every store for each product, SQLite[2] for the database system. Google Maps API[3] will be used for visualizing the optimal shopping path obtained by the shopping list user has created. Frontend will be written in React Native[4] since it enables us to develop apps using only JavaScript[5]. We will also be using Google Vision API[6] for implementing the receipt scanning system.

In this report, our aim is to provide an overview of low-level design of our system. In the first part, object design trade-offs of our system and engineering standards are described. Then, interface documentation guidelines are given as an outline for class descriptions. After that, packages with their functionalities in our system are described with class diagrams in detail. It is followed by interfaces of classes in all

packages. This way the necessary clarifications of functionalities in the software is provided.

1.1 Object Design Tradeoffs

1.1.1 Efficiency vs Portability

Our aim is to target people from diverse backgrounds. Because of that, the user's choice of platform shouldn't limit our reach. This is why we decided to rely heavily upon cross-platform technologies like React and React Native. By developing our client application on React Native, we support Android and iOS devices at the same time with minimal platform specific code. Later on if we decide to do so, we can easily start supporting desktop machines, since React works on web browsers. Since React Native is an abstraction layer above native technologies, it provides this abstraction at a cost, both at the memory level and the performance level. That means React Native incurs an efficiency impact compared to a native application [9]. We believe that blindly implementing our application in native languages is not a silver bullet, and therefore a premature optimization. Conversely, we should implement the application in an efficient way to minimize the impact of our decision by leveraging our productivity gains that comes from cross-platform development.

1.1.2 Accuracy vs Performance

Crowdsourcing is part of our application. Users may contribute to our product database by scanning their receipts. The quality of the scan is important in terms of readability. A receipt's scanned photograph must readable enough for the server-side OCR. However, before sending the scan to our servers, we need a

pre-verification check on the client-side without affecting the performance of the app to select a good candidate photo. We choose to tolerate some of the bad scans rather than irritating users to get 100% accuracy.

1.1.3 Functionality vs Usability

Farkett is an application which is used by variety of age and social groups whose common aim is to lower the price of their grocery shopping. So the core function of Farkett is to show price alternatives for the products. Rather than adding too much functionality on top of our core function, we choose to focus on usability of the software. Since our target customers are diverse, the ease of use of the app plays an important role. We decided to lean towards usability rather than adding too many features since people will mostly use the core function.

1.2 Interface documentation guidelines

In this document, the form of class names are as in ‘ClassName’ in camel case with upper case beginning and class names will be in singular form such as ‘Viewer’. The form of variables and functions are as in ‘variableName’ in camel case and ‘functionName()’ starting with lowercase letters and parentheses in the end for functions. Each class is given with its description and function. After the description and function part of each class, properties and methods of classes will be given.

Properties are listed with their names and types. Methods are listed with their return types and parameters with their types.

Overall summary of interface documentation is below:

1. Class Name
2. Class Description and Function
3. Properties (Name and Type)
4. Functions (Name, Parameters, Return Type)

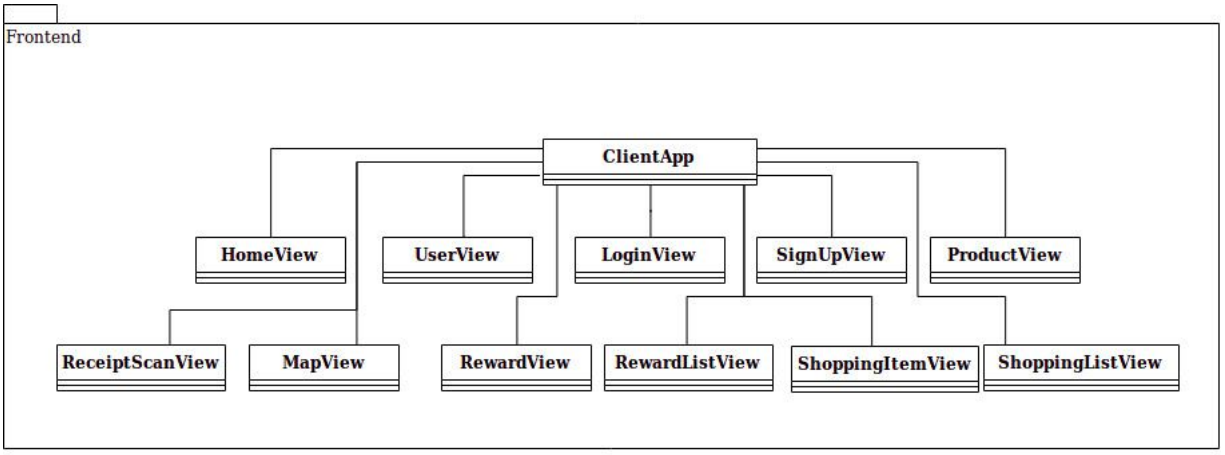
1.3 Engineering standards (e.g., UML and IEEE)

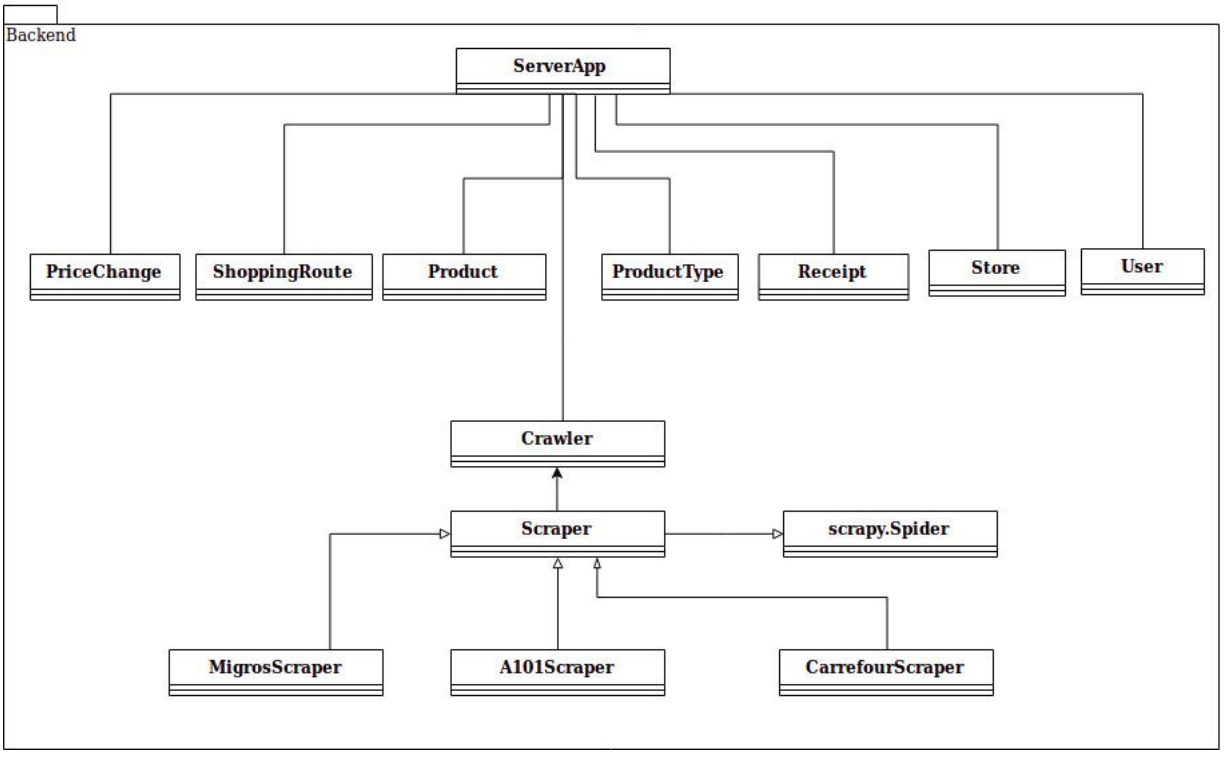
Similar with previous reports, UML[6] design principles are used in the description of class interfaces, diagrams, scenarios and use cases, subsystem compositions, and hardware-software components depiction. UML is a powerful universal modelling language which enables developers all around the globe to efficiently understand the descriptions and functionalities described by UML. “You can model just about any type of application, running on any type and combination of hardware, operating system, programming language, and network, in UML. Its flexibility lets you model distributed applications that use just about any middleware on the market.”[7]. Therefore in our reports we have preferred using UML. IEEE produces standards that are widely used and preferred by engineers, therefore this report follows IEEE citation guidelines[8].

1.4 Definitions, acronyms, and abbreviations

API: Application Programming Interface
UI: User Interface
Client: Part of the application that user interacts with
Server: Part of the application that controls data, logic and computation
UML: Unified Modelling Language
IEEE: Institute of Electrical and Electronics Engineers
OCR: Optical Character Recognition

3. Class Interfaces

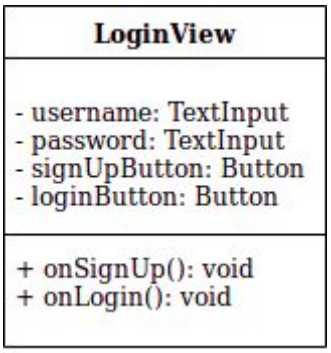




3.1 Client Side

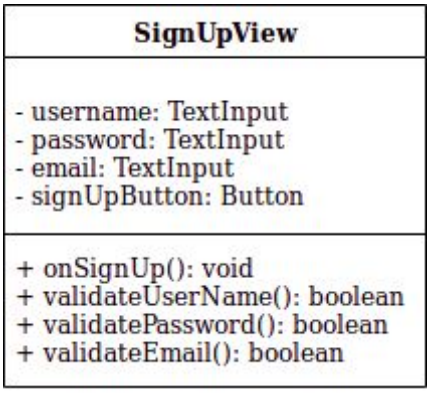
3.1.1 View

3.1.1.1 LoginView



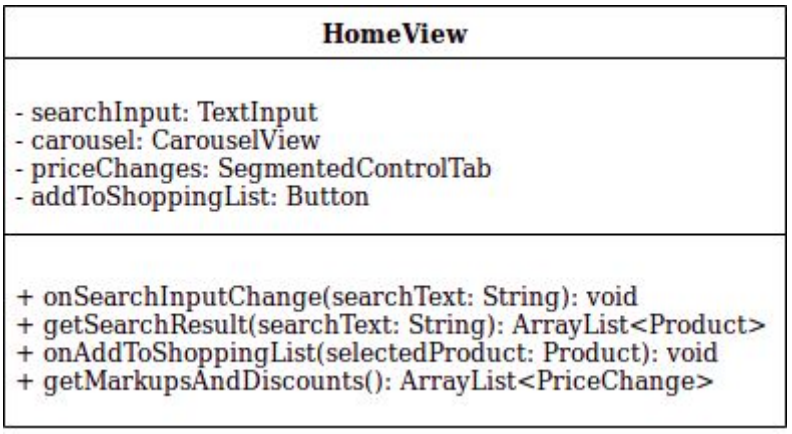
This class is responsible for the user login. onSignUp() function is called when the signUpButton is clicked and onLogin() function is called when the onLoginButton is clicked.

3.1.1.2 SignUpView



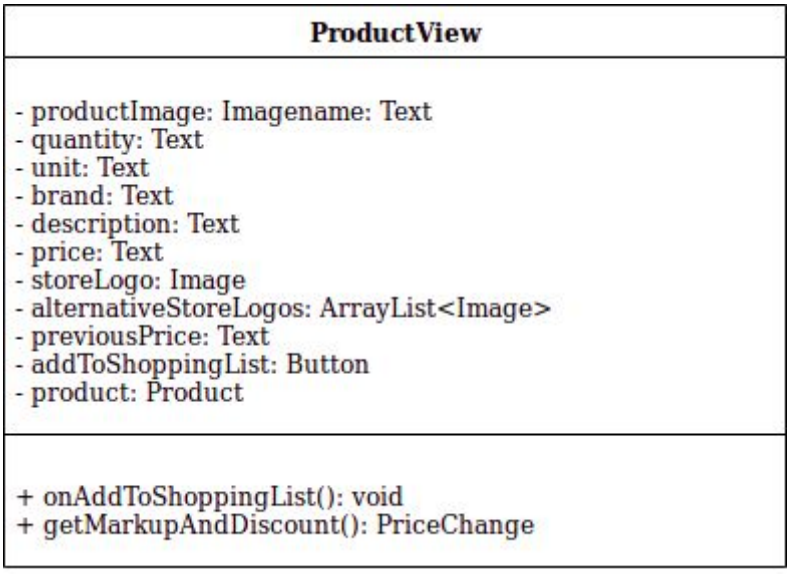
This is the class that helps user sign up to the system via username, password and email. Each input is validated with custom logic.

3.1.1.3 HomeView



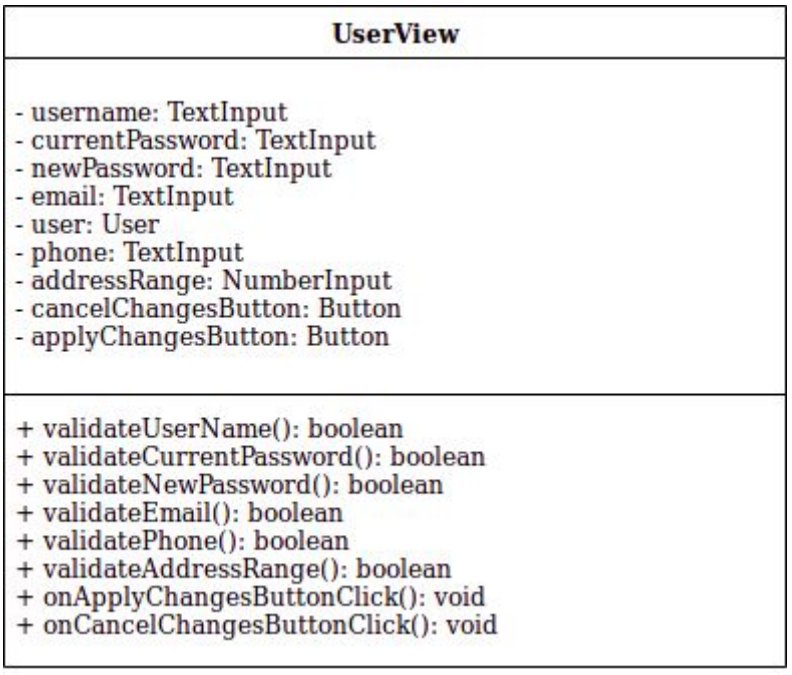
This is the main home view of the system. It allows user to search an item to add to the shopping list. After user types the name of the product into the search box, the class retrieves the items relevant to this search and displays it in Carousel View. After the user finds the item they want to add by swiping left and right, they can click add to shopping list button. Additionally, users can view the most interesting price changes.

3.1.1.4 ProductView



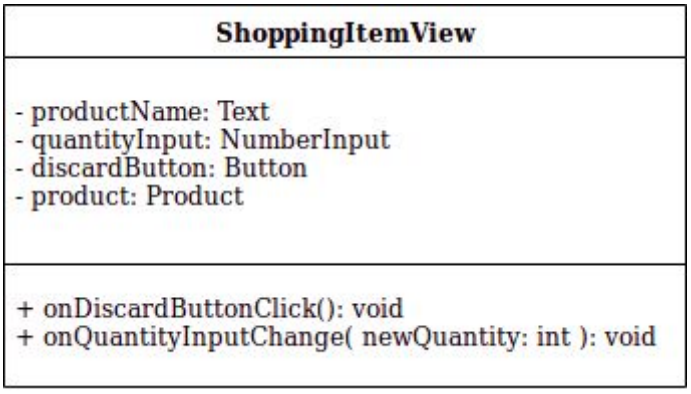
This class displays detailed information about a product. It displays a preview image, quantity, brand, description (if any), price and the store selling it. If there are any, it will also display alternative stores for this product, clicking on alternative logos enables user to compare prices in different stores. Also, it will display any discount or markup information available for this product.

3.1.1.5 UserView



In this view, user can see profile information and if they decide to do so, change it.

3.1.1.6 ShoppingItemView

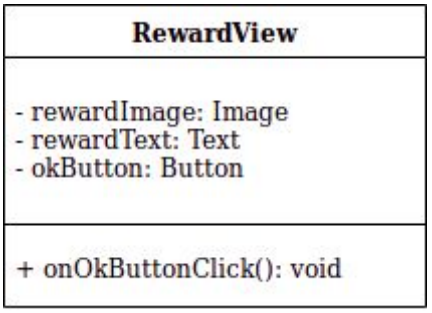


This class is responsible from displaying individual cells of shopping list. A small NumberInput enables user to buy multiples of the given product. Users can remove the item altogether.

3.1.1.7 ShoppingListView

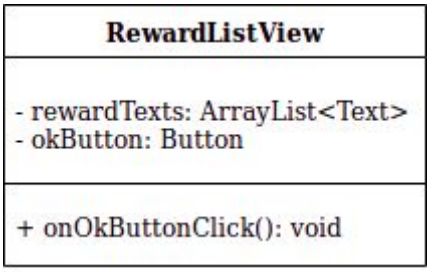
This view is responsible from displaying a shopping list for the user. If the user is satisfied with the list, they can click on start shopping button to initiate the navigation use case.

3.1.1.8 RewardView



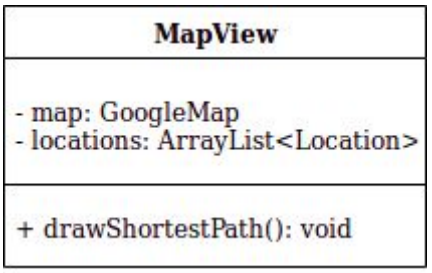
After user scans a receipt, they will rewarded however server decides to. User will see a popup displaying what type of coupon they were rewarded with an image and explanation.

3.1.1.9 RewardListView



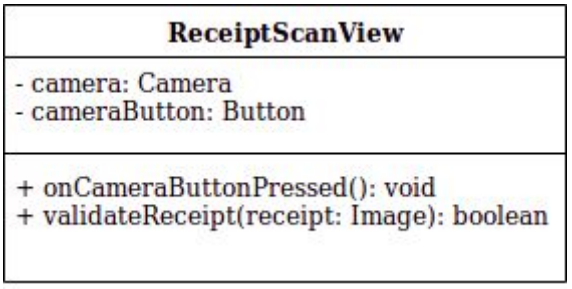
This class displays the history of rewards of the user for their receipts.

3.1.1.10 MapView



This class is responsible for showing the optimum shopping route on a map.

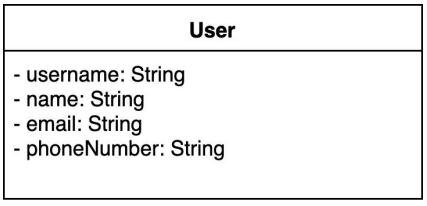
3.1.1.11 ReceiptScanView



This class is responsible for scanning and validating the receipt.

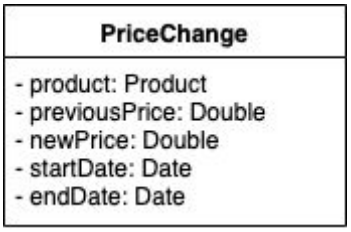
3.1.2 Model

3.1.2.1 User



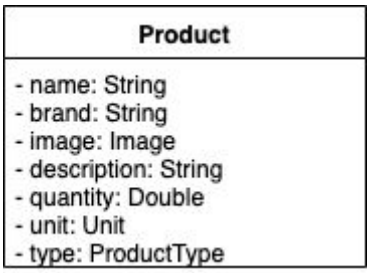
User class holds the corresponding data related to a user such as username, name, email and phone number.

3.1.2.2 PriceChange



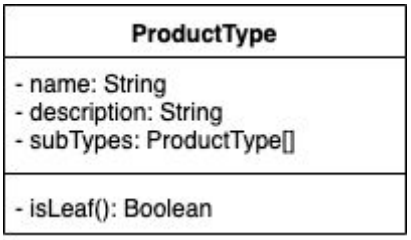
PriceChange class holds the related information about a change in an item's price. Product, previous price, new price, start and end dates are the information it holds.

3.1.2.3 Product



Product class holds the related information about a product. Name, brand, image, description, quantity, unit and type are the information it holds.

3.1.2.4 ProductType

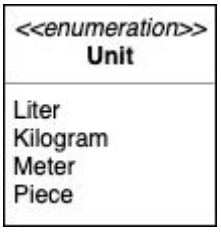


Product type class differentiates the product types. It holds related name, description and sub-type attributes and has a method that returns if the current type is a leaf.

3.1.2.5 ShoppingList

Shopping list class represents the products user added to the list.

3.1.2.6 Unit



Unit is an enumeration that demonstrates the type of the quantity to represent the amount of a product.

3.2 Server Side

In the server side design of Farkett we aimed for simplicity and high configurability for quick prototyping purposes. Therefore we adopted a “repository pattern” and we used Python 3.

In the current plan there are five packages in the server side:

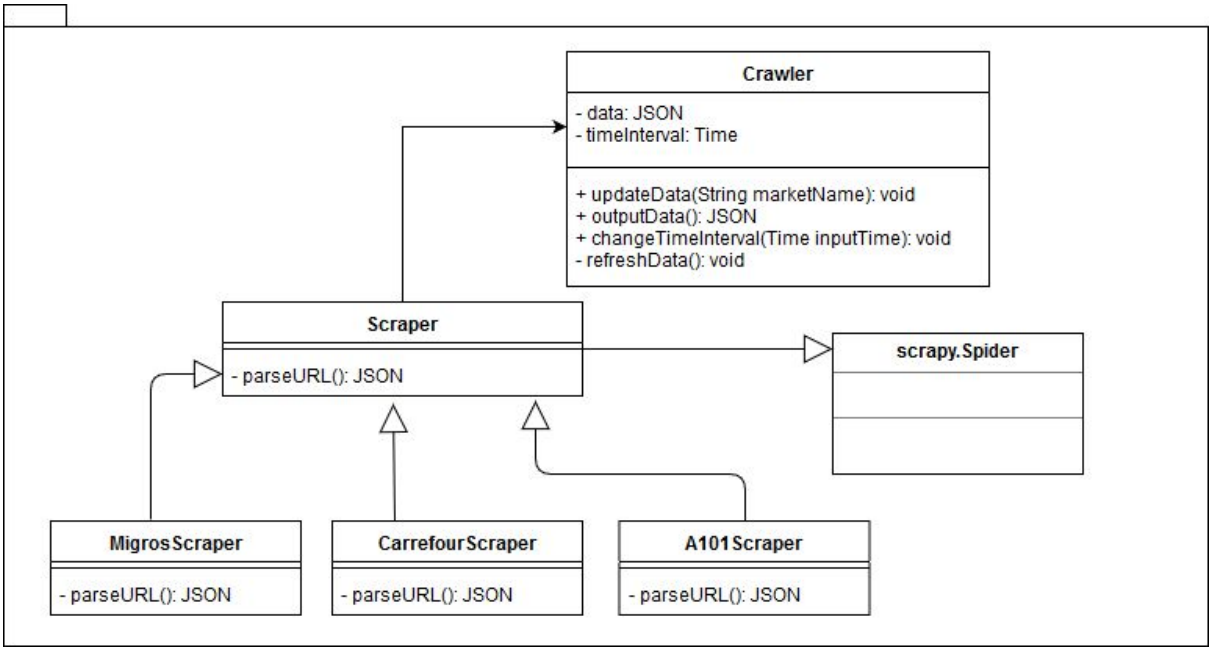
- crawler
- repository
- db-utils
- shopping-routes
- receipts

Each of these packages will be explained in detail in the following sections. All utilities from these packages are incorporated and used in a module named **app.py** which contains the REST API endpoints that accepts HTTP requests from the client side.

3.2.1 Crawler package

This package will be responsible for parsing the corresponding online markets’ websites. It will work in a modular manner and will be a black box. The package responsible for the stability of the data will retrieve and update the data using this package.
Crawler class will hold the data in JSON format. updateData method will be used from outside and will create corresponding crawlers for all markets and update this data with no parameters. updateMethod can be called with market name parameter to update a single market’s data. refreshData method will be called in the class itself to synchronize the data in a given time interval. changeTimeInterval method can be used to update this time interval. outputData will generate a JSON object to be used by other packages responsible for the database.

In each method that needs to crawl an URL, a Scraper object will be generated, extending the Spider class of the Python library Scrapy. These objects will override the parseURL method of the Scraper class and work in different ways.



3.2.1.1 Crawler

This class is responsible for the outside communication of the package. It also manages the URL crawling operations by creating Scraper objects

3.2.1.2 Scraper

This class extends the Spider class of the Python library Scrapy. It has a method that parses the given URL and handles the corresponding HTML operations. It allows modularity and maintainability for future market additions by including the parseURL, since it can be overridden easily.

3.2.1.3 MigrosScraper

Overrides the parseURL() method used to crawl Migros online shopping site,

3.2.1.4 CarrefourScraper

Overrides the parseURL() method used to crawl Carrefour online shopping site

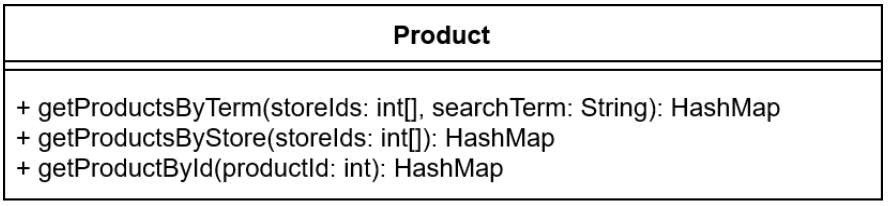
3.2.1.5 A101

Overrides the parseURL() method used to crawl A101 online shopping site

3.2.2 Repository package

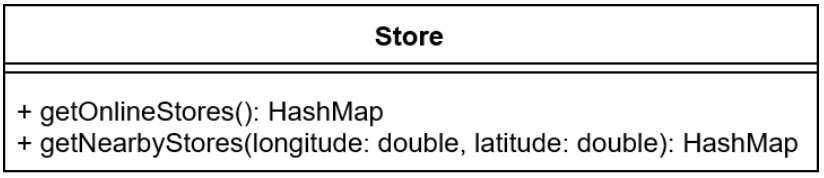
This package abstracts data manipulation functionalities so that other parts of the application can work with a simple interface. Each class contains static methods linked to database entities and have no attributes. For example “Product” class contains SQL queries that is related to “Product” table in the database. Almost all methods wraps up information that is fetched from the database into a HashMap which may later be converted to JSON which is used as an intermediate data format between the server and the client.

3.2.2.1 Product



In addition to getting products by id and store, this class also facilitates the need for searching a product with a fuzzy search term.

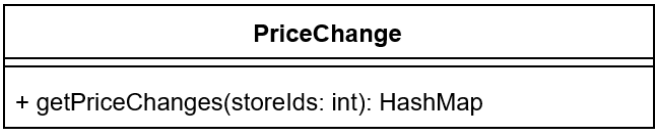
3.2.2.2 Store



This class is responsible for returning available stores and filtering them by range.

In order to find nearby stores Google’s Places API is used since it would be inefficient to store every local market in our database.

3.2.2.3 PriceChange



This class is responsible for returning markups and discounts for specified stores.

3.2.2.4 User

User
+ validateUserInfo(username: String, password: String): boolean + addUserInfo(userInfo: HashMap): boolean + updateUserInfo(userInfo: HashMap): boolean + getUserInfo(username: String): HashMap

This class is responsible for login, signup, and user profile functionalities. Note that authorization tokens is not shown in this report for clarity purposes but they will be used.

3.2.2.5 ProductType

ProductType
+ getCategoryTree(): HashMap

This class returns category tree.

3.2.3 shopping-routes package

This package is dedicated to creating efficient shopping routes.

3.2.3.1 ShoppingRoute

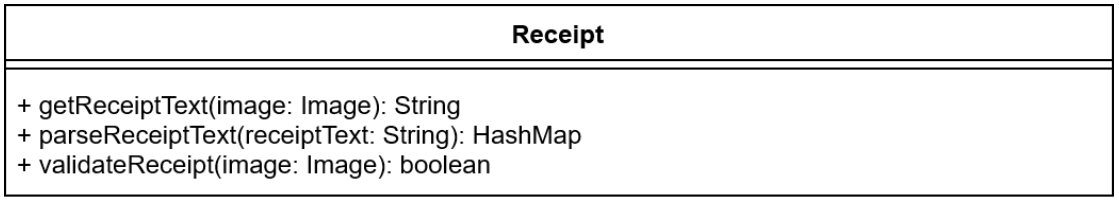
ShoppingRoute
+ getCheapestRoute(shoppingList: HashMap, longitude: double, latitude: double): HashMap + getShortestRoute(shoppingList: HashMap, longitude: double, latitude: double): HashMap + getOptimalRoute(shoppingList: HashMap, longitude: double, latitude: double): HashMap

This class has a single method that returns three shopping routes for a given shopping list and location: the cheapest, the shortest, and the most efficient. Google's Directions API will be utilized.

3.2.4 receipts package

This package is dedicated to validating and parsing receipt data coming from users and updating the database accordingly. This way the application will have data regarding shopping habits of the user. Personalization may be added later.

3.2.4.1 Receipt



This class will use Google's Cloud Vision API for optical character recognition on receipts.

Glossary

References

[1] https://www.python.org/download/releases/3.0/
[2] https://www.sqlite.org/index.html
[3] https://cloud.google.com/maps-platform/?hl=tr
[4] https://facebook.github.io/react-native/
[5] https://www.javascript.com/
[6] https://cloud.google.com/vision/?hl=tr
[7] http://www.uml.org/
[8] https://libguides.murdoch.edu.au/IEEE/text
[9] https://aaltodoc.aalto.fi/bitstream/handle/123456789/32475/master_Eskola_Rasmus_2018.pdf?sequence=1&isAllowed=y