CS491 Senior Design
Project Analysis Report

Appeatite

A Project By

Unas Sikandar Butt

Faaiz Ul Haque

Alemdar Salmoor

Supervisor

H. Altay Güvenir

Jury Members

Ozcan Ozturk

Shervin R. Arashloo

Innovative Expert

Emin Okutan

Viveka Technology Incubator

November 12th 2018

1. Introduction

Appeatite will contain a digital version of restaurants' menus displayed using a mobile application. It will be an interactive process in which both users and restaurant employees will participate. When a customer enters a restaurant that is using our service, they will have to scan a QR code placed on their dining table, using the application's camera. The users will then be prompted to a screen which will display the generic details of that restaurants menu. The menu will consist of information such as meal names, ingredients, calorie counts, images, proportions and prices. The interface of the application will be simple and abstract, in the sense that information will be shown on user's request. Then, upon selecting the desired items, the restaurant will be notified that an order has been placed and they will be able to detect from which table, as all QR codes will specify to a different table location.

The restaurant will require one employee to manage this using one simple desktop computer. We will create a web application for the restaurant to manage their orders. Furthermore, they will be given an interface allowing them to add relevant information about their food items, and create their initial menu. We have considered the possibility that some old fashioned users may wish to simply speak to someone to order, and therefore we will implement an option to call a worker to come and assist them with their order, answer any questions or take the order manually.

Since many customers have trouble deciding on their order, we will have a simple review system where previous people who have ate at the restaurant can leave a rating and small review on particular dishes.

We will also use machine learning techniques or simple algorithms to determine an expected wait time for each dish. Thus, customers can order food accordingly to the amount of time they have and can have some knowledge on how long they will be expected to wait.

2. Current system (if any)

After intensive research we have not found a single application on the market that offers the features of Appeatite.

3. Proposed system

3.1 Overview

This report will state and explain the functional, non-functional requirements in detail regarding our application for both the user's side and restaurant service's side. The pseudo requirements will be stated briefly, and the constraints mentioned in our project specifications report will be stated and further extended on from feedback we have received from our supervisor and from research we have conducted on our own. Then the system models will be mentioned and identified in detail. The scenario models, list and explain all the main types of use-cases and their activity flow for the users and restaurant staff. The class diagram shows the structure of our project and lists the different entity's and their corresponding attributes used in our application. The relation between each of these classes are also shown in our object diagram. Our UML diagram will be a visual illustration of each possible outcome named in the scenario models. Finally the activity diagram shows a simple, human readable flow of how the application will work in both user and restaurant staff perspectives. The sequence diagram, will show how the internal components of our application will function using two main examples (User ordering food, restaurant adding items to their menu). Our user-interface is simple and easy to use as shown by our mockups created for the user's point of view on the android phone application.

3.2 Functional Requirements

- Users should be able to read the QR Code within the restaurant using their mobile phone's camera
- Users should be able to view the restaurants' menu
- Users should be able to order menu item they wish to from the application
- Users should be able to call the first available restaurant staff from the application for consultation purposes
- Users should be able to see additional food information and description such as calories, meal ingredients and pictures if the restaurant representatives supply such information
- Users should be able to see reviews left by other people for the menu items.
- Users should be able to see expected delivery time of the food
- Users should be able to request cancelling their order
- Users should be able to remove ingredients from the meal if the restaurant permits such an operation

- Users should be able to view their order history
- Restaurants should be able to print the generated QR code using the client application developed for the restaurants.
- Restaurants should be able to supply the menu information using the client application.
- Restaurants should be able to supply additional food information such as calories, meal ingredients and pictures
- Restaurants should be able to mark the currently unavailable items from the menu.
- Restaurants should be able to change the prices for menu items provided that there are no pending orders for those menu items.

3.3 Nonfunctional Requirements

- Google Cloud Firestore will be used as a database of Appeatite.
- Google Cloud Functions will be used to deploy the cloud functions of Appetite
- Administration environment for cloud functions and database manipulation code will be written in Typescript using Node.js environment.
- The application will be implemented in english first, later on more language options will be made available
- To maximize user domain, after the android implementation and keeping time constraints in check an IOS version may also be developed
- Frontend application for restaurant customers will be written in Java an Kotlin programming languages
- Frontend application for restaurant staff will be written in Javascript
- Average execution of cloud functions should be in order of milliseconds

3.4 Pseudo requirements

The application will employ *Material Design* design guidelines to conform with the current market of social applications. The application will support Google's Instant App.

3.4.1 Economic Constraint

Economic constraints are one of the primary factors affecting the design of software products. To maintain the economic feasibility we have chosen to build our project on an Android Platform, as this official Integrated Development Environment(IDE) is freely accessible and does not require us to purchase any new hardware since it is able to run on both Linux and Windows Operating systems. In addition, we can run, debug and test our application using our personal android cellphones and will not need to purchase testing devices. With regards to the database we will use Cloud Firestore from Google. Since this is a Database as a Service (DBaaS) we will eliminate costs related to maintaining the database and operational costs and pay only for the

actual data reads and writes to the database. Most importantly, Cloud Firestore offers free quota that equates to 1 GiB of stored data, 50 000 document reads per day, 20 000 document writes per day, 20 000 document deletes per day and Network egress of 10 GiB/Month [1] which is more than enough for the development purposes. Node.js javascript environment, Typescript programming language and VS Code Integrated Development Environment which we are going to use for the database administration and cloud functions deployment are all open-source and free[2] [3] [4].

3.4.2 Environmental Constraint

Our application's environment will consist of eateries. The design of the product makes it very easy to integrate to the mentioned surroundings. With regards to the constraints, placement, and aesthetics of the QR code badges must be considered. A restaurant theme is important to the owner and customers thus the QR codes must be placed in such a way that they do not disrupt the restaurant's theme and are easily accessible for the customers. Another consideration must be made regarding the computer terminal which will be used by the restaurant to manage Appeatite, considering current restaurant designs and based on our personal experience most restaurants today already possess at least one computer at site.

3.4.3 Social Constraint

Integrating Appeatite system into restaurants can cause users to miss out on the one-to-one conversation with the waiters. Some users may desire this conventional one-to-one experience or simply may need some assistance. Taking into account these constraints, the system will include a call button on the user application so that if the user wants assistance they can immediately call one of the restaurant's staff member.

3.4.4 Political Constraint

The fixed and variable costs for the eating places will be decreased since our application aims to decrease the number of waiters in the given restaurant. Additionally, served food tracking system which will be provided for the restaurants may enable to let them cut the existing costs

for this kind of data manipulation. However, the waiter staff's job that will be replaced by this application will lose their hypothetical income from this job.

3.4.5 Ethical Constraint

Ethical constraints can be defined by IEEE Code of Ethics. We as developers of Appeatite, "in recognition of the importance of our technologies in affecting the quality of life throughout the world, and in accepting a personal obligation to our profession, its members, and the communities we serve, do hereby commit ourselves to the highest ethical and professional conduct and agree" not to violate the IEEE Code of Ethics[5]. In specific an ethical constraint is that our application will be unbiased in the sense that we will not put preference to certain cuisines from different cultures, but rather be accepting to all types of people from different ethnicities. Additionally, the information provided (in particular calorie count, proportions) for each dish should be ensured to be as accurate as possible. In order to receive feedback regarding this we will implement a separate section in the application where users can write their opinions about the overall experience of Appeatite's service.

3.4.6 Health Constraint

The product the we are aiming to build does not pose any explicit self hazards, since it requires the hardware that the customers already possess. To ensure the user information safety, personalized data will not be shared with any other third party. The food quality and safety of eating is by no means Appeatite's responsibility and as a disclaimer we state we are not responsible for any health problems that customers may face due to eating at a restaurant while using our application.

3.4.7 Manufacturability Constraint

If we can ensure that the product we are building can be manufactured with relative ease at minimum cost and and maximum reliability we can ensure good manufacturability. In our case after we build Appeatite we can sell the product to many eating places where additional cost will only be additional data reads and writes to the database. That is, the marginal benefit from every new restaurant will only increase while the fixed costs will stay the same. If a new restaurant is to remove their existing system, Appeatite must provide enough reliability

assurance that they can depend upon to fill out all their existing needs, without the need for their old systems.

3.4.8 Sustainability Constraint

Sustainability refers to the ability of an engineering design to perform under normal conditions for a given length of time. The use of Cloud Firestore and Cloud Functions enables us to focus on the actual logic of the database and functions while maintenance is delegated to Google company that has extremely good reputation in cloud services. The application must be maintained by our team constantly in order to keep it up to date with competing services. Without implementing new features, and responding to user critique Appeatite may lose its popularity in the future and be used less if it remains outdated.

3.4.9 Other Constraints

1. The readability for restaurant's interface and user should be easily readable and have a clean interface.
2. Once the system is tested with one restaurant it should be easily portable to add a new restaurant to become part of Appeatite's service
3. The review system may face some ethical issues. Therefore, there should be restrictions to what users can say in their reviews (profanity detector). Additionally, one user should not be able to leave two reviews at the same time in different eating places. The review system should be able to differentiate spam, and legitimate reviews.
4. The restaurant should be provided with an easy and convenient way to add their existing menu information to Appeatite's web-service. It should be hassle-free, easy and fast.
5. Since the user-application will initially be only for Android, it should be ported to iOS using third-party softwares.

3.5 System models

3.5.1 Scenarios

Scenario 1:
Use Case: Sign-Up
Actors: Jake
Entry Conditions:
Jake opens Appeatite application and is on the main registration page
Exit Conditions:
Jake is on Appeatite's home page

Main Flow Of Events:

1. Jake opens Appeatite application
2. Jake clicks the ‘Sign-Up’ button on the main page
3. Appeatite prompts Jake to a screen where he is required to input information related to setting up his account
4. Jake enters his username as ‘Jake’
5. Jake enters some keyword as his password (minimum 6 char)
6. Jake re-enters his password chosen in step 5 for validation purposes
7. Appeatite verifies whether the information given is complete, correct and is accepted by the certain restrictions
8. Appeatite creates an account under the username ‘Jake’ and registers him as an existing user
9. Jake is prompted to Appeatite’s home page

Alternate Flow #1:

9. Appeatite detects the information as incomplete and/or doesn’t satisfy given conditions
10. Appeatite prompts a relative error message and redirects Jake to step 3

Alternate Flow #2:

9. Appeatite detects the username provided already exists as a registered account
10. Appeatite prompts a relative error message and redirects Jake to step 3

<u>Scenario 2:</u>

Use Case: Sign-In

Actors: Jake

Entry Conditions:

Jake opens Appeatite and is on the main-page

Jake must have a registered account in the application

Exit Conditions:

Jake is on Appeatite’s home page

Main Flow Of Events:

1. Jake opens the Appeatite application and is prompted to the main page
2. Jake selects the Sign-In option from the main-page
3. Appeatite prompts Jake to a screen where he is required to input information related to his existing account credentials
4. Jake enters his username as ‘Jake’
5. Jake enters his corresponding password
6. Appeatite verifies whether the information given is complete, correct and the account exists in the database
7. Appeatite redirects Jake to his homepage

Alternate Flow #1:

6. Appeatite detects an error, the username and password combination did not match
7. Jake is redirected to step 3, with a relevant error message

Alternate Flow #2:

6. Appeatite detects an error, the username does not exist as a registered account
7. Jake is redirected to step 3, with a relevant error message

Scenario 3:

Use Case: ViewMenu

Actors: Jake

Entry Conditions:

Jake is currently on the home-page of Appeatite while registered into his account
Jake is sitting at a restaurant using the Appeatite service, and is ready to scan a QR code

Exit Conditions:

Jake will be prompted the menu details of the particular restaurant he is in

Main Flow Of Events:

1. Jake selects the 'Read QR' option from his main page
2. Appeatite opens the mobile's camera application
3. Jake will scan the QR code by aligning it with the camera, ensuring it is easily viewable
4. Jake will be prompted to the restaurant's abstract menu displaying all kinds of information
5. Jake can click on different information of each item upon his request such as: 'calorie count, proportions, prices, ingredients, pictures'

Alternate Flow:

Scenario 4:

Use Case: OrderItem

Actors: Jake, Restauraunt

Entry Conditions:

Jake has completed Use Case: ScanQRCode and is currently viewing the restaurant's menu details

Exit Conditions:

Jake is redirected back to the menu incase he wishes to order more

Main Flow Of Events:

1. Jake will select items from different food categories and add to his basket as he pleases
2. When finished, Jake will be able to review his basket and the items he has ordered
3. Jake can select the 'OrderItems' option
4. The order request will be sent directly to the restaurant's system
5. The order will be stored in the database
6. The order will be recorded in the user's own order history
7. The restaurant will know from the QR code information which table the order was requested from and can deliver food accordingly

Alternate Flow:

Scenario 5:

Use Case: CallStaff
Actors: Jake, StaffMember
Entry Conditions:
Jake must be logged in and on the main page of the app
Jake must have scanned the QR code, in order to call staff of a certain restauraunt
Exit Conditions:
Redirected to main page
Main Flow Of Events:
1. Jake will click the call button
2. A StaffMember will attend the call, and assist them accordingly
3. Since the QR code has been scanned the restaurant workers will know which table to attend
Alternate Flow:

Scenario 6:
Use Case: WriteReview
Actors: Jake
Entry Conditions:
Jake has completed an order and finished his meal
Exit Conditions:
Redirected to all reviews for that particular restaurant
Main Flow Of Events:
1. Jake has completed all use-cases listed in scenarios 1-4
2. Jake can now select the review page for that restaurant
3. Jake will specify the dish he ate, and comment his personal opinion in a textbox
4. Appeatite will ensure the review satisfies certain language and character constraints
5. Jake will leave a rating out of 5 stars
6. Appeatite will store the review in the restaurant's database
7. The review will be made public for all other users to view
Alternate Flow:
4. Appeatite displays an error message for too many or too less characters used, or due to profanity used in the comments
5. Appeatite will redirect the user to step 3

Scenario 7:
Use Case: ReadReview
Actors: Jake
Entry Conditions:
Jake has scanned the QR code for a particular restaurant
Exit Conditions:
Main Flow Of Events:
1. Jake has completed all use-cases listed in scenarios 1-3
2. Before making an order Jake can view the review tab by clicking it

3. A list of reviews will be displayed for that restaurant by Appeatite

Alternate Flow:

3. There are currently no reviews for this particular restaurant

Scenario 8:

Use Case: CancelOrder

Actors: Jake

Entry Conditions:

Jake has completed use-case OrderItems

Exit Conditions:

Re-prompted to restaurant's menu

Main Flow Of Events:

1. Jake must click cancel order option from the ordering tab
2. Appeatite will remove the order from their system

Alternate Flow:

3. There are currently no reviews for this particular restaurant

Scenario 9:

Use Case: ViewExpectedTime

Actors: Jake

Entry Conditions:

Jake has completed use-case OrderItems

Exit Conditions:

Jake can return to menu or close application

Main Flow Of Events:

1. Jake has successfully completed an order
2. The expected order time will be displayed

Scenario 10:

Use Case: RemoveIngredients

Actors: Jake

Entry Conditions:

Jake has completed use-case ViewMenu

Exit Conditions:

Jake is prompted to the page in order to perform use case: OrderItem

Main Flow Of Events:

1. Jake selects a dish from the menu he wishes to eat
2. (If applicable) Jake can remove certain ingredients from the dish by tapping them

Scenario 11:

Use Case: ViewOrderHistory
Actors: Jake
Entry Conditions:
Jake is on main page of Appeatite, signed into his existing account
Exit Conditions:
Jake leaves Order History tab and closes application
Main Flow Of Events:
1. Jake selects 'Recent' tab from main page which displays order history
2. Appeatite will display data taken from the database matching orders with Jake's username
3. Jake can view all orders he has made from which restaurant and what dishes he ordered
Alternative Flow:
2. There are no previous orders

Scenario 12:
Use Case: PrintQRCode
Actors: Employee A
Entry Conditions:
Employee A is on home page of client application
Exit Conditions:
Employee A is back on home page of client application
Main Flow Of Events:
1. Employee A has opened his client application on his system and is on homepage
2. Employee A will generate a QR code holding information of the restaurant's menu
3. Employee A will print the QR code using his system

Scenario 13:
Use Case: UpdateMenu (AddItem, RemoveItem, EditItem)
Actors: Employee A
Entry Conditions:
Employee A is on admin-mode of the menu
Exit Conditions:
Employee A is on view-mode of the menu
Main Flow Of Events:
1. Employee A is on his main page of client application
2. Employee A is in admin-mode of the menu where he can add/edit/remove items from the menu
3. Employee A adds an item specifying its name, prices, ingredients, calorie count, proportions, and pictures
4. Employee A saves his updates and the application stores them in the database
5. Employee A is prompted to a view-mode of the menu
Alternative Flow #1:
3. Employee A removes an item from the menu

4. The item is removed from the database, as Employee A saves changes on the application (Given that there are no pending orders for this item)
5. Employee A is prompted to the view-mode of the menu

Alternative Flow #2:

3. Employee A edits a certain dishes' name, picture, price, or ingredients
4. Employee A saves changes and the database is updated (Given that there are no pending orders for this item)
5. Employee A is prompted to the view-mode of the menu

Scenario 14:

Use Case: MarkUnavailableItems

Actors: Employee A

Entry Conditions:

Employee A is on admin-mode of the menu

Exit Conditions:

Employee A is on view-mode of the menu

Main Flow Of Events:

1. Employee A is on his main page of client application
2. Employee A is in admin-mode of the menu where he can update information of menu
3. Employee A ticks an items current availability
4. Employee A saves his updates and the application stores them in the database
5. Employee A is prompted to a view-mode of the menu

Scenario 15:

Use Case: UpdateSettings

Actors: Jake

Entry Conditions:

Jake is on settings panel of application

Exit Conditions:

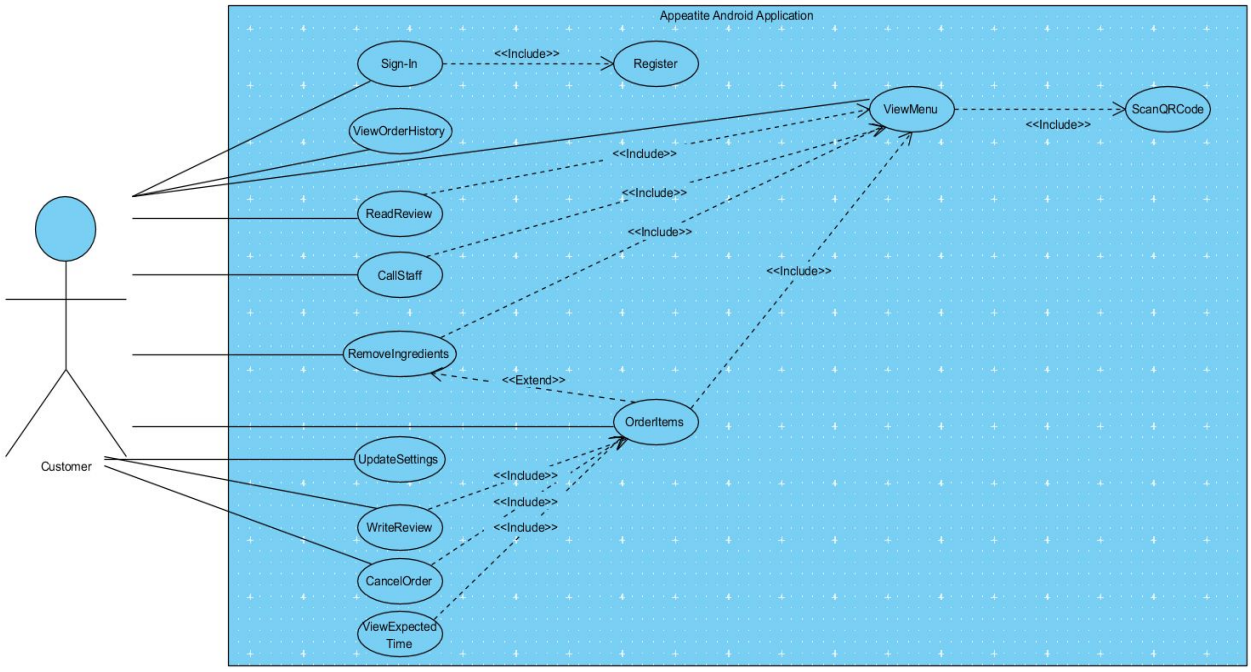
Jake returns to home page and/or closes application

Main Flow Of Events:

1. Jake is on the settings panel of Appeatite
2. Jake can select which unit he prefers to display his food energy in by selecting radio buttons
3. Jake can select what type of lighting he prefers for the application (dark, light)
4. Jake can select his preferred size of text for the application's information
5. Jake can save his settings and return to home-page

3.5.2 Use case model

Use Case Diagram #1: Customer



Use-Case Explanations:

Sign-In: Users can sign-in using their existing details that they registered into Appeatite's system. This use-case includes the 'Register' use-case because without registering users can not sign-in

Register: Users can create new accounts in order for them to use all of Appeatite's services

ViewOrderHistory: Users can view their previous order history they've made at each restaurant and what type of dishes

ReadReview: Users can read reviews for the restaurant once they have successfully scanned the QR code of the restaurant.

CallStaff: Users can call a staff employee to assist them after they have successfully scanned the QR code of the restaurant

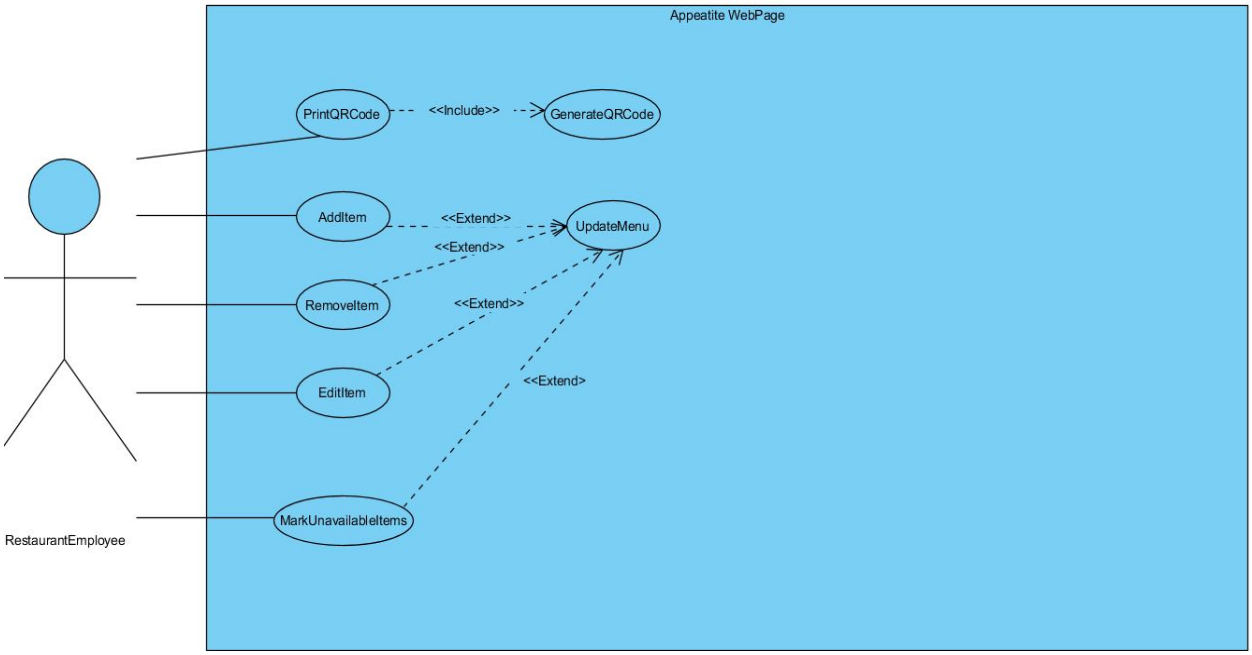
RemoveIngredients: Users can specify certain ingredients they wish to remove from the dishes they are order. This extends the OrderItems use case since they will remove ingredients before ordering items.

UpdateSettings: Users can update data units of calories, interface lighting, and text size from the settings tab

WriteReview: Users can leave a review after they have successfully ordered items and ate at the restaurant. This includes the OrderItems use case since they will not be able to make a review without actually making a purchase at the restaurant.

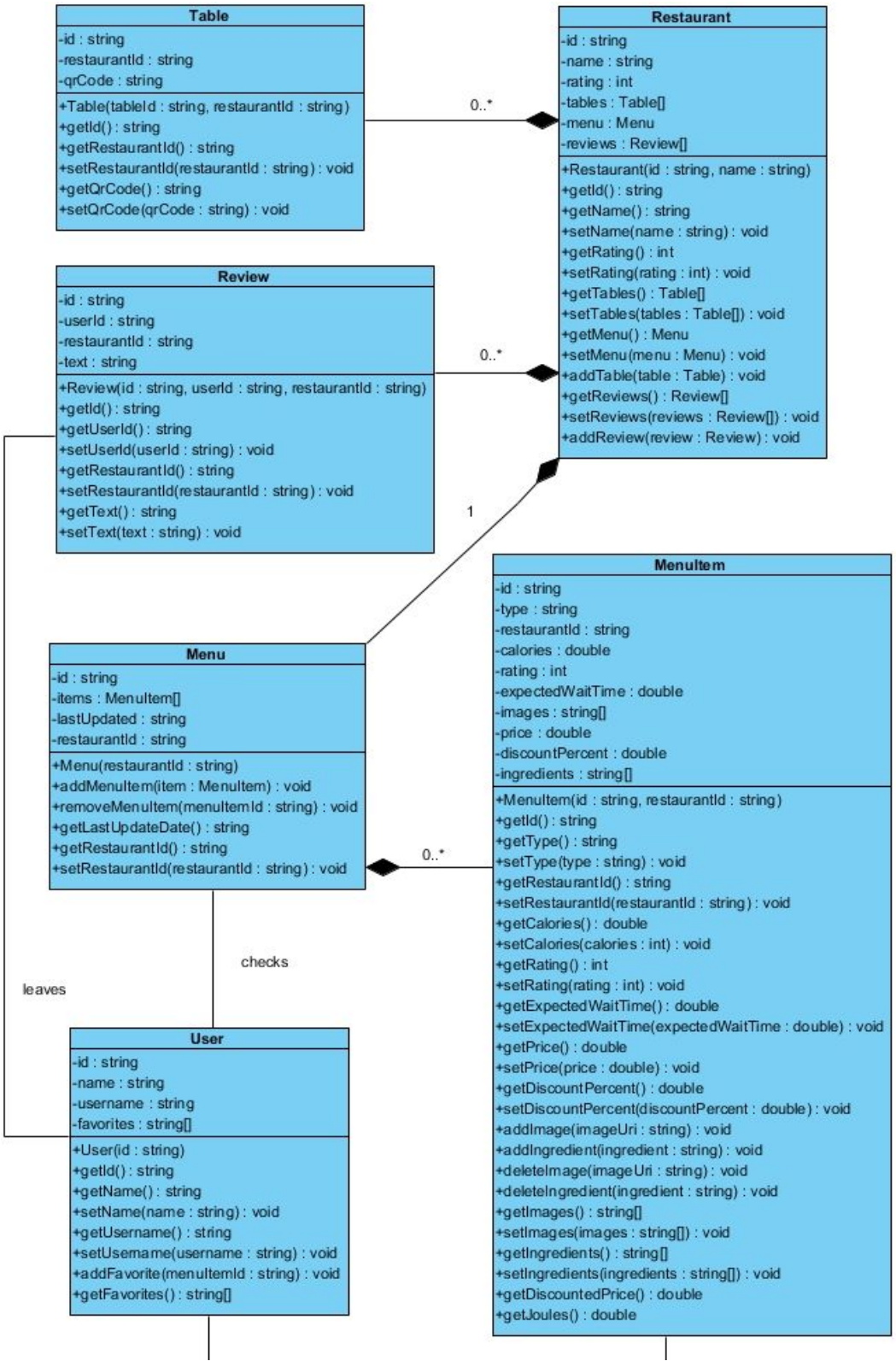
CancelOrder: Users can cancel an order if they have a made a mistake or simply wish to not eat anymore
ViewExpectedTime: After placing an order users can view the expected waiting time of their dish
OrderItems: Users can add items to their basket by selecting them after performing the ViewMenu use case and eventually order all items placed in basket
ViewMenu: Users can view the menu after successfully scanning the QR code using their mobile's camera.

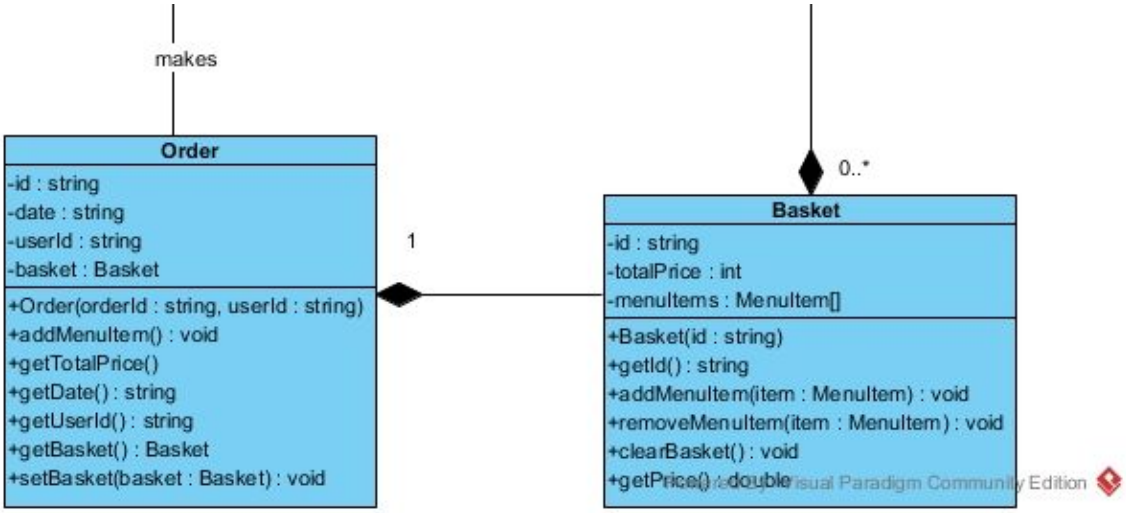
Use Case Diagram #2: Restaurant Staff



Use-Case Explanations:
GenerateQRCode: Using their client based application, the restaurant employees should be able to generate their QR code in order to place on their dining tables for users to scan
PrintQRCode: After generating their QR codes the restaurant should be able to directly print the QR codes from their system
UpdateMenu: This use-case consists of 3 variations explained below
AddItem: Restaurant staff will be able to add new dishes to their menu using an easy-to-use interface created by Appeatite's team. When adding a specific item they can specify the price, calorie intake, proportions, name, ingredients and also add relevant photographs
EditItem: Restaurant staff can edit existing dishes by changing their prices, proportions, etc.
RemoveItem: Restaurant staff can remove an item they no longer wish to serve from their menu
MarkUnavailableItems: On certain days, some dishes may not be served, hence, the restaurant can mark it as unavailable for a certain period of time

3.5.3 Object and class model





For the following classes all of the contain ids because those classes will be matched to the database object models where they will have their respective ids. Here ids are immutable and have purposes of differentiating the objects among each other since the nature of our application assumes many types of objects of similar types. All the ids are the character sequence to differentiate the instances of those classes among each other and query the database according to those ids. They are omitted in the class descriptions for brevity.

Restaurant:
data members:

- name: name of the restaurant
- rating: rating of the restaurant
- tables[]: list of table object that are in that restaurant
- menu: menu of the restaurant
- reviews[]: list of reviews that are left by the customers to this restaurant

methods:

- getter and setter methods for each data member
- addTable() - add a table to the restaurant object
- addReview() - add a review to the restaurant object

Table:
data members:

- restaurantId: id of the restaurant this table belongs to
- qrCode: unique QR code that is represented by the character sequence for this restaurant

methods:

- getter and setter methods for each data member

Review:
data members:
- userId: Id of the user whom this review belongs to
- restaurantId: id of the restaurant this review belongs to
- text: body of the review

methods
- getter and setter methods for each data member

Menu:
data members:
- items[]: list of menu items this Menu object has
- lastUpdated: date this id was last updated
- restaurantId: id of the restaurant this menu belongs to

methods
- getter and setter methods for data members
- addMenuItem(): add an instance of MenuItem to this particular menu object
- removeMenuItem(): remove MenuItem from menu object

User:
data members:
- name: name of the user
- username: the name of the user that is displayed to other users
- favorites[]: list of MenuItem objects that this user has marked as favorite

methods:
- setter and getter methods for data members
- addFavorite(): add a MenuItem ID to the list of favorites of the user

Order:
data members:
- date: date this order was made
- userId: Id of the user this order belongs to
- basket: basket of MenuItems that was purchased in this order

methods:
- necessary getter and setter methods for data members
- addMenuItem(): addMenuItem to the order, item is added to the basket
- getPrice(): the price of all MenuItems in the basket of the order

Basket:
data members:
totalPrice: total price of the MenuItems in this basket
items[]: list of Items in the basket
methods:
- getter methods for id of the Bakset object and price

- addMenuItem(): add a Menu Item to the basket
- removeMenuItem(): remove a Menu Item from the basket
- clearBasket(): clears the basket

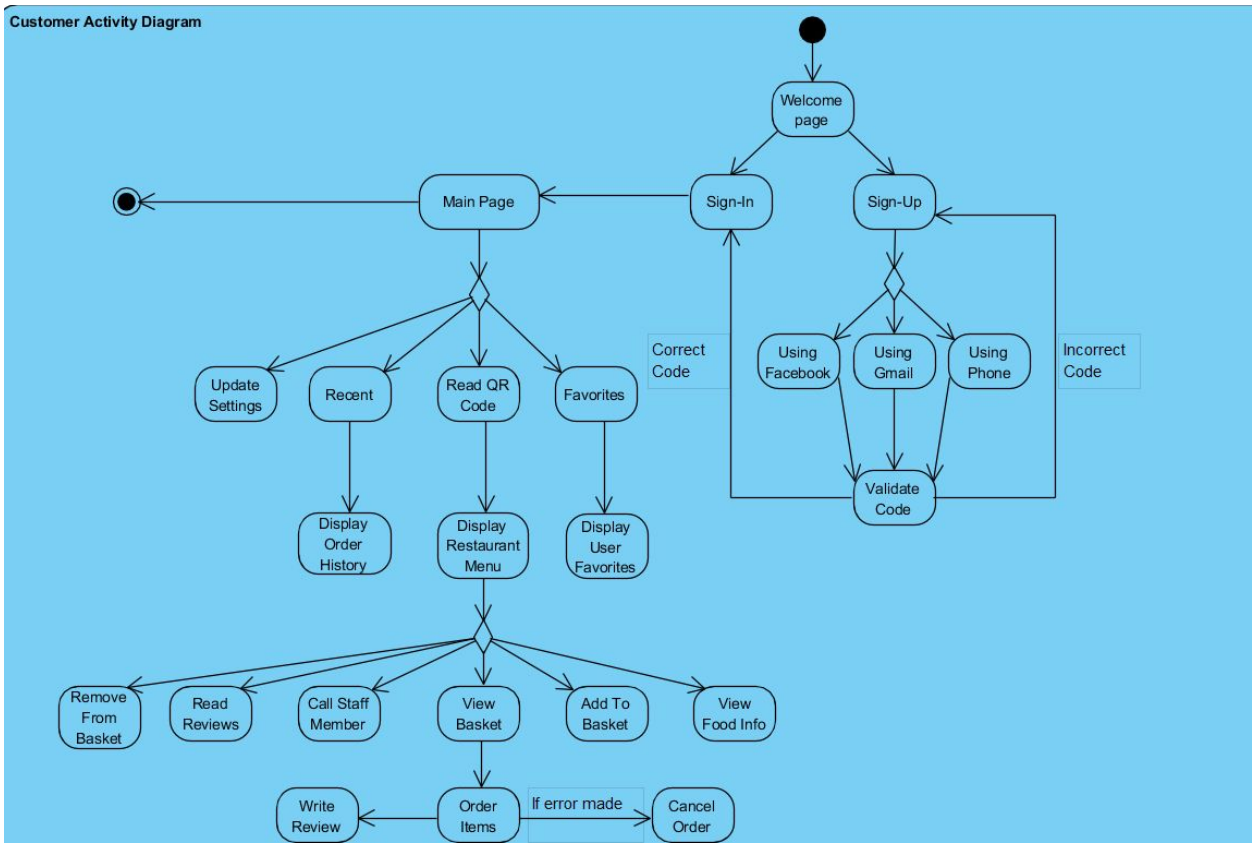
MenuItem:
data members:
- type: type of the menuItem that corresponds to the category of the food
- restaurantId: id of the restaurant this menu Item belongs to
- calories: energy of the food stored in calories
- rating: rating of the menuItem
- expectedWaitTime: wait time of the food from the time order was confirmed and brought to the user
- images[]: list of Image links of this menu item
- price: price of the menu item
- discountPecrent: discount percent from the original price
- ingredients[]: list of ingredients of the menu item

methods:
- necessary getter and setter methods
- add and remove methods for images and ingredients data members
- getDiscountedPrice(): price with discountPrice taken into accont
- getJoules(): get energy of the food in Joules instead of calories

3.5.4 Dynamic models

3.5.4.1 Activity Diagram

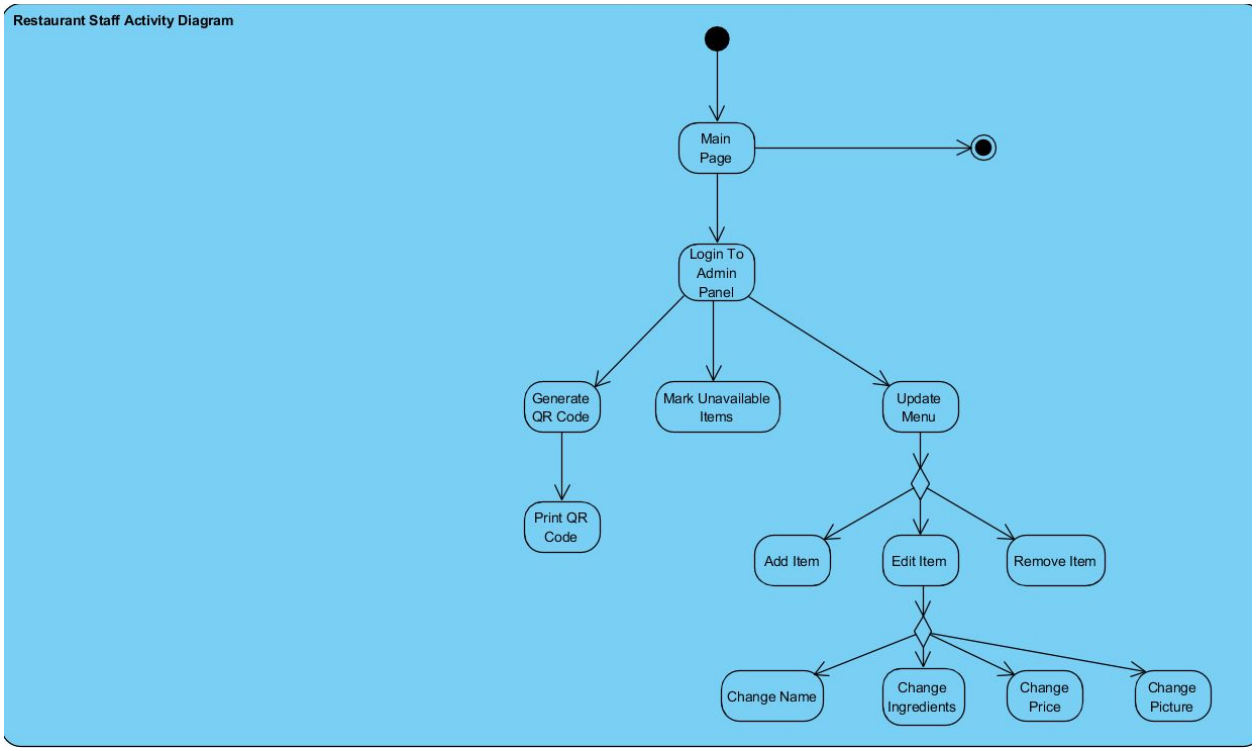
Activity Diagram #1: Customer



The initial starting point will prompt the user to the welcome page. From where new users are required to register a new account. There are three different registration options: using gmail, facebook, and by phone. After choosing a registration option and entering necessary details, the users will be required to validate the code ensuring they providing correct information. If the information is inaccurate they will be prompted to the registration page to start over. Otherwise, they will be redirected to the sign-in page where they will be asked to enter the credentials they used in registering their account to Appeatite's database. After signing-in the users will be prompted to their main page. From here they can exit the application or use the existing features. From the home page users can update their preferred settings, view their order history, display their favorite dining places, and finally if they are seated in a restaurant they can begin to scan the QR code and Appeatite can work its magic. Once the user scans the QR code of a restaurant the menu will be displayed by the restaurant and from here users can begin to view food items and their relevant information. Then they can add items they wish to purchase to their basket and make an order. Additionally, they can read reviews of the restaurant, or call a staff member if they require any assistance. Finally after successfully ordering and eating their

food, users can leave a review for the dish they ate and a rating for the restaurant's overall service.

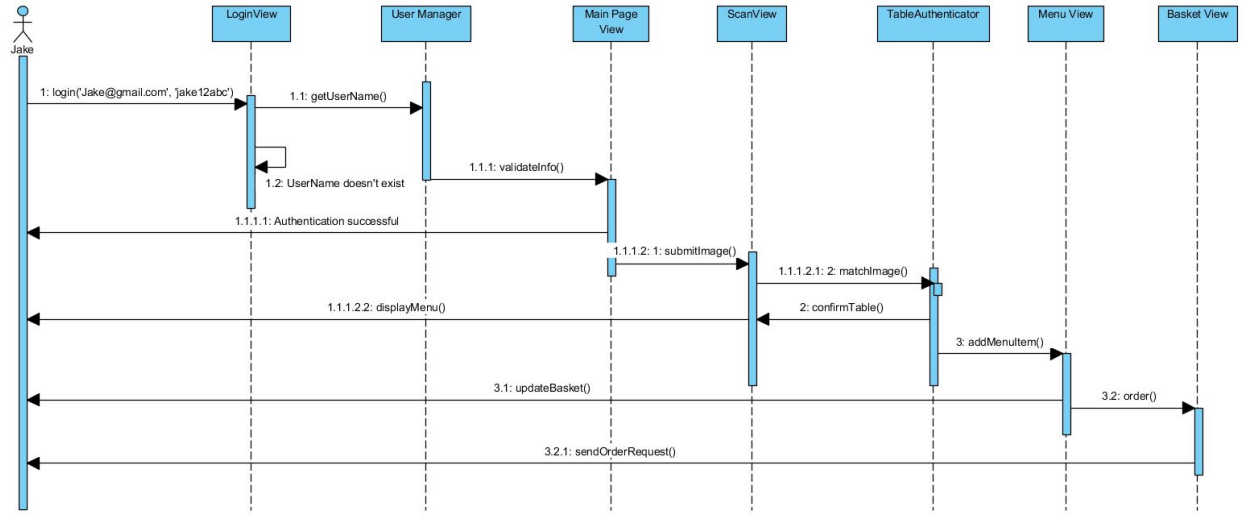
Activity Diagram #2: Restaurant Staff



An employee will monitor the client web-based application. From the main page they can either simply be in read only mode or log-in to their admin panel to make necessary changes to their menu. From here, the employees will be able to generate QR codes and print it to be placed on their dining tables. The staff can update their menu by adding, editing or removing dishes from their current menu. Additionally they can mark currently unavailable items for a specific period of time.

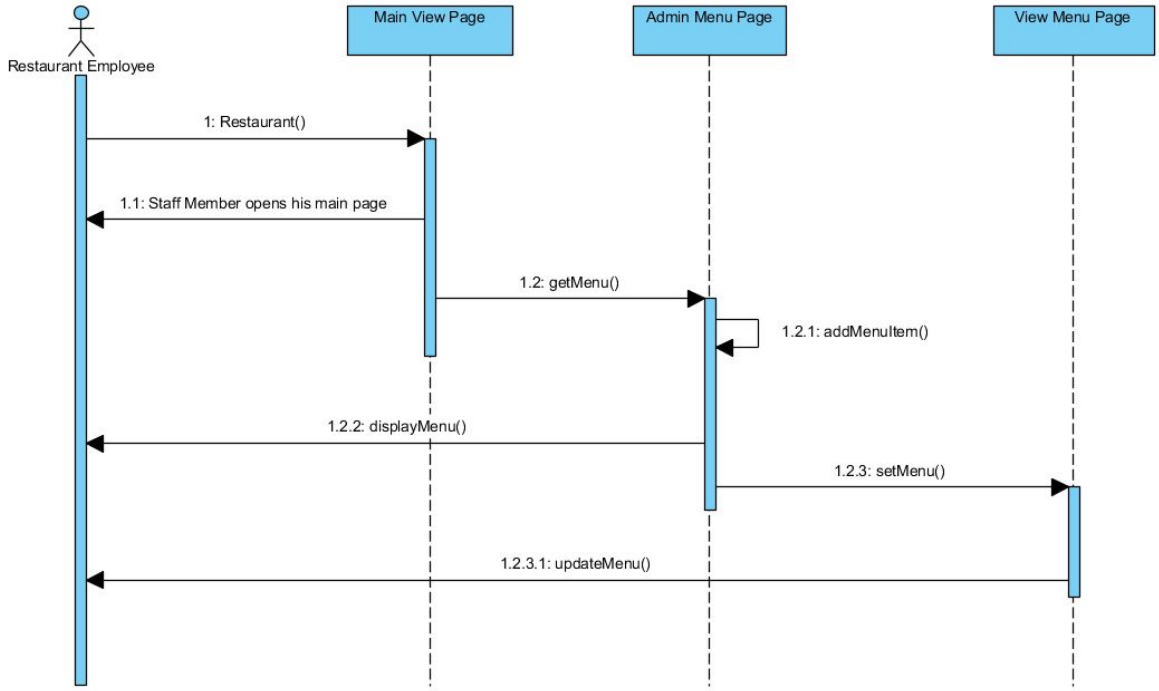
3.5.4.2 Sequence Diagram

Sequence Diagram Example #1: Customer



Here, the customer has already registered to Appeatite application successfully before. The user's name is Jake, and he attempts to log-in using his existing credentials. He is prompted to the log-in view when inputting the releveant information. The User Manager will use the function getUserName() to match the inputted username with the username stored in the database. If the username does not exist the user will be re-prompted to reenter his details. Otherwise, it will be validated correctly and authentication will be successful and the user will be prompted to his main page. From here, the user can click read a QR code and be on the ScanView page where an image is inputted of the QR code and the corresponding menu is displayed. The image is matched with the image in the database and is confirmed in the table. Then the user can update his basket by adding menu items to it and finally placing an order and sending a request to the restaurant's system.

Sequence Diagram Example #2: Restaurant



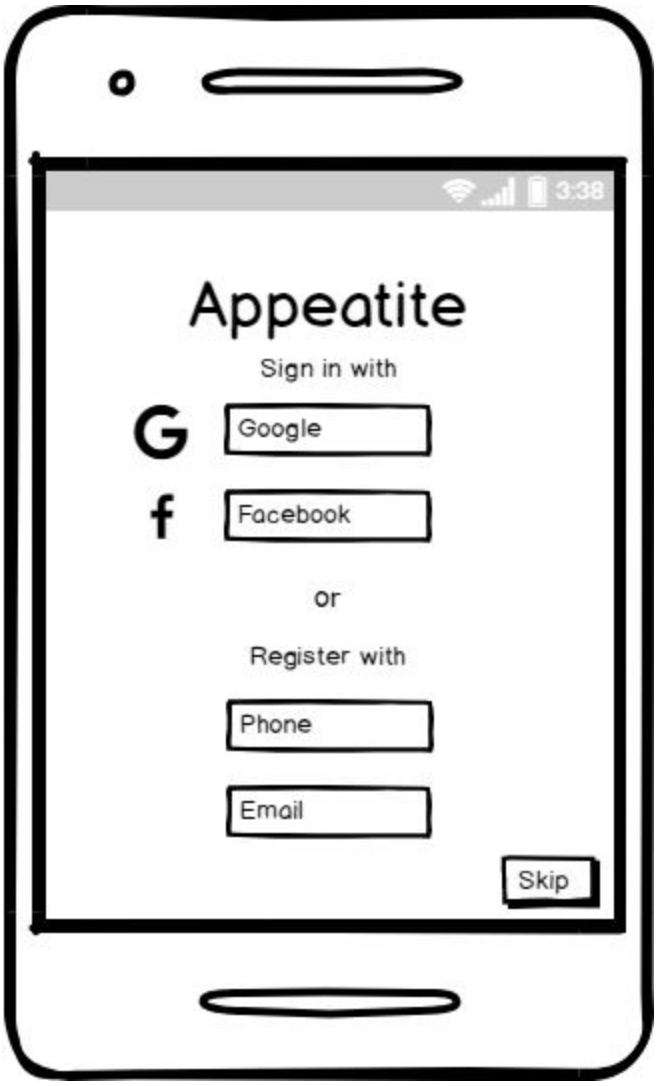
This is a simple scenario where the restaurant employee opens his admin menu page and gets the existing menu information from the database server. Then menu items are added to a menu object with relevant information, and the new menu is set to the restaurant.

3.5.5 User interface

3.5.5.1 Welcome Screen

This screen is the ‘Welcome Screen’ and is shown to the user when he launches the application for the first time. The screen contains our logo and slogan (subject to change).

3.5.5.2 Registration Page



The registration page is automatically prompted to the users after the welcome page is shown for a short moment. Users can select from four different options to register their accounts and become a part of Appeatite. Users can skip the registration however functionality will be strictly limited.

3.5.5.3 Register with Email

This page follows the home page where new users can either register using their existing facebook or gmail accounts, or register using their e-mail address. If they wish to register an account using their email address, they can simply type in their existing email account and choose a password for security purposes.

3.5.5.4 Register with Phone number

This page follows the home page where new users can either register using their existing facebook or gmail accounts, or register using their phone number. If they wish to register an account using their phone number they can simply enter their valid phone number and continue.

3.5.5.5 Code confirmation

If a user selected one of the options to register with their phone number or email they will be required to enter a confirmation code in order to verify they did not provide false information.

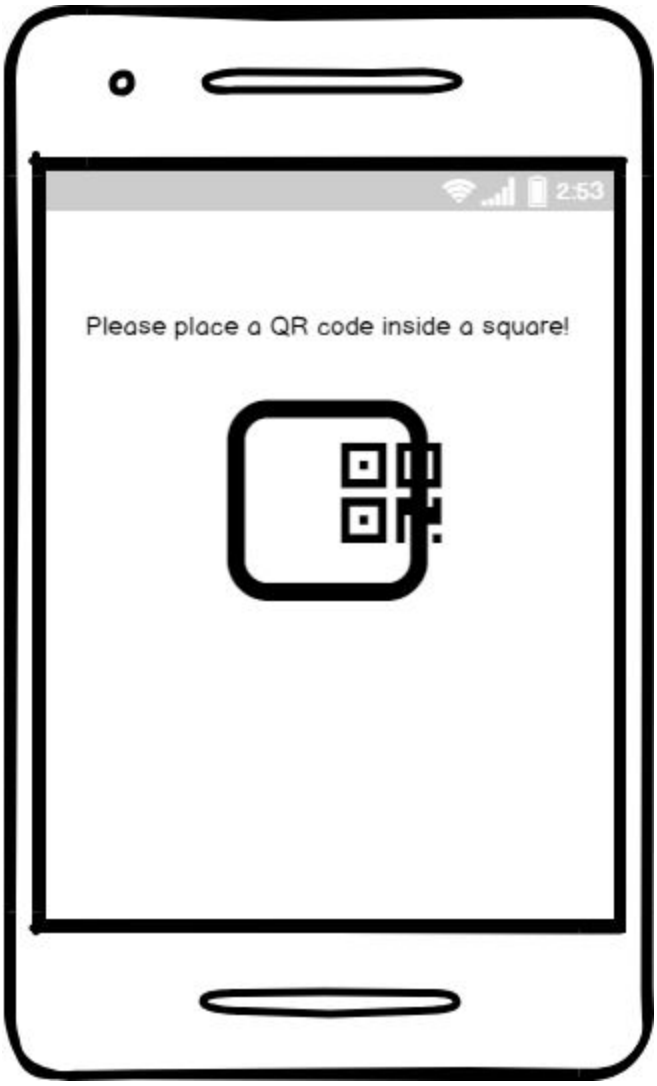
3.5.5.6 Login

For existing users, they can simply log in with their registered email addresses and corresponding passwords. Usually the log-in screen won't be asked for each time the users open the application, however, it is still required in the case a user resets his data, purchases a new device, or simply is returning to the application as an old-time user.

3.5.5.7 Home

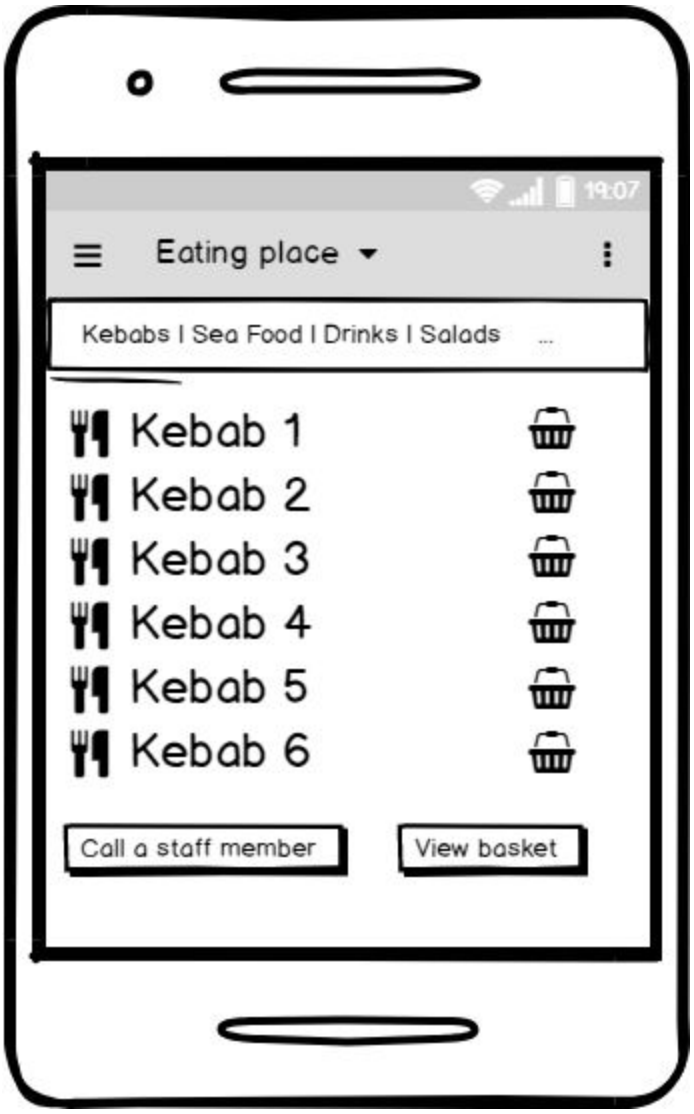
This is the home page of the user after he passes the registration or when he returns to the application as a recurrent user. He can view his order history and favorites. The QR code button is also placed here which prompts the application to open the camera of the mobile, and is ready to scan a QR code.

3.5.5.8 QR code reader



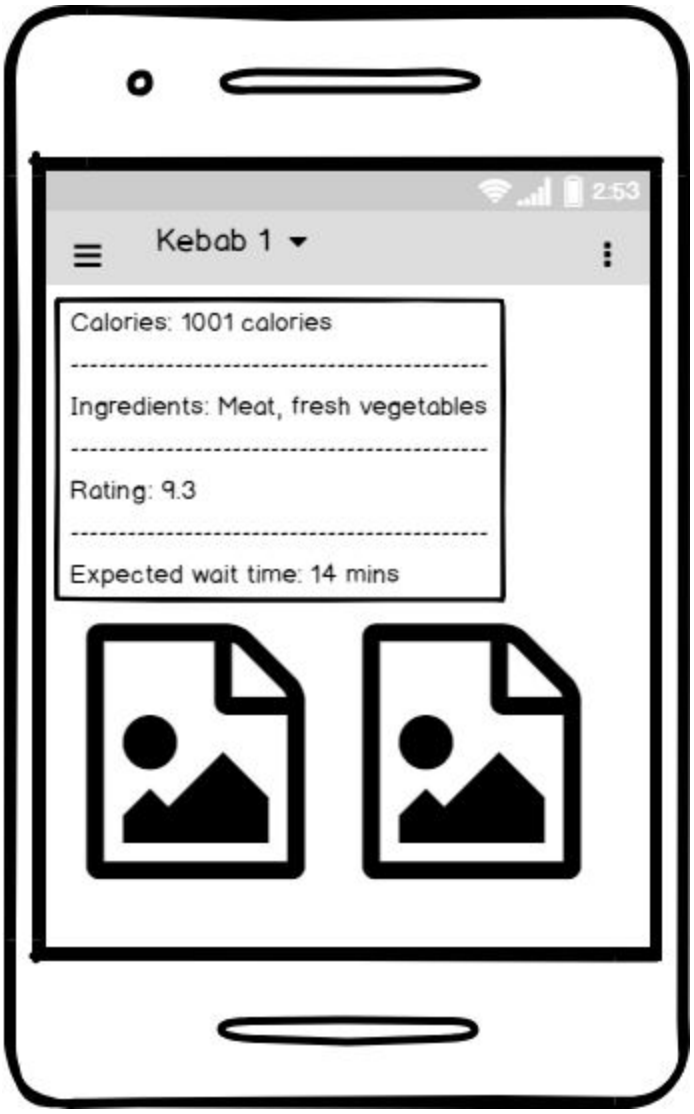
This screen mocks how the user going to be using his phone in order to read the QR code to match the restaurant from the database, and receive the menu information.

3.5.5.9 Tabs



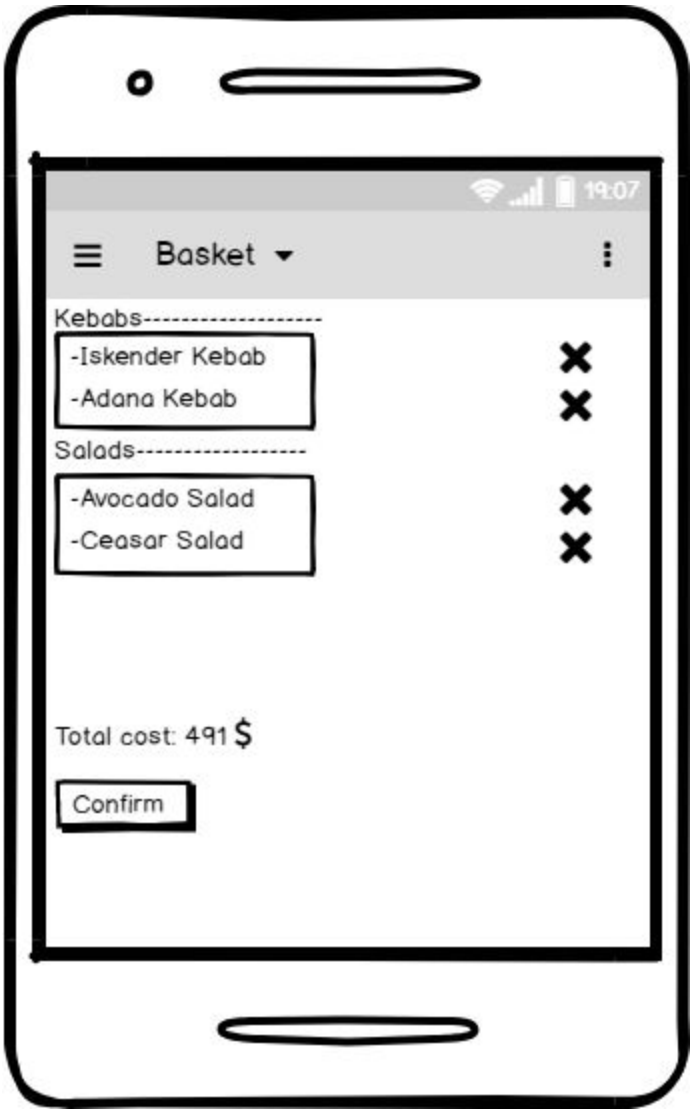
The tabs are opened after the QR code is read and successfully matched with the restaurant. The screen is populated with required information from the database for this particular restaurant. This is essentially what this restaurant can offer in terms of the food. The user can navigate through swipes between the tabs. Each tab represent a food category. Also from this screen user can request a staff member for consulting purposes. Finally, users can add specific dishes to their basket and view their basket and current items added.

3.5.5.10 Food information



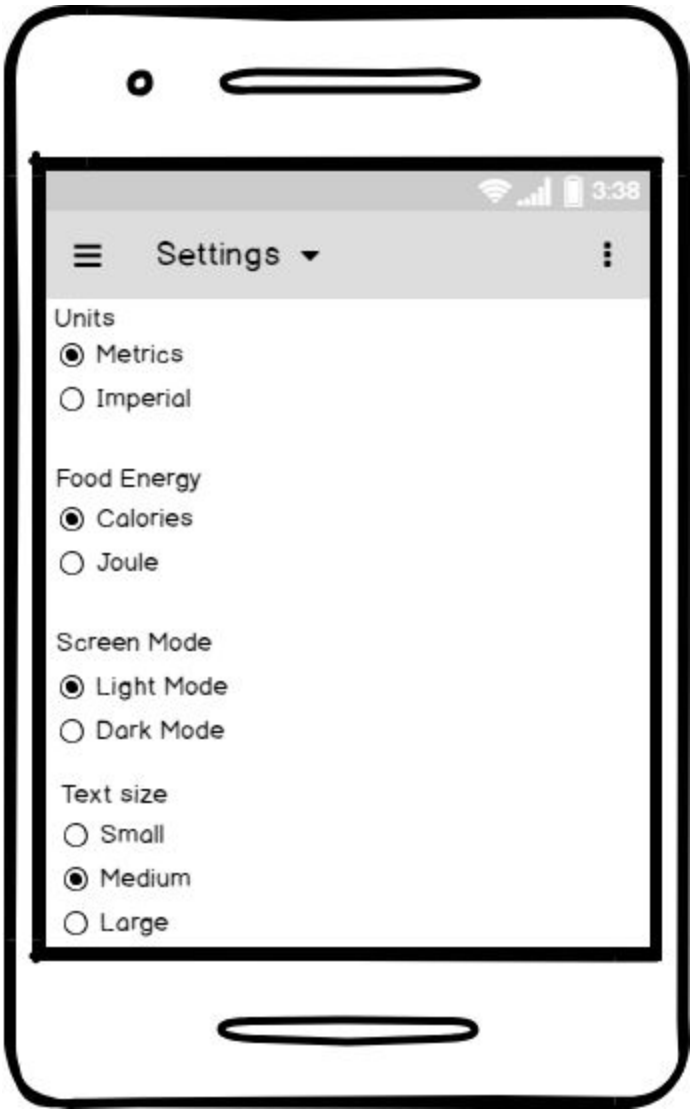
This screen opens when the user taps on the menu item. Here the detailed information about the menu item is listed such as calories, ingredients, rating, and expected wait time along with the images of the food. This allows a very abstract interface, making it convenient for the users as they can view information as they please.

3.5.5.11 Basket



This is the current basket of the user that displays the total cost is of the items in the basket is displayed to the user. User can confirm the order by pressing the corresponding button. The order will be sent to the restaurant's system, and they will begin to cook the dish. Also the menu items can be removed from the list and the total cost will be recalculated accordingly.

3.5.5.12 Settings



In the settings menu user can choose preferred unit measurement system, food energy representation, screen mode and text size.

4. References

[1] “Cloud Firestore | Usage and Limits” [Online]
https://firebase.google.com/docs/firestore/quotas [Accessed Oct 7, 2018]

[2] “Node.js” [Online]
https://nodejs.org/en/ [Accessed Oct 7, 2018]

[3] “Typescript - Javascript that scales” [Online]
https://www.typescriptlang.org/ [Accessed Oct 7, 2018]

[4] “Visual Studio - Code Editing Redefined” [Online]
https://code.visualstudio.com/ [Accessed Oct 7, 2018]

[5] ”IEEE Code of Ethics” [Online]
https://www.ieee.org/about/corporate/governance/p7-8.html [Accessd Oct 8, 2018]

[6] ”Balsamiq. Rapid, effective and fun wireframing software” [Online]
https://balsamiq.com [Accessd Nov 10, 2018]

[7] Bruegge, Bernd, and Allen H. Dutoit.
Object-oriented Software Engineering: Using Uml, Patterns and Java. Upper Saddle River,
NJ: Prentice Hall, 2003. Print.

[8] *Visual Paradigm*, Visual Paradigm, 2018