



VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Vuong Phuc Thanh

IOS MOBILE APPLICATION FOR ORDERING SYSTEM

Technology and Communication

2018

ACKNOWLEDGEMENT

Firstly, I would like to show my appreciation and my respect toward Dr. Ghodrat Moghadampour, my supervisor, for all the knowledge I have been taught during my journey studying at VAMK.

His conscientious and passionate instructions influenced to become a software engineer in the future. After two years majoring in Software Engineering, I have achieved many methods as well as skills from his courses, which I applied for the process of my final thesis. I would like to thank him for the massive support.

I would like to thank my colleague, Huy Phan. He played an important role as my customer, giving his designs and ideas for the whole mobile application. Without his collaboration, my thesis would be a mess in the user interfaces.

Also, the community of developer on Stack Overflow, Github and colleagues from the company where I completed my internship, helped me a lot with many problems that I faced during implementing period. I would like to thank them, for spending time with my questions.

Finally, I would like to express my appreciation to my parents, who supported me financially and mentally for four-years period of study. Without their encouragement, It would be difficult to complete the bachelor degree.

I want to show my gratitude to all of you, thank you.

Vaasa, 23.03.2018

Vuong Phuc Thanh

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Technology and Communication

ABSTRACT

Author	Vuong Phuc Thanh
Title	IOS Mobile Application For Ordering System
Year	2018
Language	English
Pages	68
Name of Supervisor	Ghodrat Moghadampour

The objective of this thesis was to develop a complete system for ordering goods and foods, which allow customers to order products from a mobile device.

The application includes two different views. A web-based Admin Dashboard for store managers, where they can manage orders, edit products, menus, and process the information of orders with the customers. This admin dashboard is built by Python with Django framework and SQLite.

The second view displays a list of stores and restaurants, with the details of products, foods on the menu. Clients are able to order and make payment and send their information and location to the admin side through RESTful API. This IOS application is built with Swift.

Basically, the project has achieved all the must-have requirements and has been tested by some colleagues. The application is not only a multi-functional but also provides a clear user interfaces.

The thesis can be extended for future development with potential features, which can be applied in many specific stores that require a system for online ordering.

Keywords Python, Django, SQLite, Swift, RESTful API

TABLE OF CONTENTS

ACKNOWLEDGEMENT

ABSTRACT

1	INTRODUCTION.....	10
1.1	BACKGROUND.....	10
1.2	MOTIVATIONS	10
1.3	OBJECTIVES	11
2	RELEVANT TECHNOLOGIES	12
2.1	PYTHON.....	12
2.2	DJANGO WEB FRAMEWORK	12
2.3	PIP AND VIRTUALENV	13
2.4	SQLITE	13
2.5	DJANGO OAUTH TOOLKIT AND REST FRAMEWORK SOCIAL OAUTH.....	14
2.6	BOOTSTRAP.....	14
2.7	XCODE IDE AND SWIFT PROGRAMMING LANGUAGE	15
2.8	COCOPODS.....	15
2.9	SWIFTYJSON AND ALAMOFIRE	15
2.10	STRIPE	16
2.11	RESTFUL WEB SERVICE	16
3	APPLICATION DESCRIPTION.....	18
3.1	GENERAL DESCRIPTION.....	18
3.2	QUALITY FUNCTION DEPLOYMENT.....	18
3.2.1	<i>Must – have requirements.....</i>	<i>18</i>
3.2.2	<i>Should – have requirements</i>	<i>19</i>
3.2.3	<i>Nice – to –have requirements.....</i>	<i>19</i>
3.3	USE CASE DIAGRAM	20
3.4	CLASS DIAGRAM.....	21
3.5	SEQUENCE DIAGRAM	24
3.5.1	<i>Admin Dashboard Registration Sequence Diagram.....</i>	<i>24</i>
3.5.2	<i>User Login Sequence Diagram</i>	<i>24</i>
3.5.3	<i>Adding meal Sequence Diagram.....</i>	<i>25</i>
3.5.4	<i>Editing meal Sequence Diagram</i>	<i>26</i>

3.5.5	<i>Editing account sequence diagram</i>	27
3.5.6	<i>Order page sequence diagram</i>	27
3.5.7	<i>Mobile User Login Sequence Diagram</i>	28
3.5.8	<i>Getting Restaurant List On Mobile Sequence Diagram</i>	29
3.5.9	<i>Creating Order Sequence Diagram</i>	29
3.5.10	<i>Meal Page Sequence Diagram</i>	30
3.6	COMPONENT DIAGRAM.....	31
4	DATABASE AND GUI DESIGN	33
4.1	DATABASE DESIGN.....	33
4.2	GUI DESIGN	34
4.2.1	<i>Admin Dashboard web base application</i>	35
4.2.2	<i>GUI of Mobile Application</i>	39
5	IMPLEMENTATION	43
5.1	GENERAL DESCRIPTION OF IMPLEMENTATION	43
5.2	IMPLEMENTATION OF DIFFERENT PARTS.....	44
5.2.1	<i>Implementation of Admin Dashboard web based application</i>	44
5.2.2	<i>Mobile application</i>	52
6	TESTING	59
6.1	SIGNING UP A NEW RESTAURANT.....	59
6.2	EDITING ACCOUNT INFORMATION.....	60
6.3	ADDING MEAL.....	60
6.4	EDITING MEAL	61
6.5	SIGNING WITH FACEBOOK ON MOBILE	62
6.6	GETTING LIST OF RESTAURANTS.....	62
6.7	GETTING LIST OF MEALS.....	63
6.8	MAKING AN ORDER.....	64
6.9	CHANGING ORDER STATUS.	65
7	CONCLUSION	66
	FUTURE WORKS.....	66
REFERENCES		

LIST OF FIGURES AND TABLES

Figure 1. MVT architecture.	13
Figure 2. REST api Login with Facebook process.	14
Figure 3. Stripe payment process	16
Figure 4. RESTful web service process example.	17
Figure 5. Functions for store owner.	20
Figure 6. Functions for Customer.	21
Figure 7. Diagram for View	22
Figure 8. Class Diagram for API	22
Figure 9. Class Diagram for APImanager.....	23
Figure 10. Admin Registration.....	24
Figure 11. User Login Sequence Diagram.....	25
Figure 12. Adding meal sequence diagram.....	25
Figure 13. Editing meal sequence diagram.....	26
Figure 14. Editing account sequence diagram.	27
Figure 15. Order page sequence diagram.....	27
Figure 16. Facebook Login sequence diagram	28
Figure 17. Getting Restaurant List on Mobile Sequence Diagram	29
Figure 18. Creating Order Sequence Diagram.....	30
Figure 19. Meal and Meal details Sequence Diagram	30
Figure 20. Component Diagram.....	31
Figure 21. ER diagram.	33
Figure 22. Sign in page for web application.	35
Figure 23. Sign up page.	35
Figure 24. Order page.	36
Figure 25. Meal page.	36
Figure 26. Add meal page.	37
Figure 27. Edit meal page.	37
Figure 28. Account page.	38
Figure 29. Navigation Bar.....	38
Figure 30. Log in page.	39
Figure 31, Figure 32. Restaurant list and Meal list.....	39

Figure 33. Side bar.....	40
Figure 34. Meal details page.....	41
Figure 35, Figure 36. Shopping cart, Empty cart.....	41
Figure 37, Figure 38. Order page and Payment page.....	42
Figure 39. Structure of the admin dashboard application.....	43
Figure 40. Structure of the mobile application.....	44
Figure 41. The page shows error message.....	59
Figure 42. Successfully registered.....	59
Figure 43. New information is saved.....	60
Figure 44. Filling the form.....	61
Figure 45. New meal displayed on the Meal page.....	61
Figure 46. Editing form.....	61
Figure 47. New description updated.....	62
Figure 48, Figure 49. Sign in button, Information of user from FB.....	62
Figure 50. List of restaurants which includes also a testing restaurant.....	63
Figure 51. meal list of testing restaurant.....	63
Figure 52, Figure 53. Submit address, After making payment.....	64
Figure 54. Order is sent to the Admin dashboard.....	64
Figure 55. Changing status from Admin side.....	65
Figure 56. Status of the order is changed on the mobile application.....	65

LIST OF SNIPPETS

Code Snippet 1. Setting up urls for functions.....	45
Code Snippet 2. Creating model forms.	46
Code Snippet 3. Sign up function.....	47
Code Snippet 4. Modify function for user’s account.....	47
Code Snippet 5. Function for getting data of orders and meals.	48
Code Snippet 6. Function for adding new meal.	48
Code Snippet 7. Function for editing meal.....	49
Code Snippet 8. Serializer.	50
Code Snippet 9. Creating API.	51
Code Snippet 10. Getting restaurant list function.....	53
Code Snippet 11. Searching restaurant function.	53
Code Snippet 12. Getting meal list function.....	54
Code Snippet 13. Changing quantity of meal and add to tray button function.	54
Code Snippet 14. Taking location.	55
Code Snippet 15. POST request for creating order.	56
Code Snippet 16. Getting latest order.....	57
Code Snippet 17. getting user data from Facebook.....	58

LIST OF ABBREVIATIONS

API	Application Programming Interface
JSON	Javascript Object Notation
OS	Operating system
SQL	Structured Query Language
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
SDK	Software Development Kit
HTTP	Hypertext Transfer Protocol
REST	Representational state transfer
CSS	Cascading Style Sheet
UI	User Interface
URL	Uniform Resource Locator

1 INTRODUCTION

The introduction of the project including background, objectives and motivations is discussed.

1.1 Background

People are living in the technology age, where each day there is something invented to improve human's lives. There were many technology innovations, and big machines have now evolved into smaller devices with stronger functionalities. The usage percentages of mobile phones are overtaking the market that can lead to a prediction: mobile devices are the future of technology. IOS is an operating system developed by Apple that lifted the world of technology to a new page in 2007 and it is continuously growing thanks to a powerful system and friendly design.

The RESTful web service is one of the most popular types of API, which offers an easy way for connecting, integrating and extending a software system. In practice, RESTful web service would be a reasonable choice, when making communication between separate servers and clients.

Python and Swift are two of the most used programming languages in the developer community. Both of these languages are effective and powerful. With numerous libraries and frameworks, they were used to build many big applications. Also, Python and Swift were chosen in this thesis because of readability, which makes the developer's life much easier. It allows the user to express programming thought, without many syntax errors and to easily build an application.

1.2 Motivations

The motivation for this thesis came from the idea that making a third party for ordering food can be applied in industry. Similar to Amazon or Ebay, owners of restaurants or stores related to food industry can sign up and advertise products for customers in a specific area. Due to real life situation in Vaasa or Helsinki, ordering food is mostly done by searching a website and calling the restaurant which is inconvenient when the customer must look through many restaurants.

The motivation also came from personal interests and practical skills for becoming an IOS developer as well as a full stack developer in the future.

1.3 Objectives

The project should be completed with features based on the ideas that come from the motivations. The admin dashboard should have a user friendly interface for the user to manage the products and orders, and also be easy to navigate action between pages. The mobile application should have good communication with the server using the REST web service to take all information and afterward display it on the screen. The information of users and payments should also be secured.

The services include a page for creating menu and order, list and details of the menu for the mobile application, ordering system, making payment and sending the location with Google API.

2 RELEVANT TECHNOLOGIES

This section will take a review through all the relevant tools and technologies used for the project.

2.1 Python

Python is a scripting programming language developed by Guido van Rossum in 1989. It supports many types of programming paradigms including procedural, imperative and object-oriented. /1/

Python has a simple, easy-to-use syntax that allows developers from any level to express ideas and algorithms with readability counts. One of the unique features of Python is whitespace acts as statement indentation, which help the code written by Python to be cleaner and have fewer lines compared to Java or C# languages. /1/

Python has not only a large standard library, but also huge packages of third-party libraries with a wide range of functionality, making Python powerful enough to complete any tasks with different purposes./1/

For this project, the newest version Python 3.6 is used with supported third-party libraries for stable application.

2.2 Django web framework

Django is a web framework written by Python, which was created in 2003 by Adrian Holovaty and Simon Wilison.

Django follows the Model – View –Template(MVT) software architecture, similar to MVC, which assists the application with faster development process and ability to provide multiple views./2/

Django Admin Dashboard allows user to use ‘create, read, update, delete’ interface that is generated and configured via admin models./2/

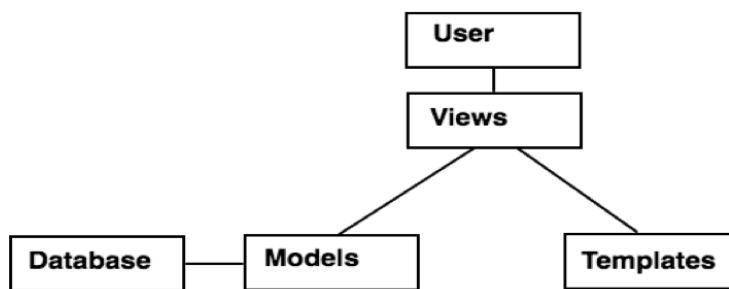


Figure 1. MVT architecture. /15/

In a Django web application, View file has a collection of functions for the application and Templates is a HTML file for user interfaces. The MVT model is slightly different from MVC, Django itself acts as a controller./2/

Django 2.0 is the latest version of Django framework with the newest release on March 2018. For this project, Django version 1.10 is used, because Django 2.0 is still under development and lack of supporting libraries.

2.3 Pip and Virtualenv

Pip is known as Python Package Manager, used for installing and updating standard or third-party libraries./3/

Virtualenv is used for separating different projects. It allows to create virtual environments with different settings and packages, so that breaking the packages installed will not happen when switching between projects./3/

2.4 SQLite

SQLite is a relational database management system which is an embedded database, the database runs as a part of the application. SQLite is a lightweight system that does not require its own process or user management./4/

By default, after installing Python and Django, the system uses SQLite for database. It is an easy to configure database system which is serverless, also the best choice for standalone applications./5/

2.5 Django OAuth toolkit and Rest framework social OAuth

OAuth is an open standard for access delegation, used to grant permission and information of the user on websites or applications without giving identification such as passwords. This methods used with accounts such as Facebook, Google, Twitter and many more. The application that has been authorized is limited by an access token given from the third-party clients. This access token will be used to protect resources which are held by the resource server./6/

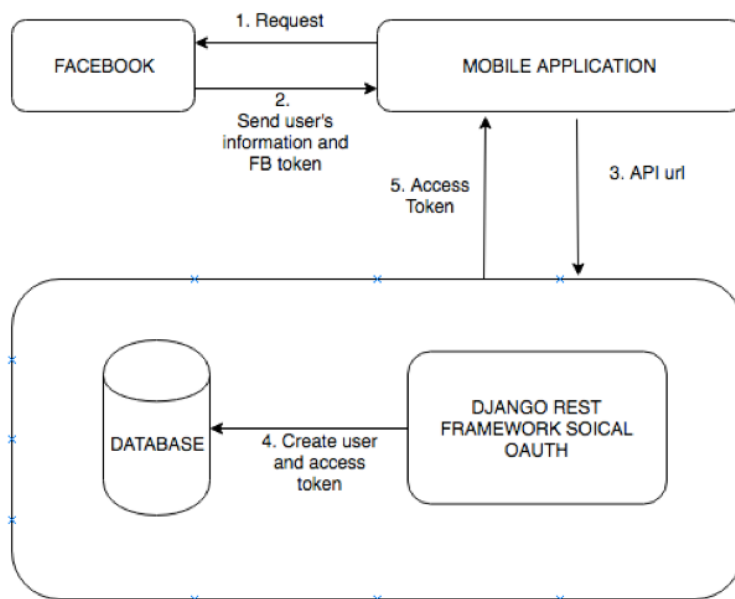


Figure 2. REST api Login with Facebook process.

For this project, the Django rest-framework Social Oauth2 module is used with a Oauth2 support for communicating between the mobile application and the server, and allows clients to login by a social media account in a most effective and secure way.

2.6 Bootstrap

Bootstrap is a front-end development framework that contains HTML and CSS based for typography, forms, buttons and many other components. One of the advantages for using Bootstrap is the grid system, developers can easily design a responsive web interface without too many details in CSS and HTML. /7/

2.7 Xcode IDE and Swift programming language

Xcode is an integrated development environment (IDE) developed by Apple which contains tools for developing applications on macOS, iOS and other operating systems from Apple. This IDE is mostly used for Swift programming and has many features that bring a massive support for developers. /8/

Swift is a general-purpose and multi-paradigm programming language developed by Apple, which was introduced in 2014. Swift is highly ranked among the developer community for a friendly design and simple syntax. /9/

Swift is a powerful and intuitive programming language for developing applications for operating systems that released by Apple. Swift code along with Xcode IDE gives an interactive environment to developers and improves complicated development processes to be faster and cleaner. /10/

2.8 CocoaPods

CocoaPods is a dependency manager for Swift and Objective-C projects, contains thousands of libraries that improve discoverability if third-party libraries.

Users can use a single text file called Podfile which CocoaPods recognize libraries in the file, fetch the resulting source code and link it an and Xcode workspace. /11/

2.9 SwiftyJSON and Alamofire

SwiftyJSON and Alamofire are both third-party libraries that installed by using CocoaPods.

Dealing with JSON with SwiftyJSON is much easier and cleaner when developers will not have to worry about writing a huge amount of code for some JSON data.

SwiftyJSON simply decreases the unreadable mess to be simpler.

Alamofire is a Swift-based HTTP networking library, which assists developers to perform networking tasks such as HTTP request, response and other methods for data from a RESTful API. /12/

2.10 Stripe

Stripe iOS SDK provides quick and easy methods to build payment experiences on iOS app. Stripe SDK can be customized with the UI screens and elements which help to improve user interfaces. The features of Stripe are simplified security, support Apple Pay and Credit Card payment and also Native UI.

For this project, Stripe is used in a simple way to obtain the idea of payment experiences with credit card. /13/

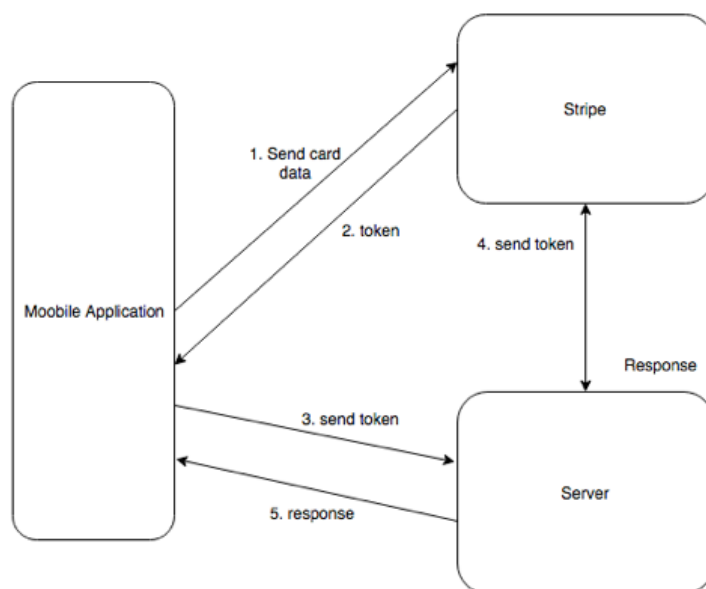


Figure 3. Stripe payment process /16/

2.11 RESTful web service

REST means representational state transfer and is an architectural style of a software. RESTful web service is for implementing such an architecture and used for client – server communication where they are both separate applications.

The RESTful API uses HTTP standard methods such as GET, POST, PUT, DELETE to process data between client and server and the results are returned in the form of JSON, XML. /14/

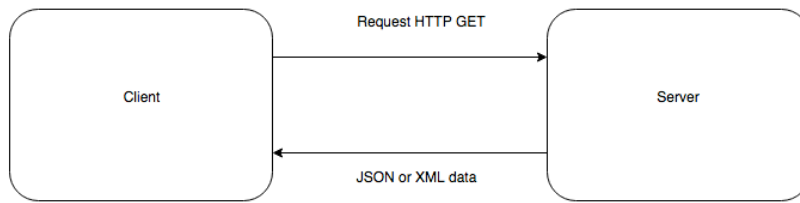


Figure 4. RESTful web service process example. /17/

3 APPLICATION DESCRIPTION

In this section, a detailed descriptions of the project and requirements are explained.

3.1 General Description

The objective of the application is to build an ordering system with separated client – server applications that communicate with each other through a web service. The application encompasses an iOS application and a web-based admin dashboard which include the following features:

- iOS application

Displays a list of restaurants and a list of products, items on the menu. Users are allowed to choose products, change the quantity and make the payment for the order.

- Admin Dashboard:

Allows users to create a restaurant, add the menu and meals for the store and check orders that have been sent from the customer. The application also allows to change the status of the orders.

3.2 Quality Function Deployment

The requirements of the project can be divided into three different parts based on the priorities: must-have, should-have and nice-to-have. Must-have features are the core functionalities of the project, while should-have and nice-to-have features are additional for future development of the application.

3.2.1 Must-have requirements

- For the iOS Client Application:
 - Searching for the name of store and restaurant.
 - Displaying the list of products and restaurants.
 - Adding products to the shopping cart, calculating the total price for the payment based on quantity and price for a single item.

- Adding customer information.
- Status of orders that have been made before.
- For the admin dashboard application:
 - Signing in/ signing out of the dashboard.
 - Signing up for registering a new restaurant.
 - Editing account information including name, phone, address and logo of the restaurant.
 - Adding products and meals to the menu with name, description, price and image of product.
 - Editing existing product on the menu.
 - Checking orders that have been sent from the client with the detail information of customers: name, address, total price and created time.
 - Changing status of the order which can be displayed to the client side.

3.2.2 Should-have requirements

- For the iOS application:
 - Logging in with a social media account
 - Making payment with a test credit card
 - Displaying the address of the customer on Google map
 - Taking user information from social media account and sending to the server when making an order
- For the admin dashboard:
 - Signing in with Facebook

3.2.3 Nice-to-have requirements

- For the iOS application:
 - Allowing the user to register with email, encrypting credential token manually and verifying the user by sending an email to the given address.
 - Chat bot or contact box for communication between users and the stores.
 - Showing recent search of restaurant and products.
 - Responsive UI.

- Restaurant rating system.
- Filtering different types of cuisine.
- For the admin dashboard:
 - Taking the data of orders and making a chart for business analysis.

3.3 Use Case Diagram

There are two different views for users, an admin dashboard application and an iOS mobile application with different user case diagrams.

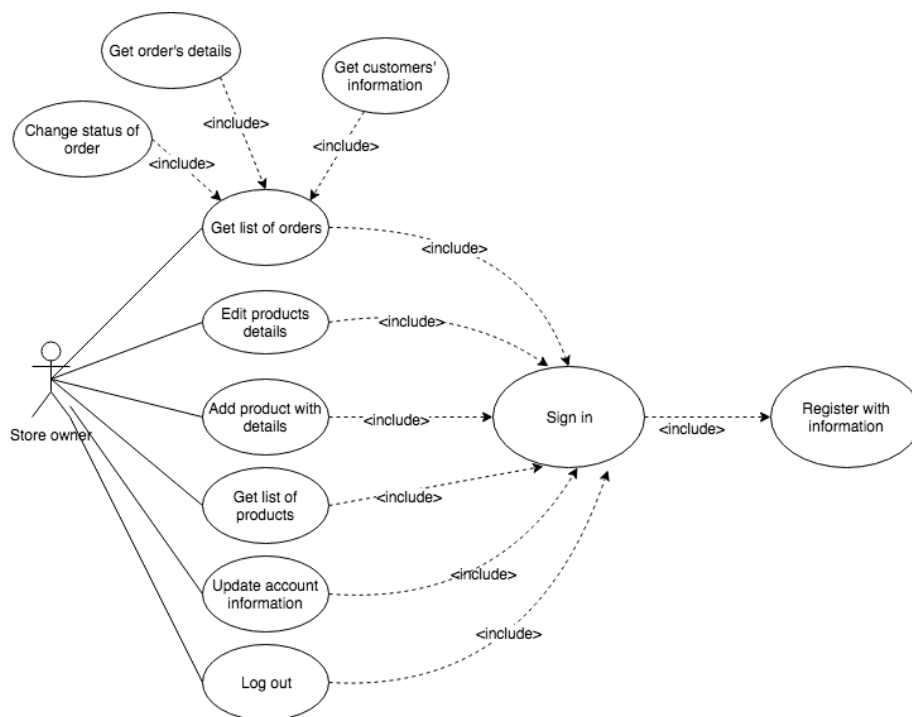


Figure 5. Functions for store owner.

From the user case diagram, the store owner can get a list of orders that are made by the customers, which includes customers' information, details of orders and changing status of orders. The owner can also add and edit product details, get a list of all published products on the menu, update information of the account and log out. All the features above require signing in from a registered account.

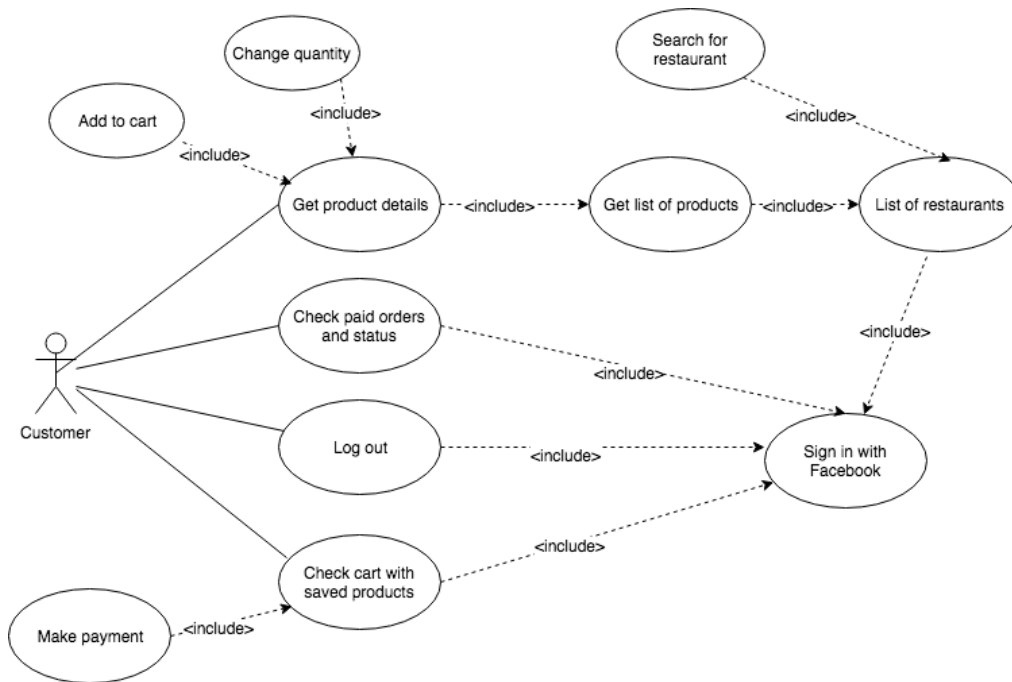


Figure 6. Functions for Customer.

As shown in this user case diagram, the customer can sign in to the application with a Facebook account, get a list of restaurants, check paid orders and their status, check cart with saved products and log out.

From the list of restaurants, the customer can search for a specific name or choose a restaurant. The process leads to a list of products that allows the customer to pick a product, change quantity and add it to the shopping cart.

In addition, the cart page shows a total price of all products that have been saved before and allows the customer to make a payment.

3.4 Class Diagram

In this project, most of the modules and functions are written based on MVT architecture. The diagram below illustrates the functions that are included in the view.py file.

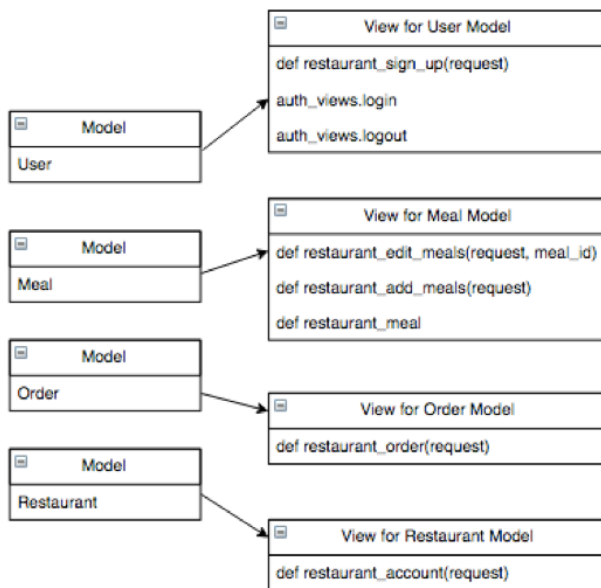


Figure 7. Diagram for View

- The View for User Model includes the method for signing up an account with the restaurant information, login and logout functions from default Django.contrib.auth
- The View for Meal Model includes functions for editing meal, adding meal and getting a list of all existing meals that have been added before.
- The View for Order Model includes a function to get orders from customers.
- The View for Restaurant Model has a method to change information of the account.

For communication between two applications, an api file is written with methods for client uses.

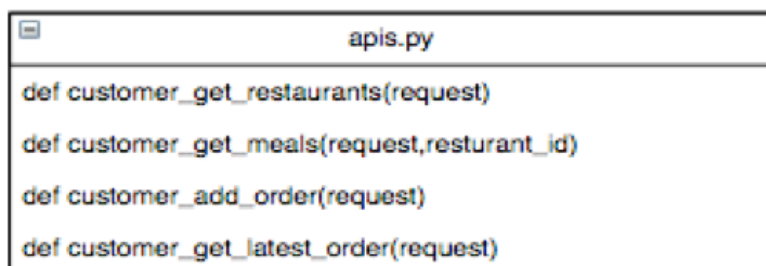
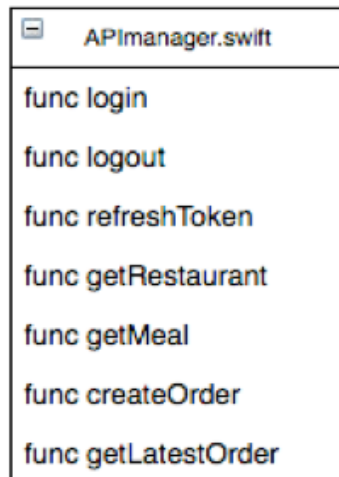


Figure 8. Class Diagram for API

- The API file includes functions for the customer to get the restaurant list, meal list, create an order and get a list of paid orders.

In the client side, the mobile application has an API manager class to handle API URL from the server, and afterward process the request and render data to the screen.



```
APImanager.swift
func login
func logout
func refreshToken
func getRestaurant
func getMeal
func createOrder
func getLatestOrder
```

Figure 9. Class Diagram for APImanager

- The API manager file includes the functions: login, logout, refreshToken, getRestaurant, getMeal, createOrder and getLatestOrder. This API manager uses Alamofire to handle HTTP request such as GET, POST with the URL from apis.py.
- These functions are also used for rendering data from the server to the mobile screen.

3.5 Sequence Diagram

3.5.1 Admin Dashboard Registration Sequence Diagram

The following sequence diagram illustrates the involvement of steps that manage a user to register and login to the admin dashboard.

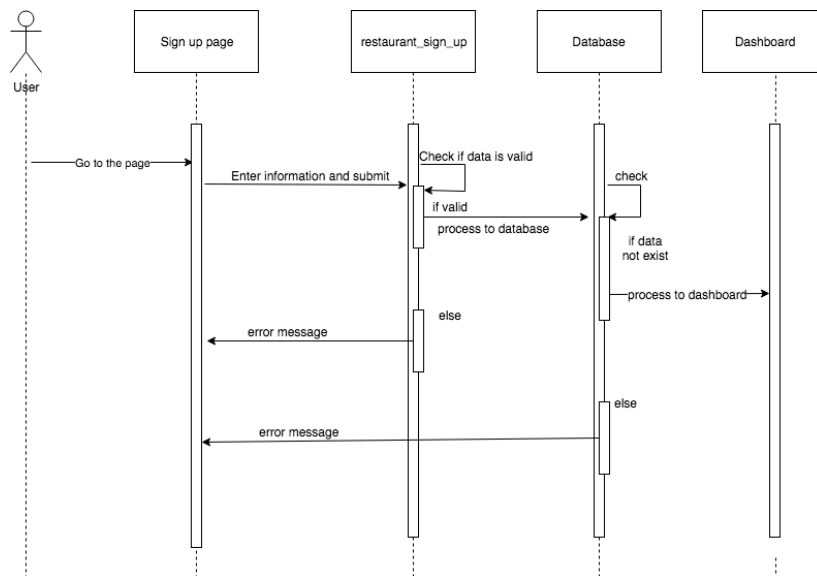


Figure 10. Admin Registration

Firstly, the user enters the information for registering an account and hit the submit button. The information of user and restaurant should match the requirements, otherwise the page will display an error message. The `restaurant_sign_up` function will be called to check if the information is valid, the application then proceeds to the next step, checking data in the database for an existing account. If there is not an existing user with the same username, the process will be completed and the user is taken to the dashboard.

3.5.2 User Login Sequence Diagram

The sequence diagram shown in Figure 11 illustrates steps that are involved in logging the process. The registered user enters username and password, the details will be checked in the database to identify the user. If username or password is not correct, an error message is generated and displayed on the browser.

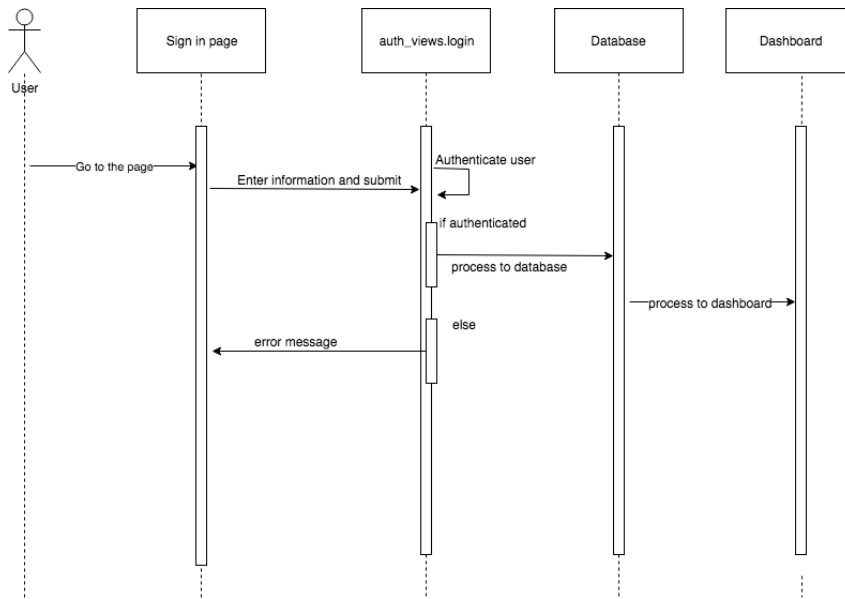


Figure 11. User Login Sequence Diagram

3.5.3 Adding meal Sequence Diagram

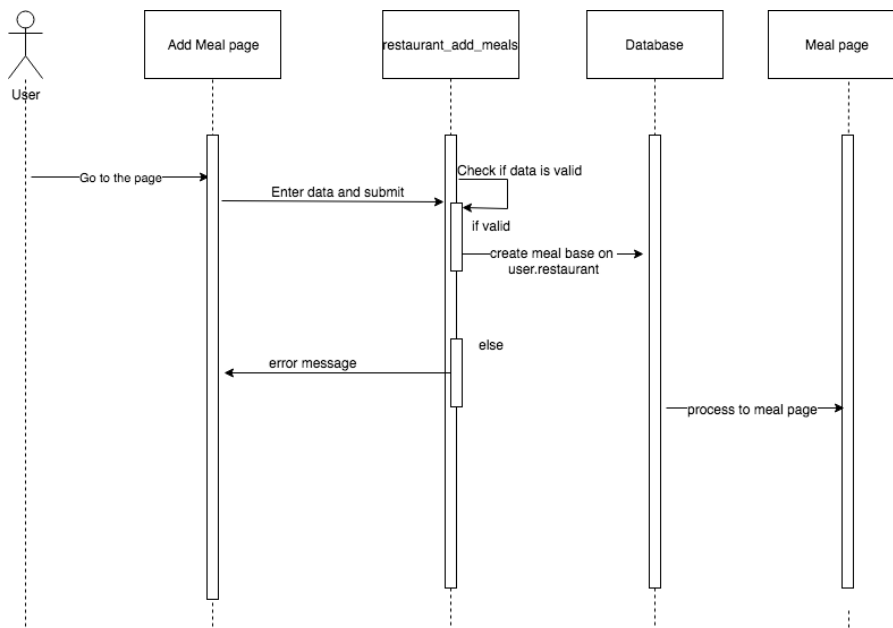


Figure 12. Adding meal sequence diagram

As shown in Figure 12, the user navigates to Add Meal Page, enters the information of the meal. The function `restaurant_add_meals` will be called to check if the data is in the right format, otherwise an error message is displayed. After checking the data, a meal is created and saved to the database based on the restaurant of current user.

3.5.4 Editing meal Sequence Diagram

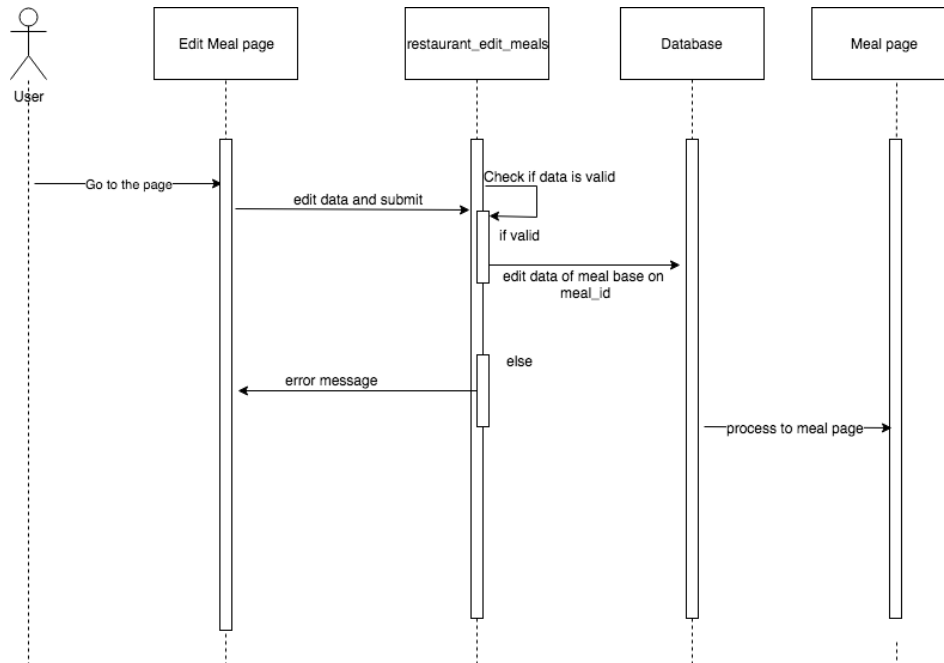


Figure 13. Editing meal sequence diagram

After choosing a meal for editing, the user can change and update information of the selected meal. The `restaurant_edit_meals` function is called to check validation of the data base on `meal_id` variable. After saving new data in the database, the user is redirected to the Meal Page with new edited information.

3.5.5 Editing account sequence diagram

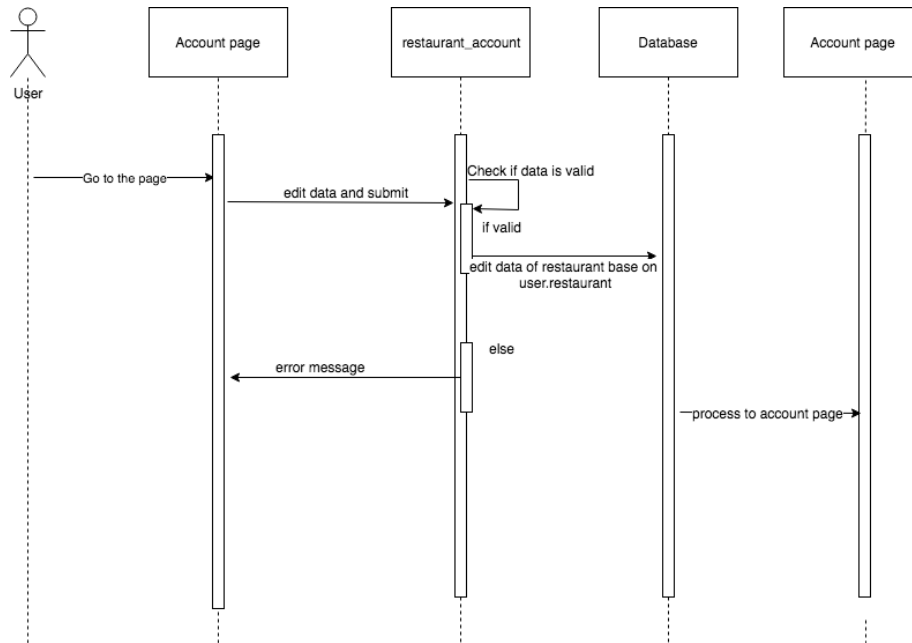


Figure 14. Editing account sequence diagram.

Figure 14 shows the steps involved when a user changes the account details. Similar to editing meal, the user changes and updates data, a function named `restaurant_account` is called to check if the data is valid, then the data will be saved to the database based on the restaurant of current user.

3.5.6 Order page sequence diagram

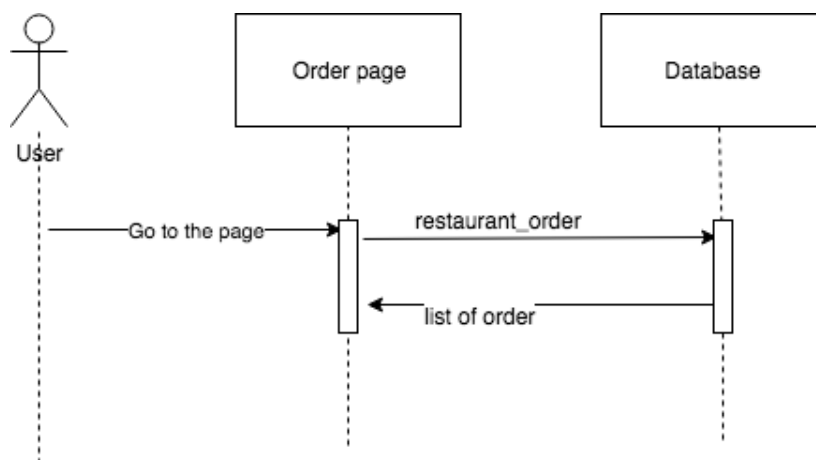


Figure 15. Order page sequence diagram

The steps to get data of orders and display orders to the browser are simple. The user navigates to the Order Page, a function `restaurant_order` is called to request all data of orders to the browser and order the data by id. The database will send the data to the Order page as a response.

3.5.7 Mobile User Login Sequence Diagram

Figure 16 below shows the steps for a user to login to the mobile application with a Facebook account.

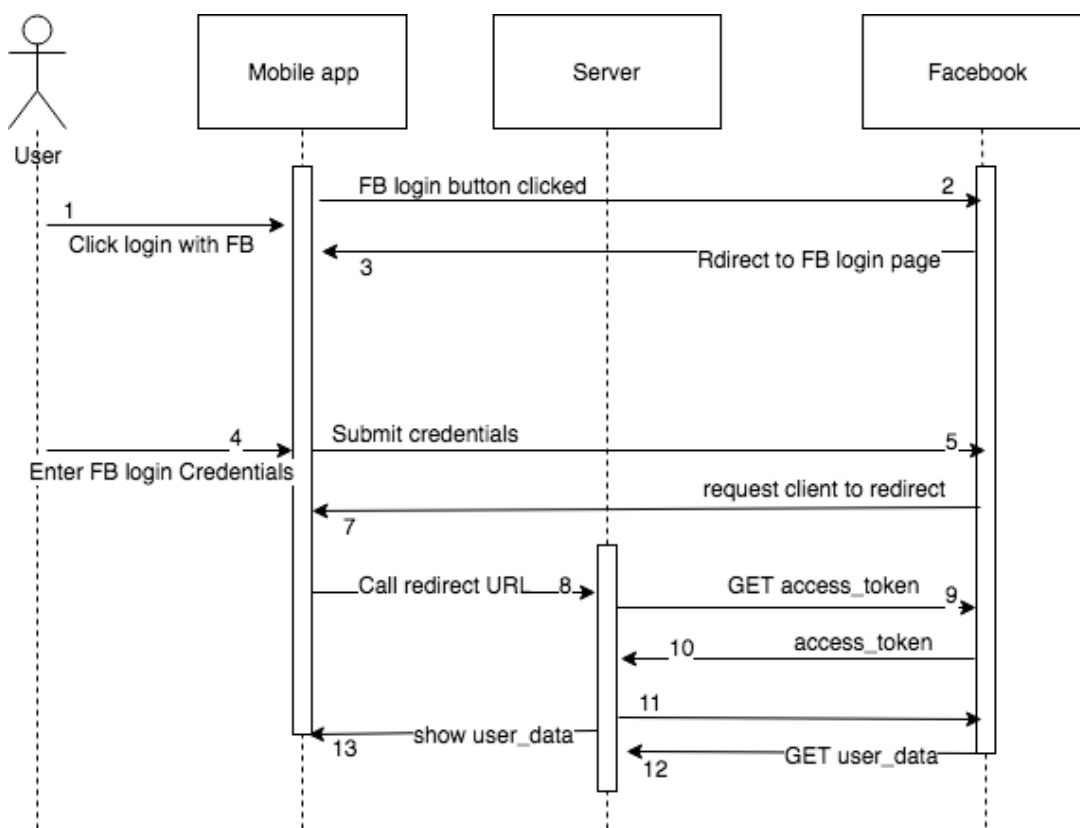


Figure 16. Facebook Login sequence diagram

First, a Login with the Facebook button is shown. The user will click that button to log-in to the mobile application. Facebook will validate the application ID and redirect the user to the Facebook login page. After entering the Facebook login credentials and submitting the data, Facebook will validate the credentials and process an access token to the Django server. This token is generated to identify the user over the application at that time. This access token will be destroyed once the user logs out of the application

and a new one will be generated when logging in. After granting an access token, the application requests user data from Facebook and fetches it to the mobile screen.

3.5.8 Getting Restaurant List On Mobile Sequence Diagram

Figure 17 below shows the steps for a user to get restaurant data from the server to the mobile application.

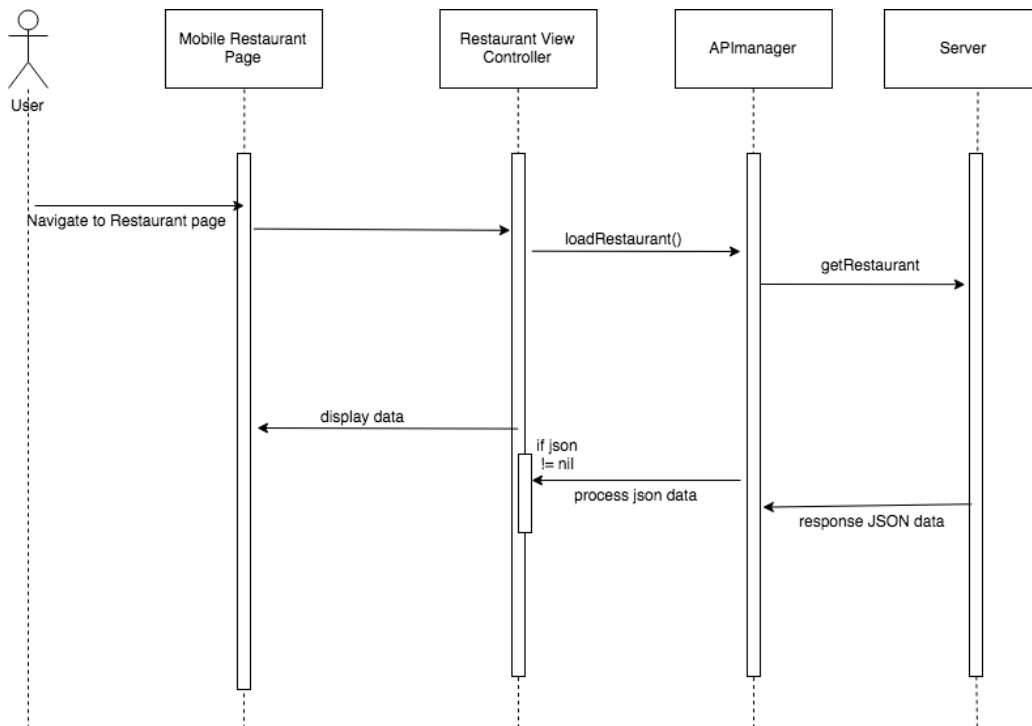


Figure 17. Getting Restaurant List on Mobile Sequence Diagram

The logged in user can navigate to the restaurant page, this page is managed by RestaurantVoewController class. The loadRestaurant function is automatically called to load all the restaurant data to the screen. This data is taken from APImanager.

The APImanager sends a get request to the server, data of restaurants are sent back as a response and processed to display on the mobile screen.

3.5.9 Creating Order Sequence Diagram

Figure 18 below illustrates the steps that are involved in creating a new order process.

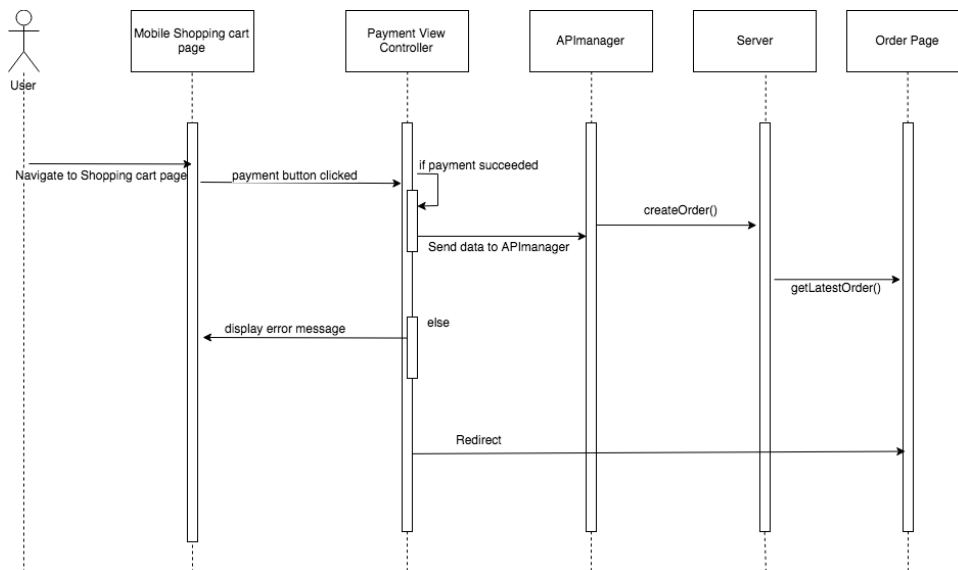


Figure 18. Creating Order Sequence Diagram

First, the user uses the payment button to make a payment for all the items in the shopping cart. The Payment View Controller will check if the payment has succeeded. If the payment fails, an error message will be displayed. After making the payment, the data of the user such as name, address and order details will be sent to the APImanager class. This data is converted to JSON type and sent to the server as a POST request. The user will be redirected to the Order Page to see status of the order that has been made.

3.5.10 Meal Page Sequence Diagram

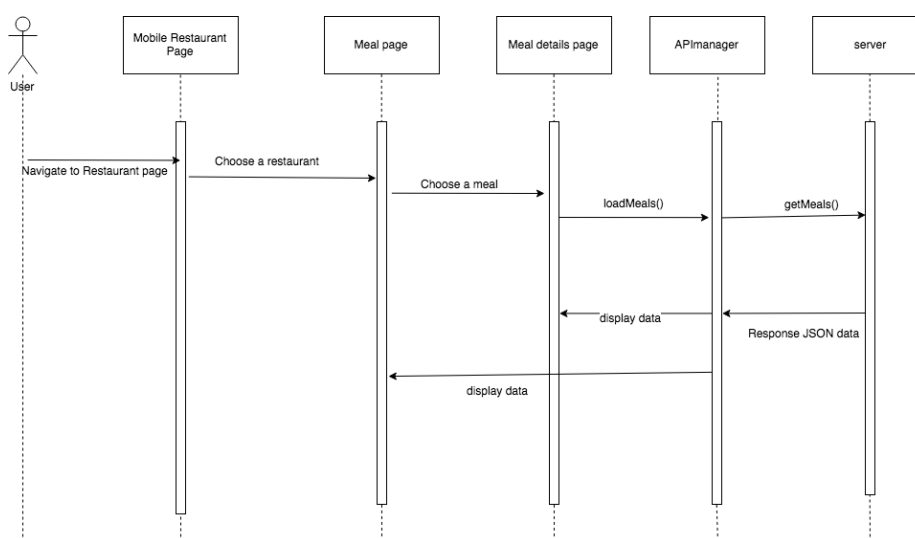


Figure 19. Meal and Meal details Sequence Diagram

First, the user needs to choose a restaurant and a meal of the selected restaurant. A function `loadMeal()` will be called to take data from API manager. The data is requested by `getMeals()` function from API manager to the server and JSON data will be sent back as a response. This JSON data is taken and fetched to the mobile screen afterwards.

3.6 Component Diagram

The component diagram below describes how components in the whole system are connected.

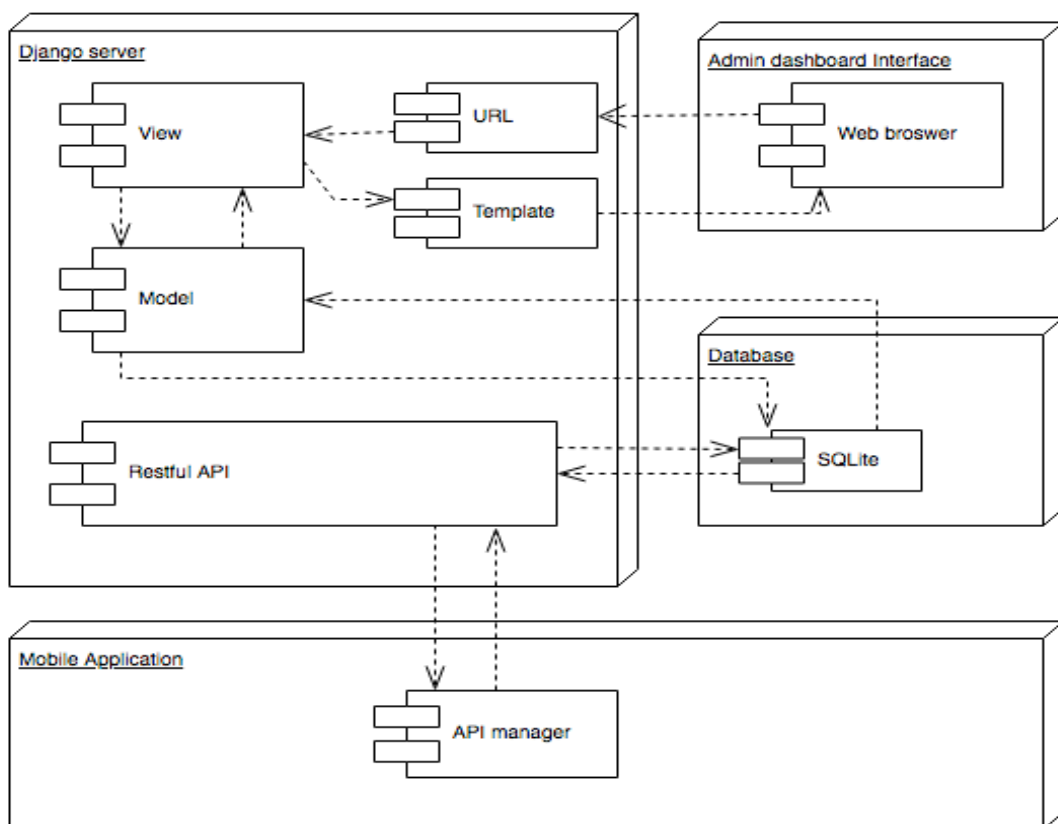


Figure 20. Component Diagram.

The application includes four major components, which are Django Server, Admin Dashboard Interface, Database and Mobile Application.

The user interfaces of the dashboard are HTML template files, which will be displayed on the web browser. Every interaction of the user with the interface on the web browser will go through URL, which locates functions that are connected between Template and

View. The function that the user interacted with will be called and then processed to Model and Database. From there data are rendered on the web browser by the reversed way.

The mobile application communicates with the server by Restful API which connected to API manager class, allows the application to render data from the database and send an HTTP request.

4 DATABASE AND GUI DESIGN

This section illustrates the design of database and GUI.

4.1 Database Design

This application used SQLite for the database, which is the default database of Django framework. As described, SQLite is a lightweight relational database management system and is the best choice for an application with light weight data.

The database includes six tables, which are User, Restaurant, Customer, Order, Meal, OrderDetails. Each table has attributes with value types and relation between the tables. The User table is default from Django. In practice, it is not necessary to write a model of the user unless of modifying.

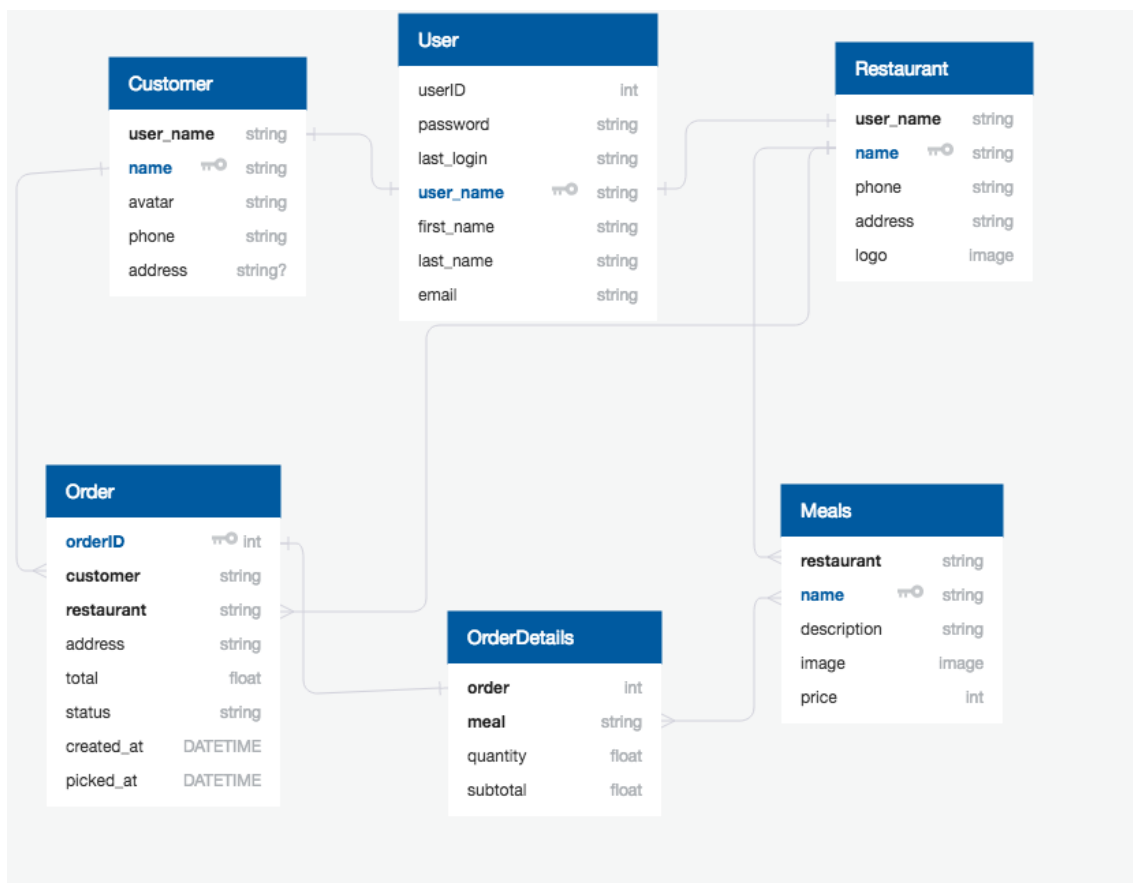


Figure 21. ER diagram.

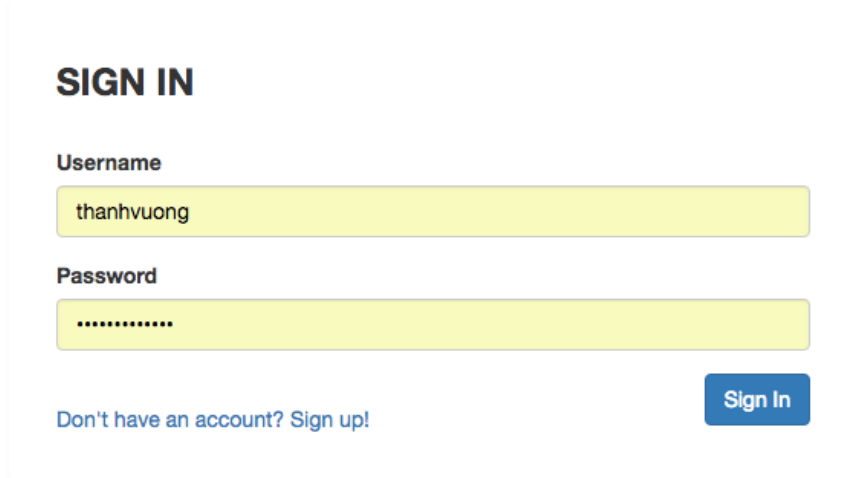
- As shown on the ER diagram, the User table has a primary key which connects to Customer table and Restaurant table³⁴ with a one-to-one relationship. This relationship is designed based on the idea that one user can have only one restaurant registered, and also one user can only be one customer at a time.
- The customer table has a one-to-many relationship with Order table, which means one customer can have many orders, but one order can only have one customer.
- The OrderDetails table has two foreign keys: order and meals. This table has a one-to-one relationship with the Order table and a many-to-many relationship with the Meals table.
- The Meal table consists of 5 attributes: restaurant, name, description, image and price. The restaurant attribute is a foreign key linked from the Restaurant table and a primary key “name” linked to the OrderDetails table.
- The Restaurant table consists of 5 attributes: user_name, name, phone, address and logo. The “name” attribute is the primary key, linked to Order table and Meals table with a one-to-many relationship. It means that one restaurant can have many orders and meals, but one order or one meal can only be in one restaurant.

Beside the tables that created the database system, there are also models from the social authentication toolkit of Django, which is automatically generated when installing the toolkit. These databases are for storing access tokens and refresh tokens sent from the client side, which is using the Facebook token for logging in.

4.2 GUI Design

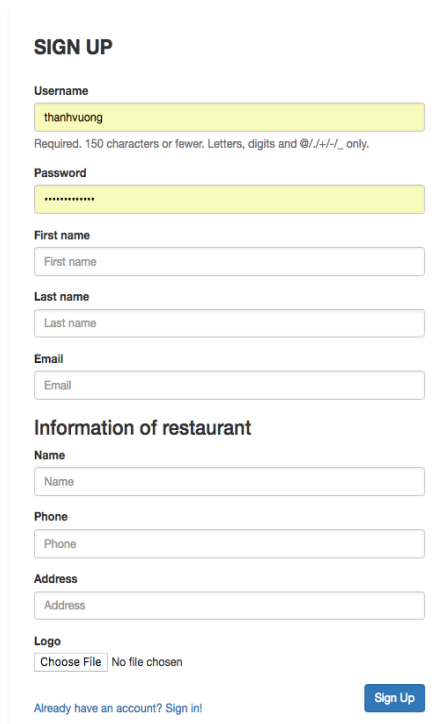
This project has two separate applications, the web based application and mobile application. The graphic user interface of the admin site is implemented with HTML, CSS, Javascript and Bootstrap for better front-end.

4.2.1 Admin Dashboard web base application



The image shows a 'SIGN IN' form. At the top, the text 'SIGN IN' is displayed in a bold, black font. Below this, there are two input fields: 'Username' and 'Password'. The 'Username' field contains the text 'thanhuong'. The 'Password' field is filled with dots. To the right of the password field is a blue button with the text 'Sign In'. Below the password field, there is a link that says 'Don't have an account? Sign up!'.

Figure 22. Sign in page for web application.

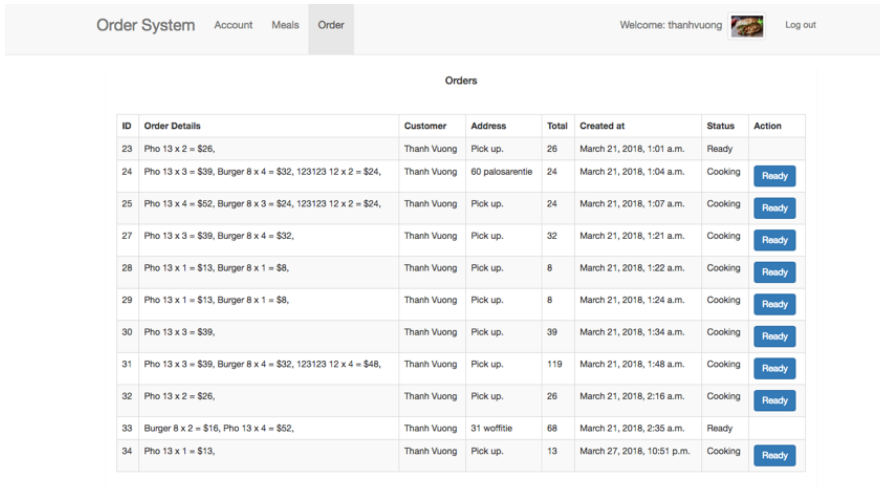


The image shows a 'SIGN UP' form. At the top, the text 'SIGN UP' is displayed in a bold, black font. Below this, there are several input fields: 'Username' (containing 'thanhuong'), 'Password' (filled with dots), 'First name', 'Last name', 'Email', 'Name', 'Phone', and 'Address'. There is also a 'Logo' section with a 'Choose File' button and the text 'No file chosen'. At the bottom right, there is a blue button with the text 'Sign Up'. At the bottom left, there is a link that says 'Already have an account? Sign in!'.

Figure 23. Sign up page.

These figures above show the “sign in” and “sign up” page for use. The design for these pages is simple. The “sign in” page has 2 input fields for user name and password and a submit button. If the user does not have any account yet, there is a link to redirect to the “sign up” page.

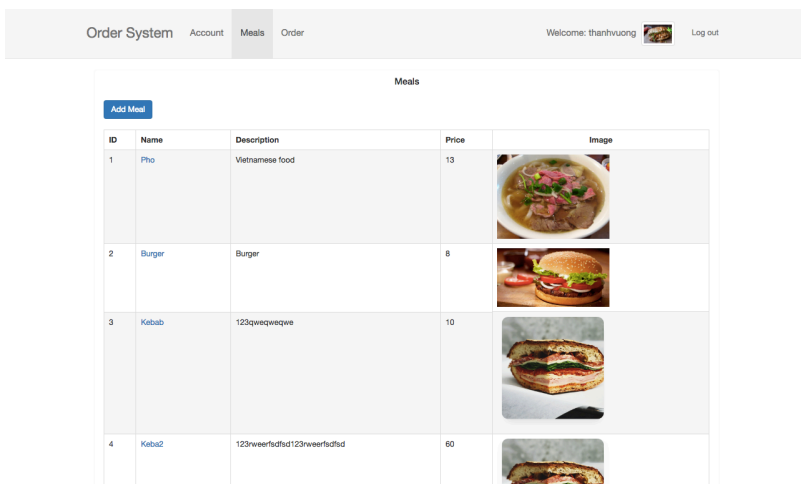
The “sign up” page has fields for users to input their information such as name, password, email and details of the restaurant with its logo. There are some required regular expressions, if the user’s inputs do not match the requirements, the page will display error messages. The email should be in the right format and all inputs should be filled.



ID	Order Details	Customer	Address	Total	Created at	Status	Action
23	Pho 13 x 2 = \$26,	Thanh Vuong	Pick up.	26	March 21, 2018, 1:01 a.m.	Ready	
24	Pho 13 x 3 = \$39, Burger 8 x 4 = \$32, 123123 12 x 2 = \$24,	Thanh Vuong	60 palosarentle	24	March 21, 2018, 1:04 a.m.	Cooking	Ready
25	Pho 13 x 4 = \$52, Burger 8 x 3 = \$24, 123123 12 x 2 = \$24,	Thanh Vuong	Pick up.	24	March 21, 2018, 1:07 a.m.	Cooking	Ready
27	Pho 13 x 3 = \$39, Burger 8 x 4 = \$32,	Thanh Vuong	Pick up.	32	March 21, 2018, 1:21 a.m.	Cooking	Ready
28	Pho 13 x 1 = \$13, Burger 8 x 1 = \$8,	Thanh Vuong	Pick up.	8	March 21, 2018, 1:22 a.m.	Cooking	Ready
29	Pho 13 x 1 = \$13, Burger 8 x 1 = \$8,	Thanh Vuong	Pick up.	8	March 21, 2018, 1:24 a.m.	Cooking	Ready
30	Pho 13 x 3 = \$39,	Thanh Vuong	Pick up.	39	March 21, 2018, 1:34 a.m.	Cooking	Ready
31	Pho 13 x 3 = \$39, Burger 8 x 4 = \$32, 123123 12 x 4 = \$48,	Thanh Vuong	Pick up.	119	March 21, 2018, 1:48 a.m.	Cooking	Ready
32	Pho 13 x 2 = \$26,	Thanh Vuong	Pick up.	26	March 21, 2018, 2:16 a.m.	Cooking	Ready
33	Burger 8 x 2 = \$16, Pho 13 x 4 = \$52,	Thanh Vuong	31 woffile	68	March 21, 2018, 2:35 a.m.	Ready	
34	Pho 13 x 1 = \$13,	Thanh Vuong	Pick up.	13	March 27, 2018, 10:51 p.m.	Cooking	Ready

Figure 24. Order page.

After signing in to the application, the user will automatically be redirected to the Order page, where all the orders from the client to the user’s restaurant are shown. The table that shows details of an order has fields such as ID, name and quantity, customer’s name, address and so on. The total price is calculated by the system. There is also an action button for the operator to change the status of orders, this status can be notified on the client screen.






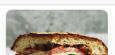
ID	Name	Description	Price	Image
1	Pho	Vietnamese food	13	
2	Burger	Burger	8	
3	Kebab	123qweqweqe	10	
4	Kebab2	123rweeefdsfd123rweeefdsfd	80	

Figure 25. Meal page.

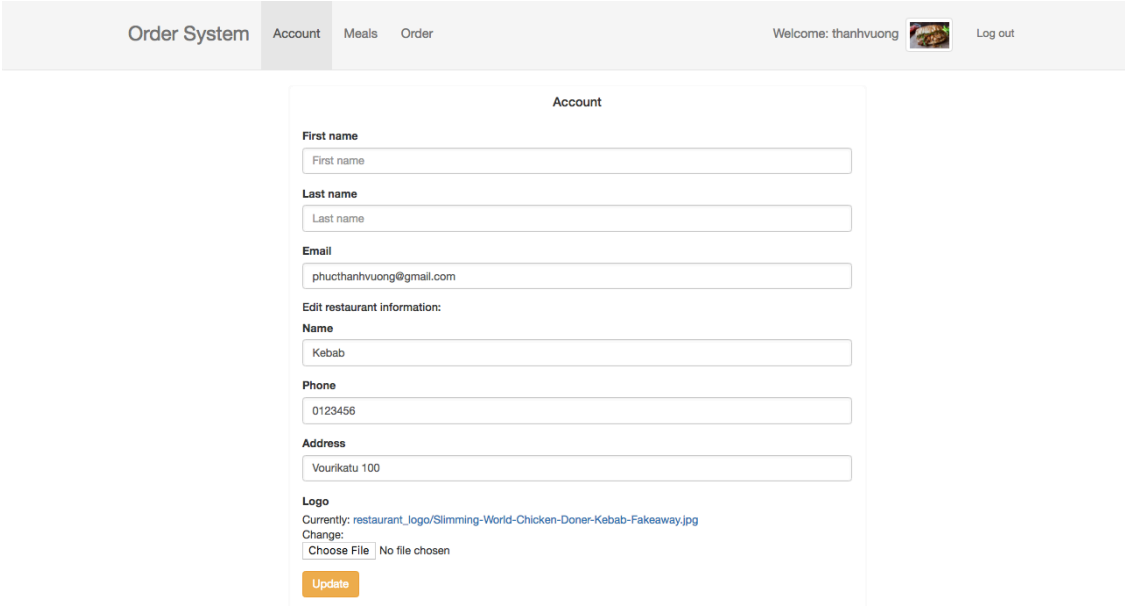
Figure 25 shows the meal page, where a list of all the meals on the menu are displayed. Each name of a meal is a link to an edit page, the user can select a meal that needs to be edited and click on the name, it will then redirect the user to an edit page. There is also a blue button for adding a new meal to the menu.


The table includes the ID of the meal, name, short description, price and picture of the meal. This information is shown on the mobile application as well.

Figure 26. Add meal page.

Figure 27. Edit meal page.

The “Add meal page” and “Edit meal page” are similar to each other. Both pages have the same input fields such as Name, Description, Image and price of the meal, but “Edit page” has information already for editing while “Add page” has only blank fields.



Order System Account Meals Order Welcome: thanhvuong  Log out

Account

First name
First name

Last name
Last name

Email
phucthanhvuong@gmail.com

Edit restaurant information:

Name
Kebab

Phone
0123456

Address
Vourikatu 100

Logo
Currently: restaurant_logo/Slimming-World-Chicken-Doner-Kebab-Fakeaway.jpg
Change:
Choose File No file chosen
Update

Figure 28. Account page.

Figure 28 shows the account page, where the user can select to modify the account information. The form is filled with information when signing up, the user can modify it and click the update button to save the changes.

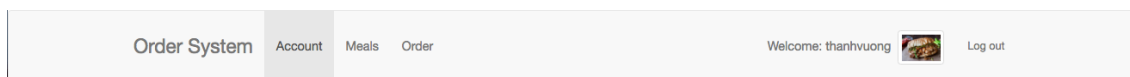


Figure 29. Navigation Bar

The navigation is designed to easily detect which page the user is using. It has three pages to navigate, an active page has a hover color that is different to others. On the right side, there are username, logo of restaurant and log out button. The brand name “Order system” will redirect the user to Order page when clicked on.

4.2.2 GUI of Mobile Application

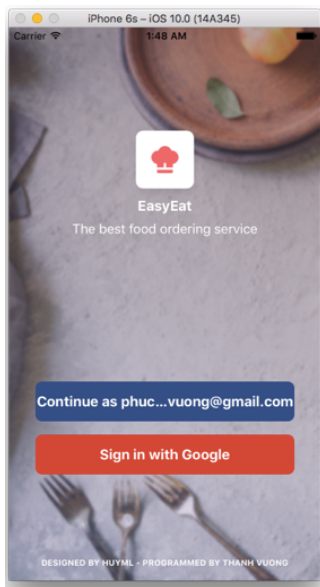


Figure 30. Log in page.

The log in page of the mobile application is designed with two buttons with a large background image. The blue button allows the user to login with their Facebook account, the application will receive their information and forward it to the server when user makes any order. The “Sign in with Google” button is not having any function at the moment and will be implemented in the future development.

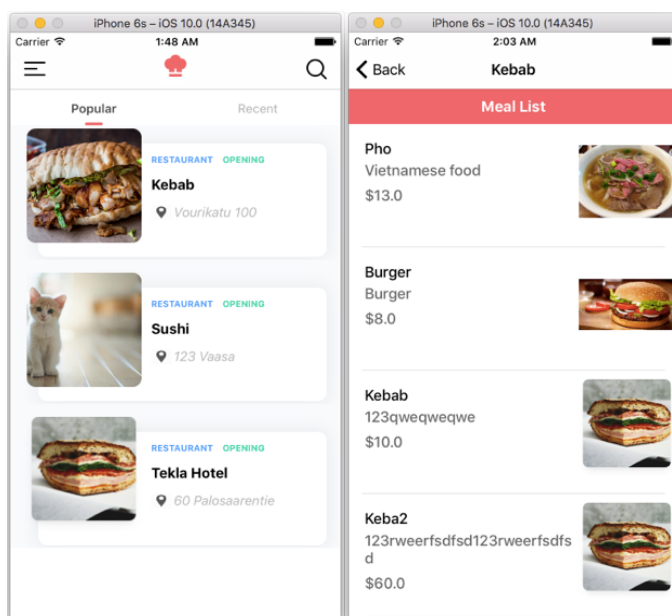


Figure 31, Figure 32. Restaurant list and Meal list.

After signing in, the application will show a list of restaurants for the user to select as shown in Figure 31. This page has two buttons on the navigation bar. On the left is a side bar (Figure 33) and on the right is a search button for search a specific name of a restaurant.

When one restaurant has been chosen, the application will segue to a Meal list. This meal list shows all the products on the menu of chosen restaurant with details such as name, description and price.

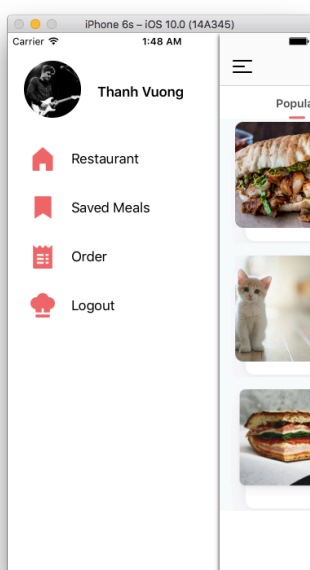


Figure 33. Side bar.

The side bar will appear when the user clicks on the right button on navigation bar. This side bar includes the profile image and user name taken from the user's Facebook account. There are four buttons: Restaurant, Saved Meals, Order and Log out. The restaurant button will redirect the user to to restaurant list page. "Saved meal" is linked to the shopping cart, where the user can see which meals they have added to the cart. "Order" is linked to a history page where it shows the order that has been paid and its status.

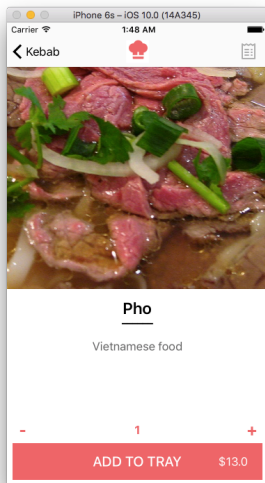


Figure 34. Meal details page.

After choosing a meal, the meal details page will appear. This page includes a closer look at the meal, name, description and price. There is a button for the user to modify the quantity. The system will calculate a total price and show it on the bottom area. There is also a red button to add the meal to the cart.

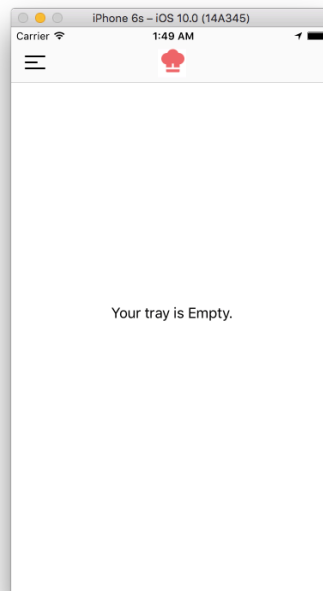
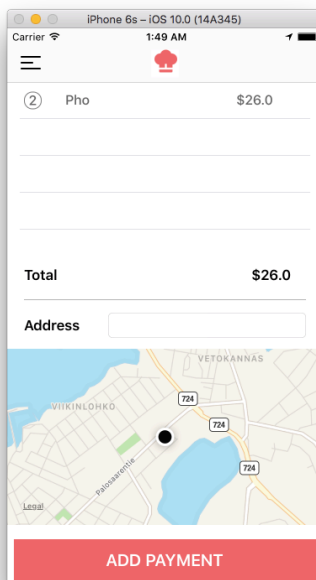


Figure 35, Figure 36. Shopping cart, Empty cart.

The shopping cart page shows all the products that the user has added to the cart on a list. Each item on the list has a number of quantity, name of meal and price for that item. The “total” label is the total price for all items after calculated and an input field for the address. The location of user can be tracked after filling in the fields if user wants the order to be delivered. A red button on the bottom will segue the user to the payment page.

In Figure 36, the app displays a message that tells the user that the shopping cart is empty.

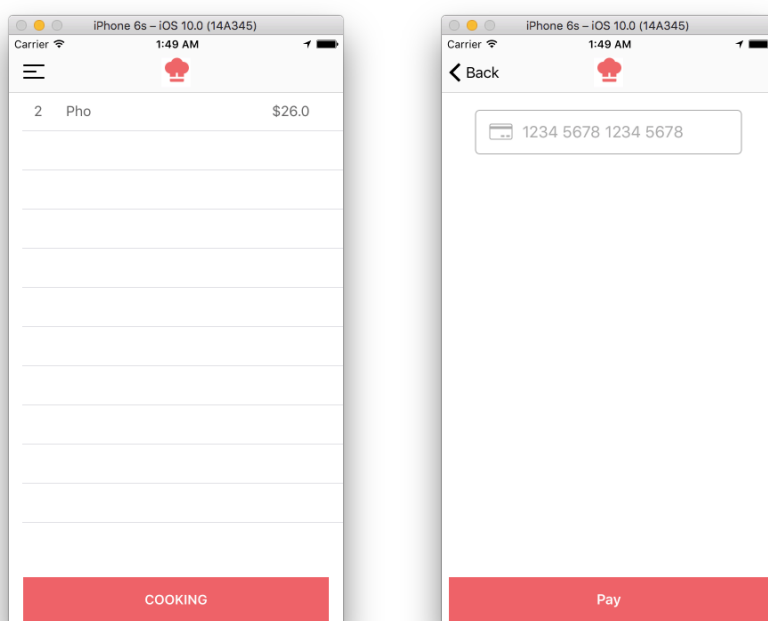


Figure 37, Figure 38. Order page and Payment page.

The order page displays a list of all the orders that the user just made and their status. This status can be changed on the admin dashboard side to notify the client when the order is ready.

Figure 38 shows the payment page, this design is taken from Stripe SDK. For the result, this page is very simple and working perfectly.

5 IMPLEMENTATION

In this section, the implementation steps of the project are discussed.

The structure of the application is described and primary methods, modules are explained afterward.

5.1 General description of implementation

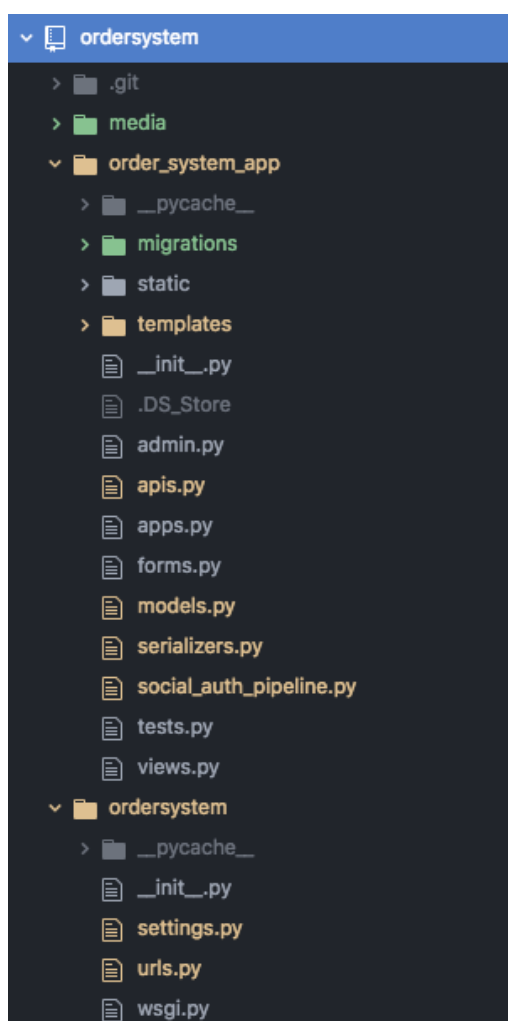


Figure 39. Structure of the admin dashboard application.

Figure 39 shows the structure diagram of the admin dashboard application. “Ordersystem” is the name of the main directory, which consists of three other child directories: “media”, “order_system_app” and “ordersystem”. The “ordersystem” is generated automatically when creating a Django project, this directory includes a settings file to handle all the applications in the project. “order_system_app” is an application where

functions and templates are written. “media” directory is for storing files such as images.

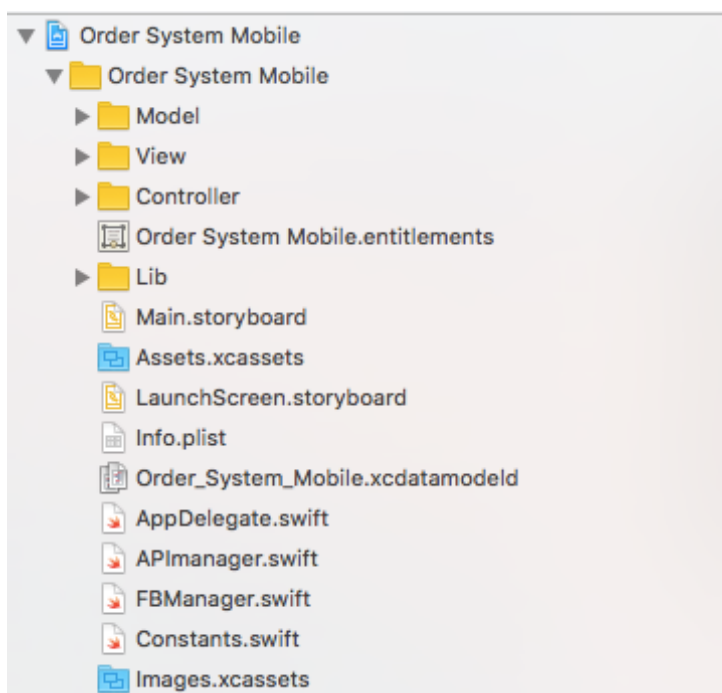


Figure 40. Structure of the mobile application.

The structure of the mobile application has three child folders: Model, View, Controller and application setting swift files. There is also an assets directory to store images.

5.2 Implementation of different parts

The implementation is divided into two parts for two different application.

5.2.1 Implementation of Admin Dashboard web based application

There are three main files that includes the all primary functions and connections for the application: views.py, apis.py and urls.py. These files are explained in this section.

- Urls.py

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.home, name='home'),

    #restaurant urls
```

```

url(r'^restaurant/sign-in/$', auth_views.login,
{'template_name': 'restaurant/sign_in.html'}, name='restaurant-sign-in'),
url(r'^restaurant/sign-out/$', auth_views.logout, {'next_page': "/"}, name='restaurant-sign-
out'),
#Sign in takes the function named login in views.py and redirect the page to sign_in.html
which called restaurant-sign-in
#for signing out it goes to the home page so it has only /
#name of url for calling it in put in <a> tag, example <a href = url 'restaurant-sign-out'
url(r'^restaurant/sign-up/$', views.restaurant_sign_up, name='restaurant-sign-up'),
url(r'^restaurant/$', views.restaurant_home, name='restaurant-home'),

url(r'^restaurant/account/$', views.restaurant_account, name='restaurant-account'),
url(r'^restaurant/meals/$', views.restaurant_meals, name='restaurant-meals'),
url(r'^restaurant/meals/add$', views.restaurant_add_meals, name='restaurant-add-meals'),
url(r'^restaurant/meals/edit/(?P<meal_id>\d+)/$',
views.restaurant_edit_meals, name='restaurant-edit-meals'),
url(r'^restaurant/order/$', views.restaurant_order, name='restaurant-order'),

#sign in sign out sign up with rest api
url(r'^auth_fb/', include('rest_framework_social_oauth2.urls')),

#/convert-token (sign in/sign up)
#/revoke-token(sign-out)

#api urls
url(r'^api/customer/restaurants/$', apis.customer_get_restaurants),
url(r'^api/customer/meals/(?P<restaurant_id>\d+)/$', apis.customer_get_meals), #\d is for
number
url(r'^api/customer/order/add/$', apis.customer_add_order),
url(r'^api/customer/order/latest/$', apis.customer_get_latest_order),
] + static(settings.MEDIA_URL, document_root = settings.MEDIA_ROOT)

#for uploading the images need to use static

```

Code Snippet 1. Setting up urls for functions.

The main purpose of this module is mapping url path expressions to the project functions in views.py and apis.py. There are different parts in an url when declaring, url for the link and the location of functions.

For example an url: `url(r'^api/customer/restaurants/$', apis.customer_get_restaurants)`. A request to `api/customer/restaurants/` would call the function that has been mapped to the url, in this case is `customer_get_restaurants` in apis.py. Some regular expression is utilized in some url for converting syntax such as `(?P<meal_id>\d+)` to take integer value.

- Forms.py

```

class UserForm(forms.ModelForm):
    email = forms.CharField(max_length=100, required = True)
    password = forms.CharField(widget=forms.PasswordInput())
    class Meta:
        model = User
        fields = ("username", "password", "first_name", "last_name", "email")

```

```

class RestaurantForm(forms.ModelForm):
    class Meta:
        model = Restaurant
        fields = ("name", "phone", "address", "logo")

class UserFormEdit(forms.ModelForm):
    email = forms.CharField(max_length=100, required = True)
    class Meta:
        model = User
        fields = ("first_name", "last_name", "email")

class MealsForm(forms.ModelForm):
    class Meta:
        model = Meals
        exclude = ("restaurant",) #except for restaurant, no need to add

```

Code Snippet 2. Creating model forms.

The main purpose of model forms is for the database-driven application, so the information that submitted by these forms will be taken to the models and will then be sent to the database. These model forms are written in a separated module for clarity and will be used in further development.

- Sign up function

```

def restaurant_sign_up(request):
    user_form = UserForm()
    restaurant_form = RestaurantForm()

    #after user click sign up button, run these function:
    if request.method == "POST":

        #get data from forms
        user_form = UserForm(request.POST)
        restaurant_form = RestaurantForm(request.POST, request.FILES)

        #check if information is valid
        if user_form.is_valid() and restaurant_form.is_valid():
            new_user = User.objects.create_user(**user_form.cleaned_data) #create new user
            object
            new_restaurant = restaurant_form.save(commit=False) #create new
            restaurant, just memory not data yet
            new_restaurant.user = new_user #assign user to
            restaurant
            new_restaurant.save() #save to database

            login(request, authenticate(
                username = user_form.cleaned_data["username"], #login
                password = user_form.cleaned_data["password"]
            ))

            return redirect(restaurant_home)

return render(request, 'restaurant/sign_up.html', {
    "user_form": user_form,
    "restaurant_form": restaurant_form

```

```
})
```

Code Snippet 3. Sign up function.

The login and logout functions are handled by Django, so only the sign up function is written manually. Firstly, `user_form` and `restaurant_form` are initialized from model forms. After the user has filled all the information and hit the sign up button, the function will be called with a POST request and take the data from the forms. If the information is valid, a user object and a restaurant object will be created and saved to the database. Lastly, the function will process logging in with the submitted data.

- Modify user's account

```
@login_required(login_url='/restaurant/sign-in/')
def restaurant_account(request):
    user_form = UserFormEdit(instance = request.user)
    restaurant_form =RestaurantForm(instance =request.user.restaurant)

    if request.method == 'POST':
        user_form = UserFormEdit(request.POST, instance = request.user)
        restaurant_form =RestaurantForm(request.POST,request.FILES, instance =
request.user.restaurant)

        if user_form.is_valid() and restaurant_form.is_valid():
            user_form.save()
            restaurant_form.save()

    return render(request, 'restaurant/account.html',{
        "user_form": user_form,
        "restaurant_form": restaurant_form
    })
```

Code Snippet 4. Modify function for user's account.

At first, the `user_form` and `restaurant_form` are initialized with data that is already saved in the database. If a POST request is sent from the user, the data will be modified and saved into the database again.

- Getting list of orders and meals.

```
@login_required(login_url='/restaurant/sign-in/')
def restaurant_meals(request):
    meals = Meals.objects.filter(restaurant = request.user.restaurant).order_by("id") #-id to
revert order
    return render(request, 'restaurant/meals.html',{'meals':meals})
#to display list of meal on meal page

@login_required(login_url='/restaurant/sign-in/')
def restaurant_order(request):
```

```

orders = Order.objects.filter(restaurant = request.user.restaurant).order_by("id") #select
restaurant of current user

if request.method == "POST":
    order = Order.objects.get(id = request.POST["id"],restaurant = request.user.restaurant)
    if order.status == Order.COOKING:
        order.status = Order.READY
        order.save()

return render(request, 'restaurant/order.html',{'orders':orders})

```

Code Snippet 5. Function for getting data of orders and meals.

Getting orders function and getting meals function are quite similar and simple, it is only necessary to request the order object and meal object from the models and sorting with an order by id. The “restaurant_order” function has also a method to change status of the order. If the user click a button to change the status, a POST request will be made, the system will process an order with a specific id and change its status from COOKING to READY.

- Adding new item to menu.

```

def restaurant_add_meals(request):
    meals_form = MealsForm()

    if request.method == "POST":
        meals_form = MealsForm(request.POST,request.FILES)
        if meals_form.is_valid():
            meals = meals_form.save(commit=False)
            meals.restaurant = request.user.restaurant
            meals.save()
            return redirect(restaurant_meals)

    return render(request, 'restaurant/add_meals.html',{
        "meals_form" : meals_form
    })

```

Code Snippet 6. Function for adding new meal.

For this function, first the model form for meals will be called. Then the function will process with a POST request, taking the data submitted by user, checking if it is valid then saving it to the database. The meal has “restaurant” variable as a foreign key, so the system can determine which restaurant the meal belongs to.

After creating a new meal successfully, the user will be redirected to the meal list page.

- Editing meal on the menu

```
@login_required(login_url='/restaurant/sign-in/')
def restaurant_edit_meals(request,meal_id):
    meals_form = MealsForm(instance=Meals.objects.get(id=meal_id)) # edit the meal based on id

    if request.method == "POST":
        meals_form =
MealsForm(request.POST,request.FILES,instance=Meals.objects.get(id=meal_id))
        if meals_form.is_valid():
            meals_form.save()
            return redirect(restaurant_meals)

    return render(request, 'restaurant/edit_meals.html',{
        "meals_form" : meals_form
    })
```

Code Snippet 7. Function for editing meal.

The function first initiated a model form with data from the database, based on the id of the chosen meal. After the user hit “Edit” button, a POST request will be sent, the function then processes to modify the data with a new one and saves it to the database if the form is valid.

- Serializer.py

```
class RestaurantSerializer(serializers.ModelSerializer):
    logo = serializers.SerializerMethodField()
    def get_logo(self,restaurant):
        request = self.context.get('request') #include also the domain not just the path
        logo_url = restaurant.logo.url

        return request.build_absolute_uri(logo_url)

    class Meta:
        model =Restaurant
        fields = ("id","name","phone","address","logo")

class MealsSerializer(serializers.ModelSerializer):
    image = serializers.SerializerMethodField()
    def get_image(self,meals):
        request = self.context.get('request') #include also the domain not just the path
        image_url = meals.image.url

        return request.build_absolute_uri(image_url)

    class Meta:
        model = Meals
        fields = ("id","name","description","image","price")

## ORDER SERIALIZER, need to transfer info from the database to json and pass it to the customer
## need to create so many Serializer to create JSON for each variable and pass it to
OrderSerializer
## otherwise it wont work
class OrderCustomerSerializer(serializers.ModelSerializer):
    name = serializers.ReadOnlyField(source = "user.get_full_name")

    class Meta:
        model = Customer
        fields = ("id","name","avatar","phone","address")
```

```

class OrderRestaurantSerializer(serializers.ModelSerializer):

    class Meta:
        model = Restaurant
        fields = ("id", "name", "phone", "address")

class OrderMealsSerializer(serializers.ModelSerializer):
    class Meta:
        model = Meals
        fields = ("id", "name", "price")

class OrderDetailsSerializer(serializers.ModelSerializer):
    meal = OrderMealsSerializer()
    class Meta:
        model = OrderDetails
        fields = ("id", "meal", "quantity", "sub_total")

class OrderSerializer(serializers.ModelSerializer):
    customer = OrderCustomerSerializer()
    restaurant = OrderRestaurantSerializer()
    order_details = OrderDetailsSerializer(many = True)
    status = serializers.ReadOnlyField(source="get_status_display")

    class Meta:
        model = Order
        fields= ("id", "customer", "restaurant", "order_details", "total", "status", "address")

```

Code Snippet 8. Serializer.

These serializers are declared to provide a useful shortcut that deals with model instances and queriesets, so complex data from the database can be easily rendered into JSON. For this step, serializers are only created to map with the data in the database, the serializing objects step will be initiated.

- APIS

```

@csrf_exempt
def customer_add_order(request):
    if request.method == "POST":
        #the access_token needed to create order based on user
        access_token = AccessToken.objects.get(token = request.POST.get("access_token"),
            expires__gt = timezone.now())

        #defining user
        customer = access_token.user.customer

        #token for stripe
        stripe_token =request.POST["stripe_token"]

        #check address, but not necessary, if user doesnt give address ~> it means it will be
        picked up

        #load orderdetails from json
        order_details = json.loads(request.POST["order_details"])

        order_total = 0
        for meal in order_details:
            order_total += Meals.objects.get(id = meal["meal_id"]).price*meal["quantity"]

        if len(order_details) > 0:

```

```

#Create a charge: this will charge customer card
charge = stripe.Charge.create(
    amount = order_total * 100, #amount in cents
    currency = "EUR",
    source = stripe_token,
    description = "Order system"
)
if charge.status != "failed":

    order = Order.objects.create(
        customer = customer,
        restaurant_id = request.POST["restaurant_id"],
        total = order_total,
        status= Order.COOKING,
        address = request.POST["address"]
    )

    for meal in order_details:
        OrderDetails.objects.create(
            order = order,
            meal_id = meal["meal_id"],
            quantity = meal["quantity"],

        )
    return JsonResponse({"status":"success"})
else:
    return JsonResponse({"status":"failed","error":"Could not complete payment."})

def customer_get_restaurants(request):
    restaurants = RestaurantSerializer(
        Restaurant.objects.all().order_by("id"),
        many=True,
        context = {"request": request}
    ).data
    return JsonResponse({"restaurants":restaurants})

def customer_get_meals(request,restaurant_id): #restaurant_id from the url
    meals = MealsSerializer(
        Meals.objects.filter(restaurant_id = restaurant_id).order_by("id"),
        many=True,
        context = {"request": request}
    ).data
    return JsonResponse({"meals":meals})
def customer_get_latest_order(request):
    access_token = AccessToken.objects.get(token = request.GET.get("access_token"),
    expires__gt = timezone.now())
    customer = access_token.user.customer
    order = OrderSerializer(Order.objects.filter(customer = customer).last()).data
    return JsonResponse({"order":order})

```

Code Snippet 9. Creating API.

These functions used the serializers from Serializer.py to convert data to JSON type. The JSON data will be used in the client side for taking data from the server and fetching it to the mobile screen.

After the user on the client side has made an order, the mobile application will send a POST request to the server, as shown in “customer_add_order” function. This function will get access_token, restaurant_id, address, order_details, stripe_token and status from the mobile client as parameters, then send it to the server with a POST request.

The system calculates the total price, takes quantities and the id of meals for the order details, send it to the server with the customer's information. There is also a method to charge money from the client's card.

5.2.2 Mobile application

In this section, the main modules of the mobile application are described.

- Getting restaurant list

```
//API for restaurant list
func getRestaurants(completionHandler: @escaping (JSON) -> Void){
    let path = "api/customer/restaurants"
    let url = baseUrl?.appendingPathComponent(path)

    refreshToken{
        Alamofire.request(url!, method: .get, parameters: nil, encoding: URLEncoding(),
headers: nil).responseJSON(completionHandler: { (response) in
            switch response.result{
                case .success(let value):
                    let jsonData = JSON(value)
                    completionHandler(jsonData)
                    break
                case .failure:
                    completionHandler(nil)
                    break
            }
        })
    }
}

func loadRestaurants(){
    APIManager.shared.getRestaurants { (json) in
        if json != nil {
            self.restaurants = []
            if let listRestaurant = json["restaurants"].array{
                for item in listRestaurant{
                    let restaurant = Restaurant(json: item)
                    self.restaurants.append(restaurant)
                }
            }
            self.tableViewRestaurant.reloadData()
        }
    }
}

func loadImage(imageView: UIImageView, urlString: String){
    let imgURL: URL = URL(string: urlString)!

    URLSession.shared.dataTask(with: imgURL){ (data,response,error) in
        guard let data = data, error == nil else { return}

        DispatchQueue.main.async(execute: {
            imageView.image = UIImage(data: data)
        })
    }
}
```

```
    }.resume()
  }
```

Code Snippet 10. Getting restaurant list function.

The function “getRestaurants” takes the API url path and handles the JSON data from the server with a GET request. The JSON data after getting from the server are put into an arraylist and then displayed to the table list.

- Searching restaurant function

```
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {
    filterRestaurants = self.restaurants.filter({(res: Restaurant) -> Bool in
        return res.name?.lowercased().range(of: searchText.lowercased()) != nil
    })
    self.tableViewRestaurant.reloadData()
}

func createSearchBar(){
    searchBar.showsCancelButton = false
    searchBar.delegate = self

    self.navigationItem.titleView = searchBar
}
```

Code Snippet 11. Searching restaurant function.

When the search icon is clicked, the “createSearchBar” will be called and create a search bar in a programmatically way. This search bar has a method to filter searching text to match with the restaurant name. The list of restaurant will also be changed when searching for a restaurant.

- Getting meal list

```
//api for getting list of meal
func getMeals(restaurantId: Int, completionHandler: @escaping(JSON) -> Void){
    let path = "api/customer/meals/\(restaurantId)"
    let url = baseUrl?.appendingPathComponent(path)

    refreshToken{
        Alamofire.request(url!, method: .get, parameters: nil, encoding: URLEncoding(),
headers: nil).responseJSON(completionHandler: { (response) in
            switch response.result{
                case .success(let value):
                    let jsonData = JSON(value)
                    completionHandler(jsonData)
                    break
                case .failure:
                    completionHandler(nil)
                    break
            }
        })
    })
}
```

```

    }
    func loadMeals(){
        if let restaurantId = restaurant?.id{
            APImanager.shared.getMeals(restaurantId: restaurantId, completionHandler: { (json)
in
                if json != nil{
                    self.meals = []
                    if let tempMeals = json["meals"].array{
                        for item in tempMeals{
                            let meal = Meal(json:item)
                            self.meals.append(meal)
                        }
                    }
                    self.tableView.reloadData()
                }
            })
        }
    }
}

```

Code Snippet 12. Getting meal list function.

The getting meal list function is similar to the getting restaurant list. First the JSON data is gotten from the server by a GET request and then process to be displayed to the user interface.

- Changing quantity of meal and add to tray button

```

let trayItem = TrayItem(meal: self.meal!, qty: self.counter)
    guard let trayRestaurant = Tray.currentTray.restaurant, let currentRestaurant =
self.restaurant
        else{
            Tray.currentTray.restaurant = self.restaurant
            Tray.currentTray.items.append(trayItem)
            return
        }

var counter = 1
@IBAction func plus(_ sender: AnyObject) {
    if counter < 99{
        counter += 1
        total_order.text = String(counter)
        if let meal_price = meal?.price{
            price.text = "$\(Float(counter) * meal_price)"
        }
    }
}

@IBAction func minus(_ sender: AnyObject) {
    if counter > 1{
        counter -= 1
        total_order.text = String(counter)
        if let meal_price = meal?.price{
            price.text = "$\(Float(counter) * meal_price)"
        }
    }
}
}

```

Code Snippet 13. Changing quantity of meal and add to tray button function.

After clicking the “Add to tray” button, an object called “trayItem” will be initiated with the detail of the meal and its quantity. This object is a child of a class of an array which is displayed on the Cart page.

A counter variable is declared for counting quantity. When the user selects the plus and minus button to change quantity of the meal, the price label will be changed respectively by the result of counter multiply by single meal price.

- Getting user location and inputting location

```
//show user's location
    if CLLocationManager.locationServicesEnabled(){

        locationManager = CLLocationManager()
        locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        locationManager.requestAlwaysAuthorization()
        locationManager.startUpdatingLocation()

        self.map.showsUserLocation = true
    }

func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    let address = textField.text
    let geocoder = CLGeocoder()
    Tray.currentTray.address = address

    geocoder.geocodeAddressString(address!) { (placemarks, error) in
        if (error != nil){
            print("error", error)
        }
        if let placemark = placemarks?.first{
            let coordinates: CLLocationCoordinate2D = placemark.location!.coordinate
            let region = MKCoordinateRegion(center: coordinates
                , span: MKCoordinateSpanMake(0.01, 0.01))

            self.map.setRegion(region, animated: true)
            self.locationManager.stopUpdatingLocation() //stop updating location of user
            when input new address

            //create pin
            let annotation = MKPointAnnotation()
            annotation.coordinate = coordinates
            self.map.addAnnotation(annotation)
        }
    }
    return true
}
```

Code Snippet 14. Taking location.

First when the user goes to the cart page and the location service is enabled, the map will automatically take the location of the user and display it on the map.

There is a text field for inputting the address of the user, the address that will be sent to the server is taken from this text field. The application uses Core Location library to convert the string from the text field and find it on the map through geo coder, and place a mark on that location afterwards.

- POST request for creating order

```
func createOrder(stripeToken: String, completionHandler: @escaping (JSON) -> Void){
    let path = "api/customer/order/add/"
    let url = baseUrl?.appendingPathComponent(path)
    let simpleArray = Tray.currentTray.items
    let jsonArray = simpleArray.map { item in
        return [
            "meal_id": item.meal.id!,
            "quantity": item.qty
        ]
    }

    if JSONSerialization.isValidJSONObject(jsonArray){
        do{
            let data = try JSONSerialization.data(withJSONObject: jsonArray, options: [])
            let dataString = NSString(data: data, encoding: String.Encoding.utf8.rawValue)!

            let params: [String: Any] = [
                "access_token": self.accessToken,
                "stripe_token": stripeToken,
                "restaurant_id": "\(Tray.currentTray.restaurant!.id!)",
                "order_details": dataString,
                "address": Tray.currentTray.address!
            ]
            refreshToken{
                Alamofire.request(url!, method: .post, parameters: params, encoding:
                URLEncoding(), headers: nil).responseJSON(completionHandler: { (response) in
                    switch response.result{
                        case .success(let value):
                            let jsonData = JSON(value)
                            completionHandler(jsonData)
                            break
                        case .failure:
                            completionHandler(nil)
                            break
                    }
                })
            }
        } catch{
            print("JSON serialization failed")
        }
    }
}
```

Code Snippet 15. POST request for creating order.

Firstly, the API url is declared. The data of every item on the Tray are taken into a jsonArray variable. The data consist of meal id and quantity, and are converted to JSON and String type with JSONSerialization and NSString methods, which are system default methods.

A POST request is made with parameters which include access token from FacebookSDK, Stripe Token which is generated when making payment, restaurant id, order details with the data of jsonArray and address.

- Getting latest order.

```
//getting latest order
func getLatestOrder(completionHandler: @escaping (JSON) -> Void) {
    let path = "api/customer/order/latest/"
    let url = baseURL?.appendingPathComponent(path)
    let params: [String: Any] = [
        "access_token": self.accessToken
    ]

    refreshToken{
        Alamofire.request(url!, method: .get, parameters: params, encoding: URLEncoding(),
headers: nil).responseJSON(completionHandler: { (response) in
            switch response.result{
                case .success(let value):
                    let jsonData = JSON(value)
                    completionHandler(jsonData)
                    break

                case .failure:
                    completionHandler(nil)
                    break
            }
        })
    }
}

func getOrder(){
    APImanager.shared.getLatestOrder { (json) in
        print(json)
        if let orderDetails = json["order"]["order_details"].array {

            self.statusLabel.text = json["order"]["status"].string!.uppercased()
            self.tray = orderDetails
            self.orderTableView.reloadData()
        }
    }
}
```

Code Snippet 16. Getting latest order.

The application takes the path of API and processes a GET request to take data of the latest order that has been made. This API takes the last order of a specific customer and sends it back to the mobile application.

The function `getOrder` is called to display the latest order and its status. First, the order details are checked if it is a valid result, then the status and a list of products will be shown on the application.

- Getting Facebook user data

```
public class func getFBUserData(completionHandler: @escaping () -> Void){
    if (FBSDKAccessToken.current() != nil){
        FBSDKGraphRequest(graphPath: "me", parameters: ["fields":"name, email,
picture.type(normal)"]).start(completionHandler: { (connection, result, error) in
            if(error == nil){
                let json = JSON(result!)
                print(json)
                User.currentUser.setInfo(json: json)
                completionHandler()
            }
        })
    }
}
```

Code Snippet 17. Getting user data from Facebook

This function requests from Facebook SDK and returns the fields of name, email, picture as JSON data type, which the application will use to display user name and image, also send it to the server when making an order.

6 TESTING

In this section, test cases and detailed description of the results are given.

6.1 Signing up a New Restaurant

- Testing Step
Go to home page, click the link for signing up. Fill all the fields and match the requirements. Click Sign up.
- Expected result
If the registration is done successfully, the user will be redirect to the main Order page. Otherwise the sign up page will show an error message.
- Actual result

SIGN UP

Username

 Required: 150 characters or fewer. Letters, digits and @/./+/-/_ only. Enter a valid username. This value may contain only letters, numbers, and @/./+/-/_ characters.

Password

First name

Last name

Email

 Enter a valid email address.

Information of restaurant

Name

Phone

Address

Logo

[Choose File](#)

[Already have an account? Sign in!](#)

Figure 41. The page shows error message.

ID	Order Details	Customer	Address	Total	Created at	Status	Action
----	---------------	----------	---------	-------	------------	--------	--------

Figure 42. Successfully registered.

6.2 Editing account information

- **Testing Step**
Go to Account page of the website, change the current information on the form.
Click update.
- **Expected result**
The modified information is saved.
- **Actual result**

The screenshot shows the 'Account' page of an 'Order System'. The navigation bar includes 'Order System', 'Account', 'Meals', and 'Order'. The user is logged in as 'thanhvuong123'. The main content area is titled 'Account' and contains a form with the following fields:

- First name:** First Name Modified
- Last name:** Last
- Email:** first.last@email.com
- Edit restaurant information:**
 - Name:** Testing restaurant
 - Phone:** 1234123123
 - Address:** testing
- Logo:** Currently: restaurant_logo/burgerking-opengraph-1.jpg, Change: Choose File | No file chosen

An orange 'Update' button is located at the bottom of the form.

Figure 43. New information is saved.

6.3 Adding meal

- **Testing Step**
Go to Meals page of the website, click a blue button for adding a meal. Fill the form and click Add to complete the process.
- **Expected result.**
The Meal should be added to the server and also displayed on the meal page.
- **Actual result**

Order System Account Meals Order Welcome: thanhvuong123 Log out

Add meal

Name
Meal test 1

Description
Description Test 1

Image
Choose File | burgerking-o...graph-1.jpg

Price
12

Add

Figure 44. Filling the form.

Order System Account Meals Order Welcome: thanhvuong123 Log out

Meals

Add Meal

ID	Name	Description	Price	Image
7	Meal test 1	Description Test 1	12	

Figure 45. New meal displayed on the Meal page.

6.4 Editing Meal

- Testing Step
Click on the name of the meal that had to be edited. Change the information on the form and click Update.
- Expected result
The meal is displayed with new information on the Meal page.
- Actual result

Order System Account Meals Order Welcome: thanhvuong123 Log out

Edit meal

Name
Meal test 1

Description
Description Test 1 Modified

Image
Currently: meals_images-burgerking-opengraph-1_gqKcZ3.jpg
Change:
Choose File | No file chosen

Price
12

Update

Figure 46. Editing form.



Order System				
Account		Meals	Order	
			Welcome: thanhvuong123	Log out
Meals				
Add Meal				
ID	Name	Description	Price	Image
7	Meal test 1	Description Test 1 Modified	12	
8	Meal test 2	Description Test 2	11	

Figure 47. New description updated.

6.5 Signing with Facebook on Mobile

- Testing step
Click Sign in with the Facebook button. After signing in the information of the user is displayed as the information from Facebook.
- Expected result
User image and name is taken from Facebook and displayed on the application.
- Actual result

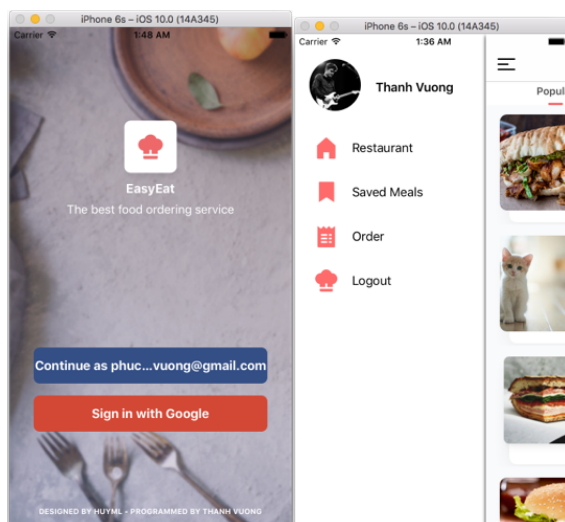


Figure 48, Figure 49. Sign in button, Information of user from FB

6.6 Getting list of restaurants

- Testing step
After signing in, a list of restaurants is displayed.

- Expected result
A list of all restaurants in the database with the restaurant that made for testing.
- Actual result

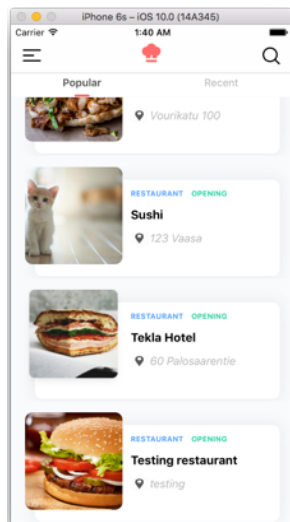


Figure 50. List of restaurants which includes also a testing restaurant.

6.7 Getting list of meals

- Testing step
From the Restaurant list page, choose a restaurant.
- Expected result
A list of meal from the menu of selected restaurant is displayed.
- Actual result

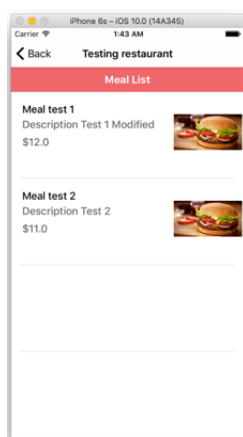


Figure 51. meal list of testing restaurant.

6.8 Making an order

- Testing step
Select a meal to order, change the quantity and add to cart.
Submit the address and process payment.
For testing, input 4242 4242 4242 4242 as a credit card and make a payment.
- Expected result
In the order page, the latest order should be shown with its status. And on the Admin side there is an order that is sent from the customer.
- Actual result

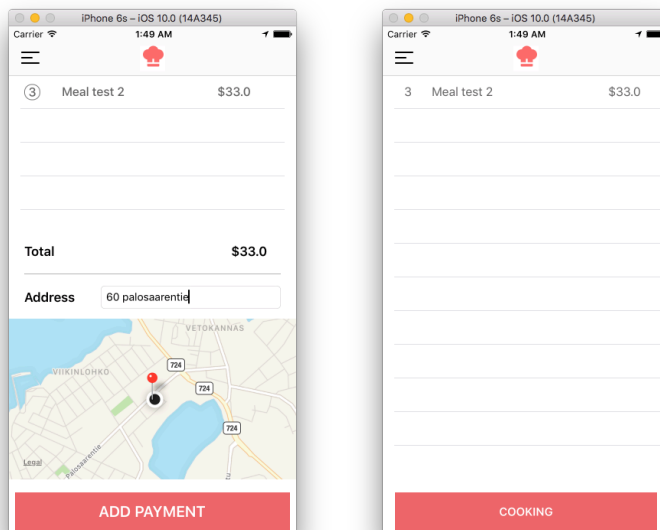


Figure 52, Figure 53. Submit address, After making payment.

ID	Order Details	Customer	Address	Total	Created at	Status	Action
36	Meal test 2 11 x 3 = \$33,	Thanh Vuong	60 palosaarentie	33	April 11, 2018, 10:49 p.m.	Cooking	Ready

Figure 54. Order is sent to the Admin dashboard.

6.9 Changing order status.

- Testing step
On the Order page, click the blue button to change status from Cooking to Ready
- Expected result
The status on the mobile application should be changed.
- Actual result

ID	Order Details	Customer	Address	Total	Created at	Status	Action
36	Meal test 2 11 x 3 = \$33,	Thanh Vuong	60 palosaarentie	33	April 11, 2018, 10:49 p.m.	Ready	

Figure 55. Changing status from Admin side.

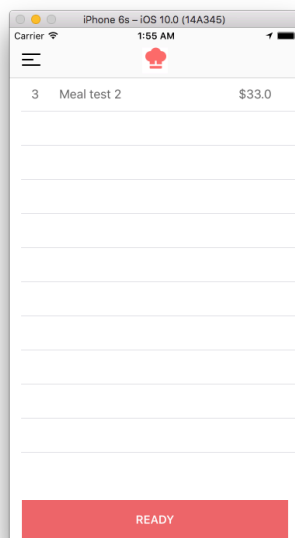


Figure 56. Status of the order is changed on the mobile application.

7 CONCLUSION

With the aim to develop a fully functional system with both web application and mobile application, the project meets nearly all the requirements of the software. The application includes also RESTful methods for communicating between server and client sides which helps the abundance of the system. Yet, this application still has a huge space for improvement for functionalities.

Users can use Mobile Ordering System to create their own restaurant with a creative menu and manage orders through the web based application. On the client side, users can easily select and make an order from many different restaurants.

Although the application seems to be simple, the implementation was not easy. There were many challenging problems that came up during developing period. One of the most confusing parts was creating API, making connection between two separate applications and especially creating an order from the mobile. Since the user's information is taken from Facebook, there are the access token and the refresh token that required to make sure the token is not expired when doing any action on the mobile. If the token is expired, it needs to refresh by adding time. For making payment, the Stripe token should be created also to communicate with the third party library and make calculation on the server. Again these API functions were really confusing for the first time implementing. Thanks to the huge community of developers in many different forums, solutions to solve this challenging parts were found.

Future Works

The initial goals of the project have been achieved, as mentioned there are still improvements for future work. The order page on the Admin Dashboard can be more detailed, the actions for the orders can be developed with more statuses. Registering for a user can also be done by social media accounts. There are numbers and data that can be analyzed and formed to a chart that is beneficial to the business. For the mobile application, the user can also log in with a Google account. Last but not least, The UI of both web app and mobile app can be implemented more responsively and friendly.

REFERENCES

/1/ Python Programming language. Accessed 23.03.2018.

[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))

/2/ Django (webframework). Accessed 23.03.2018.

[https://en.wikipedia.org/wiki/Django_\(web_framework\)](https://en.wikipedia.org/wiki/Django_(web_framework))

/3/ Installing packages using pip and virtualenv. Accessed 23.03.2018.

<https://packaging.python.org/guides/installing-using-pip-and-virtualenv/>

/4/ Writing your first app with Django. Accessed 23.03.2018.

<https://docs.djangoproject.com/en/1.7/intro/tutorial01/>

/5/ The different between SQLite and MySQL. Accessed 23.03.2018.

<https://www.quora.com/What-are-the-differences-between-SQLite-and-MySQL-Are-they-both-the-same-company>

/6/ OAuth. Accessed 23.03.2018.

<https://en.wikipedia.org/wiki/OAuth>

/7/ Bootstrap. Accessed 23.03.2018.

[https://en.wikipedia.org/wiki/Bootstrap_\(front-end_framework\)](https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework))

/8/ Xcode. Accessed 23.03.2018.

<https://en.wikipedia.org/wiki/Xcode>

/9/ Swift Programming Language. Accessed 23.03.2018.

[https://en.wikipedia.org/wiki/Swift_\(programming_language\)](https://en.wikipedia.org/wiki/Swift_(programming_language))

/10/ Swift 4. Accessed 23.03.2018.

<https://developer.apple.com/swift/>

/11/ CocoaPods. Accessed 23.03.2018.

<https://stackoverflow.com/questions/22261124/what-is-cocoapods>

/12/ Alamofire Tutorial: Getting Started. Accessed 23.03.2018.

<https://www.raywenderlich.com/147086/alamofire-tutorial-getting-started-2>

/13/ A Swift tutorial for stripe. Accessed 23.03.2018.

<https://www.appcoda.com/ios-stripe-payment-integration/>

/14/ REST and RESTful differences. Accessed 23.03.2018.

<https://stackoverflow.com/questions/1568834/whats-the-difference-between-rest-restful>

/15/ Web framework Wars: Django vs Ruby on Rails. Accessed 23.03.2018.

<https://thegeekswatch.blogspot.com/2016/08/framework-wars-django-vs-ruby-on-rails.html>

/16/ The payment process flow on Stripe. Accessed 23.03.2018.

<https://stripe.com/docs/recipes/switching-to-stripe>

/17/ RESTful web service tutorial. Accessed 23.03.2018.

<https://java2blog.com/restful-web-service-tutorial/>