

PECKISH

Overview:- Peckish is an online food ordering app for customers of a particular restaurant.

Online food ordering is the process of food delivery or takeout from a local restaurants or food cooperative through a web page or app. Much like ordering consumer goods online, many of these services allow customers to keep accounts with them in order to make frequent ordering convenient. Mobile Application is a great tool to engage with customers on the go. Having a mobile application along with a web application makes you completely accessible to your customer. Meaning, more business for you any day! Peckish has two sides to the application:-

a) Client Side:- It helps customers in placing orders from their Mobile app in their home/office/outdoors. It provides very simple and attractive User Interface to customers so that food can be ordered quickly and easily.

b) Server Side :- It helps managers to manage orders within the app. Managers can update the status of the order, see the details of the order and remove the order. Managers can also update the item, delete an item, add new items to the application.

Why use PECKISH?

- 1) With PECKISH, your sales is expected to grow.
- 2) Repeated orders and brand loyalty increases with an app.
- 3) It is a cost-effective and hassle-free way to get on the mobile platform.
- 4) Not just food ordering, many smart features available.
- 5) A Mobile Application For Restaurant and Food Ordering System
- 6) Grow your business

Features, Challenges and techniques applied:-

- 1) **Splash Activity** (See Appendix Fig.29):- The features of the application starts right from the Splash Activity. It checks if the app runs for the very first time and moves the customer to different activities depending upon the conditions.

It checks if the application is running for the first time in the following way:-

The Splash Activity has a method named `checkFirstRun()`. It creates a Shared Preference in the method and saves the current version of build in `currentVersionCode` and then look for existing `versionCode` in the `SharedPreferences` and saves it in `savedVersionCode`. If `savedVersionCode` doesn't exist it is given a flag value of -1. Then it checks for different conditions such as :-

- a) It checks if the `currentVersionCode` is same as `savedVersionCode` i.e. the app has already run and returns false. (See Appendix Fig.30)
- b) If the `savedVersionCode` doesn't exist then the `currentVersionCode` is stored in the `SharedPreferences` and it returns true since the app is running for the first time. (See Appendix Fig.31)

- c) If the `currentVersionCode > savedVersionCode` i.e. a new version of the app is installed on the device (upgrade) then it also returns true since we count an upgrade as new fresh run. (See Appendix Fig.32)

If the Splash activity is not running for the first time and there is no one logged into the app, then it takes the user to the `SignUpLoginPage`.

If the Splash activity is not running for the first time and there is someone logged into the account and the unique id is of the server, it takes the user to `DisplayActivityServer`.

If the Splash activity is not running for the first time and there is someone logged into the account and the unique id is not of the server, it takes the user to `DisplayActivity`.

If it is running for the first time, then it takes the user to the `ViewPagerActivity`. (See Appendix Fig.33)

- 2) **ViewPagerActivity** (See Appendix Fig.1 - 3):- This gives an overview of the application to the user and it just shows for the very first time. It uses a class `ViewPagerSliderAdapter` which extends `PagerAdapter` and is used to put the items from the `ArrayList` to the `ImageView` and `TextViews` created in the external layout for the `ViewPager`. Instead of creating three `ArrayList` for the `ImageView` and the two `TextViews`, a model class named `ViewPagerSliderContainer` was created and the `ArrayList` stores the object of `ViewPagerSliderContainer` class. It also :-
 - a) Hides the back button on the first page of the `ViewPager`
 - b) Changes the next button to finish button on the final pages
 - c) Keeps track of the page user is on and changes the color of the dots correspondingly.
- 3) After the `ViewPager`, it goes to the **SignUpLoginPage** (See Appendix Fig.4), which contains the slogan for the app and two buttons for going to Signup and SignIn screen.
- 4) **Sign Up Screen** (See Appendix Fig.6):- It contains four fields for creating a new account. The fields include Name, Email Address, Password, Phone Number. It contains a button for creating the account and a button for going to sign in screen if they already have an account. When signup button is clicked, it first checks for internet connection by calling a method in the Common class. If the user is not connected to the internet, it displays a toast saying please check your connection. If the user is connected to the internet a progress bar is shown and then it validates username, email, password and the phone number depending upon the different conditions. If the data is valid, then it uses an instance of `FirebaseAuth` to create a username with email and password. An `onCompleteListener` is added to the creation of the account. If the account is successfully created, visibility of the progress bar is set to gone and an instance of `User` class is created with name, email, password, phone to save the details of the user to the firebase real time database. Then using an instance of the `FirebaseDatabase`, `Users` path is referenced and `User` is added to the firebase database as a child with its primary key as the Unique Id generated by the firebase authentication and then it takes the user to the `displayActivity`. If the account is not successfully created, it displays the corresponding error message to user.
- 5) **Sign In Screen** (See Appendix Fig.5):-
 - a) It contains two `editTexts` for the email and the password. It also contains button to Login and a button to take to the signup screen if the user doesn't have an existing account. When the user clicks the login button, it checks for the internet

- connection. If the user is not connected to the internet, then a toast is displayed else it validates the email and password and sets the visibility of progress bar to visible.
- b) If the email and password is valid, it uses an instance of FirebaseAuth to sign in with email and password. An onCompleteListener is added to the logging in. If the user is successfully logged in, it uses the instance of firebase database by the referring to the unique id of the user and checks if the user is member of the staff or no. If the user is a member then it takes the user to the displayActivityServer (Server Side) and if he is not a member of the staff it takes the user to displayActivity(Client Side). It also sets the visibility of the progress bar to gone.
 - c) If the user is unable to login, it displays the corresponding error message.

6) **Display Activity** (See Appendix Fig.7) (Client Side):-

- a) This is an app drawer activity. When the app drawer on the side opens it contains the name and email of the user and contains various options to easily navigate through the app. It has a floating action button which takes the user to the cart when clicked. The floating action button also displays the number of items in the cart. The Display Activity also contains a bottom navigation view to help user to easily navigate through various components of the food. When the display activity is started, an instance of firebaseDatabase is referenced to Users table and saves the entire details of the currentUser in instance of User defined as currentUser in the Common class. Details of the user is saved to currentUser because we will be able to get the details of the user without referencing the database again and again but by directly getting it from the Common class. Then it sets the text views in app drawer to user's name and email address.
- b) Bottom Navigation contains three menus namely Main course, Drink and Dessert. When main course, Drink or Dessert menu is clicked in the bottom navigation view it loads the corresponding fragment in display activity by beginning a fragment transaction and replacing the frame layout in the Display Activity with fragment and then committing the transaction.
- c) It also updates the token for the user to the firebase database. A token is unique to each device and the app is making the use of token to send notifications to particular users. So it uses an instance of FirebaseDatabase and gets the reference of Tokens table and uploads the Token class to the database under phone number of the user as primary key. The Token class is a model class which stores the token number and Boolean value for isServerToken which will be false in client side.
- d) It also specifies what to do when an Item in App drawer is clicked. If the user click cart, it takes the user to the Cart Activity. If the user clicks Orders, it takes the user to the order status activity and if the user clicks logout, then it sign outs the user using an instance of the Firebase Auth. Also it deletes the corresponding token to the user in firebase using an instance of the firebase database, since this token no more represents this user and we do not want to send the notifications of this user to this device if user is not signed in.

7) **Fragments** (See Appendix Fig.8-9):-

a) RecyclerViewAdapterFragments:-

- 1) This adapter is used by the main course, drink and dessert fragment to load the products in the fragments. It extends RecyclerView.Adapter Class and also contains an inner class MyRecyclerViewAdapterViewHolder which extends RecyclerView.ViewHolder. MyRecyclerViewAdapterViewHolder initializes the TextViews, ImageViews to the corresponding TextViews and ImageViews from the external layout.
- 2) In the onBindViewHolder method TextViews and ImageViews are filled with data from the ArrayList. To load the image from the image id, Picasso library was used. It loads the image from the image id efficiently and effectively into the ImageView.
- 3) It checks for each item if it is added to favorites by getting the information from the Database class and display the corresponding image for favorites. It also sets a listener when the favorite ImageView is clicked on the RecyclerView and adds the corresponding product to the local database in favorites table. It also removes the item from the database if it is already in the database when clicked and changes the image of the imageview.
- 4) It also sets a listener for quick cart ImageView. It adds the item to the cart tables in the local database by using the Database Class and its addToCart method.

b) Main Course, Dessert, Drinks Fragment:- It has a search bar at the top to search through main course items. An instance of Firebase Database is used to refer to Products table in the database. Products is model class to store details of the products. A ValueEventListener is attached to the instance of firebase database to store the items from the firebase database into the ArrayList for products and suggest list for the search bar. While fetching data from the database only the products matching the corresponding type of fragments will be loaded into the arraylists. For eg:- In main course only the main course items will be added to the product list and the suggest list. Notify data set is used in the value event listener to reflect the changes in the app as soon as there are any.

c) Search bar:-

- 1) A text Change Listener is added to the Search bar. So in the onTextChanged method inside the Listener, it checks what the user has typed and updates the suggest list what matches to the user text.
- 2) A set on Search Action Listener is also attached to the Search bar. It checks when if search is not enabled it an adapter with all the items is attached to the recycler view and displays all the item. When the user clicks search, the adapter of the recycler view is changed to search adapter. Search adapter only contains the product that was searched and displays only the searched item in the RecyclerView.

d) Onclick Listener for the adapter:-

When any product is clicked, it passes the name, price, description, Image Id, and Product Id into the intent by getting it from adapter. It also moves the user to the detail Activity about the product.

8) **Detail Activity** (See Appendix Fig.11):-

- a) It uses a Collapsing toolbar layout in the XML file.
- b) It sets the name, price, description, image into the textviews and imageview by getting from extras in the intent which we passed in the display activity.
- c) It uses an instance of firebase database to get the reference to the Rating table. Then it does a query to the Rating table and gets the child where the food id in the database equals the product id of the selected product. Then adds all the rating together and keeps track of the number of rating. If the count is not zero, then it sets the rating of the rating bar to the average of all the ratings.
- d) When the floating button on the left is clicked, it displays an App Rating Dialog with Submit and Cancel buttons. A user can rate ranging from Very bad, Not Good, Quite Ok, Very Good and Excellent. And the user can type his/her comments about the food.
- e) When the user submits the rating, an instance of Rating is created. Rating is a model class which contains fields for User phone number, product id, rating, and the name of the user. Then this model class is pushed to the rating table in the firebase database.
- f) The user can select the number of items to add to the cart, by increasing/decreasing the value in the number button. After setting the value in the number button, User can click on the floating button on the right. When the floating button on the right is clicked, it creates an instance of the Database class and adds the item to the cart of the local database.
- g) When the user clicks the show comment button on the bottom of the activity, It takes the user to the Show Comment activity which displays the review of different people about the product. It also passes the product id into the intent.

9) **Show Comment** (See Appendix Fig.12):-

- a) It uses a Swipe Fresh Layout.
- b) When a user enters the show Comment Activity, it creates an instance of Firebase Recycler Adapter using the Rating Model Class and Show Comments View Holder.
- c) It sets a Refresh listener for the Swipe Layout.
- d) It gets the product id from the intent. Then it makes a query to the rating table in firebase database which gets all the ratings where the product id equals to the product id of the selected product.
- e) Then by using an instance of Firebase Recycler Options all the options are stored by building the query.
- f) Then the adapter is initialized with the options and data from the model is loaded into the holder's textviews and rating bar.
- g) It uses an external layout name show comment layout for displaying the data in the recycler views.

10) **Cart:-**

- a) RecyclerViewAdapterCart:- Cart uses this adapter for displaying the recycler view.
 - 1) RecyclerViewAdapterCart extends the RecyclerView.Adapter<CartViewHolder>.
 - 2) On create view holder method inflates the external layout namely cart_layout into the CartViewHolder.

- 3) It receives an array list from the cart activity containing Order. The arraylist contains all items added to cart. Order model class contains the product id, product name, quantity, price and image id of products.
 - 4) On bind view holder method loads the data from the array list into the textview, imageview and number button of external layout displayed in the recycler view.
 - 5) It also sets a value changed listener on the number button. When the user changes the number of items to be ordered in the cart, the total price changes correspondingly. It also updates the changes in the local database by creating an instance of the Database and using its update cart method
- b) CartViewHolder:-
- 1) CartViewHolder extends RecyclerView.ViewHolder.
 - 2) It is used to initialize the textviews, Imageview and Number button to the ones in the external layout.
 - 3) It sets a Context menu listener(Long click Listener) upon the whole recycler view and displays a menu with an option to delete the item.
- c) Activity:-
- 1) The Activity displays different items in the cart in Recycler View with the help of RecyclerViewAdapterCart.
 - 2) All the items selected for the cart is stored in the cart with the help of loadListFood method. It also computes the total price of all the items and displays in the textview at the bottom.
 - 3) It creates an instance of ApiService and uses getFCMService method from Common Class. getFCMService method uses Retrofit Client to create a class of the ApiService using the base url as <https://fcm.googleapis.com/> and creating ApiService.class.
 - 4) When someone long press the recycler view in the cart and selects delete from the context menu, it deletes the item from the arraylist. Then it cleans the cart from the local database and again fills it with the updated arraylist.
 - 5) When the user clicks place order button, it displays an alert dialog if the number of items in cart is greater than 0. An external layout is inflated into the Alert Dialog to fill the address and any comments for the food. When the user clicks submit after filling in the address and any comments, it generates an order id for the order and adds the details of the request in the Request Model Class. Request model class contains the phone number, name, address, total price, status of the item, comments and the list of items that was ordered. The Request is uploaded to Requests table in the firebase by using an instance of Firebase Database and referencing the Requests Table.
 - 6) After the order is placed, it uses an instance of Firebase Database and references the Tokens Table. It queries the Tokens table and look for the token whose is server token value is true. Then it creates an instance of the Notification Class. Notification is a model class to store the body and title for the notification. Then it creates an instance of the Sender class. Sender is Model Class created which stores the server token from the firebase database and the Notification Model Class. Then it uses the instance of ApiService and sends the notification to the particular server token. Then it checks if the notification was sent to the server. If the notification is successfully sent to the server, it displays

a toast to the user and finishes the cart activity. If it is unable to send the notification, then it displays the corresponding failure message.

11) **Order Status Activity** (See Appendix Fig.16-18):-

- a) A user can go to order status activity by clicking orders in the app drawer.
- b) It makes the use of Recycler views to display the different orders by the user.
- c) It uses an instance of the firebase database to get the reference to requests table.
- d) Then it performs a query on the requests table and filters only the requests of the user by matching the user phone number to the phone number in the requests.
- e) Then it uses Firebase Recycler Options to build the query and get the requests as a Request model class into the options.
- f) It also uses the Order View Holder to initialize the Textviews to the corresponding Textviews in the external layout for the Recycler View.
- g) Then it uses the Firebase Recycler adapter to load the items from the options into the Textviews of the external layout.
- h) Status field of Request is saved as 0,1 and 2 in the firebase. 0 means placed, 1 means on the way and 2 means shipped. So these codes should be converted to its corresponding messages while displaying in the recycler view. It uses convert code to status method to convert the code to its corresponding message and displays it in the Recycler View.
- i) On create view holder inflates the external layout for the Recycler view into the order view holder.

12) **Sending Notifications** (See Appendix Fig.19-20) (Server And Client) :-

- a) It makes use of the API service which specifies the notification will be sent in the form of json objects.
- b) It also has the authorization key in its header.
- c) It uses fcm/send as a post.
- d) Retrofit Client is used to get a client for sending the notification with the help of base url and using a Gson Converter Factory to convert the java classes into the json objects.
- e) For creating an instance of the APIService get FCM Service method can be used in the Common class. FCM Service method gets the client from the RetrofitClient class and creates an APIService class.
- f) An instance of the APIService class can be used to send the notification by calling the method sendNotification and then passing the Sender Model Class as an input.

13) **Receiving Notifications** (See Appendix Fig.21) (Service):-

- a) It makes use of the Firebase Messaging Service. There is a class MyFirebaseMessaging in the app which extends Firebase Messaging Service.
- b) It has an inbuilt method called on Message received. It is a callback method which is triggered when the device receives a notification.
- c) To display the notification in the mobile, a sendNotification method was created. It makes a channel id for the notification and also initializes the Notification Manager.
- d) The sendNotification method first checks if the android version of the device is greater than or equal to API 26. If it is, then a notification channel is created and description, lights, light color, vibration is set for the notification channel.

- e) Then it gets the notification by calling the get notification method from the instance of Remote message. Then it sets the intent i.e. the activity to be loaded when the notification is clicked.
- f) It sets the sound for the notification.
- g) Then it builds a notification with the help of NotificationCompat.Builder and sets the icon for the notification, title for the notification, text for the notification.
- h) Then it displays the notification by calling the notify method from the instance of the notification manager.

14) **Local Database** (See Appendix Fig.35):-

a) Description:-

- 1) Despite using firebase database, the app also makes use of local database stored in assets/databases/FoodDB.db.
- 2) The app uses a local database to store the cart and the list of favorites by the user.
- 3) A local database was used because the items in a cart might or the user might not order it. So there was no need to upload/download to/from the firebase database whenever the user makes any changes to the cart. So for the efficiency and effectiveness of the app, a decision for using a local database was made.

b) Database Class:-

- 1) This class is used to manage the local database.
- 2) It extends SQLiteAssetHelper class.
- 3) Fields for the name of the database were defined and a constructor was created.
- 4) There is a table named OrderDetail in the database which stores the cart. The fields of the table are Id, ProductName, ProductId, Quantity, Price, Image.
- 5) To get the cart it makes a query to the OrderDetail table in the database which selects all the columns and adds the items to a List one by one with the help of cursor.
- 6) The add to cart method takes the Order model class as an input which contains the details of the items added to the cart. Then it performs a query on the OrderDetail table and inserts the data into the table by using inbuilt execSQL method.
- 7) Clean Cart method is used to Remove all the items in the Order Detail table in the database. It makes the query on the Order Detail table and executes the query using the inbuilt methods.
- 8) There are also methods to update the cart for a particular item and a method to count the number of items in the cart.
- 9) Add to favorites method takes foodID as an input. This method is used to add the items to the favorites table in the database when the user clicks the favorites button in the recycler view of various fragments. Similarly it uses a method to remove the favorites from the table and also an is favorite method to check if a particular item is in the favorites table.

15) **Common Class** (See Appendix Fig.34):- This class contains the common fields used by many activities so it doesn't need to be written again and again. We just make use the methods of the Common Class to get the fields already stored. This is for making the app efficient and effective. For eg:- If at some place in activity, the app needs the name and the phone number of the user, it will be very inefficient to make

the calls to the firebase database whenever a request is made for the user name and phone number. So we stored the username and phone number the first time it was requested and then make use of it by directly calling from various activities.

16) **Display Activity Server** (See Appendix Fig.7):-

- a) Description:-The display activity on the server side is very similar to client side display activity in terms of the design. But from the perspective of functionalities, the display activity at the server and client side are very different.
- b) Adding a new item:-
 - 1) The display activity at the server side gives you the power to add a new item of any type.
 - 2) It keeps track of the type of the food the user is currently in by taking a flag value and changing it according to the food item when the user changes the type of food using the bottom navigation view.
 - 3) The floating action button at the bottom right corner is no more used to go to the cart. It is used to add a new item.
 - 4) When the floating action button is clicked, It shows an alert dialog with the title of Add new “type” item where type is replaced by the type of the food item they are currently in. For eg:- If they are in Dessert fragment and they click the floating action button, then it would say Add new dessert item.
 - 5) An external layout is used to set the view of the alert dialog. It has editTexts for the name of the item, description and price of the item.
 - 6) Then it has a button for selecting the image from the gallery. When this button is clicked it creates an intent for choosing an image from the gallery.
 - 7) After choosing the image, on Activity Result Callback method is called which saves the image into an URI and changes the text of the Select button to Image Selected.
 - 8) Then the user needs to click the upload button to upload the image to the firebase storage. The app generates a random name for the image. By using an instance of the firebase storage, a path to images folder in the images folder in the firebase storage is referenced and the saved uri is uploaded to the firebase storage.
 - 9) A success listener is attached to the uploading of image and displays a toast when the image is uploaded successfully. Then it gets the download url for the image and creates an instance of the Product Model class and fills it with the details of the new product.
 - 10) When the user clicks the submit button after successfully loading the image to the storage, it creates an instance of the firebase database and references the products table in it and uploads the details of the new product to the firebase database.
 - 11) Since we are using real time database, it shows real time changes in the app. As soon as we upload a new product, the product is shown in the other users application.

17) **Fragments (Server Side)** (See Appendix Fig.8-9):-

- a) Description:-

Same as display activity on the server side, the fragments over the server side are very similar to fragments over the client side in terms of the design but completely different in terms of the functionalities.

b) RecyclerViewAdapterFragmentsServer:-

- 1) The recycler view in the fragments make use of the RecyclerViewAdapterFragmentsServer which extends RecyclerView.Adapter.
- 2) There are no click listeners for quick cart and add to favorites over the server side as they are not required by the manager.
- 3) Instead there is a create context menu listener over the recycler views which creates a menu of update and delete when the recycler view in the fragments is pressed for a long time.

c) Main Course, Dessert and Drinks:-

- 1) The items matching the type are displayed in the recycler view of the respective fragments. When a user long presses the recycler view, it displays a context menu showing options of update and delete.
- 2) When a user clicks update, a alert dialog is shown. An external layout is used as a view for the alert dialog. It has editTexts for name, description and price of the item and the name, description and price of the selected item is loaded into the edit text so that it is convenient for the user to know the details of the item.
- 3) He can make any change in the name, price and description of the item. It also has button for choosing a new image for the item. When the user clicks the button for choosing the image, a new intent is opened to choose the image from the gallery. The image is saved as an uri by the call back method on activity result.
- 4) Then the user needs to click the upload button to upload the image to the firebase storage. The app generates a random name for the image. By using an instance of the firebase storage, a path to images folder in the images folder in the firebase storage is referenced and the saved uri is uploaded to the firebase storage.
- 5) A success listener is attached to the uploading of image and displays a toast when the image is uploaded successfully. Then it gets the download url for the image and creates an instance of the Product Model class and it sets the image id of the product to the new image id.
- 6) When the user clicks the submit button, it make changes to details of the product as per the changes in the edit text and then by using an instance of the firebase database, it updates the details of the item in the firebase database.

18) **Order Status Server** (See Appendix Fig.26):-

- a) When the user clicks on orders in the app drawer over the server side, it takes the user to the order status activity. It shows all the orders that have been requested.
- b) When the manager enters the order status activity, it makes a reference to the requests table in the firebase database.
- c) It uses Firebase Recycler options to build a query where it gets all the requests from the requests table.
- d) Then it uses the firebase recycler adapter, to load the data in the options to the recycler views.

- e) It makes use of the Order view holder server to connect the recycler view to the external layout.
 - f) It has three buttons edit, remove and details.
 - g) When the manager clicks edit, it shows an Alert Dialog with three options place, on my way, and shipped. The manager can choose one of the three options. When the manager changes the status of the order, it updates the status of the order in the firebase database.
 - h) After changing the order status, it uses an instance of the firebase database and makes a reference to the Tokens table. It gets the token for the user whose order status was updated by matching the phone number.
 - i) Then it stores the body and title of the notification in Notification model class. Then it fills the Sender model class with the token number and an instance of the Notification model class. Then it sends the notification to the person whose order status was changed saying your order status was updated.
- 19) **Order Detail Server Side** (See Appendix Fig.26):-
- a) Order detail activity:-
 - 1) It displays the order id at the top, phone number, total price of the order, Address, and the comment of the person who ordered the food.
 - 2) Then it displays a textview and under the textview a recycler view is attached to it.
 - b) Order Detail Adapter:-
 - 1) It makes use of the Order Detail Adapter to load the items from the list to the external layout for the recycler view.
 - 2) It gets the detail of the current request from the common class and displays it in the recycler view.
 - c) MyViewHolder:-
 - 1) It initializes the textviews to the textviews in the external layout.

Installation guide:- The application requires following synchronizations in order to run successfully:

- Make sure that android studio has latest build tools installed (28.0.0)
- It should have the latest versions of google play services.
- In order to render the application successfully, go to preferences -> Build, Execution, Deployment -> Instant Run and disable it.
- Install the intel HAXM accelerator before running the application on the emulator. Check the following screenshot to make sure that you have all the SDK tools required.
- Make sure that the gradle file is properly synced and there is no error in it.
- Go to Build -> Clean Project or Rebuild Project to clear all the unnecessary cache and perform clean installation.
- System Requirements

Windows	Mac	Linux
---------	-----	-------

<ul style="list-style-type: none"> • Microsoft Windows 7/8/10 (32 or 64 bit) • 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator. • 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image) • 12800 x 800 minimum screen resolution 	<ul style="list-style-type: none"> • Mac OS x 10.10 (Yosemite) or higher, up to 10.13 (macOS High Sierra) • 3 GB minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator • 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image) • 1280 x 800 minimum screen resolution 	<ul style="list-style-type: none"> • GNOME or KDE desktop Tested on Ubuntu 14.04 LTS, Trusty Tahr (64-bit distribution capable of running 32-bit applications) • 64-bit distribution capable of running 32-bit applications • GNU C Library (glibc) 2.19 or later • 3 GB RAM minimum, 8 GB Ram recommended ; plus 1 GB for the Android emulator • 2 GB of available disk space minimum, 4 GB recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image) • 1280 x 800 minimum screen resolution
---	--	---

Functions and user manual:-

1) Splash Screen and Slide View Pager:-

a) Description:- Slide screen activities and Slide view pagers are incorporated in our application to make new user feel familiar about the offerings that app provides in general. For an example, we can display that how it is different for the other application or what are the top features of the application.

b) User Manual:-

- 1) Install the app
- 2) After installing the app, the user will see the name and icon of the application in the list of the apps in the mobile phone.
- 3) Then click the icon and the app will open.
- 4) When the app is opened, it displays a splash screen to the user.
- 5) After the splash screen, the user will see the different screens since the app is running for the first time after the install.
- 6) The three different screen tells about the offerings of the app.
- 7) The user can either slide or click on buttons to navigate through the different screens.
- 8) He can click on finish button to end the description.

2) Login and sign in Activity:-

- a) Description:- Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices. We have used firebase authentication services in order to validate user credentials. From the login screen user will be redirected to either admin dashboard or to the regular screen depending on their role assigned in the database.
- b) User Manual:-
 - 1) To create a new account, click on signup button on the signup login page.
 - 2) Enter the name, phone number, email and password for the account as per the requirements and click on sign up.
 - 3) A new account will be created for the user.
 - 4) To login using an old account, just click on sign in button on the signup login page and enter the email address, password for the account.
 - 5) Click on log in button.
 - 6) If the email address and password for the account is correct, the user will be logged in.

3) Optimized Search bar and Favorite Items:-

- a) Description:- Search bar optimization was necessary as we wanted to display recommendation when user clicks on the search bar and as soon he starts typing, list should be updated to match users search query. In addition to that we were able to implement functionality which is very specific to the navigation. For ex. If user search when he is on drinks menu, all the possible results that are available in the drinks menu are displayed.
- b) User Manual:-
 - 1) For using the search feature, click on the search bar on the top of the fragments.
 - 2) Then it will display the suggestion depending upon which fragment the user is currently in.
 - 3) The list update as he types.
 - 4) After writing the name of the item, he has to click on the search icon on the keyboard.
 - 5) Then the item is displayed the user and he/she can click the item and add to cart.
 - 6) For using the favorites feature, just click the heart icon right next to the quick cart to add the item to the list of favorites.

4) Review and Ratings:-

- a) Description:- For any online ordering application or website, it is very important that user can provide and read reviews from the other user about a specific item. By keeping that in mind we have integrated Review and Ratings system where user can freely rate the food item and provide their personalized comment.
- b) User Manual:-
 - 1) The user needs to be in the detail activity for writing a review about the food.
 - 2) The user needs to click on the star floating button on the left hand side in the details of the food.
 - 3) Then a dialog will appear and the user can select the number of stars for the food item and can also type a comment for the food
 - 4) Then the user can submit his rating by clicking the submit button.
 - 5) The user can also view the reviews about the food by the other people.

<ul style="list-style-type: none"> 6) To do so, the user needs to click the show comment button at the bottom of the detail activity 7) When the user clicks the show comment button, it takes the user to a new activity where it shows all the comments for the selected food item.
<p>5) Add to Cart / Quick Cart:-</p> <ul style="list-style-type: none"> a) Description:- These buttons are used by the user to add the items to the cart. Implementing the quick cart functionality not only fasten the ordering process but it also provide smooth transition between different activities. We have used local databases to store user favorites, cart items. Local databases can be useful in the situations where you have to restrict server calls and make sure that request are sent to the server only when it's necessary. b) User Manual:- <ul style="list-style-type: none"> 1) The user can add the item to the cart by using the quick cart button. 2) The user can see the quick cart button at the bottom right of each product and can click it to quickly add the items to the cart without seeing the description about the item. 3) He can also add the items to the cart by going into the detail activity and selecting the number of items to be ordered. 4) User can view cart items by clicking on the FAB (Floating Action Button) at the bottom of the screen in the display activity. 5) The list of the items added to the cart is shown to the user. 6) The number of items can be changed by clicking the +/- button. 7) The user can also delete the item by long pressing the item and clicking on delete.
<p>6) Order Processing and Notifications:-</p> <ul style="list-style-type: none"> a) Description:- In our application, notifications play very important role. In order to make those notification appear on user screen with minimum delay integration of FCM was crucial. b) User Manual:- <ul style="list-style-type: none"> 1) The user can place the order by clicking on the place order button in the bottom of the Cart. 2) Then the user needs to fill his/her address and any comments for food if he would like. 3) Once he press yes and confirm the order, notification is sent to administrator that new order has been received. 4) Administrator can click on the notification and see the order details instantly.
<p>7) Navigational Elements:-</p> <ul style="list-style-type: none"> a) Description:-Navigation refers to the interactions that allow users to navigate across, into, and back out from the different pieces of content within your app. To make sure that all the user can access all the functionality regardless of the activity he's currently in, we have implemented navigation drawer and bottom navigation menus. The interesting part about the navigation drawer and navigation menu is that functionalities of both of them depends on user role. For ex. If user role is administrator depending on which menu he is (main menu, drinks, dessert), he can add item to that particular menu while for regular use depending on which menu he is, he'll be able to search item specific to that.

b) User manual:-

- 1) User can make use of the app drawer activity by clicking the icon on the top left corner of the display activity and then can select the menu items from the drawer.
- 2) The user can click on different types of food items using the bottom navigation view and can view the food items.

8) Add, Update and Remove Items:-

A) Description:- As an administrator, it is very important to manage all the items that are in menu and to do that swiftly is a challenging task. In order to add new item, you have to click on floating action button at the bottom with += sign. Once that button is clicked, an pop-up screen will appear with information form, where administrator can add name, description, price and Image for the specific item. While uploading image you have to click on select image button first and once you select the image, you can press on upload image button. After you fill all the information press YES and new item will appear on both the screens (user and admin) instantly.

Before clicking on the new item floating button, make sure that you are in the right navigation menu. As mentioned earlier, if you are in drinks menu, the new item that you are trying to add will be added in drinks menu . Same thing applies for main menu and dessert menu. To remove the specific item from the menu Long click on the specific item and select delete action. As soon as you click on delete, item will get deleted from the menu instantly.

You can update specific item by long clicking on the item and select the update option. As soon as you click on update it well pop-up a screen that consist all the information on current item. This functionality is really useful when you want to make any changes in current item. For ex. If you want to increase or reduce price of any specific item, you can do it with the help of this functionality.

B) User manual:

- 1) Manager can use this feature by clicking the add, update and remove buttons from the server side, to make changes in the menu of the restaurant. He can add new items. By using update button he can make any changes to the item as in to change the price. If any item is no longer available in the store, by clicking the remove button the item will be deleted.

9) Update order and Notification:-

A)Description:- Administrator can go to the orders from navigation drawer. This will show the list of orders that are placed by the user. There are three options available for admin at this point, either he can view the details of a specific order, by clicking on details button it will show all the individual item along with the price. You can also view the comment made by that specific order on the same screen. Clicking on remove button will remove the order from the order list.

You can edit the status of order by clicking on edit button. There are 3 options from the drop down menu which you can choose depending on the status of the order. Placed, on

the way and shipped. As soon as you select any of the option and click yes, a notification will be sent to customer informing the status update. This allows customer to keep in loop with the whole ordering process.

B) User manual:-

1) This function is on the server side for tracking the food order. The manager gets the order notification from the user.

2) After viewing the order the manager can change the status to “on the way” while preparing the order and notifies the user about it.

3) Once, the order completes, he can change the status of the order to “shipped” and the user will get the notification correspondingly.

10) Session Management:-

A) Description:- It was very important for us that each time user closes / minimize the application and comes back, he should be able to access the functionality specified by its role. For ex. If administrator minimize the apps and reopens it, rather than asking for the credentials it will automatically login based on the last session.

B) User manual:-

1) This feature is available on the user side as well as on the client side, when they close the app and open it again.

Incomplete functions:-

We did not work on showing the restaurants using the google play API as our app was targeting a single restaurant with only one store. So, we decided to not implement the google API since it won't be of any use to the user.

We did add many other things that we did not mention in the proposal such as Server side, sending notifications and much more.

DISCLAIMER:-


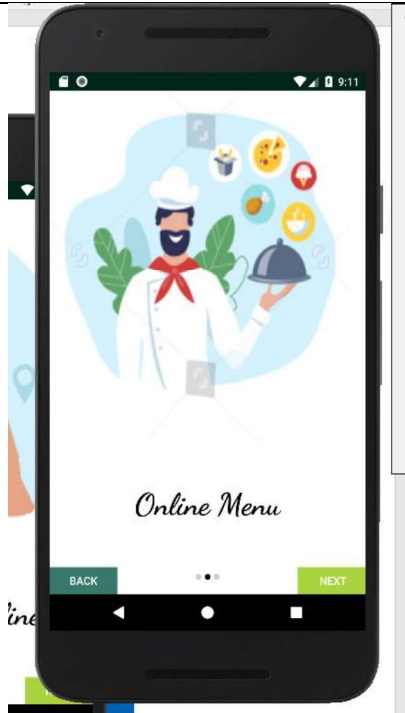
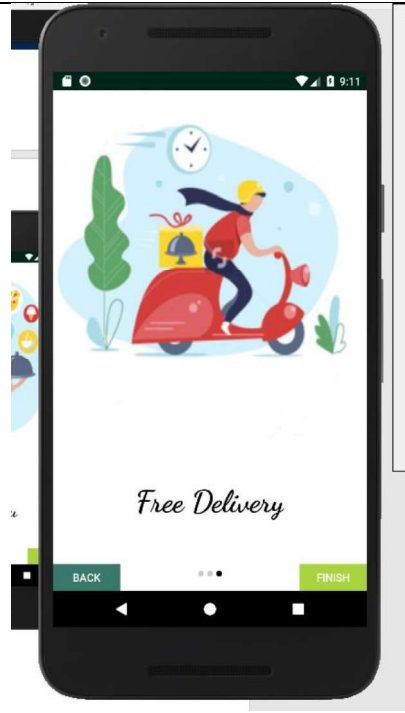
This enclosed document is the outcome of the project for the class Mobile Development-2 and represent the original work done by the group members mentioned at the starting of the document under the guidance of Prof. Ivan Wong.

References:-

Simplified Coding (Youtube)

Google Documentation

APPENDIX:-

		
Fig. 1	Fig. 2	Fig. 3

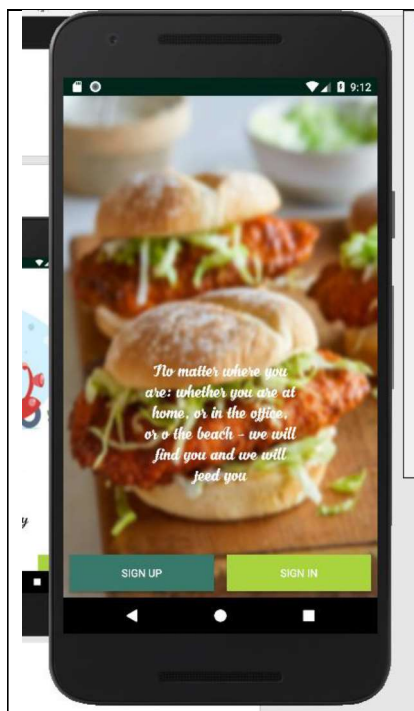


Fig. 4

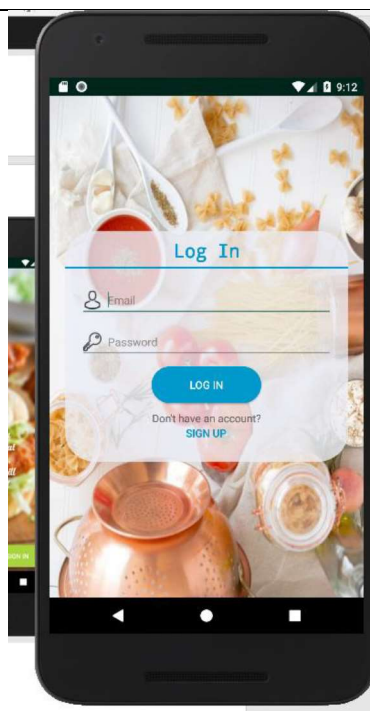


Fig. 5

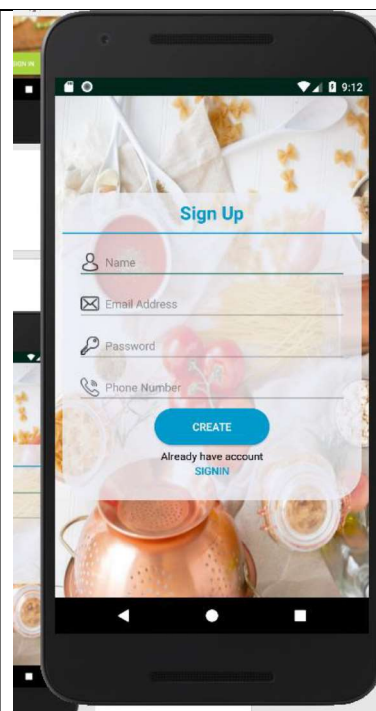


Fig. 6

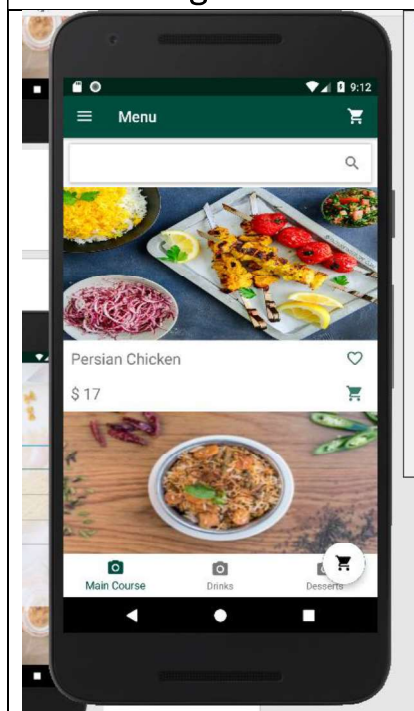


Fig. 7

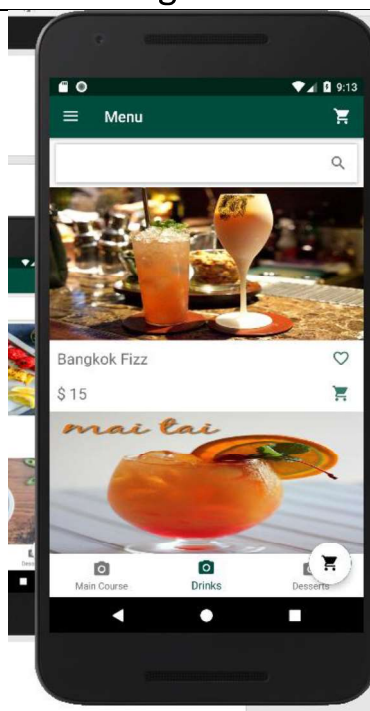


Fig. 8

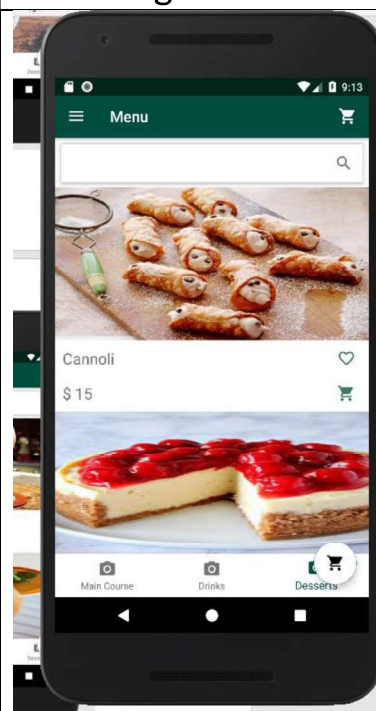


Fig. 9

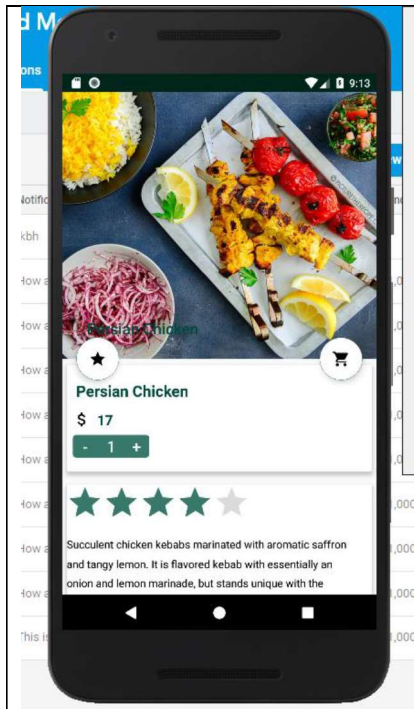


Fig. 10

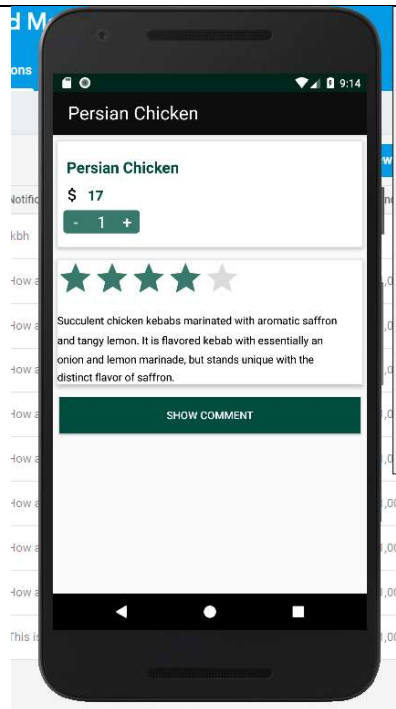


Fig. 11

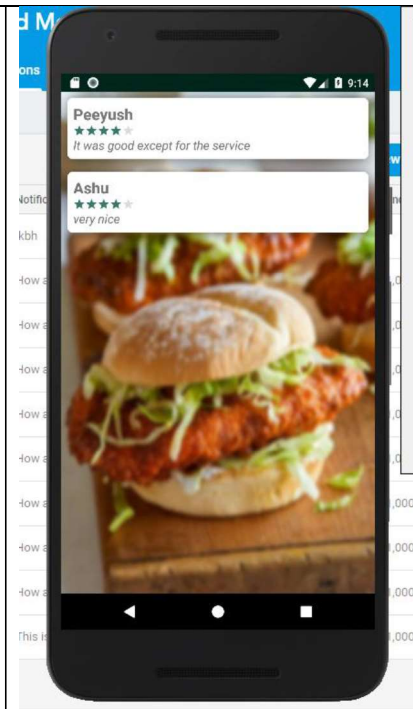


Fig. 12

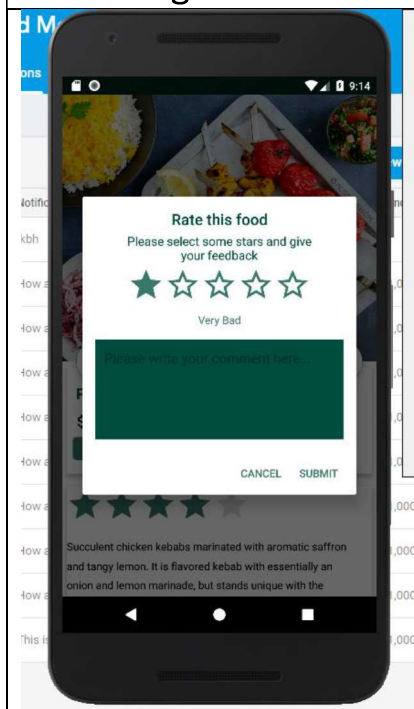


Fig. 13

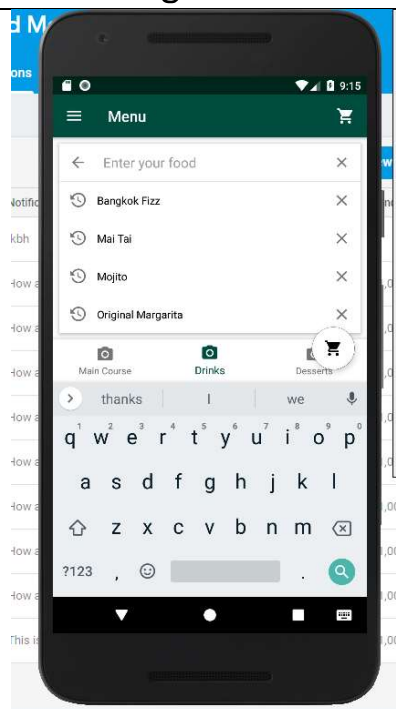


Fig. 14

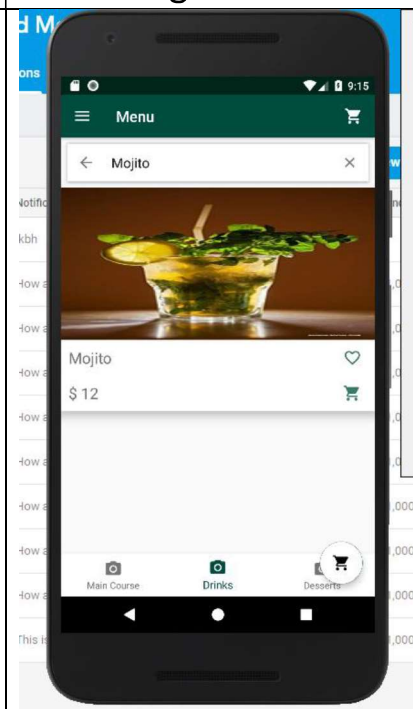


Fig. 15

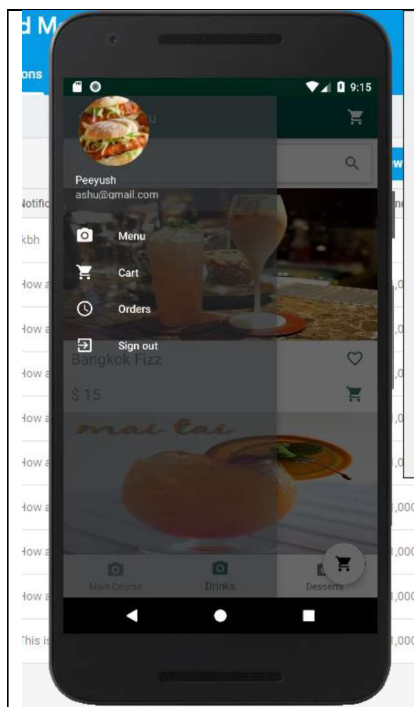


Fig. 16

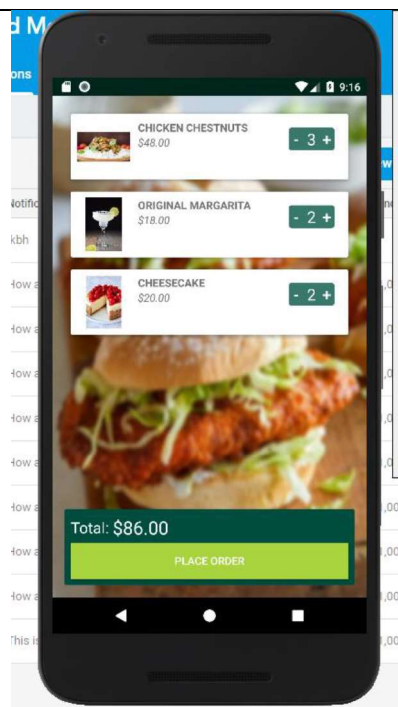


Fig. 17

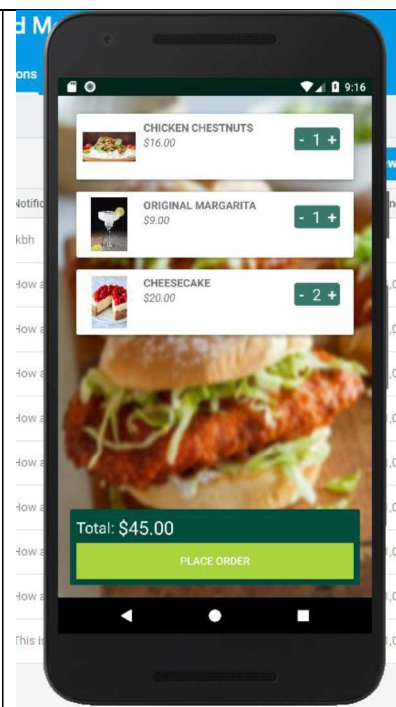


Fig. 18

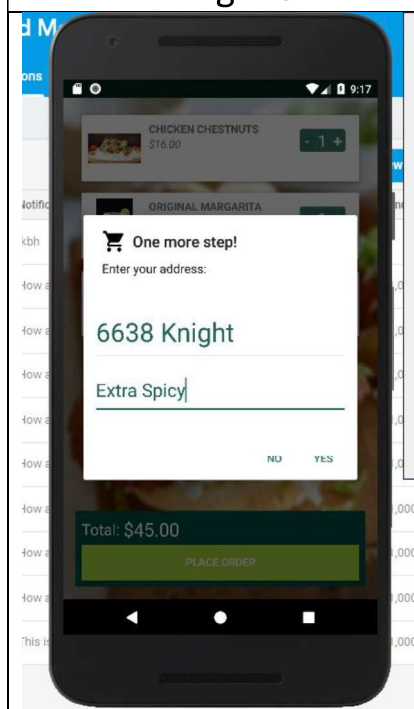


Fig. 19

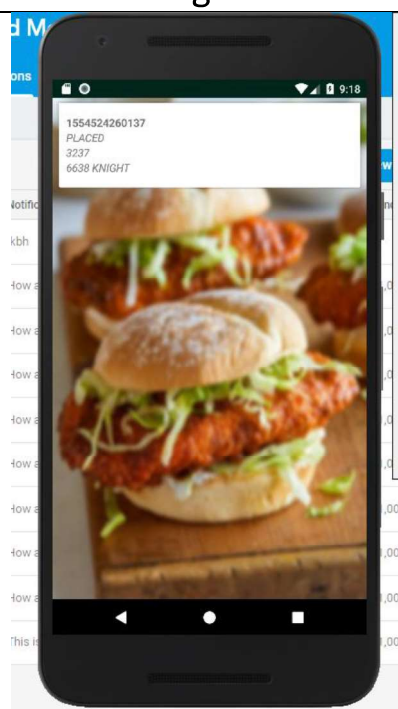


Fig. 20

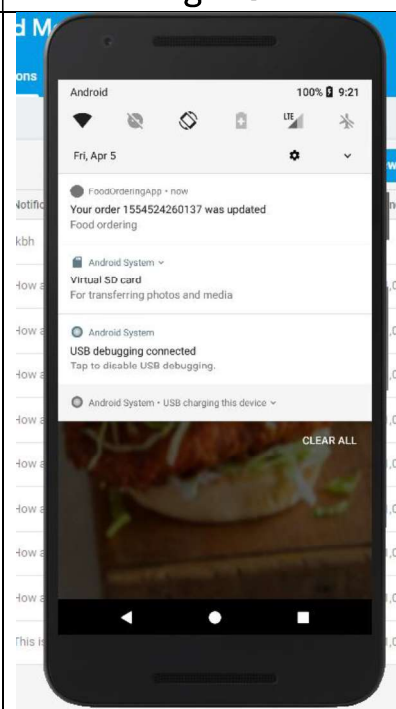


Fig. 21

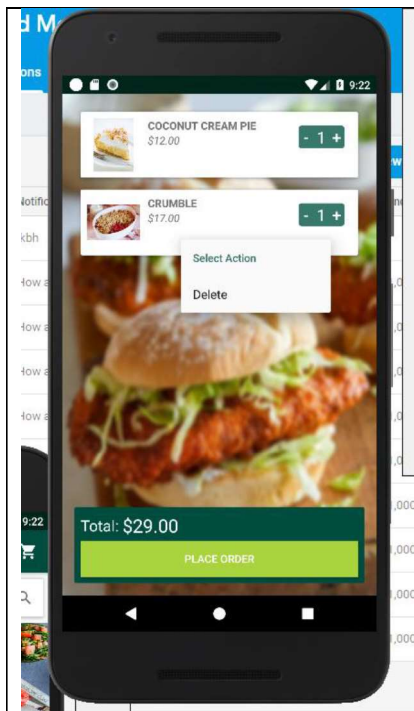


Fig. 22

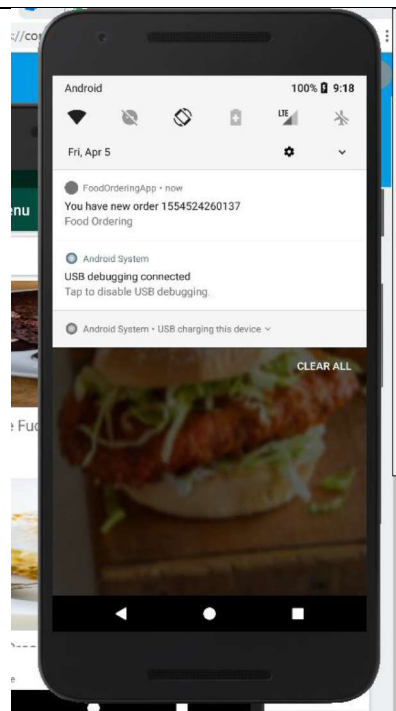


Fig. 23

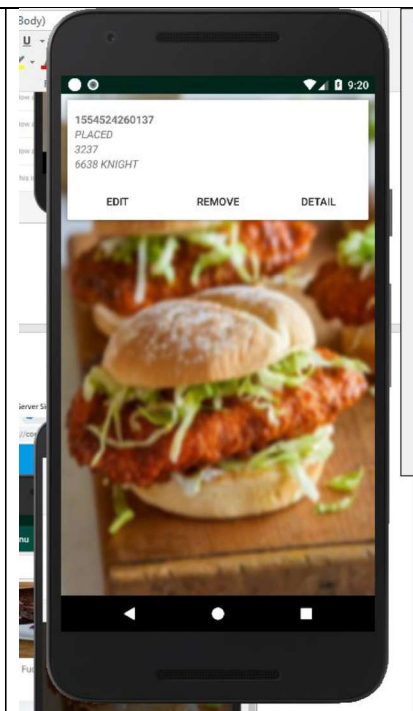


Fig. 24

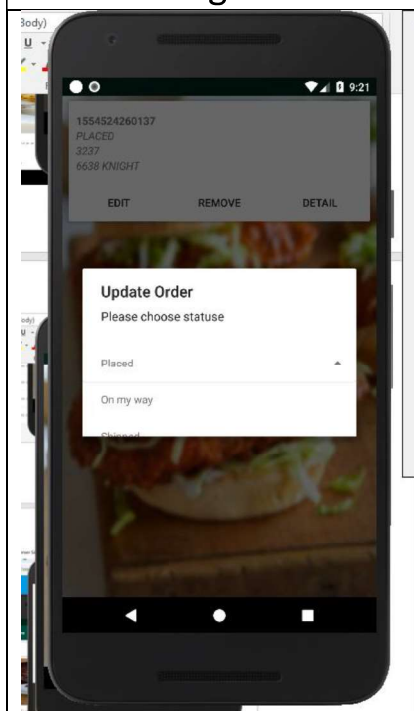


Fig. 25

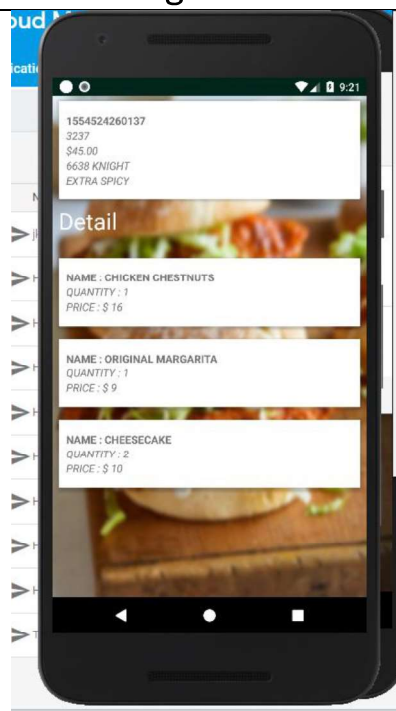


Fig. 26

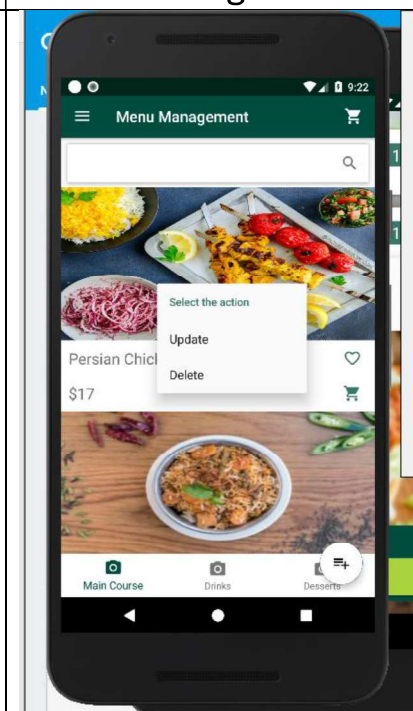


Fig. 27

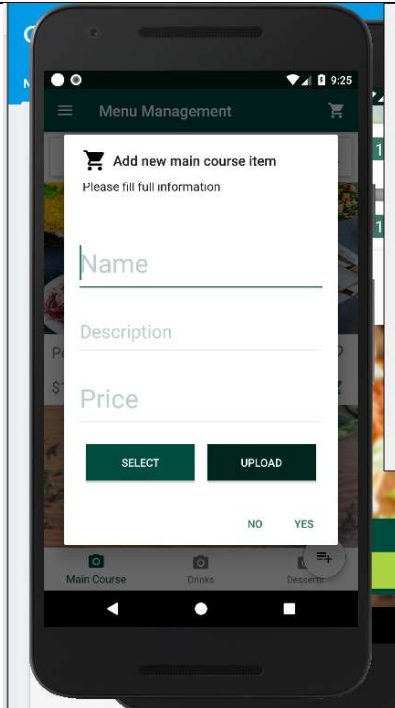


Fig. 28



Fig. 29

```

76
77 private boolean checkFirstRun() {
78
79     String PREFS_NAME = "MyPrefsFile";
80     final String PREF_VERSION_CODE_KEY = "version_code";
81     final int DOESNT_EXIST = -1;
82
83     // Get current version code
84     int currentVersionCode = BuildConfig.VERSION_CODE;
85     SharedPreferences prefs = getSharedPreferences(PREFS_NAME, MODE_PRIVATE);
86
87     // Get saved version code
88     int savedVersionCode = prefs.getInt(PREF_VERSION_CODE_KEY, DOESNT_EXIST);
89
90     // Check for first run or upgrade
91     if (currentVersionCode == savedVersionCode)
92     {
93
94         // This is just a normal run
95         return false;
96     }
97

```

Fig. 30

```

    else if (savedVersionCode == DOESNT_EXIST)
    {
        // Update the shared preferences with the current version code
        prefs.edit().putInt(PREF_VERSION_CODE_KEY, currentVersionCode).apply();
        return true;
        // TODO This is a new install (or the user cleared the shared preferences)
    }

```

Fig. 31

```

else if (currentVersionCode > savedVersionCode)
{
    // Update the shared preferences with the current version code
    prefs.edit().putInt(PREF_VERSION_CODE_KEY, currentVersionCode).apply();
    return false;
    // TODO This is an upgrade
}

```

Fig.32

```

 mAuth=FirebaseAuth.getInstance();

TimerTask newtask;
if (checkFirstRun()==false && mAuth.getCurrentUser()==null)
{
    newtask=(TimerTask) () -> {
        startActivity(new Intent( packageContext: SplashActivity.this,SignupLoginPage.class));
        finish();
    };
}
else if(checkFirstRun()==false && mAuth.getCurrentUser()!=null && mAuth.getCurrentUser().getUid().equals("jw2LsLGEC1fvWtkkqlJ1H210NB62"))
{
    newtask=(TimerTask) () -> {
        startActivity(new Intent( packageContext: SplashActivity.this,DisplayActivityServer.class));
        finish();
    };
}
else if(checkFirstRun()==false && mAuth.getCurrentUser()!=null)
{
    newtask=(TimerTask) () -> {
        startActivity(new Intent( packageContext: SplashActivity.this,DisplayActivity.class));
        finish();
    };
}
else
{
    newtask=(TimerTask) () -> {
        startActivity(new Intent( packageContext: SplashActivity.this,ViewPagerActivity.class));
        finish();
    };
}

Timer myTimer=new Timer();
myTimer.schedule(newtask, delay: 1000);
}

```

Fig.33

```

10
11
12
13 public class Common {
14     public static User currentUser;
15     public static Request currentRequest;
16
17     public static final String INTENT_FOOD_ID="FoodId";
18
19     private static final String BASE_URL="https://fcm.googleapis.com/";
20     public static APIService getFCMService()
21     {
22         return RetrofitClient.getClient(BASE_URL).create(APIService.class);
23     }
24
25     public static APIService getFCMClient()
26     {
27         return RetrofitClient.getClient(BASE_URL).create(APIService.class);
28     }
29
30     public static final String UPDATE="Update";
31     public static final String DELETE="Delete";
32     public static final int PICK_IMAGE_REQUEST=71;
33
34     public static String convertCodeToStatus(String status) {
35         if(status.equals("0"))
36             return "Placed";
37         else if(status.equals("1"))
38             return "On my way";
39         else
40             return "Shipped";
41     }
42
43     public static boolean isConnectedToInternet(Context context)
44     {
45         ConnectivityManager connectivityManager=(ConnectivityManager)context.getSystemService(Context.CONNECTIVITY_SERVICE);
46
47         if(connectivityManager!=null)
48         {
49             NetworkInfo[] info=connectivityManager.getAllNetworkInfo();
50             for (NetworkInfo networkInfo : info)
51             {
52                 if(networkInfo.isConnected())
53                     return true;
54             }
55             return false;
56         }
57         return false;
58     }
59 }

```

Fig 34

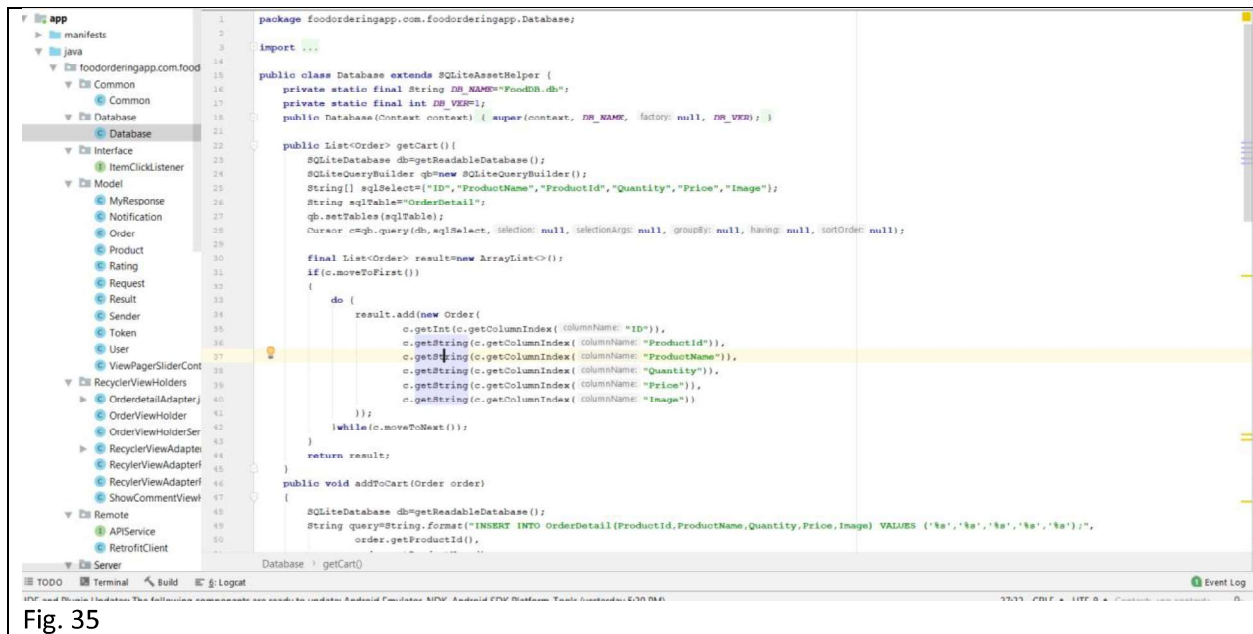


Fig. 35