

Tangito

Kunal Dhupar

Hrishikesh D S

Anandh Rajan S

Question statement:

Application like McD/Swiggy , below are the flow

Register and login

User can easily reach the restaurant by using application(Gmap, Current location, and marked near by restaurent locations)

View available Product (Multi-language support, REST api call)

Add to cart (stored values in internal DB or similar)

My order (Edit order)

Place order

expectation :

Need to using any design pattern

No hard coded values

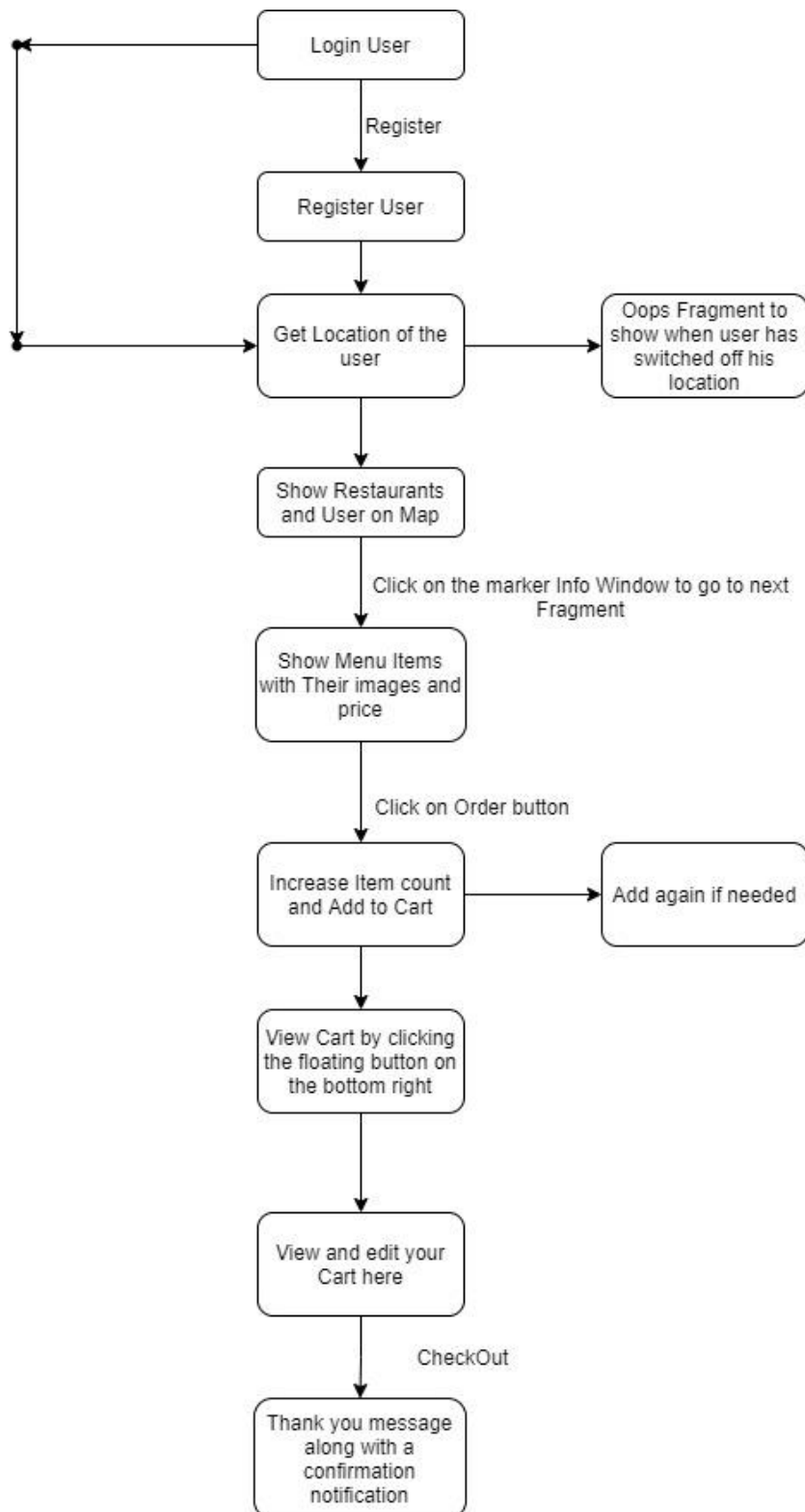
Nit and clean project structure/packages

No crashes

Introduction

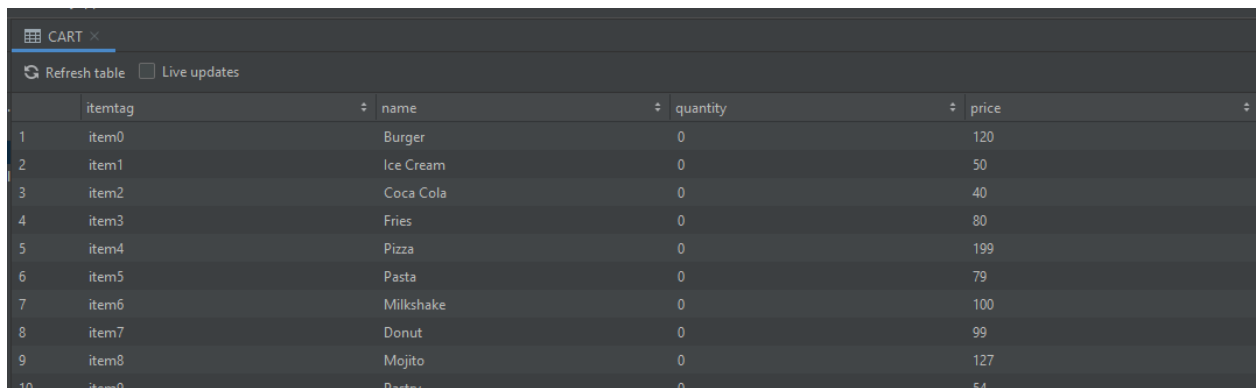
Tangito is a food chain application that was developed by keeping food chain applications like KFC and McDonalds as reference. This app allows its users to get Tangito restaurants near in his/her city, and show the locations on a Google Map. The user can select any of the restaurant in his/her city and can select from a variety of food items. The selected food items can be added to cart, which is later editable depending on the user's choice and the total bill is calculated and presented to the user, who can checkout and later view his order history as well.

Flow Diagram of Tangito



Schema

Cart(internal DB)



	itemtag	name	quantity	price
1	item0	Burger	0	120
2	item1	Ice Cream	0	50
3	item2	Coca Cola	0	40
4	item3	Fries	0	80
5	item4	Pizza	0	199
6	item5	Pasta	0	79
7	item6	Milkshake	0	100
8	item7	Donut	0	99
9	item8	Mojito	0	127
10	item9	Pastor	0	54

Internal Cart DB is populated on login/register from firebase with itemtag,name,quantity,price

DBwrapper API

DBWAPPER api is used to execute changes on the internal CART db. DBWrapper has a DBHelper reference which provides a layer of separation

Whenever A database operation needs to take place, create a reference and call the appropriate function

```
Val wrapper= DBWrapper(context)//pass context
```

```
wrapper.executefun()//execute one of the following
```

```
fun saveCart(itemtag : String,itemname : String,quantity : Int=0,price : Int=0) : Long
```

Called on login/register.Adds a new row/item to CART.Returns no of rows affected by db operation(should be 1)

```
fun retrieveCart() : Cursor
```

Returns a cursor with all items in CART. Iterate over the cursor to retrieve data

```
fun retrieveCartAsList() : ArrayList<Food>
```

Returns an arraylist of type <Food>(see Item.kt) with all CART items

```
fun retrieveCartAsString() : String
```

Returns a comma separated name-quantity-price,name-quantity-price.... string of CART items

```
fun getQuantityOfTag(itemtag : String) : Int
```

Returns quantity of specific item in CART

```
fun getTotalQuantity() : Int
```

Returns $\Sigma(\text{quantity})$ in CART

```
fun updateQuantity(itemtag : String, count : Int) : Int
```

Updates quantity of tag(ex-item0) to count

```
fun updateAllQuantitytoZero()
```

Updates all quantities to 0 in CART. Called when user checks out

```
fun deleteAll()
```

Deletes CART. call on logout

```
fun incrementQuantity(itemtag : String)
```

Pass the item tag as(ex- "item0"). Increments quantity by 1 for passed item. Call on plus button

```
fun decrementQuantity(itemtag : String)
```

Pass the item tag as(ex- "item0"). Decrement quantity by 1 for passed item. Call on minus button

```
fun retrieveDataFromTag(itemtag : String): Cursor
```

Pass the item tag(ex-"item0") Returns a cursor with single row

```
fun calculateBill() : Int
```

Return total amount calculated from db $\Sigma(\text{quantity}) * (\text{price})$

```
fun addRowsFromFirebase()
```

Makes a firebaseWrapper call to retrieve items from realtime database, populates the db on login/register

FIREBASE(FIRESTORE/REALTIME DATABASE)

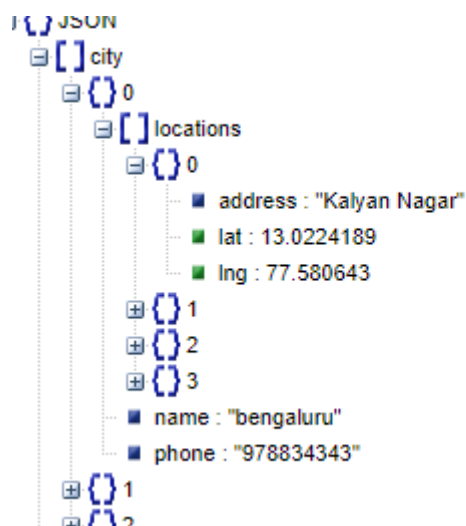
Firestore is used to store User details(collection)(email,name,phone ,previous order(collection))

Users is a collection. Each user document has orders collection. Each order document details a single order made by the user.(date,location,order,total)

Realtime Database is used to store cities/locations and menu

fooddeliveryapp-aa531-default-rtdb contains city and menu

city



menu



FirestoreWrapper API

Use this API to do firestore operations

```
fun getListOfCities(callbackfun : (List<String>->(Unit)))
```

Wherever function is called, retrieve list of cities in callback lambda. Lambda is used because firestore calls are asynchronous

```
fun getMenu(callbackfun : (List<Item>->(Unit)))
```

Wherever function is called, retrieve list of menu items in callback lambda. Lambda is used because firestore calls are asynchronous

```
fun getLocationsOfCity(cityname: String, callbackfun: (List<Location>) -> Unit)
```

Wherever function is called, retrieve list of locations (passed as cityname) in callback lambda. Lambda is used because firestore calls are asynchronous

```
fun getUserFromId(UID : String, callback : (User)->(Unit))
```

Wherever function is called, retrieve all data of current user (passed as UID) in callback lambda. Lambda is used because firestore calls are asynchronous

```
fun getPreviousOrdersById(UID : String, callback  
:(List<Orders>->(Unit)))
```

Wherever function is called, retrieve list of previous orders made by current user (passed as UID) in callback lambda. Lambda is used because firebase calls are asynchronous

```
fun resetEmail(email : String, callback :(String)->(Unit))
```

WSends a reset email to passed email parameter. Use callback to show snackbar/update UI accordingly with the result

Activities:

1) Main Activity-

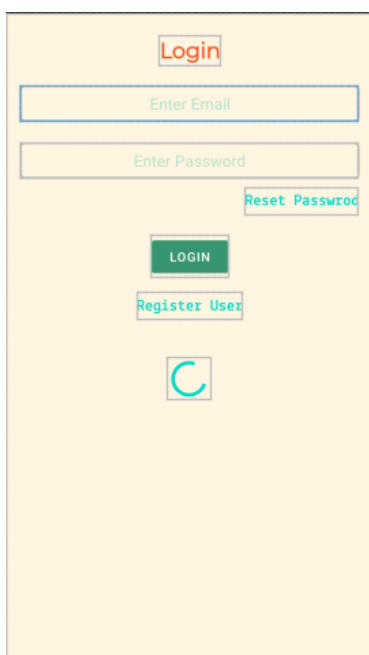
The first activity in our application. Mainly used for Authentication purpose. It consists of

Splash Screen-

Shown initially when the app loads while the user is being verified.

Login Fragment-

The user can login using his prior credentials. Firebase authentication is used to verify the user and if login is successful it will navigate to the next activity.

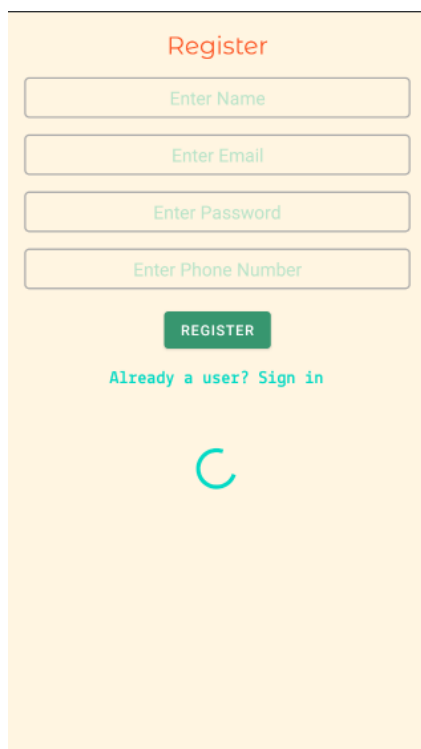


If it's a new user, he can click on "Register User" which will navigate to Register fragment.

It also has a feature called Reset Password which sends an email to the user's mail address for a password reset.

Register Fragment-

The user can create a new account whose details will be stored in the firebase for authentication purposes. After the user registers, they will be redirected to the next activity.

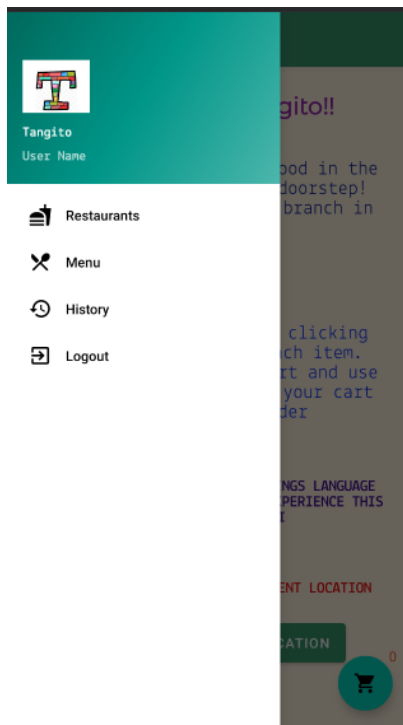
A screenshot of a mobile application's 'Register' screen. The screen has a light orange background. At the top, the word 'Register' is written in orange. Below it are four white input fields with rounded corners, each containing a green placeholder text: 'Enter Name', 'Enter Email', 'Enter Password', and 'Enter Phone Number'. Below the input fields is a green button with the word 'REGISTER' in white. Under the button, there is a link that says 'Already a user? Sign in' in green. At the bottom center, there is a large, light blue circular loading spinner.

If it's an old user he can navigate to Login fragment and login.

2)Delivery Activity-

This is the next activity of the application which consists of all the required functionalities. It consists of a navigation drawer activity for easy accessibility of the user.

The components of this activity are-



Find Location

GetLocations Fragment

This fragment is the first screen visible to the user after logging in or after reopening the app. Here we take the current location of user using FusedLocationProviderClient class and update it by implementing its methods.

The location comes in the form of latitude and longitude which help us in getting the city name using Geocoder. All of these values are passed to the next Map Fragment.

Functions used:

fun getCityName(latitude,longitude): Using the Geocoder function we can obtain the city name of any location by passing the latitude and longitude. Function returns city name of the user current location.

fun isLocationEnabled(): To check if the location provider(GPS or network) is switched on or not. Returns a Boolean value

fun onRequestPermissionsResult(): checks if location permission is granted or not

```
fun getLastLocation()
```

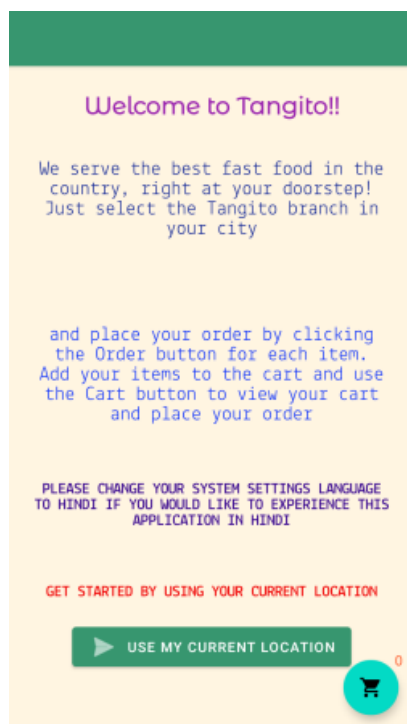
Gets last known location of user using FusedLocationProviderClient class by calling NewLocationData() and calls getCityName. Checks if city is amongst the one where Tangito is available, if not opens OopsFragment. This function is called onResume lifecycle of the fragment

```
fun NewLocationData()
```

Gets the location by setting up LocationRequest, with High Priority

```
fun locationCallBack()
```

Responds to the request and sends the latitude, longitude and city name to the map fragment as a bundle.



Maps

Maps Fragment-

In this fragment we set the markers of the user's current location as well as the restaurants location from the Firebase. We setup a click listener on the infoWindow of these markers, which stores the location name and opens the menu of our food chain. Location name is displayed along with username in the drawer navigation.

Functions-

- **onMapReadyCallback**: This callback is triggered when the map is ready to be used. This is where we can add markers or lines, add listeners or move the camera. We fetch restaurant markers from Firebase as a list and add them to the mapView. We also add User marker from location fetched by GetLocations Fragment
- **setOnInfoWindowClickListener**: Calls the saveAddress and goToMenu() functions
- **saveAddress**: stores the location name of the clicked marker window in the shared preference.
- **goToMenu**: Navigates to the next Fragment



Slide Menu-

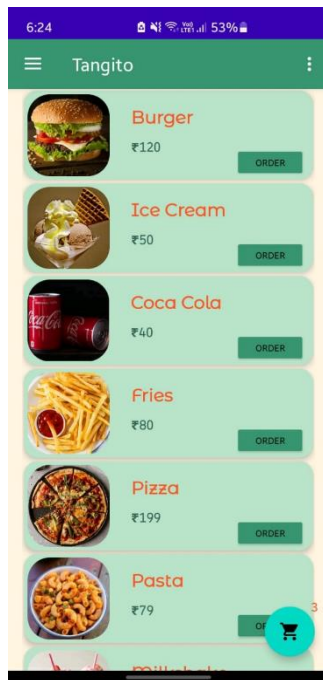
Logout Fragment-

A dialog box which prompts the user and confirms if they want to log out of the application.

Menu-

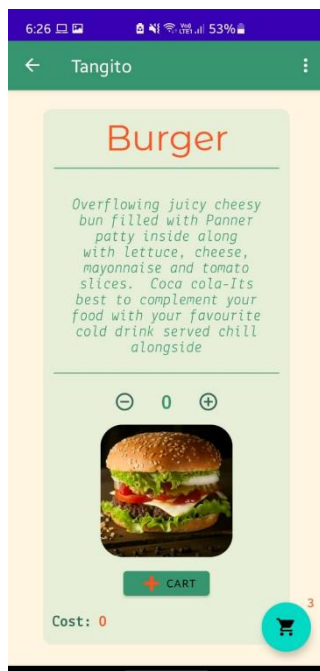
a) Menu Fragment-

It consists of a recycler view of all the menu items that have been obtained from a firebase call. A recycler view adapter is used to populate the data.



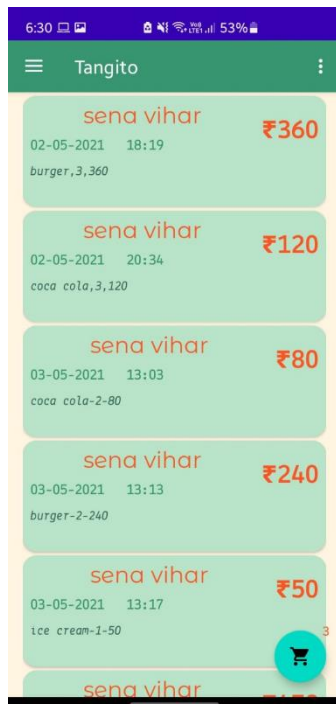
b) Menu Details Fragment-

Once the user wants to order a food item, they can click on the order button which will redirect them to this fragment where a detail overview of the food item is given to the user. The total cost of individual items is given here and the user can add the desired item to the cart.



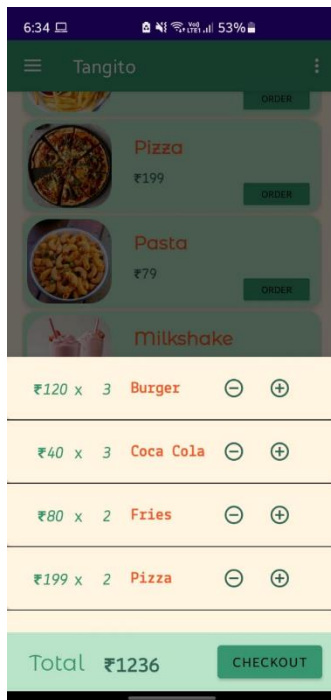
iii) **Orders Fragment-**

This fragment consists of the user's previous orders. It has a recycler view of the total bill, the restaurant location and the orders.



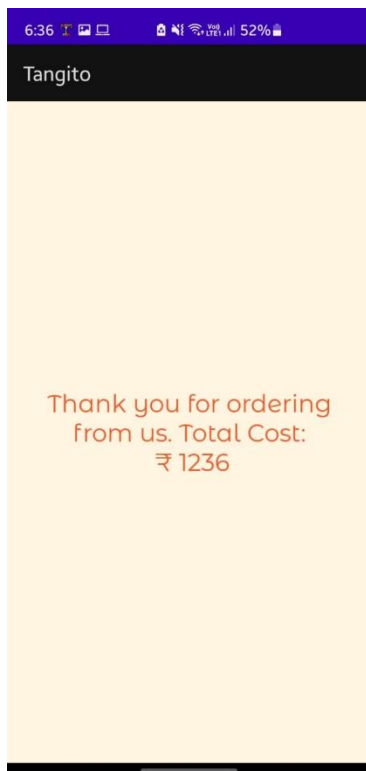
Bottom Sheet Dialog-

This consists of the cart which can be called when the user clicks on the Floating action button at the bottom corner of the screen. In this cart, it consists of a list view of all the items stored in the local database, the user can see all of his items which were added to the cart and they can also edit the number of items in the cart. In the bottom, the total bill of the user is displayed and the user can check out.



3)Check out Activity-

The final activity of the application after the user orders. The total bill is displayed and also a notification is sent to the user from the app.



CREDENTIALS

Firebase Console is linked to a google mail account

Email- groupfourcapgemii@gmail.com

Password- Capgemini@44

Project link - <https://console.firebase.google.com/u/1/project/fooddeliveryapp-aa531/overview>

Google cloud developer console - <https://console.cloud.google.com/apis/dashboard?pli=1&authuser=1&project=clear-practice-311813>

Maps API KEY - AlzaSyD20z2lneNh_NO55s2VB_S-fZMJtpNowAY

Firebase API KEY - AlzaSyBLoH9YWfamICEN1Is16-6KMFsG5j0kC1o

PROJECT INFO (Firebase)

```
"project_number": "262265243541",  
"project_id": "fooddeliveryapp-aa531",  
"storage_bucket": "fooddeliveryapp-aa531.appspot.com"
```