SmartInternz Guided Project

Snack Squad

A Customizable Snack Ordering and Delivery App

Project Documentation

Team ID - **Team-591097**

Team Members

Shreyansh Singh (shreyansh.singh2021@vitbhopal.ac.in)

Tirth Sheth (tirth.pinkal2021@vitbhopal.ac.in)

Akash Gupta (akash.gupta2021@vitbhopal.ac.in)

Dhruv Anil (dhruv.anil2021@vitbhopal.ac.in)

Contents

1. Introduction

1.1. Project Overview

A project that demonstrates the use of Android Jetpack Compose to build a UI for a snack squad app. Snack Squad is a sample project built using the Android Compose UI toolkit. It demonstrates how to create a simple e-commerce app for snacks using the Compose libraries.The user can see a list of snacks, and by tapping on a snack,and by tapping on the "Add to Cart" button, the snack will be added to the cart. The user can also see the list of items in the cart and can proceed to checkout to make the purchase.

1.2. Purpose

The app is designed with the sole aim of simplifying snack acquisition. In a market flooded with diverse applications, our unique selling point is an exclusive focus on snacks. We're carving out a niche to cater specifically to those delightful moments when you crave a quick and tasty treat. Imagine a dedicated space where snack enthusiasts can effortlessly browse, select, and satisfy their snack cravings without the distraction of unrelated options. It's snack time, simplified.

2. Literature Survey

2.1. Existing Problem

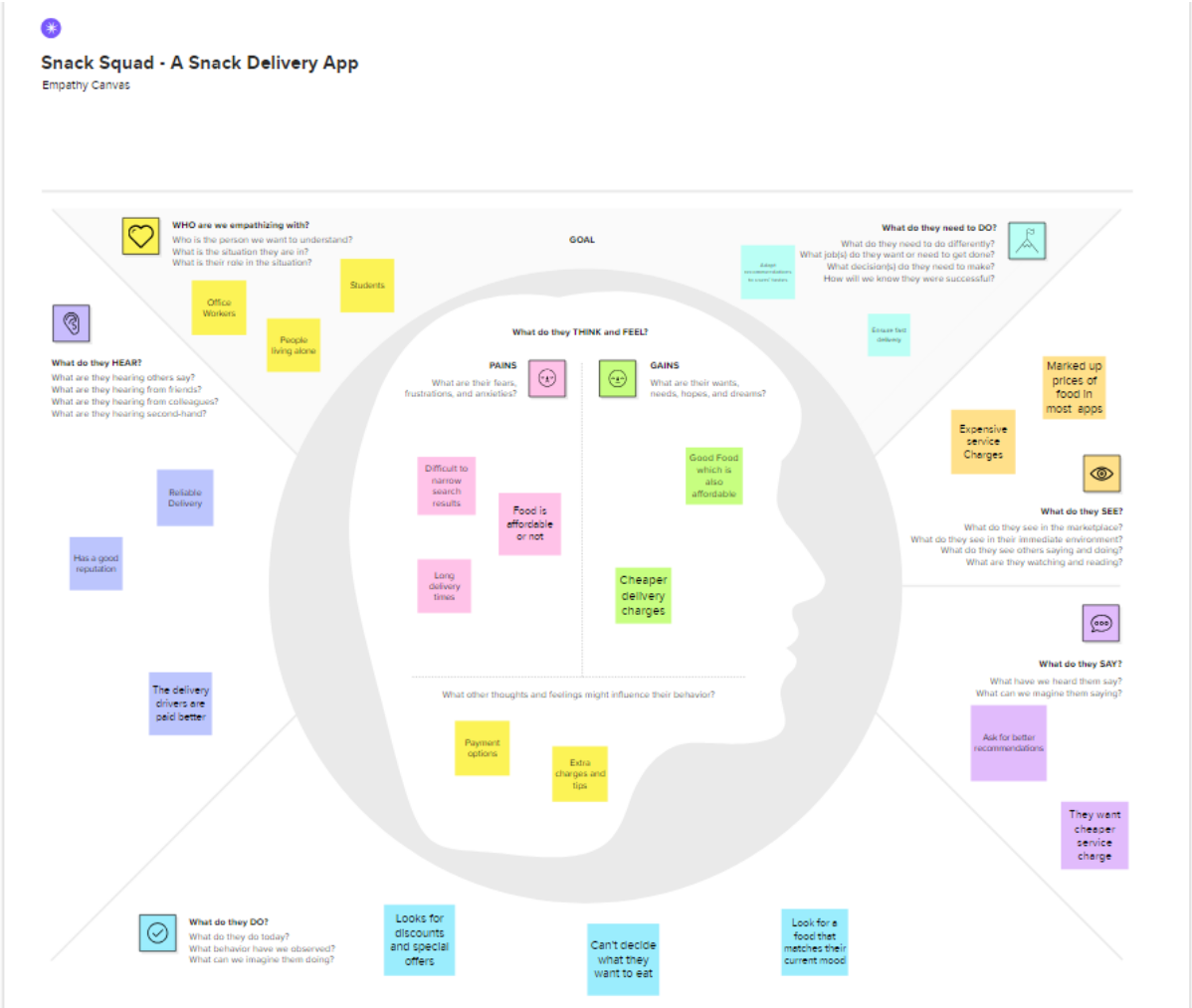
While other food delivery apps cast a wide net, offering an extensive array of cuisines and dishes, our app is set to stand out by honing in on a niche that often gets overlooked—snacks. While competitors may present users with a vast menu of meal options, we're zooming in on the delightful world of quick bites and treats. It's a strategic move to cater specifically to those moments when all you need is a satisfying snack, without the distraction of sifting through a plethora of meal choices. In a world of culinary abundance, we're simplifying the snack game, making it easier than ever to indulge in your favorite munchies.

2.2. Problem Statement Definition

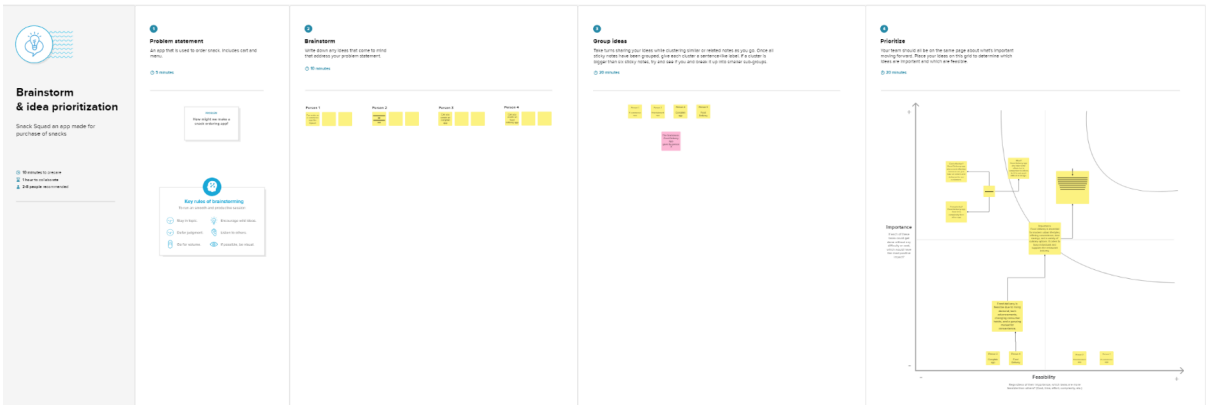
In a sea of food delivery apps, each vying for attention with their diverse offerings, there's a noticeable gap—one that we're eager to fill. Surprisingly, there's a dearth of apps exclusively dedicated to the world of snacks. While others may provide a smorgasbord of meal options, our app is pioneering a focused approach, catering specifically to snack aficionados. Picture a platform where the spotlight is solely on those tantalizing, bite-sized delights that make snack time a cherished moment. It's a unique niche that sets us apart in the food delivery landscape, providing a dedicated space for snack enthusiasts to effortlessly satisfy their cravings. In a market dominated by meals, we're carving out a special place just for snacks.

3. Ideation and Proposed Solutions

3.1. Empathy Map



3.1. Ideation and Brainstorming



4. Requirements Analysis

4.1 Functional Requirement

Discovering vendors, exploring a variety of delectable food options, adding them to your virtual cart, and seamlessly completing the payment process—our app streamlines the entire experience. You can easily search for your preferred vendors, peruse their tempting offerings, select your desired items, and conveniently proceed to checkout. The payment process is designed to be smooth and hassle-free, ensuring that your culinary cravings are satisfied with efficiency and ease.

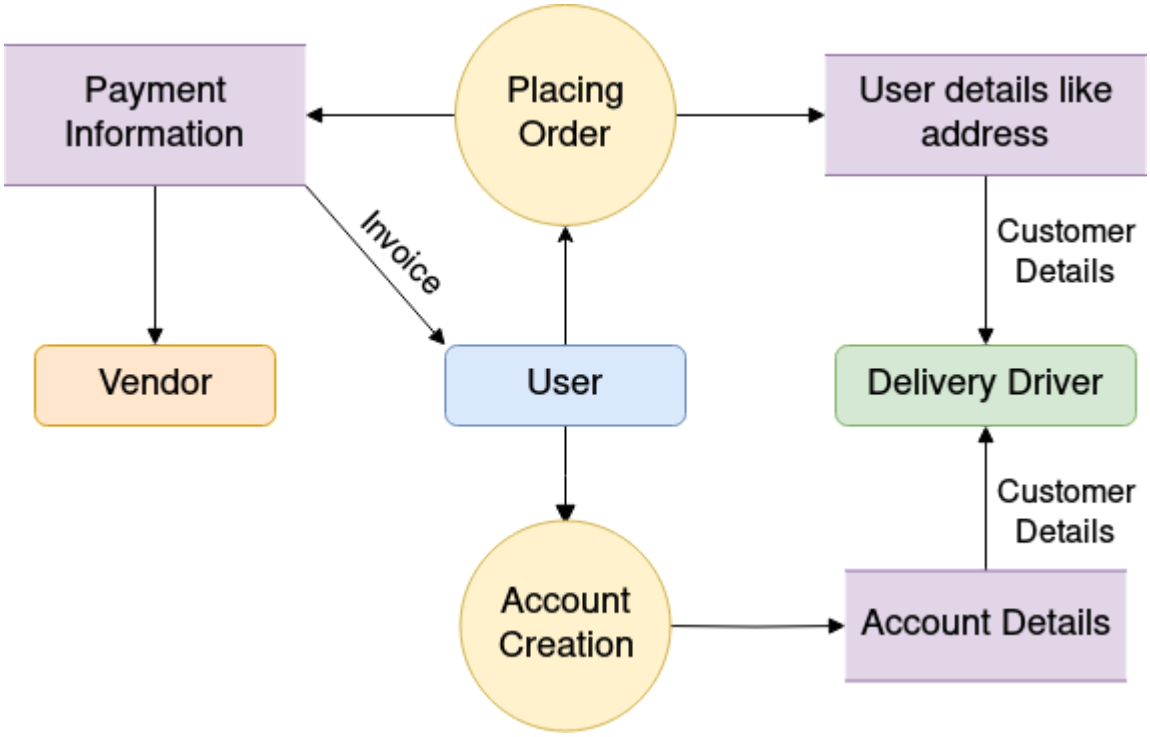
4.2 Non Functional Requirement

Our app boasts swift and efficient performance, ensuring a seamless experience even in areas with slower internet connections. It's designed to maintain usability and responsiveness, catering to users who may not have access to high-speed internet. Whether you're in a bustling city or a remote location, our app is committed to delivering a consistently smooth and user-friendly interface, regardless of your internet speed.

5. Project Design

5.1.1. Data Flow Diagram

The data flow diagram of our project is as follows



5.1.2. User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story/Task	Acceptance Criteria	Priority	Release
Customer (Mobile user)	Registration	USN 1	As a user, I can register for food delivery app Snack Squad by entering my email, password, and confirming my password	I can access my account, And food item list.	High	Sprint-1
		USN 2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN 3	As a user, I can register for the		Low	Sprint-2

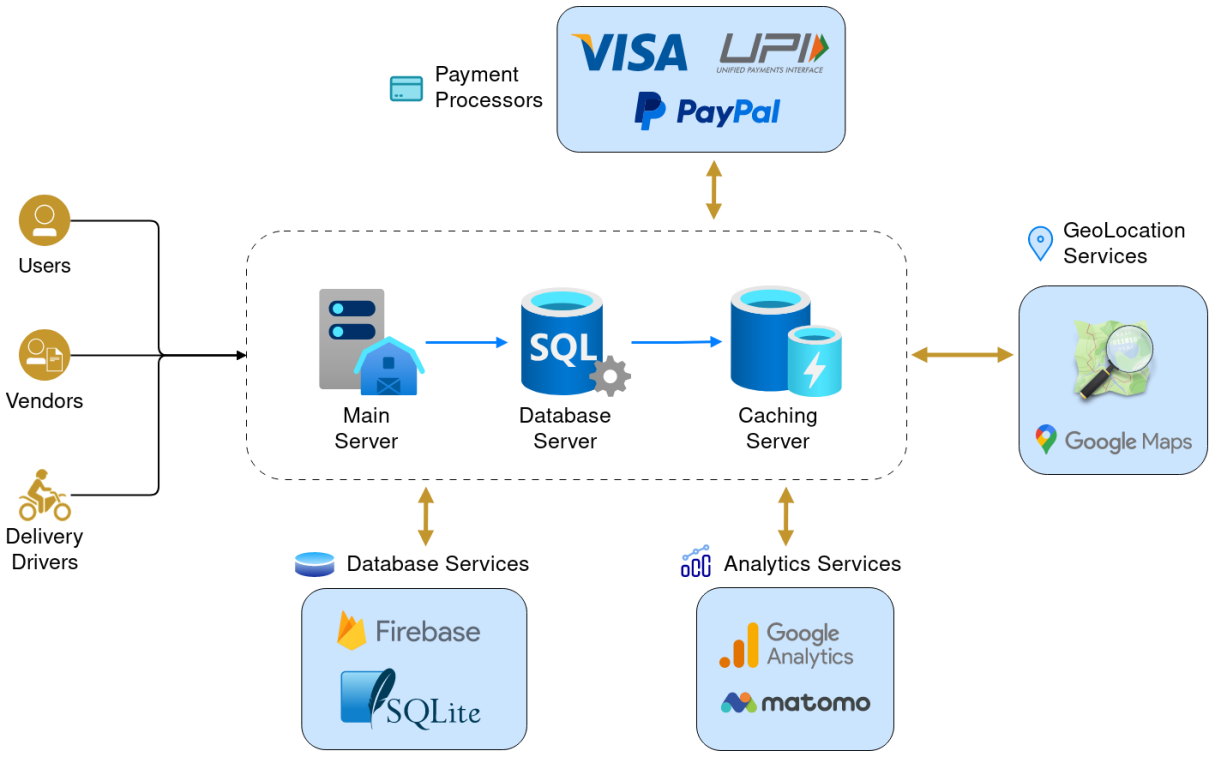
			applicati on through google			
	Login	USN 4	As a user, I can log into the applicati on by entering email & password		High	Sprint-1
	Order	USN 5	As a user, I can easily order food from the app	I can access the vendors, with their menus and photos	High	Sprint-2
Customer Care Executive	Help Line	USN 6	I can easily call helpline number and resolve all my issues	Contact page	Medium	Sprint-4
Administr ator	Managem ent	USN 7	I can add and modify items in the database	Add and modify items	High	Sprint-2

5.2. Solution Architecture

Our snack delivery app can be divided into the following layers :–

- **Frontend:** This layer is responsible for interacting with the user and displaying the application's UI.
- **Backend:** This layer contains the business logic of the application, such as managing user accounts, processing orders, and tracking deliveries.
- **Database:** This layer is responsible for interacting with the database to store and retrieve data.

The following diagram shows a high-level overview of the solution architecture:



Frontend –

The frontend layer is responsible for the following:

- Displaying the application's UI to the user
- Handling user input, such as search queries, order placement, and payment
- Providing feedback to the user, such as order confirmation and delivery status

It is implemented using Kotlin and Jetpack Compose. It follows the Material Design 3 specifications.

Backend –

The backend layer is implemented using Kotlin and it is responsible for the following:

- Managing user accounts
- Processing orders
- Tracking deliveries
- Communicating with the frontend and the database.

We can use services like Google Analytics and Matomo to get analytics about the users to decide our future priorities and areas of improvement.

We should add an option to pay by cash on delivery and also common payment processors like Visa and Paypal so people can use their Debit or Credit Cards for payment. Adding UPI payment option is also important as many people in India are using UPI apps like PayTM, Google Pay, PhonePe, etc to pay instead of Debit/Credit Cards.

GeoLocation services like Google Maps or OpenStreetMaps are important to select the delivery location and display the current location of the delivery driver.

Database –

The data access layer is responsible for the following:

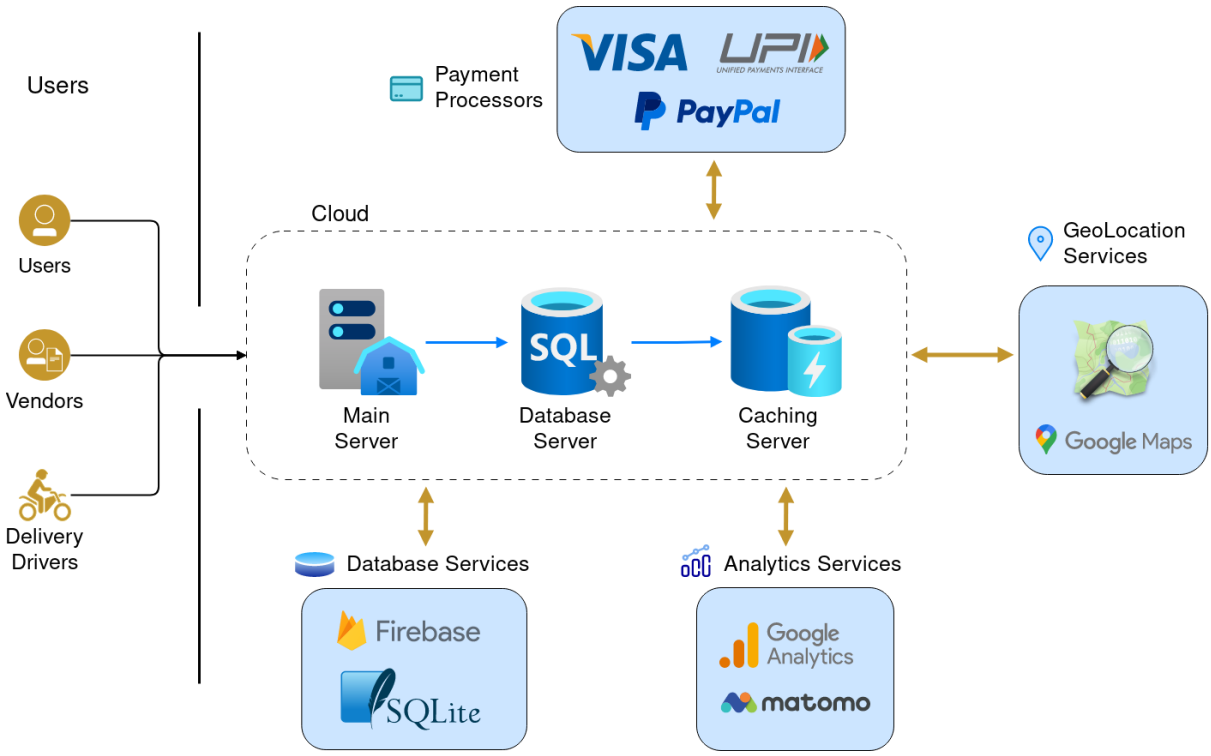
- Storing and retrieving data from the database
- Performing complex queries
- Managing database transactions

The databases are implemented using SQLite for storing a local database for things like items in the user's cart and Firebase Firestore for Cloud database which stores the list of vendors and their menu items.

6. Project Planning and Scheduling

6.1. Technical Architecture

The technical architecture of our application is as follows -



Components and Technologies –

S. No	Component	Description	Technology
1.	User Interface	How the user interacts with the application.	Kotlin, Jetpack Compose
2.	Application Logic	Logic for backend processes in the application.	Kotlin
3.	Local Database	Database stored in the phone's storage	SQLite, Room
4.	Cloud Database	Database stored in the cloud	Firebase Firestore
5.	External API	API used for payment	Razorpay

Application Characteristics –

S. No	Characteristics	Description	Technology
1.	Open Source Frameworks	Open Source frameworks used in the application	Jetpack Compose
2.	Security Implementation	Security methods implemented in the application	Secure payment transactions using Razorpay
3.	Availability	The app is very robust to failure	Google Cloud to store the database.
4.	Performance	App tries to be highly performant	Few requests are made to the server. Caching is used to increase performance.

6.2. Sprint Planning and Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	2
Sprint-1	Registration	USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	2
Sprint-2	Registration	USN-4	As a user, I can register for the application through Gmail	2	Low	1
Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password	2	High	1
Sprint-2	Dashboard	USN-6	As a user, I can easily order food from the app	4	High	3
Sprint-3	Administrator	USN-7	I can add and modify items in the database	2	Medium	2
Sprint-4	Customer Care	USN-8	I can easily call helpline number and resolve all my issues	2	Medium	4

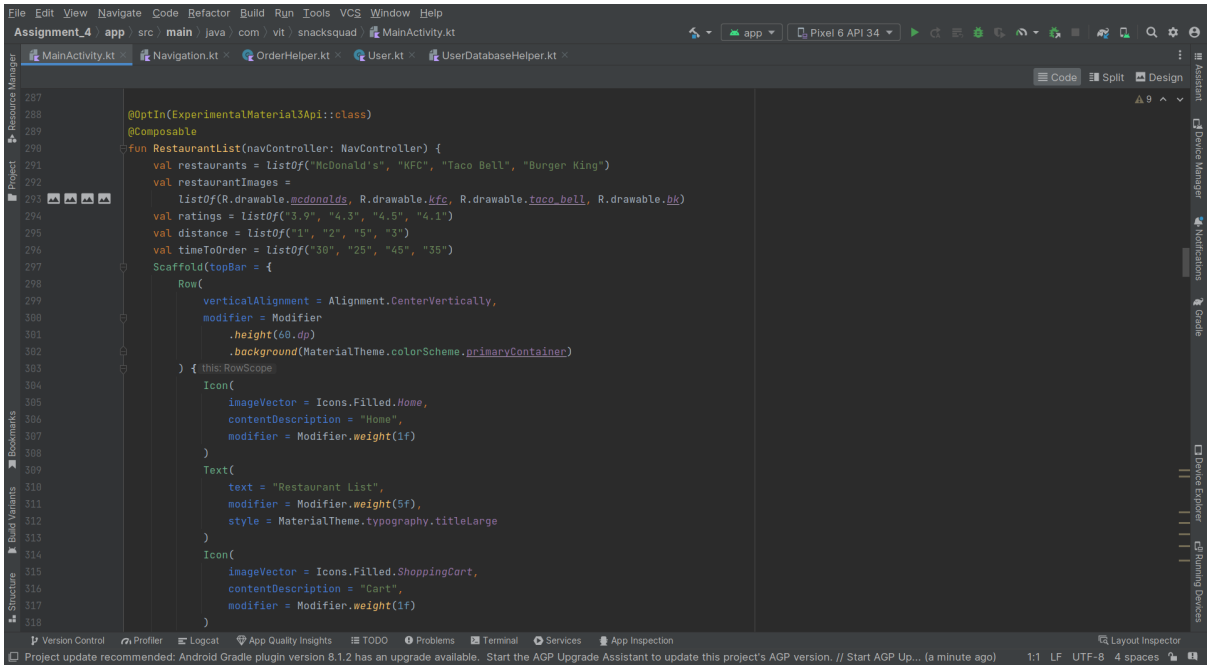
6.3. Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	5	6 Days	18 Oct 2023	23 Oct 2023	5	23 Oct 2023
Sprint-2	6	6 Days	24 Oct 2023	29 Oct 2023	6	29 Oct 2023
Sprint-3	2	4 Days	30Oct 2023	2 Nov 2023	2	2 Nov 2023
Sprint-4	2	4 Days	3 Nov 2023	6 Nov 2023	2	6 Nov 2023

7. Coding and Solutioning

7.1. Feature 1 - Find Snack Vendor and order food

The app allows us to find new snack vendors in our city, see their menu and order get food delivery.



```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun RestaurantList(navController: NavController) {
    val restaurants = listOf("McDonald's", "KFC", "Taco Bell", "Burger King")
    val restaurantImages =
        listOf(R.drawable.mcdonalds, R.drawable.kfc, R.drawable.taco_bell, R.drawable.bk)
    val ratings = listOf("3.9", "4.3", "4.5", "4.1")
    val distance = listOf("1", "2", "5", "3")
    val timeToOrder = listOf("30", "25", "45", "35")
    Scaffold(topBar = {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier
                .height(60.dp)
                .background(MaterialTheme.colorScheme.primaryContainer)
        ) { this: RowScope
            Icon(
                imageVector = Icons.Filled.Home,
                contentDescription = "Home",
                modifier = Modifier.weight(1f)
            )
            Text(
                text = "Restaurant List",
                modifier = Modifier.weight(5f),
                style = MaterialTheme.typography.titleLarge
            )
            Icon(
                imageVector = Icons.Filled.ShoppingCart,
                contentDescription = "Cart",
                modifier = Modifier.weight(1f)
            )
```

```
}, content = { it: PaddingValues
    LazyColumn(contentPadding = it, content = { this: LazyListScope
        itemsIndexed(restaurants) { this: LazyItemScope x, item ->
            Card(
                onClick = { navController.navigate( route: "Menu/" + restaurants[x]) },
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 20.dp, vertical = 10.dp),
                elevation = CardDefaults.cardElevation(20.dp)
            ) { this: ColumnScope
                Column(
                    modifier = Modifier.fillMaxSize(), verticalArrangement = Arrangement.Top
                ) { this: ColumnScope
                    Image(
                        painter = painterResource(id = restaurantImages[x]),
                        contentDescription = "McDonald's",
                        contentScale = ContentScale.FillWidth,
                        modifier = Modifier.height(200.dp)
                    )
                    Row(
                        verticalAlignment = Alignment.CenterVertically,
                        horizontalArrangement = Arrangement.Start,
                        modifier = Modifier.padding(
                            horizontal = 15.dp, vertical = 10.dp
                        )
                    ) { this: RowScope
                        Text(
                            text = item,
                            style = MaterialTheme.typography.headlineMedium,
                            modifier = Modifier.weight(4f)
                        )
                        Badge(
```

```kotlin
package com.vit.snacksquad

data class Order(
    var userName: String? = null,
    var userEmail: String? = null,
    var phoneNo: String? = null,
    var completed: Boolean = false,
    var address: String? = null,
    var paymentMethod: String? = null,
    var restaurant: String? = null,
    var date: String? = null
)
```

```kotlin
    fun insertUser(user: User) {
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_FIRST_NAME, user.firstName)
        values.put(COLUMN_EMAIL, user.email)
        values.put(COLUMN_PASSWORD, user.password)
        db.insert(TABLE_NAME, nullColumnHack: null, values)
        db.close()
    }

    @SuppressLint("Range")
    fun getUserByUsername(username: String): User? {
        val db = readableDatabase
        val cursor: Cursor =
            db.rawQuery( sql: "SELECT * FROM $TABLE_NAME WHERE $COLUMN_FIRST_NAME = ?", arrayOf(username))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
        return user
    }
}
```

7.2. Database Schema

SQLite

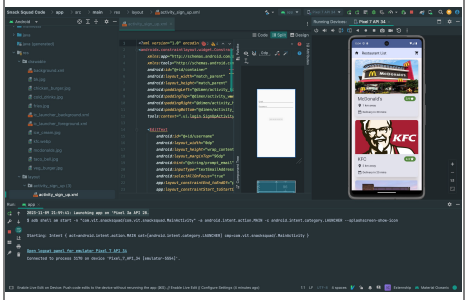
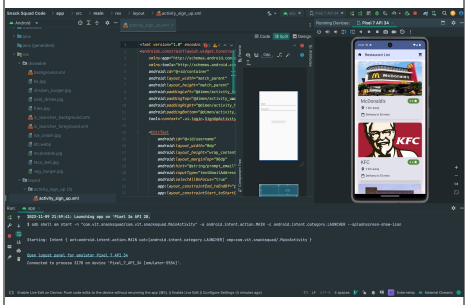
user_table	
PK	column_id: Int
	first_name: String
	email: String
	password: String

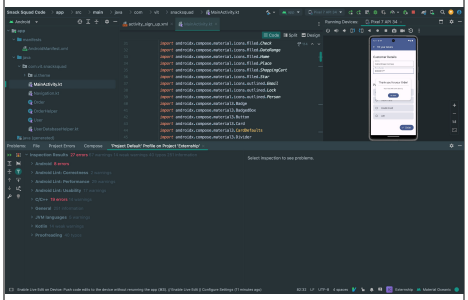
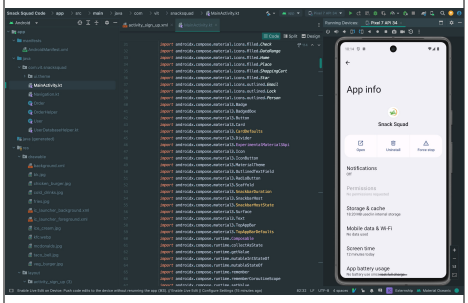
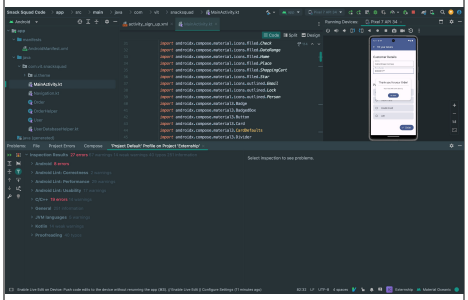
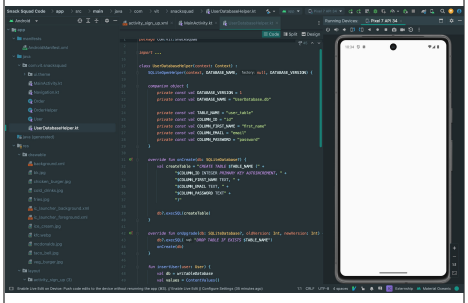
Firebase
Firestore

Orders	
PK	order_id: Int
	userName: String
	userEmail: String
	phoneNo: String
	completed: boolean
	address: String
	paymentMethod: String
	restaurant: String
	date: String

8. Performance Testing

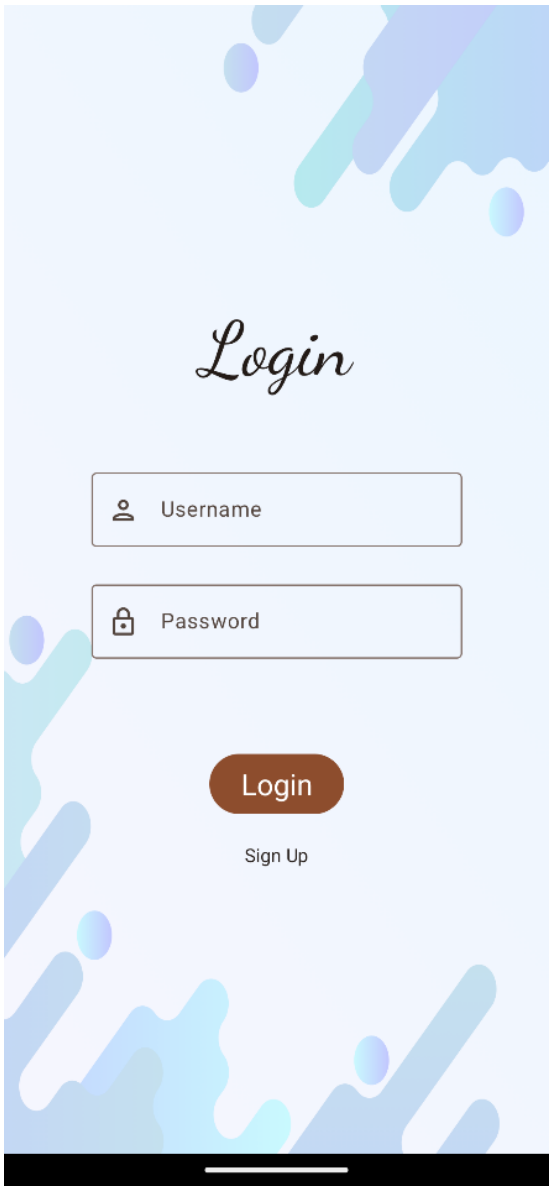
8.1. Performance Metrics

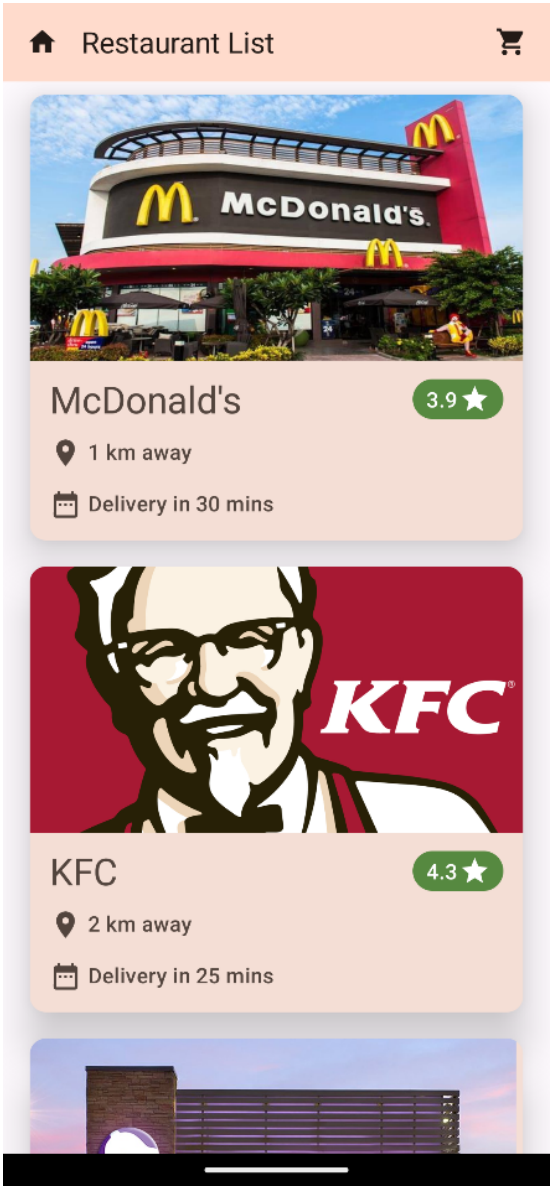
S.No.	Parameter	Values	Screenshot
1.	Metrics	App Launch Time- 2023-11-09 21:59:41: Launching app on 'Pixel 7 API 34. For "Snack Squad," key app launch time considerations include optimizing code for a swift startup, efficient caching, and preloading essential resources. Balancing a delightful splash screen and progressive loading ensures a speedy and user-friendly snacking experience.	
2.	Metrics	Screen Render Time-90ms	

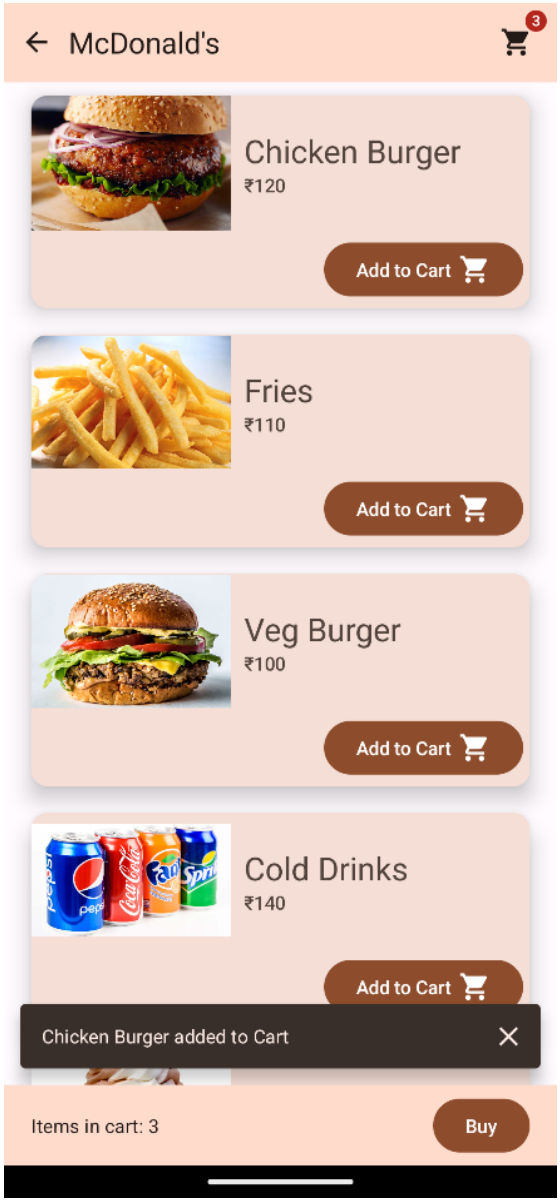
3.	Metrics	Code Quality-27 weak errors	
4.	Usage	App Size-40.79 MB Customer Experience- My family and I have all used and tried the app, and we found it to have a user-friendly interface. We are entirely satisfied with the app, and our overall experience has been good.	
5.	Performance	Error and Crash Rates- Android 8 errors C/C++ 19 error Crash rate-((Number of crash occurrences / total number of sessions)) * 100 =0.1%. And per user it is about <1%	
		Database Query Performance-1ms	

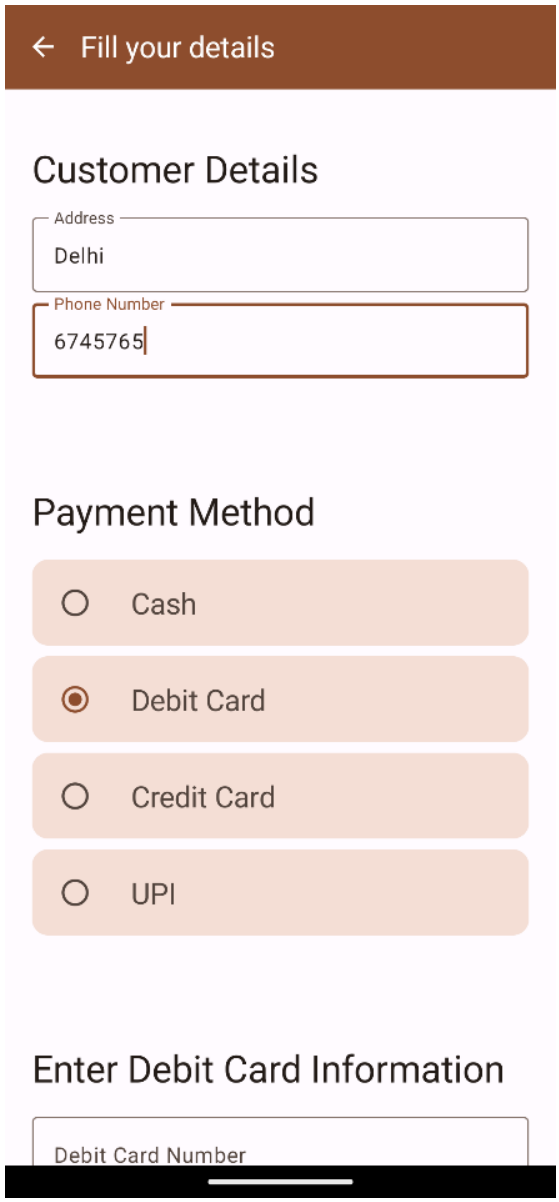
9. Results

9.1. Output Screenshots






McDonald's
3
Chicken Burger
₹120
Add to Cart
Fries
₹110
Add to Cart
Veg Burger
₹100
Add to Cart
Cold Drinks
₹140
Add to Cart
Chicken Burger added to Cart
Items in cart: 3
Buy


Fill your details
Customer Details
Address
Delhi
Phone Number
6745765
Payment Method
Cash
Debit Card
Credit Card
UPI
Enter Debit Card Information
Debit Card Number

10. ADVANTAGES AND DISADVANTAGES

Advantages of Snack Ordering Apps:

- Convenience: Snack ordering apps make it easy to order snacks from your favorite stores or restaurants without having to leave your home or office. You can simply browse the app's menu, add items to your cart, and checkout with a few taps.
- Variety: Snack ordering apps offer a wide variety of snacks to choose from, including both healthy and unhealthy options. This makes it easy to find the perfect snack for any occasion.
- Discounts and promotions: Snack ordering apps often offer discounts and promotions on their products. This can save you money on your snack purchases.
- Tracking your orders: Snack ordering apps allow you to track your orders in real time. This way, you always know when your snacks will arrive.

Disadvantages of Snack Ordering Apps

- Unhealthy choices: Snack ordering apps can make it easy to overindulge in unhealthy snacks. This is because the apps often display tempting pictures of snacks and make it easy to add items to your cart without thinking about the calories or other nutritional information.
- Food waste: Snack ordering apps can contribute to food waste. This is because it is easy to over-order snacks, especially if you are ordering from multiple apps at the same time.
- Reliability: Snack ordering apps may not always be reliable. For example, the app may crash or your order may be delayed.

11. CONCLUSION

Hence a snack ordering app with an add to cart feature and a checkout feature is made. So its easy to order snacks door to door.

12. FUTURE SCOPE

In the future we can add more snack delivery partners. Also add better payment gateway and allow for pre orders.

13.APPENDICES

The snack squad app is made which helps to order snacks in a faster and efficient way . It includes various features like add to cart and checkout.I has helped the user get a faster access to snack ordering.

It is quite affordable and provides quality snacks to the user .

14. SOURCE CODE

https://github.com/smartinternz02/SI-GuidedProject-587574-1697029005/tree/main/Project%20Development%20Phase/Snack%20Squad%20Code

15. Project Demo Link

https://drive.google.com/file/d/1kO1o1aBAXLrv0FhZgkQFECxNHXIMDymo/view