



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

9th November 2023

Snack Squad: A Customizable Snack Ordering

By

D SRUJAY - 21BCE1975

B JASWANTH SAI SIMHA GANESH - 21BRS1486

K VAMSHI KRISHNA - 21BPS1126

SREERAMA SRI SAI MANJUNATH-21BCE5031

Project Report Format

1. INTRODUCTION

- 1.1 Project Overview
- 1.2 Purpose

2. LITERATURE SURVEY

- 2.1 Existing problem
- 2.2 References
- 2.3 Problem Statement Definition

3. IDEATION & PROPOSED SOLUTION

- 3.1 Empathy Map Canvas
- 3.2 Ideation & Brainstorming

4. REQUIREMENT ANALYSIS

- 4.1 Functional requirement
- 4.2 Non-Functional requirements

5. PROJECT DESIGN

- 5.1 Data Flow Diagrams & User Stories
- 5.2 Solution Architecture

6. PROJECT PLANNING & SCHEDULING

- 6.1 Technical Architecture
- 6.2 Sprint Planning & Estimation
- 6.3 Sprint Delivery Schedule

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

- 7.1 Feature 1
- 7.2 Feature 2
- 7.3 Database Schema (if Applicable)

8. PERFORMANCE TESTING

- 8.1 Performance Metrics

9. RESULTS

- 9.1 Output Screenshots

10. ADVANTAGES & DISADVANTAGES

11. CONCLUSION

12. FUTURE SCOPE

13. APPENDIX

Source Code

GitHub & Project Demo Link

Introduction

Date	09 November 2023
Team ID	Team-590962
Project Name	SNACK SQUAD

13.1 Project Overview

In response to the increasing demand for convenient and personalized snack experiences, Snack Squad aims to introduce a user-friendly mobile application. This app will cater to the cravings of busy professionals, students, and snack enthusiasts by providing a customizable platform for ordering and delivering a variety of snacks. The primary objective is to streamline the snack ordering process, offering users a diverse range of options while ensuring a seamless and enjoyable experience.

Key Features:

User Registration and Authentication: Secure account creation and login functionality for users.

Customizable Snack Selection: A vast array of snacks categorized by type, cuisine, and dietary preferences. Customization options allowing users to personalize their snack orders.

13.2 Purpose

The purpose of Snack Squad, a Customizable Snack Ordering and Delivery App, is to revolutionize the way users experience and satisfy their snack cravings. In a fast-paced world where convenience is paramount, Snack Squad aims to provide a seamless and personalized solution for individuals looking to enjoy a diverse range of snacks. The app's primary purpose is to offer users a user-friendly platform that caters to their unique preferences and dietary choices.

The app's goal is not only to streamline the snack procurement process but also to create an engaging and trustworthy ecosystem. Features like user reviews and ratings contribute to transparency, allowing customers to make informed decisions about their snack choices and delivery experiences.

Literature Survey

Date	09 November 2023
Team ID	Team-590962
Project Name	SNACK SQUAD

2.1 Existing problem

Addressing existing problems such as technical issues, delivery logistics, snack variety, security concerns, and communication strategies will be crucial for Snack Squad to thrive in a competitive market and fulfill its promise of delivering a customizable and delightful snack ordering experience.

2.2 Problem Statement Definition

the problem statement for Snack Squad encompasses addressing technical disruptions, optimizing logistical processes, enhancing snack variety, fortifying security measures, and improving customer communication.

Successfully resolving these challenges will be pivotal in realizing the app's vision of delivering a customizable and delightful snack ordering experience for users.

Ideation Phase

Brainstorm & Idea Prioritization Template

Date	18 October 2022
Team ID	Team-590962
Project Name	SNACK SQUAD
Maximum Marks	4 Marks

Brainstorm & Idea Prioritization Template:


Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Reference: <https://www.mural.co/templates/empathy-map-canvas>

Step-1: Team Gathering, Collaboration and Select the Problem Statement

Template



Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

🕒 10 minutes to prepare

🕒 1 hour to collaborate

👤 2-8 people recommended

➔

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

🕒 10 minutes

A

Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B

Set the goal
Think about the problem you'll be focusing on solving in the brainstorming session.

C

Learn how to use the facilitation tools
Use the Facilitation Superpowers to run a happy and productive session.

Open article

➔

1

Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

🕒 5 minutes

PROBLEM

This document is intended to help you define the problem and prepare to address the problem in your brainstorming session. It is designed to help you define the problem in a way that is specific, measurable, achievable, relevant, and time-bound. It is also designed to help you define the problem in a way that is framed as a "How Might We" statement. This will be the focus of your brainstorming session.

Key rules of brainstorming

To run an smooth and productive session

🗨️ Stay in topic.

💡 Encourage wild ideas.

👂 Defer judgment.

👂 Listen to others.

🗨️ Go for volume.

👁️ If possible, be visual.



Need some inspiration?
See a finished version of this template to help start your work.

Open example ➔

Step-2: Brainstorm, Idea Listing and Grouping

3 Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

Person 1

Person 2

Person 3

Person 4

Person 5

Person 6

Person 7

Person 8

TIP You can select a sticky note and drag the pencil icon to start drawing.

4 Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP Add comparable tags to sticky notes to make it easier to find, browse, organize, and categorize important data as themes within your mind.

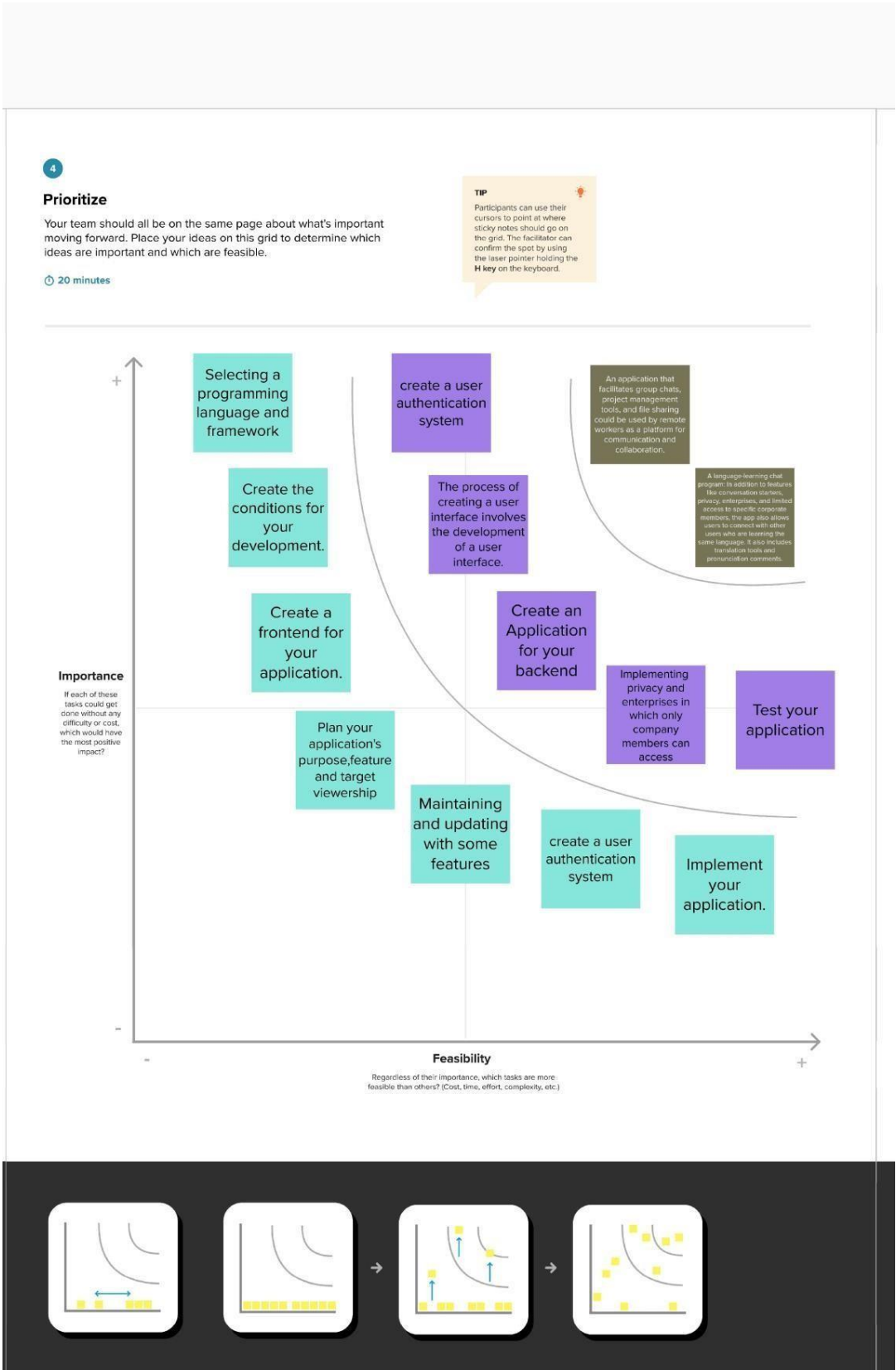
An application that facilitates group chats, project management tools, and file sharing could be used by remote workers as a platform for communication and collaboration.

A language-learning chat program. In addition to features like conversation starters, privacy, enterprises, and limited access to specific corporate members, the app also allows users to connect with other users who are learning the same language. It also includes translation tools and pronunciation comments.

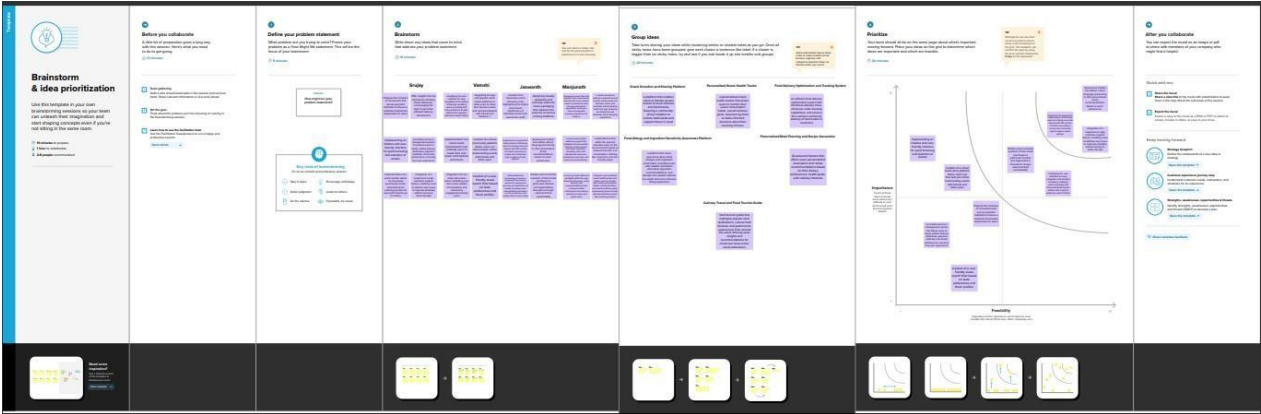
→

→

Step-3: Idea Prioritization



All Steps :



Ideation Phase

Empathize & Discover

Date	18 October 2022
Team ID	Team-590962
Project Name	SNACK SQUAD
Maximum Marks	4 Marks

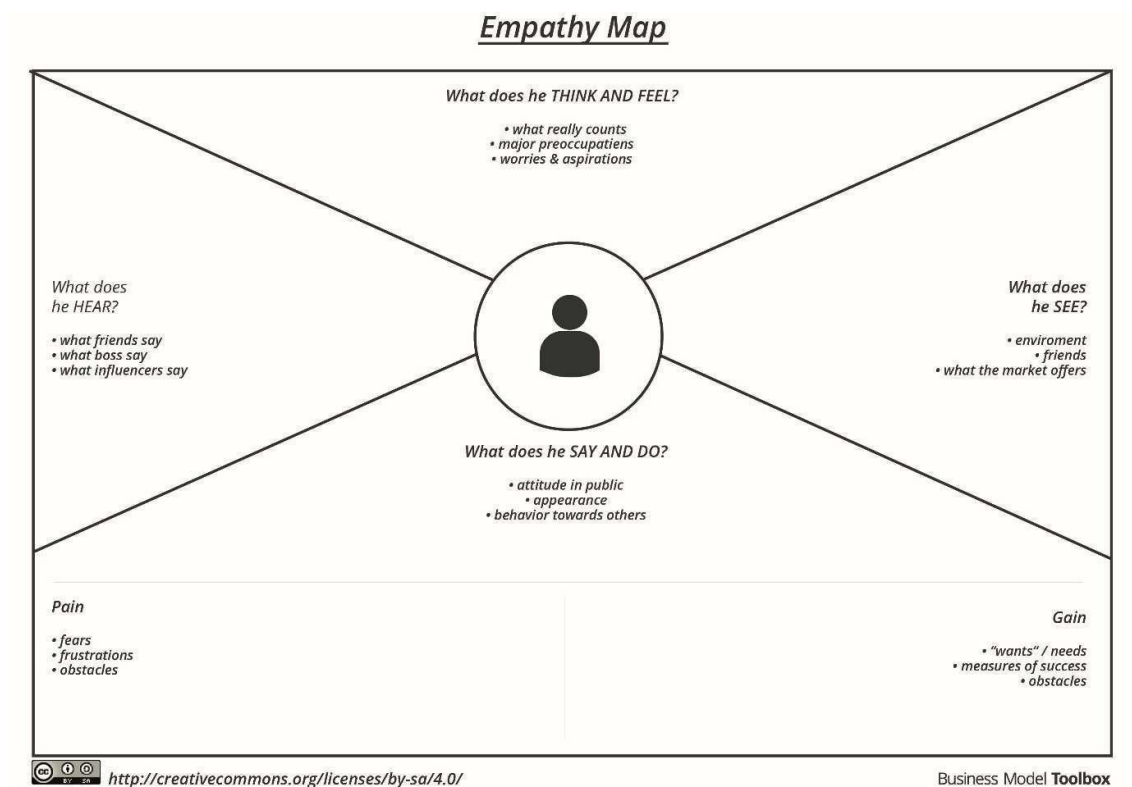
Empathy Map Canvas:

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.

It is a useful tool to help teams better understand their users.

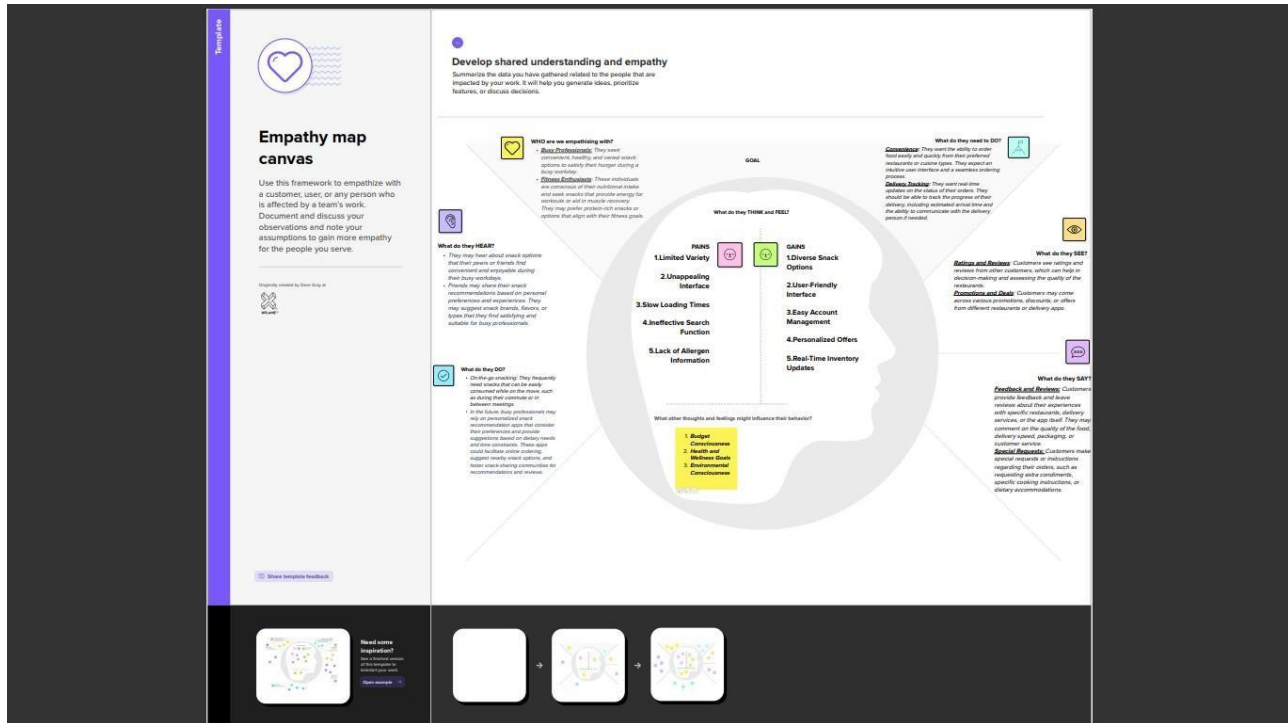
Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

Example:



Reference: <https://www.mural.co/templates/empathy-map-canvas>

Snack Squad - A Customizable Snack Ordering and Delivery App



Project Design Phase-II
Data Flow Diagram & User Stories

Date	22-10-2023
Team ID	590962
Project Name	Snacks Squad
Maximum Marks	4 Marks

Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

Data Flow:

User Registration Data: Flows from the Users to User Registration and Profile Management.

Snack Catalog Data: Flows from the Catalog Management to the Catalog Database and to the Users.

Order Data: Flows between Users, Order Management, and the Order Database.

Recommendation Data: Flows from User Profiles Database to User Recommendations.

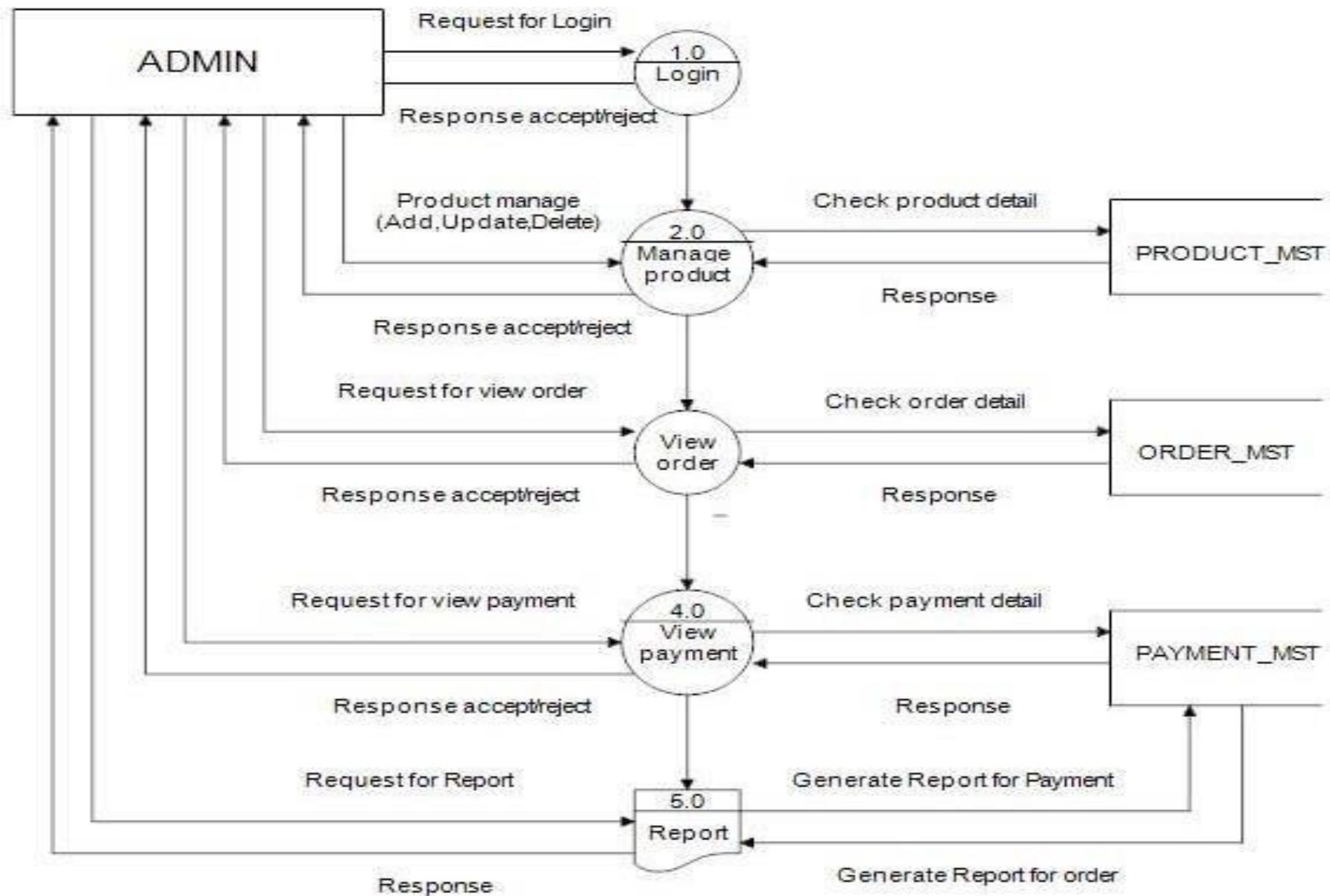
Notification Data: Flows between Notifications and Communication processes and the Notifications Database.

Search and Filter Data: Flows between Users, Search and Filtering, and the Snack Catalog Database.

Payment Data: Flows between Users, Payment Processing, and external payment systems.

Review and Feedback Data: Flows between Users, User Feedback and Reviews, and the Reviews and Feedback Database.

Support Request Data: Flows between Users and User Support.



User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customers	User Account Management	USN-1	As a customer, I want to be able to create a user profile with my name, contact information, and address so that I can place food orders and have them delivered to the correct location.	Successfully created a user profile with name, contact information, and address.	High	Sprint 1
Customers	Ordering	USN-2	As a customer, I want to add items to my cart, review my order, specify any special instructions, and place the order for delivery or pickup.	Able to add items to the cart, review order, and place the order.	High	Sprint 1
Restaurants	Menu Management	USN-3	As a restaurant, I want to create and manage my menu, including adding, updating, and removing items, to keep it up-to-date for customers.	Successfully added, updated, or removed menu items.	High	Sprint 2
Customers	Payment	USN-4	As a customer, I want to pay for my order online through various payment methods, including credit card, PayPal, or digital wallets.	Successfully made a payment using various payment methods.	High	Sprint 3
Delivery Personnel	Order Delivery	USN-5	As a delivery person, I want to receive order details, navigate to the customer's location, and mark the order as delivered upon completion.	Successfully received order details and marked orders as delivered.	High	Sprint 3
Customers	Order History	USN-6	As a customer, I want to view my order history, including past orders and their statuses, for reference and reordering.	Able to view order history and reorder from it.	medium	Sprint 4
Customers	Ratings and Reviews	USN-7	As a customer, I want to provide ratings and reviews for the food I've ordered to help other customers make informed decisions.	Able to provide ratings and reviews for food items.	medium	Sprint 4
Administrators	Admin Dashboard	USN-8	As an administrator, I want access to an admin dashboard to manage user accounts, resolve disputes, and oversee the app's functionality and security.	Successfully accessed and performed administrative tasks through the admin dashboard.	high	Sprint 5

Administrators	Testing & quality assurance	USN-8	conduct thorough testing of the model and web interface to identify and report any issues or bugs. fine-tune the model hyperparameters and optimize its performance based on user feedback and testing results.	we could create web application	medium	Sprint 5
----------------	-----------------------------	-------	---	---------------------------------	--------	----------

Project Design Phase-I
Solution Architecture

Date	22 September 2023
Team ID	590962
Project Name	Snacks Squad
Maximum Marks	4 Marks

Solution Architecture:

App streamlines access to a wide array of wholesome snack choices, promoting nutritional balance and fostering a culture of mindful eating. By offering a user-friendly platform for personalized and convenient snack selection, it encourages users to make informed dietary decisions that align with their health goals and preferences. With its emphasis on diverse and nutritious options, the app contributes to improved overall well-being, instilling a sense of wellness and satisfaction among users. Through its efficient and accessible approach, it cultivates a healthier snacking routine, ultimately leading to a positive impact on users' lifestyles and long-term health. Moreover, by encouraging the consumption of wholesome snacks, the app contributes to a broader societal shift towards healthier eating habits and wellness-conscious living.

Solution Architecture Diagram

User Interface Layer
(Mobile Application Interface)
Interactive and User-Friendly Design
Customizable Snack Selection Options

Business Logic Layer
(Application & Database Servers)
Inventory Management Systems
Recommendation Engines
Secure Order Processing

Networking Layer
(APIs, Push Notifications, CDN)

Seamless Communication via APIs
Real-time Push Notifications for Orders
Efficient Content Delivery via CDN

Data Management Layer
(User Profiles, Inventory, Orders)
Secure and Efficient Data Storage
Reliable User Profile Management
Streamlined Order History Maintenance

Personalization & Customization Layer

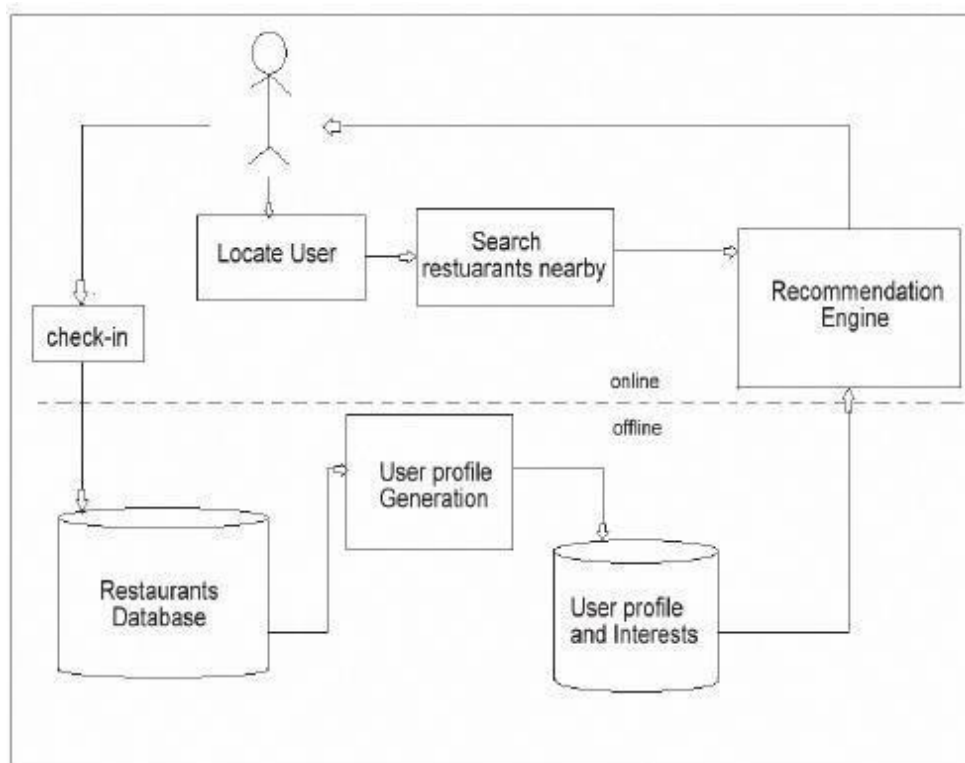
Personalized Snack Recommendations
Customization Options for Users
Dietary Preference Management

Third-Party Integrations Layer

Secure Payment Gateway Integration
Seamless Delivery Service Integration
Comprehensive Analytics Integration

Scalability & Performance Layer

Load Balancers for User Traffic Management
Caching Mechanisms for Data Optimization
Autoscaling Capabilities for User Demand



Project Design Phase-I
Proposed Solution Template

Date	22 September 2023
Team ID	590962
Project Name	Snacks Squad
Maximum Marks	2 Marks

Proposed Solution Template:

Project team shall fill the following information in proposed solution template.

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	<i>The problem revolves around the limited availability of convenient, healthy snack options for individuals with busy lifestyles. Many traditional vending machines and snack options often lack variety and fail to accommodate different dietary needs, leading to unhealthy snacking habits and a lack of accessible healthier choice.</i>

2.	Idea / Solution description	<i>The proposed solution is a mobile application that provides users with a diverse selection of customizable, healthy snack options. Users can easily navigate the app to select, customize, and order snacks based on their preferences and dietary requirements. The app offers a convenient ordering process, with options for delivery or pick-up at easily accessible locations, making it a hassle-free snacking solution for users on the go.</i>
3.	Novelty / Uniqueness	<i>The app's uniqueness lies in its integration of a personalized nutrition recommendation system. This system considers user preferences, dietary restrictions, and health goals to suggest suitable snack options, encouraging users to make healthier choices. Additionally, the app leverages AI-driven algorithms to recommend innovative and unique snack combinations, introducing users to new and exciting healthy snack alternatives they may not have discovered.</i>
4.	Social Impact / Customer Satisfaction	<i>By promoting healthier snacking habits, the app contributes to improved overall well-being and encourages users to make mindful dietary choices. It aims to provide customer satisfaction by offering a wide range of snack options that cater to diverse dietary requirements, including vegan, gluten-free, and low-sugar options. This inclusivity ensures that a broader range of users can find suitable and satisfying snacks through the app, thereby enhancing overall customer satisfaction and loyalty.</i>

5.	Business Model (Revenue Model)	<p><i>The app's business model combines a subscription-based approach for premium users, granting them access to exclusive snacks and personalized nutrition plans. Meanwhile, regular users can access the app on a transaction-based model. Additionally, the app generates revenue through partnerships with local snack suppliers, providing an avenue for featured product placements and promotions within the app, thereby creating additional revenue streams.</i></p>
6.	Scalability of the Solution	<p><i>The app's architecture is built with scalability in mind, allowing for seamless expansion to multiple locations and integration with various third-party delivery services. The use of cloud-based infrastructure ensures the app can handle an increasing number of users and orders without compromising performance. Moreover, the app's design allows for the easy integration of new features and enhancements, ensuring it can adapt to evolving user needs and market demands over time.</i></p>

Coding And Solutioning

Date	09 November 2023
Team ID	Team-590962
Project Name	SNACK SQUAD

SIGNIN-

```
@Composable
fun AuthScreen(navController: NavHostController) {
    val context = LocalContext.current
    val sharedPreferences = context.getSharedPreferences("main", Context.MODE_PRIVATE)

    // Load the background image
    val backgroundImage = painterResource(id = R.drawable.bg)

    Box(
        modifier = Modifier.fillMaxSize()
    ) {
        Image(
            painter = backgroundImage,
            contentDescription = null,
            contentScale = ContentScale.Crop,
            modifier = Modifier.fillMaxSize()
        )
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(horizontal = 20.dp),
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            val selectedTab = remember {
                mutableStateOf(0)
            }
            TabLayout(
                selectedIndex = selectedTab.value,
                items = listOf(
                    "Sign In" to {
                        SignIn(
                            navController = navController,
                            sharedPreferences = sharedPreferences
                        )
                    },
                    "Sign Up" to {
                        SignUp(
                            navController = navController,
                            sharedPreferences = sharedPreferences
                        )
                    }
                ),
                onTabClick = {
                    selectedTab.value = it
                }
            )
        }
    }
}
```

```

    )
  }
}

```

```

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SignIn(
    navController: NavController,
    sharedPreferences: SharedPreferences
) {
    val rememberMeChecked = remember {
        mutableStateOf(false)
    }
    val email = remember {
        mutableStateOf("")
    }
    val password = remember {
        mutableStateOf("")
    }
    val showPassword = remember {
        mutableStateOf(false)
    }
    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Spacer(modifier = Modifier.height(22.dp))
        AppTextField(
            value = email.value,
            onValueChange = {
                email.value = it
            },
            label = "E-Mail Address"
        )
        Spacer(modifier = Modifier.height(16.dp))
        AppTextField(
            value = password.value,
            onValueChange = {
                password.value = it
            },
            label = "Password",
            visualTransformation = if (showPassword.value)
                VisualTransformation.None
            else
                PasswordVisualTransformation(),
            trailing = {
                Icon(
                    modifier = Modifier.clickable {
                        showPassword.value = !showPassword.value
                    },
                    painter = painterResource(
                        id = if (showPassword.value)
                            R.drawable.ic_eye_off else R.drawable.ic_eye_open
                    ),

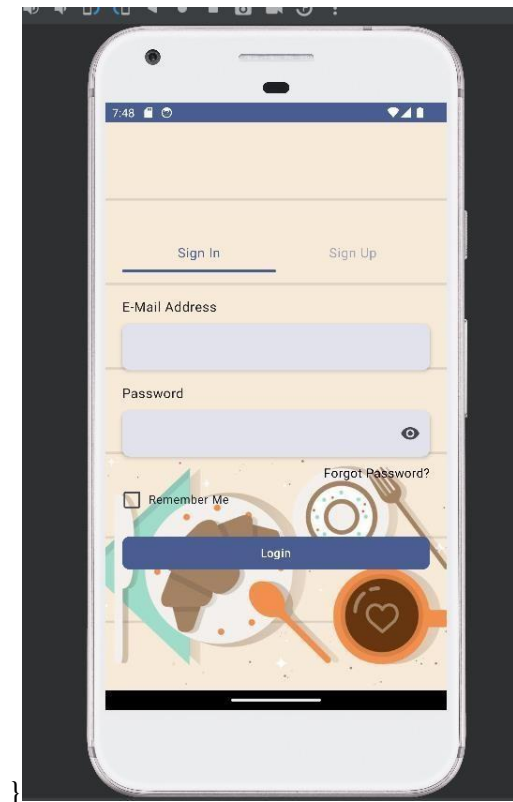
```



```

        contentDescription = null
    )
}
)
Spacer(modifier = Modifier.height(8.dp))
Row(
    modifier = Modifier.fillMaxWidth(),
    horizontalArrangement = Arrangement.End
) {
    Text(text = "Forgot Password?", color = Color.Black, fontSize = 15.sp)
}
Spacer(modifier = Modifier.height(8.dp))
Row(
    modifier = Modifier.fillMaxWidth(),
    horizontalArrangement = Arrangement.Start,
    verticalAlignment = Alignment.CenterVertically
) {
    CompositionLocalProvider(LocalMinimumTouchTargetEnforcement provides false) {
        Checkbox(checked = rememberMeChecked.value, onCheckedChange = {
            rememberMeChecked.value = it
        })
    }
    Text(
        text = "Remember Me",
        fontSize = 14.sp,
        modifier = Modifier.padding(horizontal = 8.dp)
    )
}
Spacer(modifier = Modifier.height(32.dp))
Button(
    modifier = Modifier
        .fillMaxWidth()
        .height(40.dp),
    onClick = {
        sharedPreferences.edit().apply {
            putBoolean("loggedIn", true)
            putString("email", email.value)
        }
        .apply()
        navController.navigate("home") {
            popUpTo(0)
        }
    }, shape = RoundedCornerShape(10.dp)
) {
    Text(text = "Login", fontFamily = ubuntuFont)
}
}

```



SIGNUP=

```

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SignUp(
    navController: NavController,
    sharedPreferences: SharedPreferences
) {
    val rememberMeChecked = remember {
        mutableStateOf(false)
    }
    val email = remember {
        mutableStateOf("")
    }
    val password = remember {
        mutableStateOf("")
    }
    val passwordRepeat = remember {
        mutableStateOf("")
    }
    val showPassword = remember {
        mutableStateOf(false)
    }
    val showPasswordRepeat = remember {
        mutableStateOf(false)
    }
    Column(
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Spacer(modifier = Modifier.height(22.dp))
    }
}

```

```

AppTextField(
    value = email.value,
    onValueChange = {
        email.value = it
    },
    label = "E-Mail Address"
)
Spacer(modifier = Modifier.height(16.dp))
AppTextField(
    value = password.value,
    onValueChange = {
        password.value = it
    },
    label = "Password",
    visualTransformation = if (showPassword.value)
        VisualTransformation.None
    else
        PasswordVisualTransformation(),
    trailing = {
        Icon(
            modifier = Modifier.clickable {
                showPassword.value = !showPassword.value
            },
            painter = painterResource(
                id = if (showPassword.value)
                    R.drawable.ic_eye_off else R.drawable.ic_eye_open
            ),
            contentDescription = null
        )
    }
)
Spacer(modifier = Modifier.height(16.dp))
AppTextField(
    value = passwordRepeat.value,
    onValueChange = {
        passwordRepeat.value = it
    },
    label = "Password",
    visualTransformation = if (showPasswordRepeat.value)
        VisualTransformation.None
    else
        PasswordVisualTransformation(),
    trailing = {
        Icon(
            modifier = Modifier.clickable {
                showPasswordRepeat.value = !showPasswordRepeat.value
            },
            painter = painterResource(
                id = if (showPasswordRepeat.value)
                    R.drawable.ic_eye_off else R.drawable.ic_eye_open
            ),
            contentDescription = null
        )
    }
)
Spacer(modifier = Modifier.height(32.dp))

```

```

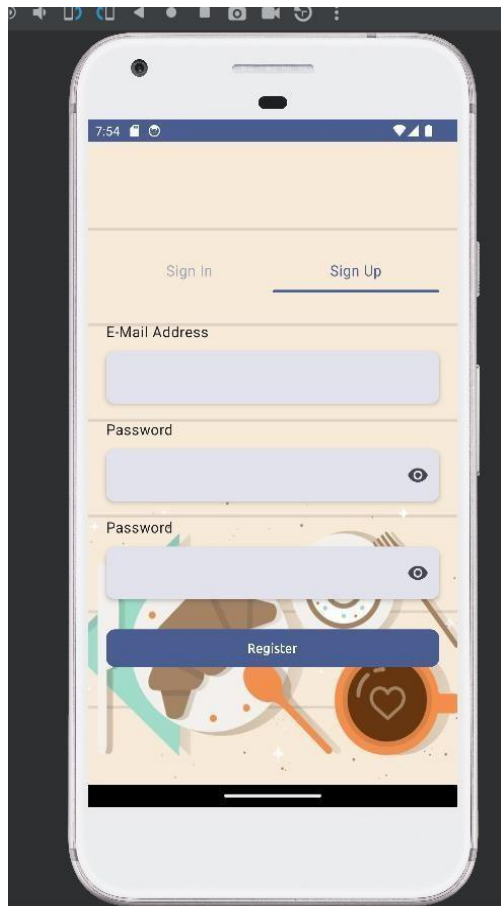
Button(
    modifier = Modifier
        .fillMaxWidth()
        .height(40.dp),
    onClick = {
        sharedPreferences.edit().apply {
            putBoolean("loggedIn", true)
            putString("email", email.value)
        }
        .apply()
        navController.navigate("home") {
            popUpTo(0)
        }
    }, shape = RoundedCornerShape(10.dp)
) {
    Text(text = "Register", fontFamily = ubuntuFont)
}
}

```

```

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AppTextField(
    value: String,
    onValueChange: (String) -> Unit,
    label: String,
    visualTransformation: VisualTransformation = VisualTransformation.None,
    trailing: (@Composable () -> Unit)? = null
) {
    Column {
        Text(text = label)
        Spacer(modifier = Modifier.height(8.dp))
        Box(
            modifier = Modifier
                .shadow(3.dp, RoundedCornerShape(10.dp))
                .fillMaxWidth()
                .clip(RoundedCornerShape(10.dp))
        ) {
            TextField(
                modifier = Modifier.fillMaxWidth(),
                value = value,
                onValueChange = onValueChange,
                visualTransformation = visualTransformation,
                singleLine = true,
                colors = TextFieldDefaults.textFieldColors(
                    focusedIndicatorColor = Color.Transparent,
                    unfocusedIndicatorColor = Color.Transparent
                ),
                trailingIcon = trailing
            )
        }
    }
}

```



Homescreen_

```

@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun HomeScreen(navController: NavController, onFoodSelected: (Food) -> Unit, cartItems:
List<CartItem>) {
    val uiController = rememberSystemUiController()
    uiController.isStatusBarVisible = false

    Scaffold(
        topBar = {
            TopAppBar(
                title = {
                    Box(
                        modifier = Modifier.fillMaxWidth(),
                        contentAlignment = Alignment.Center
                    ) {
                        Text(text = "Our Menu", fontFamily = ubuntuFont)
                    }
                },
            ),
        navigationIcon = {
            IconButton(onClick = { }) {
                androidx.compose.material3.Icon(imageVector = Icons.Rounded.Menu, contentDescription =
"Menu")
            }
        },
        actions = {

```

```

        IconButton(onClick = { }) {
            androidx.compose.material3.Icon(imageVector = Icons.Outlined.FavoriteBorder,
contentDescription = "Fav icon")
        }

        Box(
            modifier = Modifier.clickable {

                val totalItems = cartItems.sumBy { it.quantity }

                navController.navigate("cartScreen/${cartItems.sumBy { it.quantity }}") {
                    launchSingleTop = true
                    restoreState = true
                }
            },
            contentAlignment = Alignment.Center
        ) {
            androidx.compose.material3.Icon(imageVector = Icons.Outlined.ShoppingCart,
contentDescription = "ShoppingCart icon")
            if (cartItems.isNotEmpty()) {
                Text(
                    text = cartItems.sumBy { it.quantity }.toString(),
                    modifier = Modifier
                        .background(MaterialTheme.colorScheme.primary)
                        .padding(4.dp),
                    color = Color.Black
                )
            }
        },
        backgroundColor = Color(0xFFE0A9A5),
        contentColor = Color.Black
    )
}
) {
    Box(
        modifier = Modifier.fillMaxSize()
    ) {
        Image(
            painter = painterResource(id = R.drawable.backgrounds),
            contentDescription = null,
            contentScale = ContentScale.Crop,
            modifier = Modifier.fillMaxSize()
        )

        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(horizontal = 8.dp)
        ) {
            Spacer(modifier = Modifier.height(16.dp))

            val selectedFoodType = remember {
                mutableIntStateOf(0)
            }

            val distinctFoods = foods.distinctBy { it.name }

```

```

val foodsState = remember { mutableStateListOf(*distinctFoods.toArray()) }
var cartItemsState by remember { mutableStateOf(cartItems) }
val onLikeChange: (Food) -> Unit = { food ->
    val index = foodsState.indexOfFirst { it.name == food.name }
    if (index != -1) {
        val updatedFood = foodsState[index].copy(liked = !food.liked)
        foodsState[index] = updatedFood

        val updatedCartItems = cartItemsState.toMutableList().apply {
            val existingCartItem = find { it.food == updatedFood }
            if (updatedFood.liked) {
                if (existingCartItem == null) {
                    add(CartItem(updatedFood, 1))
                } else {
                    val updatedCartItem = existingCartItem.copy(quantity = existingCartItem.quantity +
1)
                    set(indexOf(existingCartItem), updatedCartItem)
                }
            } else {
                existingCartItem?.let {
                    if (it.quantity > 1) {
                        val updatedCartItem = it.copy(quantity = it.quantity - 1)
                        set(indexOf(existingCartItem), updatedCartItem)
                    } else {
                        remove(existingCartItem)
                    }
                }
            }
        }

        cartItemsState = updatedCartItems
    }
}

```

```

Spacer(modifier = Modifier.height(40.dp))
TabLayout(
    items = listOf(
        "Hot" to {
            Foods(
                items = foodsState.filter { it.type == FoodType.Hot },
                onLikeChange = onLikeChange,
                onTap = { food ->
                    onFoodSelected(food)
                }
            )
        },
        "Sweet" to {
            Foods(
                items = foodsState.filter { it.type == FoodType.Sweet },
                onLikeChange = onLikeChange,
                onTap = { food ->
                    onFoodSelected(food)
                }
            )
        },
    ),
)

```



```

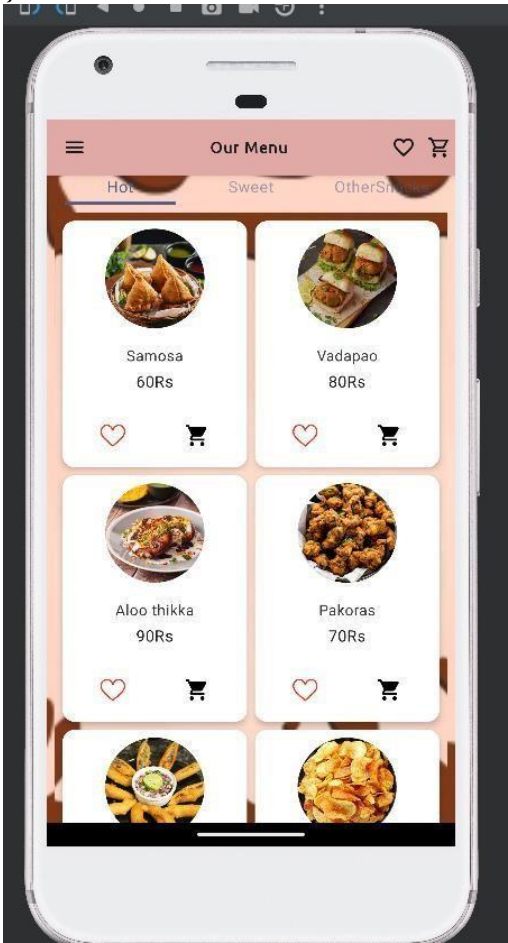
        defaultElevation = 4.dp
    ),
    colors = CardDefaults.cardColors(
        containerColor = Color.White
    )
) {
    Column(
        modifier = Modifier.fillMaxWidth(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Spacer(modifier = Modifier.height(8.dp))
        Image(
            modifier = Modifier
                .size(100.dp)
                .clip(CircleShape),
            painter = painterResource(id = food.image),
            contentDescription = food.name,
            contentScale = ContentScale.Crop
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(text = food.name, fontSize = 15.sp, color = Color(0xff383838))
        Spacer(modifier = Modifier.height(2.dp))
        Text(text = "${food.price}Rs")
        Spacer(modifier = Modifier.height(10.dp))
    }

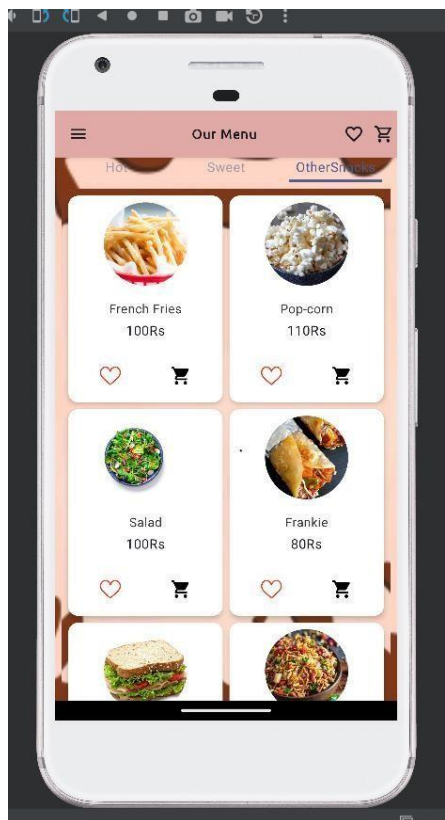
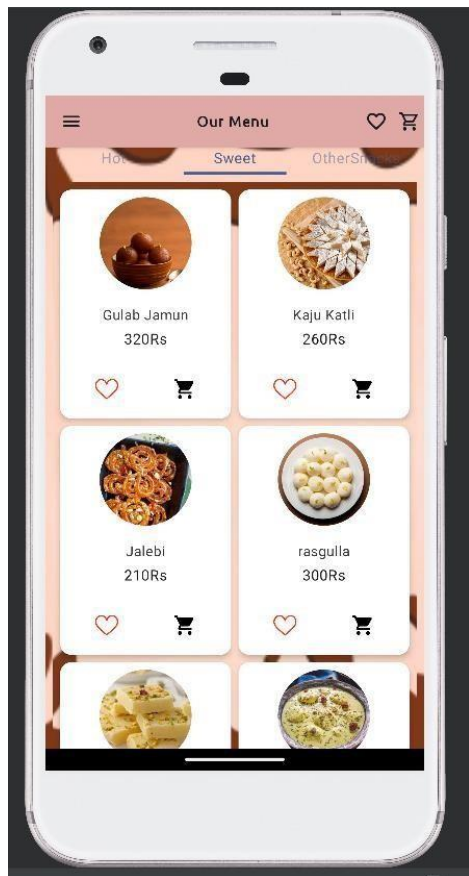
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        IconButton(
            onClick = {

                onLikeChange(food)
            },
            modifier = Modifier
                .weight(1f)
        ) {
            Image(
                painter = painterResource(
                    id =
                        if (food.liked) R.drawable.ic_like else R.drawable.ic_unlike
                ),
                contentDescription = null,
                modifier = Modifier.size(25.dp)
            )
        }
        data class totalItems(
            val food: Food,
            val quantity: Int
        )

        IconButton(

```





Foodscreen-

```
@OptIn(ExperimentalMaterial3Api::class)
```

```
@Composable
```

```
fun FoodScreen(navController: NavController, food: Food) {
```

```

Scaffold(
  topBar = {
    CenterAlignedTopAppBar(
      modifier = Modifier.shadow(8.dp),
      navigationIcon = {
        Row {
          Spacer(modifier = Modifier.width(8.dp))
          Icon(
            imageVector = Icons.Rounded.ArrowBack,
            contentDescription = null,
            modifier = Modifier
              .size(30.dp)
              .clickable {
                navController.popBackStack()
              }
          )
        }
      },
      title = {
        Text(text = "Food", fontFamily = ubuntuFont)
      }
    )
  }
) {
  padding ->
  Column(
    modifier = Modifier
      .fillMaxSize()
      .padding(padding)
      .verticalScroll(rememberScrollState())
  ) {
    Image(
      modifier = Modifier
        .fillMaxWidth()
        .heightIn(max = 200.dp),
      painter = painterResource(id = food.image),
      contentDescription = null,
      contentScale = ContentScale.Crop
    )
    Spacer(modifier = Modifier.height(16.dp))
    Column(modifier = Modifier.padding(horizontal = 16.dp)) {
      Text(
        text = food.name,
        fontSize = 18.sp,
        fontWeight = FontWeight.Bold
      )
      Spacer(modifier = Modifier.height(16.dp))
      Text(text = "${food.price}Rs", fontSize = 17.sp)
      Spacer(modifier = Modifier.height(16.dp))
      Text(
        text = "Description",
        fontSize = 14.sp,
        fontWeight = FontWeight.Bold
      )
      Spacer(modifier = Modifier.height(2.dp))
      Text(
        text = food.description ?: "",

```

```

        fontSize = 13.sp,
        textAlign = TextAlign.Justify,
        color = Color(0xff313131)
    )
}
Spacer(modifier = Modifier.height(16.dp))
Divider(thickness = 2.dp)
Spacer(modifier = Modifier.height(16.dp))
Text(
    text = "Recommended Foods:",
    modifier = Modifier.padding(horizontal = 8.dp),
    fontWeight = FontWeight.Bold,
    fontSize = 17.sp
)
Spacer(modifier = Modifier.height(16.dp))
LazyRow(horizontalArrangement = Arrangement.spacedBy(16.dp)) {
    items(foods) { food ->
        RecommendedFood(food = food, navController = navController, onTap = { food ->
            // Do Something you want..
        })
    }
}
Spacer(modifier = Modifier.height(16.dp))
Divider(thickness = 2.dp)
Spacer(modifier = Modifier.height(16.dp))
Column(modifier = Modifier.padding(horizontal = 8.dp)) {
    Text(
        text = "Rating",
        fontSize = 18.sp,
        fontWeight = FontWeight.Bold
    )
    Spacer(modifier = Modifier.height(8.dp))
    Text(
        text = food.reviews ?: "",
        fontSize = 13.sp,
        textAlign = TextAlign.Justify,
        color = Color(0xff313131)
    )
}
Spacer(modifier = Modifier.height(8.dp))
}
}
}

```

```

@OptIn(ExperimentalMaterial3Api::class)

```

```

@Composable

```

```

fun RecommendedFood(food: Food, navController: NavController, onTap: (Food) -> Unit) {

```

```

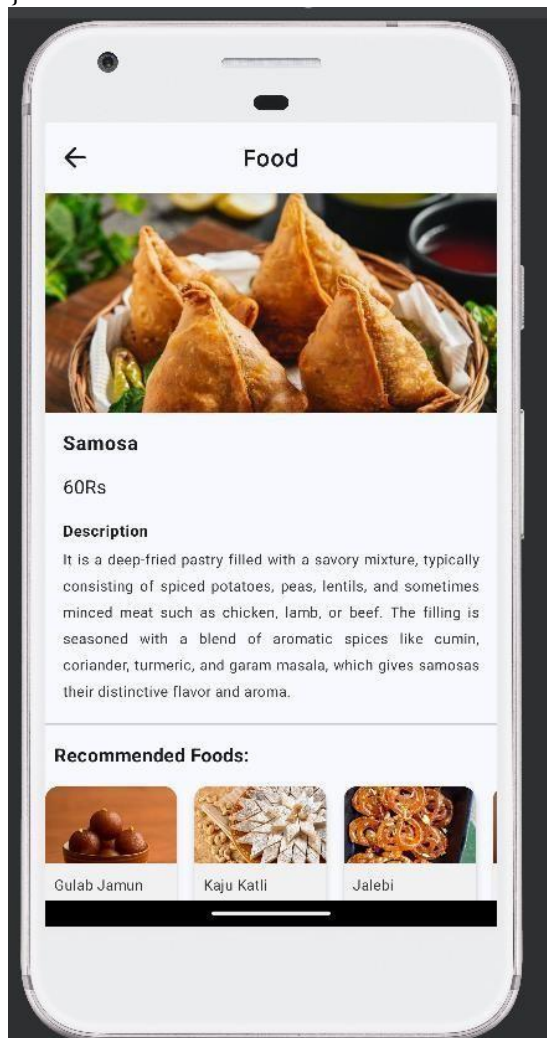
    Card(
        elevation = CardDefaults.cardElevation(defaultElevation = 4.dp),
        colors = CardDefaults.cardColors(
            containerColor = Color(0xffff1f1f)
        ),
        onClick = {
            // Navigate to the food section
            navController.navigate("food/${food.name}")
        }
    )
}

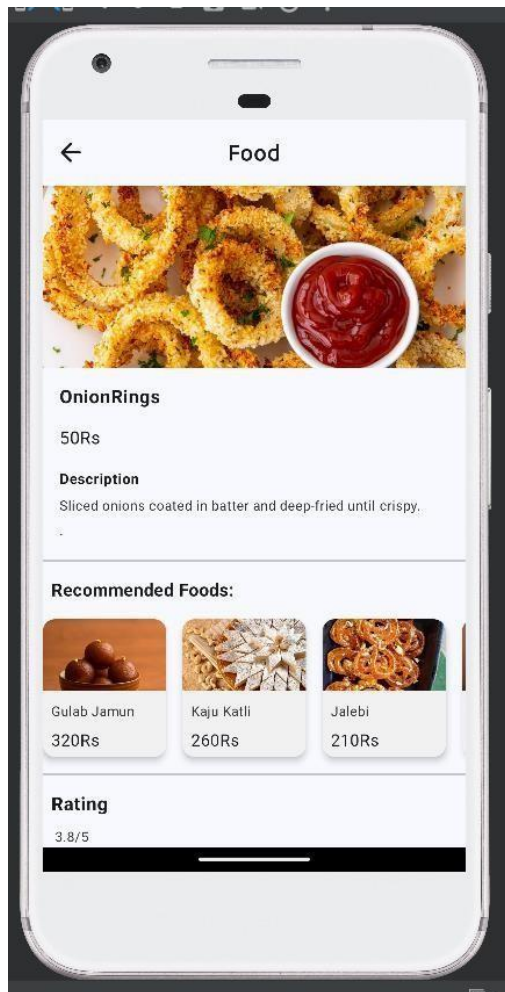
```

```

    ) {
        Column {
            Image(
                modifier = Modifier.sizeIn(
                    maxWidth = 120.dp,
                    maxHeight = 70.dp
                ),
                painter = painterResource(id = food.image),
                contentDescription = food.name,
                contentScale = ContentScale.Crop
            )
            Column(modifier = Modifier.padding(horizontal = 8.dp)) {
                Spacer(modifier = Modifier.height(8.dp))
                Text(text = food.name, color = Color(0xff313131), fontSize = 13.sp)
                Spacer(modifier = Modifier.height(4.dp))
                Text(text = "${food.price}Rs")
                Spacer(modifier = Modifier.height(4.dp))
            }
        }
    }
}

```





Cartscreen-

```

@SuppressLint("UnusedBoxWithConstraintsScope")
@Composable
fun CartScreen(
    cartItems: List<CartItem>,
    totalItems: Int,
    onAddMoreClick: () -> Unit,
    onPlaceOrderClick: () -> Unit
) {
    BoxWithConstraints(
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.background)
    ) {
        val backgroundImage = painterResource(id = R.drawable.signupbg)

        Image(
            painter = backgroundImage,
            contentDescription = null,
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .fillMaxSize()
                .alpha(0.3f)
        )
    }
}

```

```

Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp),
    verticalArrangement = Arrangement.Center
) {

    Text(
        text = "Total items in the cart: ${cartItems.sumBy { it.quantity }}",
        fontWeight = FontWeight.Bold,
        fontSize = 24.sp
    )
    Spacer(modifier = Modifier.height(16.dp))

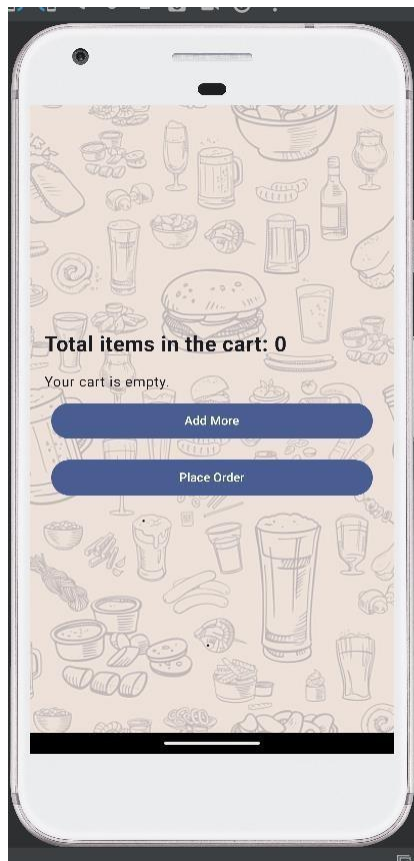
    if (cartItems.isNotEmpty()) {

        for (item in cartItems) {
            Text(text = "${item.food.name} - Quantity: ${item.quantity}")
            Spacer(modifier = Modifier.height(8.dp))
        }
    } else {
        Text(text = "Your cart is empty.")
    }

    Button(
        onClick = { onAddMoreClick() },
        modifier = Modifier
            .padding(8.dp)
            .fillMaxWidth()
    ) {
        Text("Add More")
    }

    Button(
        onClick = { onPlaceOrderClick() },
        modifier = Modifier
            .padding(8.dp)
            .fillMaxWidth()
    ) {
        Text("Place Order")
    }
}
}
}

```

Mainactivity-

```

@SuppressLint("UnusedBoxWithConstraintsScope")
@Composable
fun CartScreen(
    cartItems: List<CartItem>,
    totalItems: Int,
    onAddMoreClick: () -> Unit,
    onPlaceOrderClick: () -> Unit
) {
    BoxWithConstraints(
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.background)
    ) {
        val backgroundImage = painterResource(id = R.drawable.signupbg)

        Image(
            painter = backgroundImage,
            contentDescription = null,
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .fillMaxSize()
                .alpha(0.3f)
        )

        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp),

```

```

        verticalArrangement = Arrangement.Center
    ) {

        Text(
            text = "Total items in the cart: ${cartItems.sumBy { it.quantity }}",
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp
        )
        Spacer(modifier = Modifier.height(16.dp))

        if (cartItems.isNotEmpty()) {

            for (item in cartItems) {
                Text(text = "${item.food.name} - Quantity: ${item.quantity}")
                Spacer(modifier = Modifier.height(8.dp))
            }
        } else {
            Text(text = "Your cart is empty.")
        }

        Button(
            onClick = { onAddMoreClick() },
            modifier = Modifier
                .padding(8.dp)
                .fillMaxWidth()
        ) {
            Text("Add More")
        }

        Button(
            onClick = { onPlaceOrderClick() },
            modifier = Modifier
                .padding(8.dp)
                .fillMaxWidth()
        ) {
            Text("Place Order")
        }
    }
}

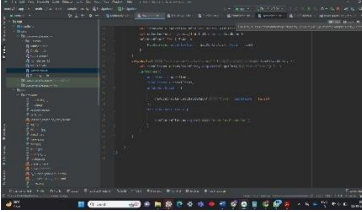
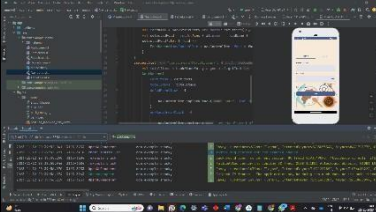
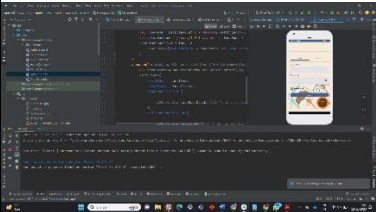
```

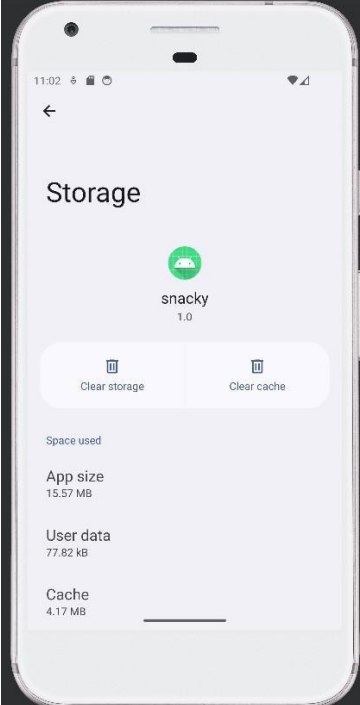
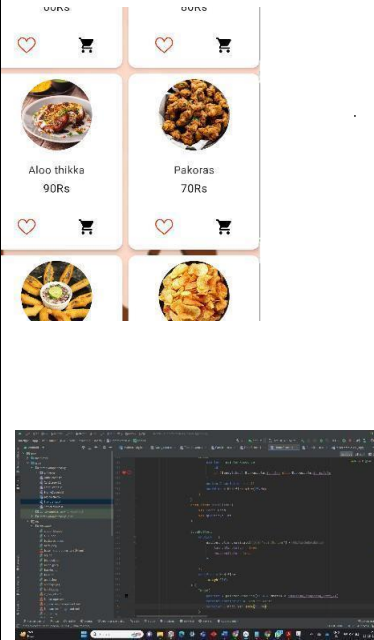
Project Development Phase Model Performance Test

Date	09 November 2023
Team ID	Team-590962
Project Name	Project – SNACK SQUAD
Maximum Marks	10 Marks

Model Performance Testing:

Project team shall fill the following information in model performance testing template.

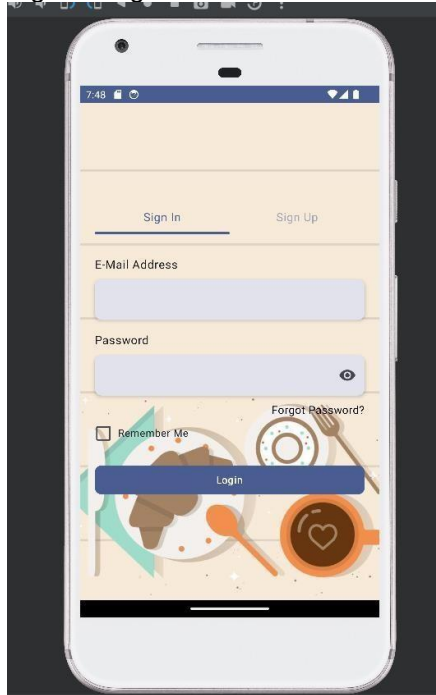
S.No.	Parameter	Values	Screenshot
1.	Metrics	<p>App Launch Time- 3s Screen Render Time- 1s Code Quality-</p> <p>https://github.com/smartinternz02/SI-GuidedProject-587302-1697649760</p>	  

2.	Usage	App Size- 15.57 MB Customer Experience- Good	
3.	Performance	Error and Crash Rates- Database Query Performance-	

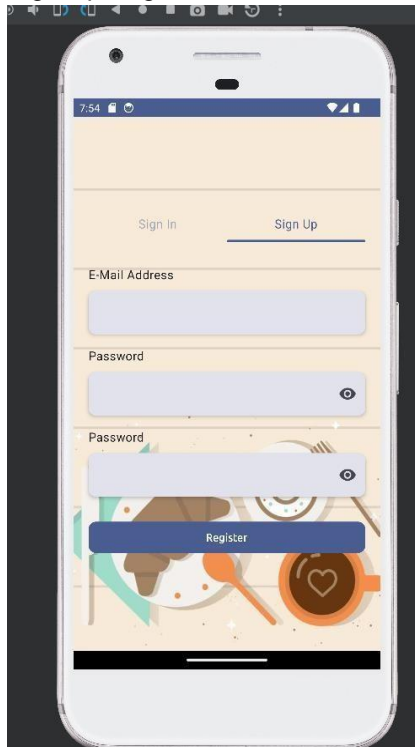
Results

Date	06 November 2023
Team ID	Team-590962
Project Name	Project – SNACK SQUAD

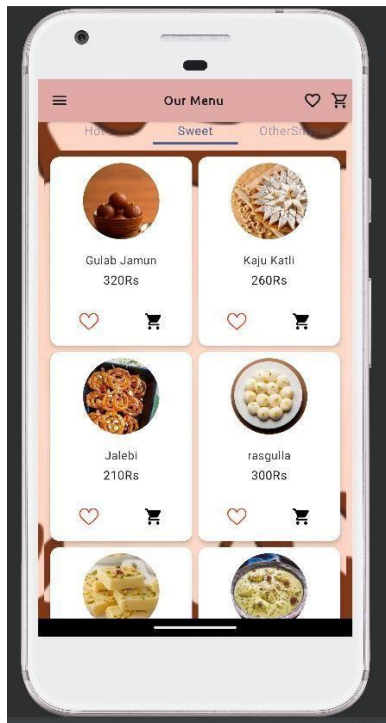
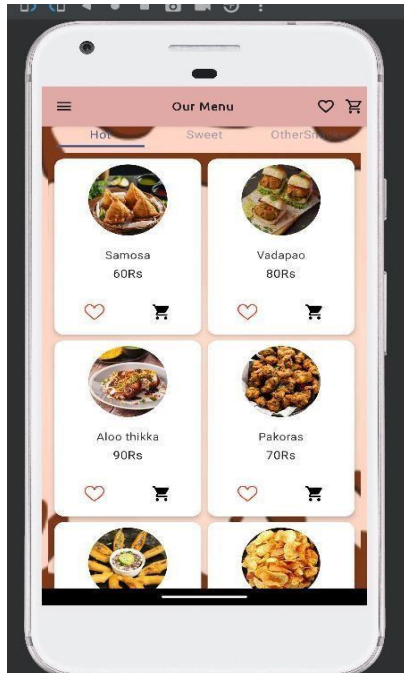
1) Sign In Page



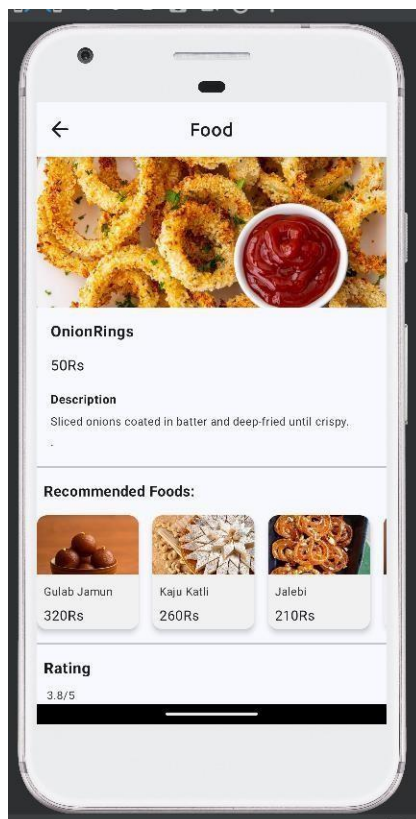
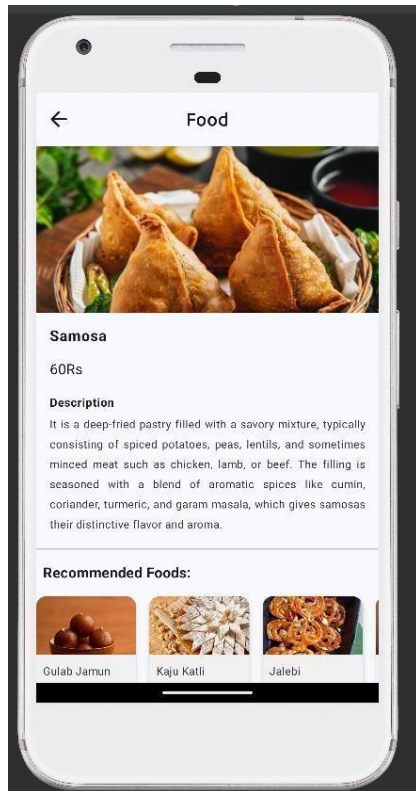
2) Sign Up Page



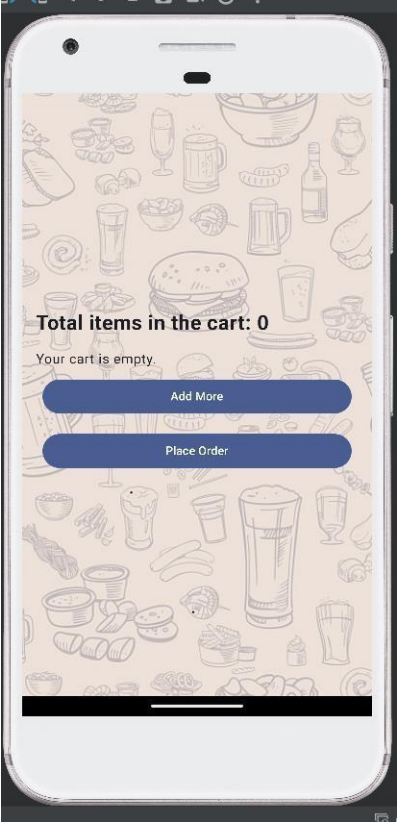
3) Home Screen



Foodscreen-



CART SCREEN



Advantages And Disadvantages

Date	09 November 2023
Team ID	Team-590962
Project Name	Project – SNACK SQUAD

Creating an enterprise-level Snack Squad app using Kotlin for backend development and XML for frontend design brings with it a set of advantages and disadvantages.

Advantages:

Kotlin for Backend:

Conciseness and Readability: Kotlin is known for its concise syntax, which can enhance code readability and maintainability.

Interoperability: Kotlin seamlessly interoperates with Java, providing access to the rich Java ecosystem and libraries.

XML for Frontend:

Declarative UI: XML allows for a declarative way of defining user interfaces, making it easier to visualize and understand the layout structure.

Separation of Concerns: XML enables a clear separation between the layout structure and business logic, promoting a modular and maintainable codebase.

Full-Stack Kotlin:

Consistent Language: Using Kotlin for both backend and frontend development creates a consistent development experience and reduces the learning curve for developers.

Android Development Experience:

Android Studio Support: Kotlin is officially supported by Android Studio, the preferred IDE for Android app development, providing a seamless development experience.

Coroutines for Concurrency:

Concurrency Management: Kotlin's native support for coroutines simplifies asynchronous programming, improving the efficiency of handling concurrent tasks.

Community Support:

Growing Community: Kotlin has a growing community of developers, which means more resources, tutorials, and third-party libraries are available.

Disadvantages:

Learning Curve:

Adoption Challenges: While Kotlin is gaining popularity, some developers might be more familiar with Java or other languages, potentially posing a learning curve for the team.

XML Complexity:

Verbosity: XML layouts can become verbose, especially for complex UIs, leading to increased file sizes and potentially impacting readability.

Tooling for XML:

Limited Tooling Support: Compared to more modern UI frameworks, XML might have limited tooling support for UI design and might not provide the same level of convenience.

Potential for Duplication:

Code Duplication: The separation of UI and logic can sometimes lead to duplication of information between XML layouts and Kotlin code, requiring careful synchronization.

Backend Framework Choices:

Limited Kotlin Backend Frameworks: While there are several Kotlin-compatible backend frameworks, the selection might be more limited compared to more established languages like Java or Node.js.

XML Updates:

Manual XML Updates: Changes to the UI may require manual updates to XML layouts, potentially leading to more maintenance effort.

Android-Specific Focus:

Android-Centric: While Kotlin is versatile, the focus on Android development might limit the potential for using the same codebase for other platforms without additional adaptations.

11. CONCLUSION

In conclusion, Snack Squad, a Customizable Snack Ordering and Delivery App, aims to redefine the way users experience and satisfy their snack cravings. By offering a user-friendly and customizable platform, the app seeks to address the evolving needs of busy professionals, students, and snack enthusiasts, providing a seamless and delightful snacking experience.

The project envisions a comprehensive solution with a diverse snack catalog, real-time order tracking, secure payment integration, and a user-friendly interface. Through customizable options, users can tailor their snack orders, fostering a sense of personalization and catering to individual tastes and preferences.

The development plan includes a robust technology stack, encompassing React Native, Node.js, and secure cloud services, ensuring scalability, security, and cross-platform compatibility. The phased approach, from planning and design to deployment and future enhancements, demonstrates a commitment to a systematic and well-executed development lifecycle.

Despite the numerous advantages presented by Snack Squad, it is essential to acknowledge potential challenges, such as technical disruptions, logistical issues, and security concerns. Proactive measures, including regular testing, security audits, and a customer-centric approach, are crucial to overcoming these challenges and ensuring the app's success in a competitive market.

12. FUTURE SCOPE

The future scope for Snack Squad, a Customizable Snack Ordering and Delivery App, extends beyond its initial launch, presenting opportunities for innovation, expansion, and enhancement. Here are several avenues for future development and growth:

IoT Integration:

Explore the integration of Internet of Things (IoT) devices to create smart snacking experiences. For example, smart packaging that provides additional information about the snack, or IoT-enabled appliances for re-heating or preparing certain snacks.

Geographic Expansion:

Consider expanding the app's reach to additional cities or regions. Collaborate with local snack vendors to diversify the snack catalog and appeal to a broader audience.

Collaborations and Partnerships:

Forge collaborations with popular snack brands, local vendors, or even restaurants to offer exclusive or special edition snacks through the app. Partnerships can enhance the app's offerings and attract a wider customer base.

Enhanced Personalization:

Implement advanced algorithms and machine learning to analyze user preferences and provide personalized snack recommendations. This can improve user engagement and satisfaction.

Augmented Reality (AR) Features:

Explore the integration of AR features for a more immersive snacking experience. For instance, users could use AR to visualize how a customized snack would look before placing an order.

Subscription Models:

Introduce subscription-based models where users can subscribe to receive curated snack boxes regularly. This can enhance customer retention and provide a steady revenue stream.

Analytics and Data-Driven Insights:

Leverage analytics tools to gather insights into user behavior, popular snacks, and regional preferences. Use this data to optimize the app, tailor promotions, and enhance the overall user experience.

Social Integration:

Integrate social media features to allow users to share their snack experiences, reviews, and customized creations. Social sharing can contribute to organic marketing and user acquisition.

Voice Command Integration:

Explore voice command integration for hands-free ordering. Voice-activated features can enhance accessibility and convenience for users.

Sustainability Initiatives:

Implement sustainability initiatives such as eco-friendly packaging options, sourcing snacks from

sustainable producers, or collaborating with environmentally conscious brands.

Multi-Language Support:

Consider adding multi-language support to cater to a diverse user base, especially in regions with multiple languages spoken.

Enhanced Customer Engagement Strategies:

Develop and implement targeted marketing campaigns, loyalty programs, and referral incentives to keep users engaged and foster brand loyalty.

In summary, the future scope for Snack Squad involves a combination of technological advancements, strategic partnerships, and a continuous commitment to meeting the evolving needs of users. By staying agile and embracing emerging trends, Snack Squad can position itself as a dynamic and innovative player in the competitive snack ordering and delivery market.

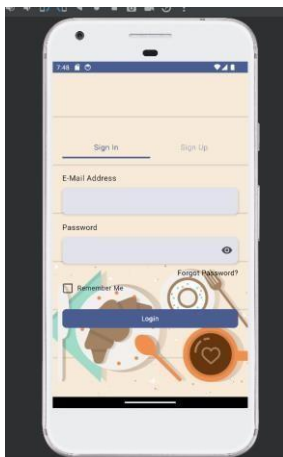
Appendix

Date	09 November 2023
Team ID	Team-590962
Project Name	Project – SNACK SQUAD

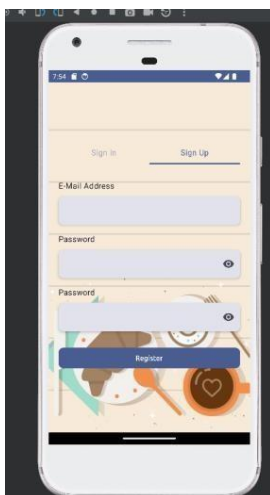
The appendix of this Enterprise SNACK App Documentation serves as a supplementary section that provides readers with additional resources and in-depth insights into various aspects of our enterprise chat app. In this introduction, we provide an overview of the content you can expect to find in the appendix and the value it adds to your understanding of the app's development, features, and usage.

Appendix A: User Interface Designs

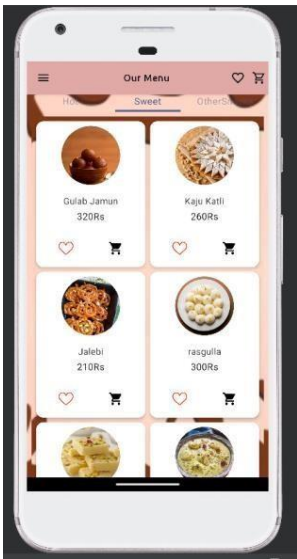
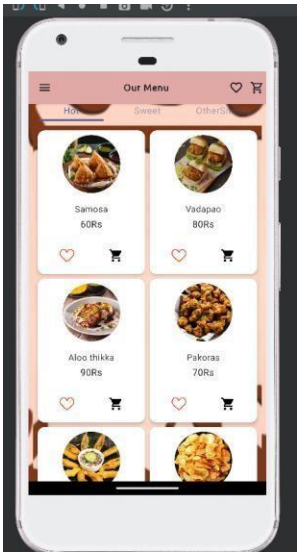
Login Screen



Sign Up Screen



Home Screen



Appendix B: Technical Specifications

This app is made using Kotlin and XML in Android Studio IDE. We used Firebase at the backend to authenticate and communicate between the users.

Appendix D: User Documentation

In this section, we'll provide a comprehensive user documentation to help you make the most out of our enterprise chat app. It includes detailed instructions on how to get started, use the app's features, and troubleshoot common issues.

- **Getting Started:** The app just needs an android device with API level 34
- **Navigating the App:** The app is easily navigable in an intuitive way using touch.
- **Chatting Basics:** The app
- **Contacts and Groups:** Contacts and groups are managed by the admins
- **Advanced Features:** Describes voice/video calls, screen sharing, and file sharing.
- **Security and Privacy:** Covers user authentication, encryption, and privacy settings.

Appendix E: Test Cases and Results

If you conducted testing, this appendix could include a list of test cases, test scripts, and the results of testing, including any issues or bugs encountered.

Appendix F: Data Privacy and Security Policies

The data is completely private and is handled with care.

Appendix G: Team Members, TEAM CODE 590962

1. D SRUJAY
2. B JASWANTH SAI SIMHA GANESH
3. K VAMSHI KRISHNA
4. SREERAMA SRI SAI MANJUNATH

