SNACK SQUAD: A CUSTOMISIBLE SNACK ORDERING AND DELIVERY APP

PROJECT REPORT

Submitted by

JINO R J(20203111506232)

LIJIN T(20203111506233)

MERBIN SAJI S(20203111506235)

MESHAK S(20203111506236)

ABINESH J D(20203111506204)

Submitted to Manonmaniam Sundaranar University. Tirunelveli

In partial fulfilment for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

Under the Guidance of

Prof.D.H.KITTY SMAILIN M.Sc.,M.Phil

DEPARTMENT OF PG COMPUTER SCIENCE

NESAMONY MEMORIAL CHRISTIAN COLLEGE,

MARTHANDAM

KANYAKUMARI DISTRICT -629165

DEPARTMENT OF PG COMPUTER SCIENCE

NESAMONY MEMORIAL CHRISTIAN COLLEGE

MARTHANDAM-629165, KANYAKUMARI DISTRICT.

BONAFIDE CERTIFICATE

This is to certify that the project work entitled "**SNACK SQUAD: A CUSTOMISIBLE SNACK ORDERING AND DELIVEY APP"** is a bonafide record work done by **JINO R J(20203111506232), LIJIN T (20203111506233), MERBIN SAJI S(20203111506235), MESHAK S(20203111506236), ABINESH J D (20203111506204)** during the academic year 2020-2023 in partial fulfilment of the requirements for the award of the degree in **BACHELOR OF SCIENCE IN COMPUTER SCIENCE** of Manonmaniam Sundaranar University, Tirunelveli.

Dr.D.Latha M.Sc., M.Phil., Ph.D.,
Head of the Department,
Department of PG Computer Science.

Prof.D.H.KITTY SMAILIN M.Sc.,M.Phil
Project Supervisor,
Department of PG Computer Science

DECLARATION

I hereby declare that the project work entitled "**SNACK SQUAD: A CUSTOMISIBLE SNACK ORDERING AND DELIVERY APP"** is an original work done by me in partial fulfilment of the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE.** The study has been carried under the guidance of **Prof.D.H.KITTY SMAILIN M.Sc.,M.Phil Department of PG COMPUTER SCIENCE, Nesamony Memorial Christian College, Marthandam.** I declare that this work has not been submitted elsewhere for the award of any degree.

Place: Marthandam.

Jino R J(20203111506232)

Lijin T(20203111506233)

Merbin sajiS(20203111506243)

Meshak S (20203111506236)

Abinesh J D(20203111506204)

ACKNOWLEDGEMENT

First of all I offer my prayers to god almighty for blessing me to complete this internship programme successfully

I express my sincere thanks to**, Dr.K.Paul Raj M.Sc., M.Phil., M.Ed., M.Phil(Edu).,Ph.D.,** Principal Nesamony Memorial Christian college, for his official support to do my work.

I express my profound thanks to **Dr. Dr.D.Latha M.Sc., M.Phil., Ph.D.,** HOD, Department of PG Computer Science for his encouragement given to undertake this project.

I extend my deep sense of gratitude to **, Prof. D.H.KITTY SMAILIN M.Sc.,M.Phil** Department of PG Computer Science for guiding me in completing this project work successfully.

My special thanks to other Faculty members of Department of PG Computer Science for Guiding me in Completing this project.

Then I want to thank for Naanmudhalvan smartinternz team for giving me as a wonderful opportunity for doing a android project and gave more knowledge about the android application by free course videos.

I also wish to extend a special word of thanks to my parents, friends and all unseen hands that helped me for completing this project work successfully.

CONTENTS

1. INTRODUCTION

The "A Customisable Snack Ordering and Delivery App" has been developed to override the problems prevailing in the practicing manual system. This software is supported to eliminate and, In some cases, reduce the hardships faced by this existing system. Moreover, this system is designed for the particular need of the company to carry out operations in a smooth and effective manner. The application is reduced as much as possible to avoid errors while entering the data. It also provides error message while entering invalid data. No formal knowledge is needed for the user to use this system. Thus, by this all it proves it is user-friendly. Online Snack Ordering System, as described above, can lead to error free, secure , reliable and fast management system. It can assist the user to concentrate on their other activities rather to concentrate on the record keeping. Thus, it will help organization in better utilization of resources. Every organization, whether big or small, has challenges to overcome and managing the information of Category, Food Item, Order, Payment, Confirm Order. Every Customisable Snack Ordering and delivery app needs different Food Item needs; therefore, we design exclusive employee management systems that are adapted to your managerial requirements. This is designed to assist in strategic planning and will help you ensure that your organization is equipped with the right level of information and details for your future goals. Also, for those busy executives who are always on the go, our systems come with remote access features, which will allow you to manage your workforce anytime, at all times. These systems will ultimately allow you to better manage resources.

1.1 Overview

In the age of technology, where many things are available at your fingertips, people who love going to the shops after a long day still have to stand in long queues outside the auditorium to get snacks, try to remember other people's orders and miss some parts of the movie, hence not having a satisfactory experience at the movies.

A customisable Snack Ordering app aims at solving the problem of leaving the auditorium and provides the option of ordering from inside the theatre. The group order feature adds the convenience of saving time in remembering all the order items that other people of the group want.

The online food delivery is a service that allows the user to order food from a desired food outlet via the internet. This can be done either by going directly to the website and placing an order or by using a mobile phone application. The introduction of online food delivery system has been a convenient addition, which has not only reduced long queues, but has also decreased the waiting time for ordered food delivery. The online food delivery system has already been adopted throughout the globe and its performance has been relatively good.

The key players in the industry have been relying on partnerships and acquisition as the prominent strategies to help boost their growth in the market. The global online food market is driven by rise in internet penetration coupled with increase in the working population and surge in the food & beverage industry. The online food delivery market is also supplemented by growth in mobile phone dependency. However, unwillingness of big food outlets to adopt this system along with the potential technical and infrastructural issues hinder the growth of this market. Moreover, too much competition and lack of loyal customers also act as a threat to this market. Technological breakthroughs and infrastructural improvements, especially in the emerging nations are expected to drive the growth of the market in the future.

The global online food delivery market is segmented based on the delivery model and region. Based on delivery model, the market is segmented into the traditional delivery model, aggregators, and new delivery model. By region, the global online food delivery market is studied across North America, Europe, Asia-Pacific, and LAMEA.

Objective of Project on Online Food Ordering System:

The main objective of the Project on Online Food Ordering System is to manage the details of Food Item, Category, Customer, Order, Confirm Order. It manages all the information about Food Item, Payment, Confirm Order, Food Item. The project is totally built at administrative end and thus only the administrator is guaranteed the access. The purpose of the project is to build an application program to reduce the manual work for managing the Food Item, Category, Payment, Customer. It tracks all the details about the Customer, Order, Confirm Order.

Functionalities provided by Online Food Ordering System are as follows: Provides the searching facilities based on various factors. Such as Food Item, Customer, Order, Confirm Order Online Food Ordering System also manage the Payment details online for Order details, Confirm Order details, Food Item. It tracks all the information of Category, Payment, Order etc Manage the information of Category Shows the information and description of the Food Item, Customer To increase efficiency of managing the Food Item, Category It deals with monitoring the information and transactions of Order. Manage the information of Food Item Editing, adding and updating of Records is improved which results in proper resource management of Food Item data. Manage the information of Order Integration of all records of Confirm Order

Step 1: Download Android Studio

To set up Android Studio, you need to first download the IDE from the official Android Studio download page. Choose the version that is compatible with your operating system, and download the installer. Android Studio is available for Windows, macOS, and Linux. Once the download is complete, run the installer and follow the instructions to install Android Studio on your computer.

Step 2: Install the Required Components

During the installation process, Android Studio will prompt you to install the required components. These include the Android SDK, Android Virtual Device (AVD) Manager, and the Android Emulator. The Android SDK is a collection of libraries and tools that developers use to build Android applications. The AVD Manager is used to create and manage virtual devices for testing applications. The Android Emulator is a virtual device that allows developers to test their applications without having to use a physical device.

Step 3: Configure Android Studio

After installing Android Studio, you need to configure it before you can start using it. When you launch Android Studio for the first time, you will be prompted to configure the IDE. Choose the “Standard” configuration and click on “Next”. In the next screen, you can choose the theme of the IDE and click on “Next” again. You can also customize the settings based on your preferences.

Step 4: Create a New Project

Once Android Studio is configured, you can start creating your first Android application. To create a new project, click on “Start a new Android Studio project” on the welcome screen, or select “New Project” from the “File” menu. You will be prompted to choose the project name, package name, and other project details. You can also choose the minimum SDK version, which determines the minimum version of Android that the application can run on.

Step 5: Build Your Application

Once your project is created, you can start building your application using the various tools and features provided by Android Studio. You can use the visual layout editor to design the user interface, write code in Java or Kotlin, and use the Android SDK to access device features such as the camera, sensors, and GPS. You can also use the built-in debugging tools to troubleshoot issues and optimize your application.

Step 6: Test Your Application

Testing your application is an important step in the development process. Android Studio comes with an emulator that allows you to test your application on different virtual devices. You can also connect your Android device to your computer and test your application directly on the device. Use the “Run” button in Android Studio to launch your application and test it on the emulator or device. You can also use the built-in profiler to analyse the performance of your application and identify any bottlenecks or performance issues. In conclusion, setting up Android Studio is a crucial step in developing Android applications. By following these steps, you can easily set up Android Studio on your computer and start building high-quality Android applications. Android Studio provides a powerful set of tools and features that make the development process easier and more efficient.

Components in Android Studio

1. Manifest File

The AndroidManifest.xml file is a crucial component of any Android application. It provides essential information about the application to the Android operating system, including the application's package name, version, permissions, activities, services, and receivers. The manifest file is required for the Android system to launch the application and to determine its functionality. Here are some of the key uses of the manifest file in an Android application:

Declaring Application Components: The manifest file is used to declare the various components of an Android application, such as activities, services, and broadcast receivers. These components define the behaviour and functionality of the application, and the Android system uses the manifest file to identify and launch them.

Specifying Permissions: Android applications require specific permissions to access certain features of the device, such as the camera, GPS, or storage. The manifest file is used to declare these permissions, which the Android system then checks when the application is installed. If the user has not been granted the required permissions, the application may not be able to function correctly.

Defining App Configuration Details: The manifest file can also be used to define various configuration details of the application, such as the application's name, icon, version code and name, and supported screens. These details help the Android system to identify and manage the application properly.

Declaring App-level Restrictions: The manifest file can be used to declare certain restrictions at the app level, such as preventing the application from being installed on certain devices or specifying the orientation of the app on different screens.

In summary, the manifest file is an essential part of any Android application. It provides important information about the application to the Android system and enables the system to launch and manage the application correctly. Without a properly configured manifest file, an Android application may not be able to function correctly, or it may not be installed at all.

2. Build.gradle

Gradle

Build.gradle is a configuration file used in Android Studio to define the build settings for an Android project. It is written in the Groovy programming language and is used to configure the build process for the project. Here are some of the key uses of the build.gradle file:

Defining Dependencies: One of the most important uses of the build.gradle file is to define dependencies for the project. Dependencies are external libraries or modules that are required by the project to function properly. The build.gradle file is used to specify which dependencies the project requires, and it will automatically download and include those dependencies in the project when it is built.

Setting Build Options: The build.gradle file can also be used to configure various build options for the project, such as the version of the Android SDK to use, the target version of Android, and the signing configuration for the project.

Configuring Product Flavours: The build.gradle file can be used to configure product flavours for the project. Product flavours allow developers to create different versions of their application with different features or configurations. The build.gradle file is used to specify which product flavours should be built, and how they should be configured.

Customizing the Build Process: The build.gradle file can also be used to customize the build process for the project. Developers can use the build.gradle file to specify custom build tasks, define build types, or customize the build process in other ways.

Overall, the build.gradle file is a powerful tool for configuring the build process for an Android project. It allows developers to define dependencies, configure build options, customize the build process, and more. By understanding how to use the build.gradle file, developers can optimize the build process for their projects and ensure that their applications are built correctly and efficiently.

3. Git

Git is a popular version control system that allows developers to track changes to their code and collaborate with other team members. Android Studio includes built-in support for Git, making it easy to manage code changes and collaborate with others on a project. Here are some of the key uses of Git in Android Studio:

Version Control: Git allows developers to track changes to their code over time. This means that they can easily roll back to a previous version of their code if needed, or review the changes made by other team members.

Collaboration: Git enables multiple developers to work on the same codebase simultaneously. Developers can work on different features or parts of the code base without interfering with each other, and merge their changes together when they are ready.

Branching and Merging: Git allows developers to create branches of their code base, which can be used to work on new features or bug fixes without affecting the main codebase. When the changes are complete, the branch can be merged back into the main codebase.

Code Review: Git allows team members to review each other's code changes before they are merged into the main codebase. This can help ensure that the code is of high quality and meets the project's requirements.

Android Studio includes a built-in Git tool that allows developers to perform common Git tasks directly within the IDE. Developers can create new repositories, clone existing ones, and manage branches and commits. Android Studio also provides a visual diff tool that makes it easy to see the changes made to the code base over time. To use Git in Android Studio, developers need to first initialize a Git repository for their project. Once the repository is set up, they can use the Git tool in Android Studio to manage changes to their code, collaborate with others, and review code changes.

In summary, Git is a powerful version control system that is essential for managing code changes and collaborating with other team members. Android Studio includes built-in support for Git, making it easy for developers to manage their code changes directly within the IDE.

4. Debug

Debugging is an essential part of software development, and Android Studio provides a robust set of debugging tools to help developers identify and fix issues in their applications. Here are some of the key uses of debugging in Android Studio.

Identifying Issues: Debugging helps developers identify issues in their code by allowing them to inspect variables, evaluate expressions, and step through the code line by line. This allows developers to pinpoint exactly where a problem is occurring and fix it more quickly.

Optimizing Performance: Debugging can also be used to optimize the performance of an application by identifying bottlenecks or areas of inefficient code. By profiling an application while it is running, developers can identify areas of the code that are causing slow performance and make changes to improve performance.

Testing and Validation: Debugging is also useful for testing and validating an application. By stepping through code and inspecting variables, developers can ensure that the application is behaving as expected and that it is producing the desired output.

Android Studio provides a comprehensive set of debugging tools, including breakpoints, watches, and the ability to evaluate expressions in real time. Developers can use these tools to inspect variables, step through code, and identify issues in their applications.

To use the debugging tools in Android Studio, developers need to first configure their project for debugging by adding breakpoints to their code. Breakpoints are markers that tell the debugger to pause execution at a certain point in the code. Once the breakpoints are set, developers can run their application in debug mode and step through the code line by line, inspecting variables and evaluating expressions as they go.

In summary, debugging is a critical part of software development, and Android Studio provides a robust set of debugging tools to help developers identify and fix issues in their applications. By using these tools, developers can optimize performance, test and validate their code, and improve the quality of their applications.

5. App Inspection

Inspector

App Inspection is a feature in Android Studio that allows developers to inspect and debug their Android applications. It provides a suite of tools for analysing the performance of the application, identifying and fixing errors, and optimizing the code. Here are some of the key features and uses of App Inspection:

Performance Analysis: App Inspection provides tools for analysing the performance of an Android application. Developers can use these tools to identify performance bottlenecks, such as slow database queries or inefficient network requests, and optimize the code to improve performance.

Error Detection and Debugging: App Inspection allows developers to detect and debug errors in their Android applications. It provides tools for tracking down errors and identifying the root cause of the issue, making it easier to fix bugs and improve the stability of the application.

Memory Management: App Inspection provides tools for managing the memory usage of an Android application. Developers can use these tools to identify memory leaks and optimize the code to reduce memory usage, which can improve the performance and stability of the application.

Network Profiling: App Inspection includes tools for profiling network traffic in an Android application. Developers can use these tools to monitor network requests, identify slow or inefficient requests, and optimize the code to improve network performance.

Overall, App Inspection is a valuable tool for Android developers. It provides a suite of tools for analysing and debugging Android applications, identifying and fixing errors, and optimizing the code for improved performance and stability. By using App Inspection, developers can ensure that their Android applications are of the highest quality and provide the best possible user experience.

6. Build Variants

Build variants in Android Studio are different versions of an Android app that can be built from the same source code. They are typically used to create multiple versions of an app that target different device configurations or use cases. Build variants are configured in the build.gradle file and can be built and installed separately from each other. Here are some examples of how build variants can be used.

Debug and Release Variants: The most common use of build variants is to create a debug variant and a release variant of an app. The debug variant is used for testing and debugging the app during development, while the release variant is used for production and is optimized for performance and stability.

Flavours: Build variants can also be used to create different flavours of an app, which can have different features or configurations. For example, an app might have a free version and a paid version, or a version that targets tablets and a version that targets phones.

Build Types: Build variants can also be used to create different build types, which can have different build options or signing configurations. For example, an app might have a debug build type and a release build type, each with its own set of build options.

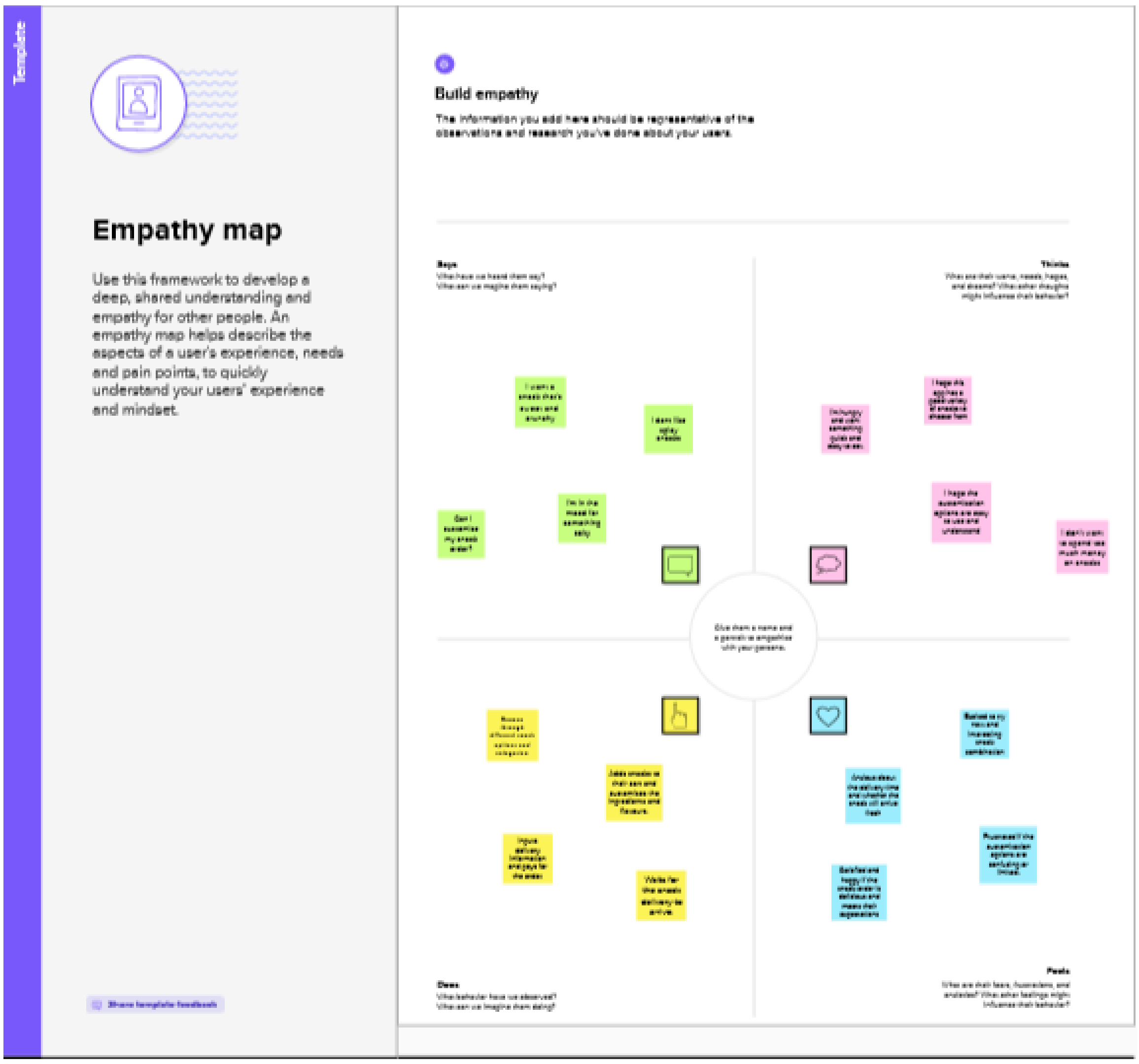
Overall, build variants are a powerful tool for Android developers. They allow developers to create different versions of an app from the same source code, which can save time and improve the quality of the app. By using build variants, developers can easily target different device configurations or use cases, create different versions of the app with different features or configurations, and optimize the app for performance and stability.

1.2 Purpose

The purpose of Online Snack Ordering Project is to automate the existing manual system by the help of computerized equipment's and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for alonger period with easy accessing and manipulation of the same. The required software and hardware are easily available and easy to work with. Online Snack Ordering System, as described above, can lead to error free, secure, reliable and fast management system. It can assist the user to concentrate on their other activities rather to concentrate on the record keeping. Thus it will help organization in better utilization of resources. The organization can maintain computerized records without redundant entries. That means that one need not be distracted by information that is not relevant, while being able to reach the information. The aim is to automate its existing manual system by the help of computerized equipment's and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. Basically the project describes how to manage for good performance and better services for the clients.

2. Problem Definition & Design Thinking

Empathy map:



Empathy map

Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.

Build empathy

The information you add here should be representative of the observations and research you've done about your users.

Says

What have we heard them say? What can we imagine them saying?

I want a snack that's sweet and crunchy

I don't like salty snacks

Can I customize my snack order?

I'm in the mood for something salty

Thinks

What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?

I'm hungry and want something quick and easy to eat.

I hope the app has a good variety of snacks to choose from

I hope the customization options are easy to use and understand

I don't want to spend too much money on snacks

Give them a name and a persona to empathize with your persona.

Does

What behavior have we observed? What can we imagine them doing?

Browse through different snack options and categories

Adds snacks to their cart and customizes the ingredients and flavours.

Inputs delivery information and pays for the order

Waits for the snack delivery to arrive.

Feels

What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

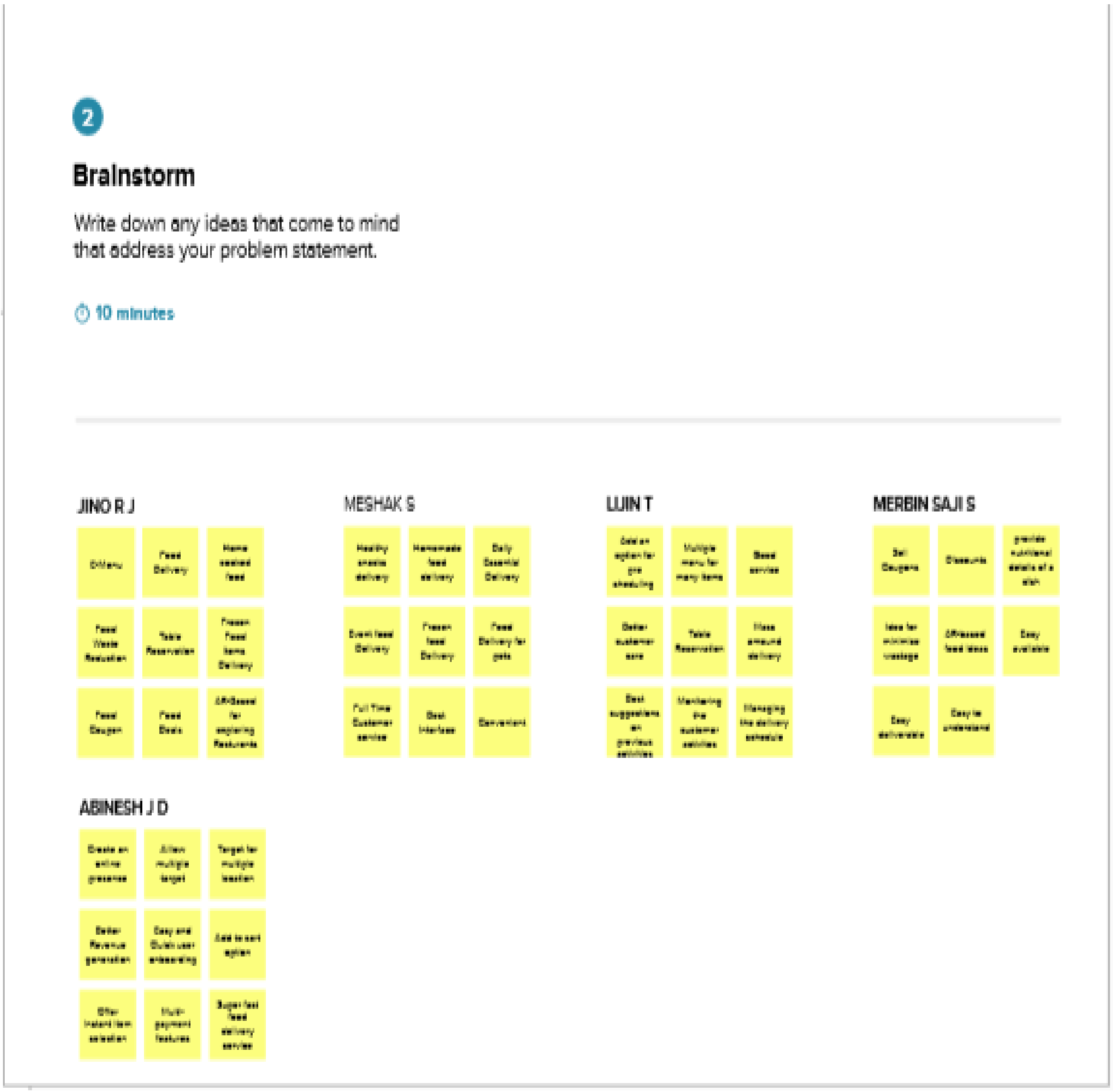
Excited to try new and interesting snack combination

Anxious about the delivery time and whether the snack will arrive fresh

Frustrated if the customization options are confusing or limited.

Satisfied and happy if the snack order is delicious and meets their expectations

Ideation & Brain Storming Map:



2

Brainstorm

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

10 minutes

JINO R J

- E-Menu
- Food Delivery
- Home cooked food
- Food Waste Reduction
- Table Reservation
- Frozen Food items Delivery
- Food Coupon
- Food Deals
- AR-Based for exploring Restaurants

MESHAK S

- Healthy snacks delivery
- Homemade food delivery
- Daily Essential Delivery
- Event food Delivery
- Frozen food Delivery
- Food Delivery for pets
- Full Time Customer service
- Best Interface
- Convenient

LIJIN T

- Add an option for pre scheduling
- Multiple menu for many items
- Good service
- Better customer care
- Table Reservation
- Mass amount delivery
- Best suggestions on previous activities
- Monitoring the customer activities
- Managing the delivery schedule

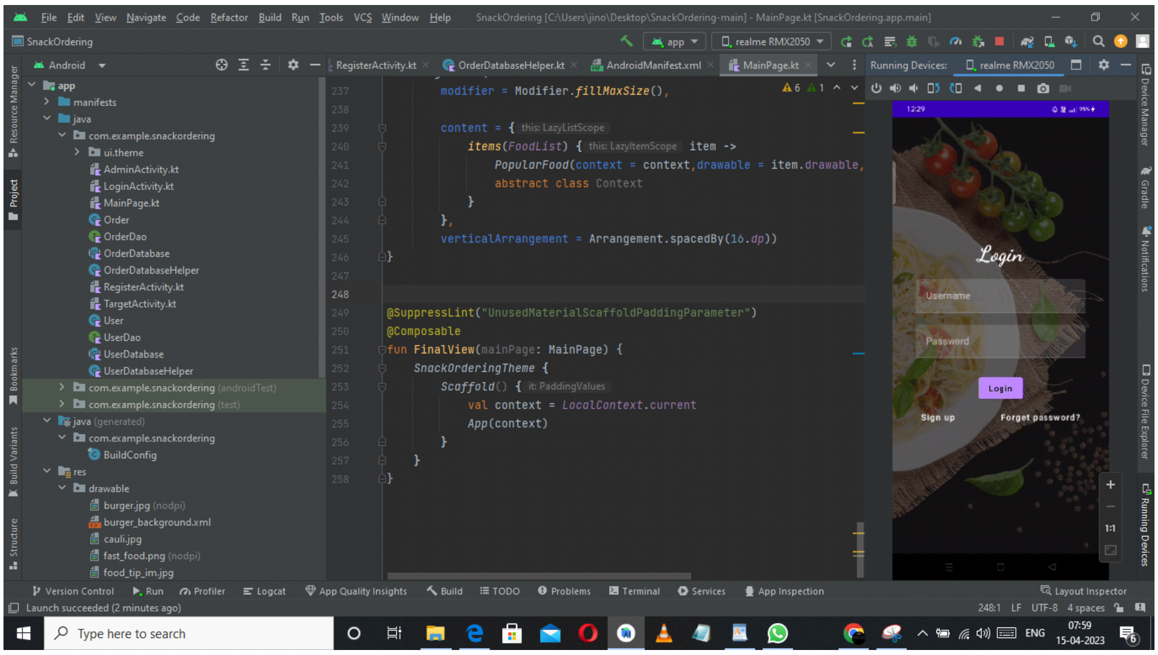
MERBIN SAJI S

- Sell Coupons
- Discounts
- provide nutritional details of a dish
- Idea for minimize wastage
- AR-based food ideas
- Easy available
- Easy deliverable
- Easy to understand

ABINESH J D

- Create an online presence
- Allow multiple target
- Target for multiple location
- Better Revenue generation
- Easy and Quick user onboarding
- Add to cart option
- Offer instant item selection
- Multi-payment features
- Super fast food delivery service

3. RESULT

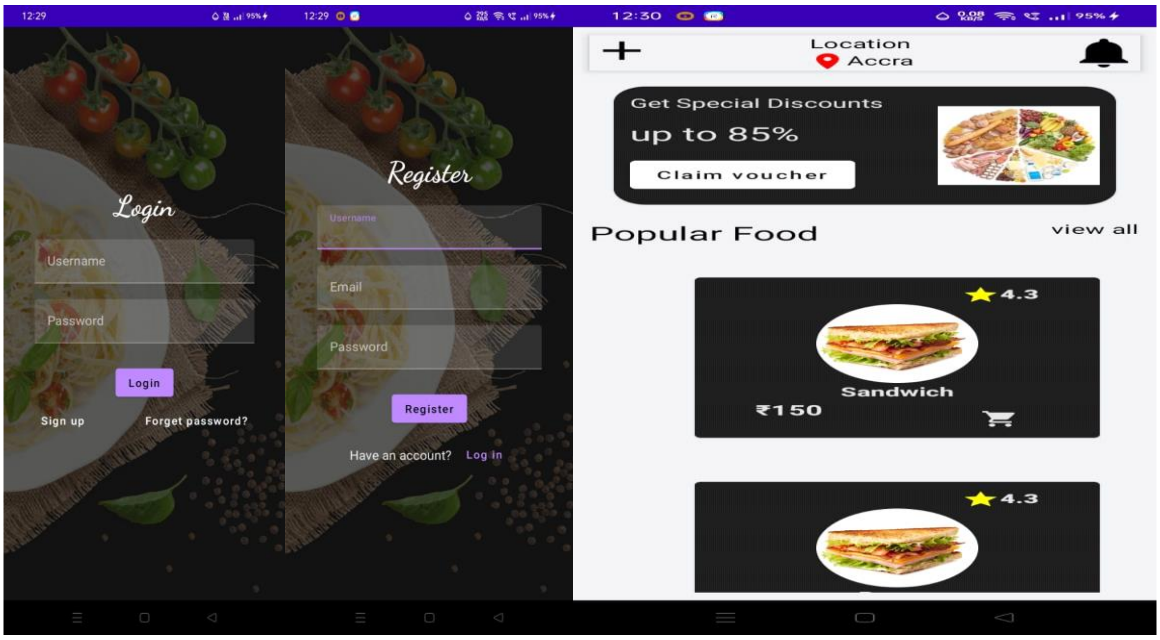

SnackOrdering [C:\Users\jino\Desktop\SnackOrdering-main] - MainPage.kt [SnackOrdering.app.main]

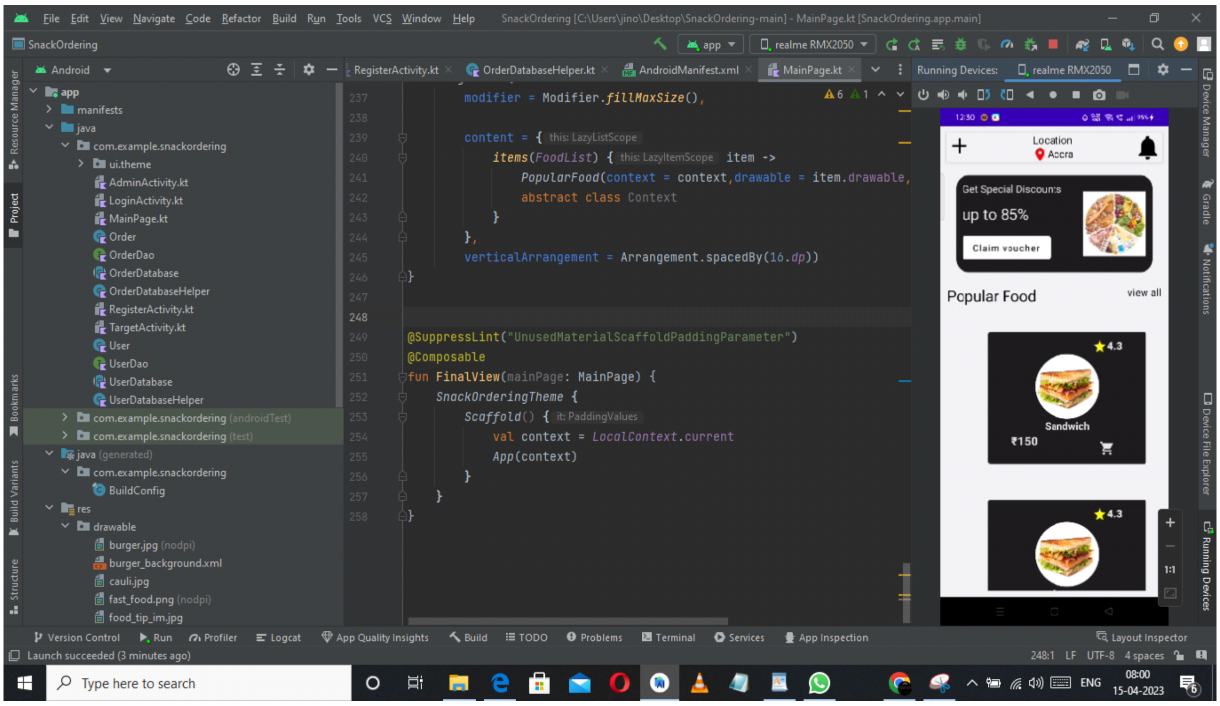
```
        modifier = Modifier.fillMaxSize(),

        content = {
            items(FoodList) { item ->
                PopularFood(context = context,drawable = item.drawable,
                abstract class Context
            }
        },
        verticalArrangement = Arrangement.spacedBy(16.dp))
}


@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun FinalView(mainPage: MainPage) {
    SnackOrderingTheme {
        Scaffold() {
            val context = LocalContext.current
            App(context)
        }
    }
}
```

Launch succeeded (2 minutes ago)



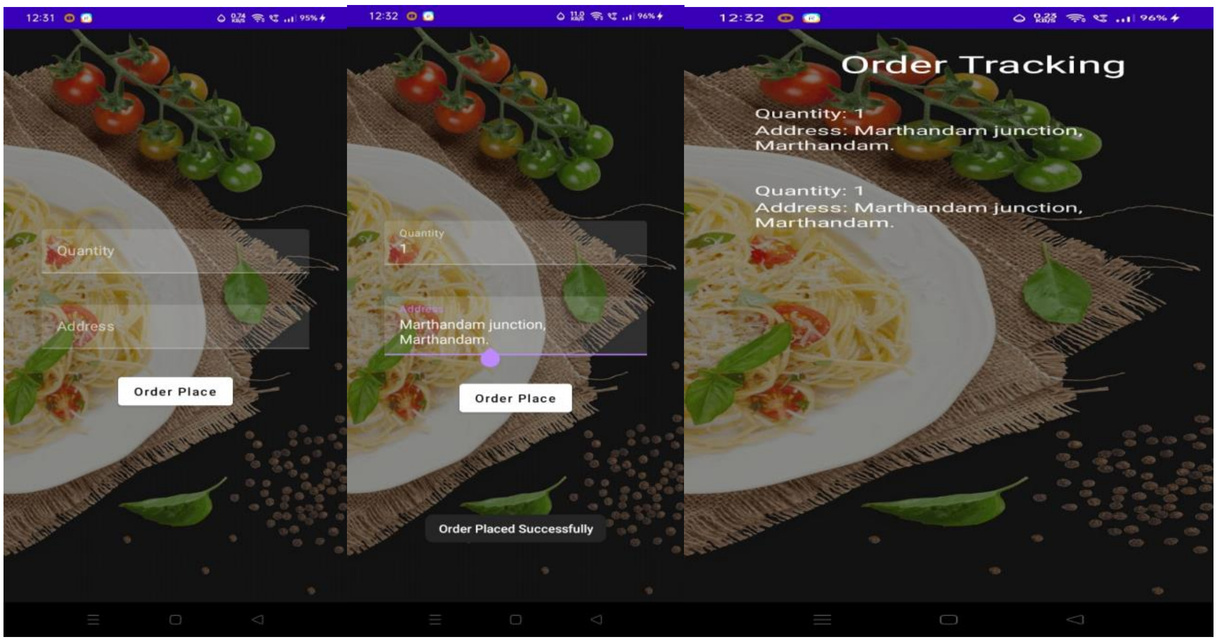

SnackOrdering [C:\Users\jino\Desktop\SnackOrdering-main] - MainPage.kt [SnackOrdering.app.main]

```
            modifier = Modifier.fillMaxSize(),

            content = { this: LazyListScope
                items(FoodList) { this: LazyItemScope  item ->
                    PopularFood(context = context,drawable = item.drawable,
                    abstract class Context
                }
            },
            verticalArrangement = Arrangement.spacedBy(16.dp))
}


@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun FinalView(mainPage: MainPage) {
    SnackOrderingTheme {
        Scaffold() { it: PaddingValues
            val context = LocalContext.current
            App(context)
        }
    }
}
```

Launch succeeded (3 minutes ago)



Quantity
1
Address
Marthandam junction, Marthandam.

Order Placed Successfully

Order Tracking

Quantity: 1
Address: Marthandam junction, Marthandam.

Quantity: 1
Address: Marthandam junction, Marthandam.

4. ADVANTAGES AND DISADVANTAGES

ADVANTAGES

i. Exposure to new customers.

ii. Online Snack ordering is convenient.

iii. More business opportunities.

iv. Stay ahead of the competition.

Greater reach.

v. Better customer data.

DISADVANTAGES

i. Price

ii.Limited Menu

iii. Preperation

iv. Quality of the food may suffer

v. The vibe of the resturant is missing

5. APPLICATION

The use of " A Customisable Snack Ordering and Delivery App" can be a convenient one that can be used in many application areas . That can be done by using mobile phones . We can implient it in on theatre's as much as possible to reduce the time and work done. We can also apply this in small retailer shops and also in big retailers to makes them so fast and convenient and easy to maintanable one.

Snack Ordering App can be applicable in many real time usage for the snack delivery at any place that the range of the application that has been designed.

CONCLUSION

Our project is only a humble venture to satisfy the needs to manage their project work. Several user friendly coding have also adopted. This package shall prove to be a powerful package in satisfying all the requirements of the school. The objective of software planning is to provide a frame work that enables the manger to make reasonable estimates made within a limited time frame at the beginning of the software project and should be updated regularly as the project progresses.

This project contains three parts, Background Management Platform, Website Public Page and Android Application. The Background Management Platform was implemented with Kotlin, XML and JPA framework. Website Public Page was achieved with Servlet/JSP and JavaBean. The Android framework was used in the Android Application. This project was a typical combination between a website and an Android application. The aim of the project was to help the restaurant owner to improve the efficiency of managing, meanwhile, help the customer to purchase snacks in different platforms easily. By now, the core function of this project has been implemented. The owner and employees in the company can manage snack and handle snack orders and so on. On the public page, customers can view snack information and purchase snacks. Also, the customer can order snacks from the Android platform. Developing the application made it possible to learn and practice the whole processes of agile development with Android studio frameworks as well as Android framework.

7. FUTURE SCOPE

In Android Studio using Kotlin, it can be summarized that the future scope of the project circles around maintaining information regarding:

🕐We can add printer in future.

🕐We can give more advance software for Online Snack Ordering System including more facilities.

🕐We will host the platform on online servers to make it accessible worldwide.

🕐Integrate multiple load balancers to distribute the loads of the system

🕐Create the master and slave database structure to reduce the overload of the database queries.

🕐Implement the backup mechanism for taking backup of code base and database on regular basis on different servers.

The above-mentioned points are the enhancements which can be done to increase the applicability and usage of this project. Here we can maintain the records of Snack Item and Category. Also, as it can be seen that now-a-days the players are versatile, i.e. so there is a scope for introducing a method to maintain the Online Food Ordering System. Enhancements can be done to maintain all the Snack Item, Category, Customer, Order, Confirm Order.

We have left all the options open so that if there is any other future requirement in the system by the user for the enhancement of the system then it is possible to implement them. In the last we would like to thanks all the persons involved in the development of the system directly or indirectly. We hope that the project will serve its purpose for which it is develop there by underlining success of process.

8. APPENDIX

Database 1 (Create User Data Class)

```
package com.example.snackordering

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "user_table")
data class User(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?,
    @ColumnInfo(name = "email") val email: String?,
    @ColumnInfo(name = "password") val password: String?,

    )
```

Create An UserDao Interface

```
package com.example.snackordering

import androidx.room.*

@Dao
interface UserDao {

    @Query("SELECT * FROM user_table WHERE email = :email")
    suspend fun getUserByEmail(email: String): User?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUser(user: User)

    @Update
    suspend fun updateUser(user: User)

    @Delete
    suspend fun deleteUser(user: User)
}
```

Create An UserDatabase Class

```
        package com.example.snackordering
```

```
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase


@Database(entities = [User::class], version = 1)
abstract class UserDatabase : RoomDatabase() {


    abstract fun userDao(): UserDao


    companion object {


        @Volatile
        private var instance: UserDatabase? = null


        fun getDatabase(context: Context): UserDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    UserDatabase::class.java,
                    "user_database"
                ).build()
                instance = newInstance
                NewInstance
            }
        }
    }
}
```

Create An UserDatabaseHelper Class

```
package com.example.snackordering

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class UserDatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "UserDatabase.db"

        private const val TABLE_NAME = "user_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_FIRST_NAME = "first_name"
        private const val COLUMN_LAST_NAME = "last_name"
        private const val COLUMN_EMAIL = "email"
        private const val COLUMN_PASSWORD = "password"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        val createTable = "CREATE TABLE $TABLE_NAME (" +
                "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "$COLUMN_FIRST_NAME TEXT, " +
                "$COLUMN_LAST_NAME TEXT, " +
                "$COLUMN_EMAIL TEXT, " +
                "$COLUMN_PASSWORD TEXT" +
                ")"

        db?.execSQL(createTable)
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }

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

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

    @SuppressLint("Range")
    fun getAllUsers(): List<User> {
        val users = mutableListOf<User>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        if (cursor.moveToFirst()) {
            do {
                val user = User(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                    lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                    email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                    password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
                )
                users.add(user)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return users
    }

}
```

DATABASE 2

Create Order Data Class

```
package com.example.snackordering

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "order_table")
data class Order(
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "quantity") val quantity: String?,
    @ColumnInfo(name = "address") val address: String?,
)
```

Create OrderDao Interface

```
package com.example.snackordering


            import androidx.room.*


            @Dao
            interface OrderDao {


                @Query("SELECT * FROM order_table WHERE address= :address")
                suspend fun getOrderByAddress(address: String): Order?


                @Insert(onConflict = OnConflictStrategy.REPLACE)
                suspend fun insertOrder(order: Order)


                @Update
                suspend fun updateOrder(order: Order)


                @Delete
                suspend fun deleteOrder(order: Order)
            }
```

Create OrderDatabase Class

```
package com.example.snackordering
```

```
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase


@Database(entities = [Order::class], version = 1)
abstract class OrderDatabase : RoomDatabase() {


    abstract fun orderDao(): OrderDao


    companion object {


        @Volatile
        private var instance: OrderDatabase? = null


        fun getDatabase(context: Context): OrderDatabase {
            return instance ?: synchronized(this) {
                val newInstance = Room.databaseBuilder(
                    context.applicationContext,
                    OrderDatabase::class.java,
                    "order_database"
                ).build()
                instance = newInstance
                newInstance
            }
        }
    }
}
```
Create OrderDatabaseHelper Class

```
package com.example.snackordering

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class OrderDatabaseHelper(context: Context) :
        SQLiteOpenHelper(context, DATABASE_NAME, null,DATABASE_VERSION){
```

```
    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "OrderDatabase.db"

        private const val TABLE_NAME = "order_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_QUANTITY = "quantity"
        private const val COLUMN_ADDRESS = "address"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        val createTable = "CREATE TABLE $TABLE_NAME (" +
                "${COLUMN_ID} INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "${COLUMN_QUANTITY} Text, " +
                "${COLUMN_ADDRESS} TEXT " +
                ")"

        db?.execSQL(createTable)
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }

    fun insertOrder(order: Order) {
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_QUANTITY, order.quantity)
        values.put(COLUMN_ADDRESS, order.address)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }



    @SuppressLint("Range")
    fun getOrderByQuantity(quantity: String): Order? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_QUANTITY = ?", arrayOf(quantity))
        var order: Order? = null
        if (cursor.moveToFirst()) {
            order = Order(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                quantity = cursor.getString(cursor.getColumnIndex(COLUMN_QUANTITY)),
                address = cursor.getString(cursor.getColumnIndex(COLUMN_ADDRESS)),
            )
        }
        cursor.close()
        db.close()
        return order
    }
    @SuppressLint("Range")
    fun getOrderById(id: Int): Order? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE $COLUMN_ID =
?", arrayOf(id.toString()))
        var order: Order? = null
        if (cursor.moveToFirst()) {
            order = Order(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                quantity = cursor.getString(cursor.getColumnIndex(COLUMN_QUANTITY)),
                address = cursor.getString(cursor.getColumnIndex(COLUMN_ADDRESS)),
            )
        }
        cursor.close()
        db.close()
        return order
    }

    @SuppressLint("Range")
    fun getAllOrders(): List<Order> {
        val orders = mutableListOf<Order>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        if (cursor.moveToFirst()) {
            do {
                val order = Order(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    quantity = cursor.getString(cursor.getColumnIndex(COLUMN_QUANTITY)),
                    address = cursor.getString(cursor.getColumnIndex(COLUMN_ADDRESS)),
                )
                orders.add(order)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return orders
    }

}
```

Creating LoginActivity.Kt With Database

```
package com.example.snackordering
```

```
import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
import com.example.snackordering.ui.theme.SnackOrderingTheme

class LoginActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            SnackOrderingTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    LoginScreen(this, databaseHelper)
                }
            }
        }
    }
}
@Composable
fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {

    Image(painterResource(id = R.drawable.order), contentDescription = "",
        alpha =0.3F,
        contentScale = ContentScale.FillHeight,

    )
```

```
var username by remember { mutableStateOf("") }
var password by remember { mutableStateOf("") }
var error by remember { mutableStateOf("") }

Column(
    modifier = Modifier.fillMaxSize(),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center
) {

    Text(
        fontSize = 36.sp,
        fontWeight = FontWeight.ExtraBold,
        fontFamily = FontFamily.Cursive,
        color = Color.White,
        text = "Login"
    )
    Spacer(modifier = Modifier.height(10.dp))

    TextField(
        value = username,
        onValueChange = { username = it },
        label = { Text("Username") },
        modifier = Modifier.padding(10.dp)
            .width(280.dp)
    )

    TextField(
        value = password,
        onValueChange = { password = it },
        label = { Text("Password") },
        modifier = Modifier.padding(10.dp)
            .width(280.dp)
    )

    if (error.isNotEmpty()) {
        Text(
            text = error,
            color = MaterialTheme.colors.error,
            modifier = Modifier.padding(vertical = 16.dp)
        )
    }

    Button(
        onClick = {
            if (username.isNotEmpty() && password.isNotEmpty()) {
                val user = databaseHelper.getUserByUsername(username)
                if (user != null && user.password == password) {
```

```
                error = "Successfully log in"
                context.startActivity(
                    Intent(
                        context,
                        MainPage::class.java
                    )
                )
                //onLoginSuccess()
            }
                if (user != null && user.password == "admin") {
                    error = "Successfully log in"
                    context.startActivity(
                        Intent(
                            context,
                            AdminActivity::class.java
                        )
                    )
                }
                else {
                    error = "Invalid username or password"
                }

        } else {
            error = "Please fill all fields"
        }
    },
    modifier = Modifier.padding(top = 16.dp)
) {
    Text(text = "Login")
}
Row {
    TextButton(onClick = {context.startActivity(
        Intent(
            context,
            MainActivity::class.java
        )
    )}
    )
    { Text(color = Color.White,text = "Sign up") }
    TextButton(onClick = {
    })

    {
        Spacer(modifier = Modifier.width(60.dp))
        Text(color = Color.White,text = "Forget password?")
    }
}
}
```

```
}
private fun startMainPage(context: Context) {
    val intent = Intent(context, MainPage::class.java)
    ContextCompat.startActivity(context, intent, null)
}
```

Creating MainActivity.Kt With Database

```
package com.example.snackordering

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
import com.example.snackordering.ui.theme.SnackOrderingTheme

class MainActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            SnackOrderingTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {

                    RegistrationScreen(this,databaseHelper)
                }
            }
        }
    }
}
```

```
@Composable
fun RegistrationScreen(context: Context, databaseHelper: UserDatabaseHelper) {

    Image(
        painterResource(id = R.drawable.order), contentDescription = "",
        alpha =0.3F,
        contentScale = ContentScale.FillHeight,

        )

    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {

        Text(
            fontSize = 36.sp,
            fontWeight = FontWeight.ExtraBold,
            fontFamily = FontFamily.Cursive,
            color = Color.White,
            text = "Register"
        )

        Spacer(modifier = Modifier.height(10.dp))
        TextField(
            value = username,
            onValueChange = { username = it },
            label = { Text("Username") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)

        )

        TextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("Email") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)
        )

        TextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Password") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp)
        )


        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }

        Button(
            onClick = {
                if (username.isNotEmpty() && password.isNotEmpty() && email.isNotEmpty()) {
                    val user = User(
                        id = null,
                        firstName = username,
                        lastName = null,
                        email = email,
                        password = password
                    )
                    databaseHelper.insertUser(user)
                    error = "User registered successfully"
                    // Start LoginActivity using the current context
                    context.startActivity(
                        Intent(
                            context,
                            LoginActivity::class.java
                        )
                    )

                } else {
                    error = "Please fill all fields"
                }
            },
            modifier = Modifier.padding(top = 16.dp)
        ) {
            Text(text = "Register")
```

```
        }
        Spacer(modifier = Modifier.width(10.dp))
        Spacer(modifier = Modifier.height(10.dp))

        Row() {
            Text(
                modifier = Modifier.padding(top = 14.dp), text = "Have an account?"
            )
            TextButton(onClick = {
                context.startActivity(
                    Intent(
                        context,
                        LoginActivity::class.java
                    )
                )
            })

            {
                Spacer(modifier = Modifier.width(10.dp))
                Text(text = "Log in")
            }
        }
    }
}
private fun startLoginActivity(context: Context) {
    val intent = Intent(context, LoginActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
```

Creating MainPage.Kt File

```
package com.example.snackordering

import android.annotation.SuppressLint
import android.content.Context
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.annotation.DrawableRes
import androidx.annotation.StringRes
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.Text
import androidx.compose.ui.unit.dp
import androidx.compose.ui.graphics.RectangleShape
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat.startActivity
import com.example.snackordering.ui.theme.SnackOrderingTheme

import android.content.Intent as Intent1


class MainPage : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            SnackOrderingTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    FinalView(this)
                    val context = LocalContext.current
                    //PopularFoodColumn(context)
                }
            }
        }
    }
}



@Composable
fun TopPart() {

    Row(
        modifier = Modifier
            .fillMaxWidth()
            .background(Color(0xffeceef0)),  Arrangement.SpaceBetween
    ) {
        Icon(
            imageVector = Icons.Default.Add,  contentDescription  = "Menu Icon",
            Modifier

                .clip(CircleShape)
                .size(40.dp),
            tint  = Color.Black,
        )
        Column(horizontalAlignment  = Alignment.CenterHorizontally)  {
            Text(text = "Location",  style = MaterialTheme.typography.subtitle1,  color = Color.Black)
            Row {
                Icon(
                    imageVector = Icons.Default.LocationOn,
                    contentDescription  = "Location",
                    tint  = Color.Red,
                )
                Text(text = "Accra" , color  = Color.Black)
            }

        }
        Icon(
            imageVector = Icons.Default.Notifications,  contentDescription  = "Notification  Icon",

            Modifier
                .size(45.dp),
            tint  = Color.Black,
        )
    }
}

@Composable
fun CardPart() {
    Card(modifier  = Modifier.size(width  = 310.dp,  height = 150.dp),  RoundedCornerShape(20.dp))  {
        Row(modifier  = Modifier.padding(10.dp),  Arrangement.SpaceBetween) {
            Column(verticalArrangement  = Arrangement.spacedBy(12.dp))  {
                Text(text = "Get Special  Discounts")
                Text(text = "up  to 85%", style = MaterialTheme.typography.h5)
                Button(onClick  = {}, colors  = ButtonDefaults.buttonColors(Color.White))  {
                    Text(text = "Claim  voucher",  color  = MaterialTheme.colors.surface)
                }
            }
            Image(
                painter = painterResource(id  = R.drawable.food_tip_im),
                contentDescription  = "Food  Image", Modifier.size(width  = 100.dp,  height  = 200.dp)
            )
        }
    }
}



@Composable
fun PopularFood(
    @DrawableRes drawable: Int,
    @StringRes text1: Int,
    context: Context
) {
    Card(
        modifier = Modifier
            .padding(top=20.dp, bottom = 20.dp, start = 65.dp)
            .width(250.dp)

    ) {
        Column(
            verticalArrangement = Arrangement.Top,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Spacer(modifier = Modifier.padding(vertical = 5.dp))
            Row(
                modifier = Modifier
                    .fillMaxWidth(0.7f), Arrangement.End
            ) {
                Icon(
                    imageVector = Icons.Default.Star,
                    contentDescription = "Star Icon",
                    tint = Color.Yellow
                )
                Text(text = "4.3", fontWeight = FontWeight.Black)
            }
            Image(
                painter = painterResource(id = drawable),
                contentDescription = "Food Image",
                contentScale = ContentScale.Crop,
                modifier = Modifier
                    .size(100.dp)
                    .clip(CircleShape)
            )
            Text(text = stringResource(id = text1), fontWeight = FontWeight.Bold)
            Row(modifier = Modifier.fillMaxWidth(0.7f), Arrangement.SpaceBetween) {
                /*TODO Implement Prices for each card*/
                Text(
                    text = "$50",
                    style = MaterialTheme.typography.h6,
                    fontWeight = FontWeight.Bold,
                    fontSize = 18.sp
                )

                IconButton(onClick = {

                    //var no=FoodList.lastIndex;
                    //Toast.
                    val intent = Intent1(context, TargetActivity::class.java)
                    context.startActivity(intent)

                }) {
                    Icon(
                        imageVector = Icons.Default.ShoppingCart,
                        contentDescription = "shopping cart",
                    )
                }
            }
        }
    }
}




private val FoodList = listOf(
    R.drawable.sandwish to R.string.sandwich,
    R.drawable.sandwish to R.string.burgers,
    R.drawable.pack to R.string.pack,
    R.drawable.pasta to R.string.pasta,
    R.drawable.tequila to R.string.tequila,
    R.drawable.wine to R.string.wine,
    R.drawable.salad to R.string.salad,
    R.drawable.pop to R.string.popcorn
).map { DrawableStringPair(it.first, it.second) }

private data class DrawableStringPair(
    @DrawableRes val drawable: Int,
    @StringRes val text1: Int
)


@Composable
fun App(context: Context) {

    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(Color(0xffeceef0))
            .padding(10.dp),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Surface(modifier = Modifier, elevation = 5.dp) {
            TopPart()
        }
        Spacer(modifier = Modifier.padding(10.dp))
        CardPart()

        Spacer(modifier = Modifier.padding(10.dp))
        Row(modifier = Modifier.fillMaxWidth(), Arrangement.SpaceBetween) {
            Text(text = "Popular Food", style = MaterialTheme.typography.h5, color = Color.Black)
            Text(text = "view all", style = MaterialTheme.typography.subtitle1, color = Color.Black)
        }
        Spacer(modifier = Modifier.padding(10.dp))
        PopularFoodColumn(context) // <- call the function with parentheses
    }
}




@Composable
fun PopularFoodColumn(context: Context) {

    LazyColumn(
        modifier = Modifier.fillMaxSize(),

        content = {
            items(FoodList) { item ->
                PopularFood(context = context,drawable = item.drawable, text1 = item.text1)
                abstract class Context
            }
        },
        verticalArrangement = Arrangement.spacedBy(16.dp))
}


@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun FinalView(mainPage: MainPage) {
    SnackOrderingTheme {
        Scaffold() {
            val context = LocalContext.current
            App(context)
        }
    }
}
```

Creating TargetActivity.Kt

```
package com.example.snackordering

import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.textInputServiceFactory
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.core.content.ContextCompat
import com.example.snackordering.ui.theme.SnackOrderingTheme

class TargetActivity : ComponentActivity() {
    private lateinit var orderDatabaseHelper: OrderDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        orderDatabaseHelper = OrderDatabaseHelper(this)
        setContent {
            SnackOrderingTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier
                        .fillMaxSize()
                        .background(Color.White)

                ) {
                    Order(this, orderDatabaseHelper)
                    val orders = orderDatabaseHelper.getAllOrders()
                   Log.d("swathi", orders.toString())
                }
            }
        }
    }
}

@Composable
fun Order(context: Context, orderDatabaseHelper: OrderDatabaseHelper){
    Image(painterResource(id = R.drawable.order), contentDescription = "",
        alpha =0.5F,
    contentScale = ContentScale.FillHeight)
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center) {

        val mContext = LocalContext.current
        var quantity by remember { mutableStateOf("") }
        var address by remember { mutableStateOf("") }
        var error by remember { mutableStateOf("") }


        TextField(value = quantity, onValueChange = {quantity=it},
            label = { Text("Quantity") },
           keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp))

        Spacer(modifier = Modifier.padding(10.dp))

        TextField(value = address, onValueChange = {address=it},
            label = { Text("Address") },
            modifier = Modifier
                .padding(10.dp)
                .width(280.dp))

        Spacer(modifier = Modifier.padding(10.dp))


        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }
        Button(onClick = {
            if( quantity.isNotEmpty() and address.isNotEmpty()){
                val order = Order(
                    id = null,
                    quantity = quantity,
                    address = address
                )
                orderDatabaseHelper.insertOrder(order)
            Toast.makeText(mContext, "Order Placed Successfully", Toast.LENGTH_SHORT).show()}
        },
            colors = ButtonDefaults.buttonColors(backgroundColor = Color.White))
        {
            Text(text = "Order Place", color = Color.Black)
        }


    }
}
private fun startMainPage(context: Context) {
    val intent = Intent(context, LoginActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}
```

Creating AdminActivity.Kt

```
package com.example.snackordering

import android.icu.text.SimpleDateFormat
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.LazyRow
import androidx.compose.foundation.lazy.items
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.snackordering.ui.theme.SnackOrderingTheme
import java.util.*
```

```
class AdminActivity : ComponentActivity() {
    private lateinit var orderDatabaseHelper: OrderDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        orderDatabaseHelper = OrderDatabaseHelper(this)
        setContent {
            SnackOrderingTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    val data=orderDatabaseHelper.getAllOrders();
                    Log.d("swathi" ,data.toString())
                    val order = orderDatabaseHelper.getAllOrders()
                    ListListScopeSample(order)
                }
            }
        }
    }
}

@Composable
fun ListListScopeSample(order: List<Order>) {
    Image(
        painterResource(id = R.drawable.order), contentDescription = "",
        alpha =0.5F,
        contentScale = ContentScale.FillHeight)
    Text(text = "Order Tracking", modifier = Modifier.padding(top = 24.dp, start = 106.dp, bottom = 24.dp
), color = Color.White, fontSize = 30.sp)
    Spacer(modifier = Modifier.height(30.dp))
    LazyRow(
        modifier = Modifier
            .fillMaxSize()
            .padding(top = 80.dp),

        horizontalArrangement = Arrangement.SpaceBetween
    ){
        item {

            LazyColumn {
                items(order) { order ->
                    Column(modifier = Modifier.padding(top = 16.dp, start = 48.dp, bottom = 20.dp)) {
                        Text("Quantity: ${order.quantity}")
                        Text("Address: ${order.address}")
                    }
                }
            }
        }

    }
}
```

Modifying AndroidManifest.Xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@drawable/fast_food"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.SnackOrdering"
        tools:targetApi="31">
        <activity
            android:name=".AdminActivity"
            android:exported="false"
            android:label="@string/title_activity_admin"
            android:theme="@style/Theme.SnackOrdering" />
        <activity
            android:name=".LoginActivity"
            android:exported="true"
            android:label="SnackSquad"
            android:theme="@style/Theme.SnackOrdering">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".TargetActivity"
            android:exported="false"
            android:label="@string/title_activity_target"
            android:theme="@style/Theme.SnackOrdering" />
        <activity
            android:name=".MainPage"
            android:exported="false"
            android:label="@string/title_activity_main_page"
            android:theme="@style/Theme.SnackOrdering" />
        <activity
            android:name=".MainActivity"
            android:exported="false"
            android:label="MainActivity"
            android:theme="@style/Theme.SnackOrdering" />
    </application>

</manifest>
```