A CUSTOMIZABLE SNACK OREDERING AND DELIVERY APP

Team Members	NM ID
SANTHOSH S	9239E76A12E0D27873E47222BED10DA2
GIRITHARAN E	B280C6A9C09E89EA2334209433E5823C
VIGNESH K	2DCA7564C6D1241A0CD4A4ABC78D4E41
HARISH M	65ED2B64CA5D3B685B648E35FFC48271

Project Description:

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

Required Initial Steps:

Step 1:

Download Android Studio

- Visit the official Android Studio download page
- [https://developer.android.com/studio]
- Review and accept the terms and conditions if prompted.
- The download will begin automatically.

Step 2:

Install Android Studio:

For Windows

- Run the installer: Double-click the downloaded .exe file.
- Follow the installation wizard:
- Choose installation options (use default settings unless you have specific preferences).

- Android Studio requires a minimum of 4 GB of RAM (recommended 8 GB), and a screen resolution of at least 1280x800.
- Install Android SDK: The installer will automatically install the Android SDK, Android Emulator, and other necessary tools.
- Complete installation: Click Finish once the process is complete.
- Follow the setup wizard: Android Studio will guide you through the rest of the installation.

Step 3:

First Launch and Setup:

- Choose UI Theme**: Select between Light or Dark theme.
- Install SDK Components**: Android Studio will download necessary SDK components (such as the latest Android platform and tools). This might take some time, depending on your internet speed.

Step 4:

Configure Android Studio:

- Install additional SDK packages: You might need to install certain packages depending on your Android development needs (e.g., specific Android API versions or system images for the emulator).
- Set up the Android Emulator: If you plan to use the Android Emulator for testing, you can set it up through the AVD (Android Virtual Device) Manager.

Step5:

Verify Installation:

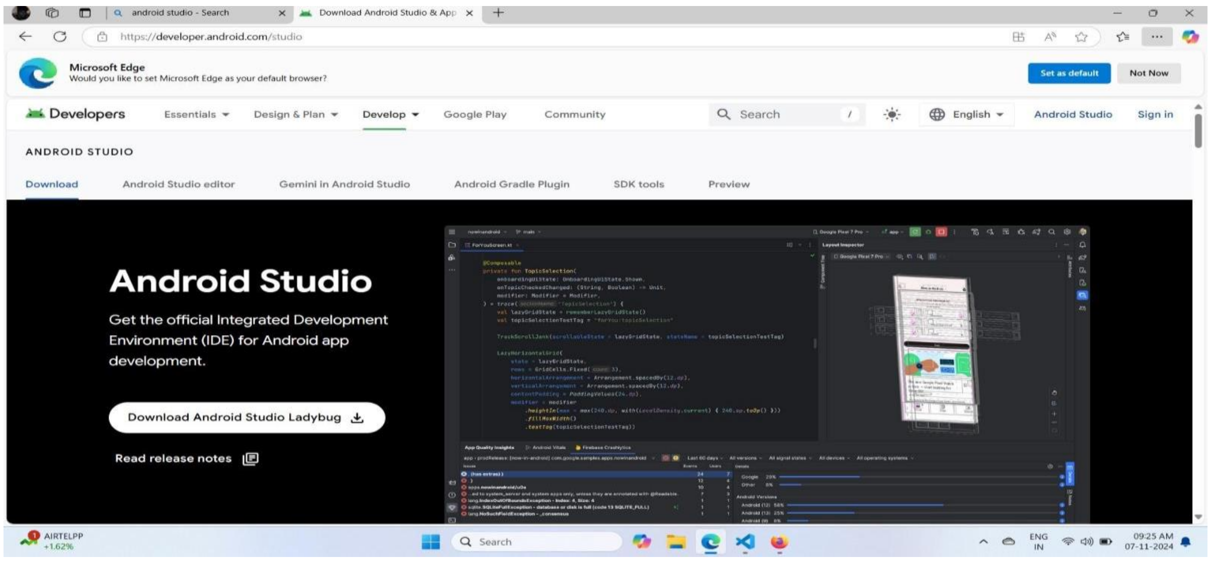
- Go to File → New → New Project.
- Choose a template (e.g., Empty Activity).
- Configure the project (e.g., name, package name).
- Click Finish.

Step 6:

Install Android Device/SDK Drivers (If needed):

- You have enabled USB debugging on the device (Settings → About phone → Tap Build number 7 times to unlock developer options → Developer options → Enable USB debugging).
- You may need to install specific drivers for your device (especially on Windows).

<u>**Screen Shorts:**</u>



Would you like to set Microsoft Edge as your default browser?

Get the official Integrated Development Environment (IDE) for Android app development.

Creating a New Project in an Android Studio

1. **Open Android Studio**
   - Launch Android Studio. If you're opening it for the first time, it may take a few moments to initialize.
2. **Start a New Project**
   - On the Welcome Screen, click "Start a new Android Studio project".
   - If you have an existing project open, go to File > New > New Project.
3. **Choose a Template**
   - Android Studio will prompt you to choose a project template. You can choose from various options, like:

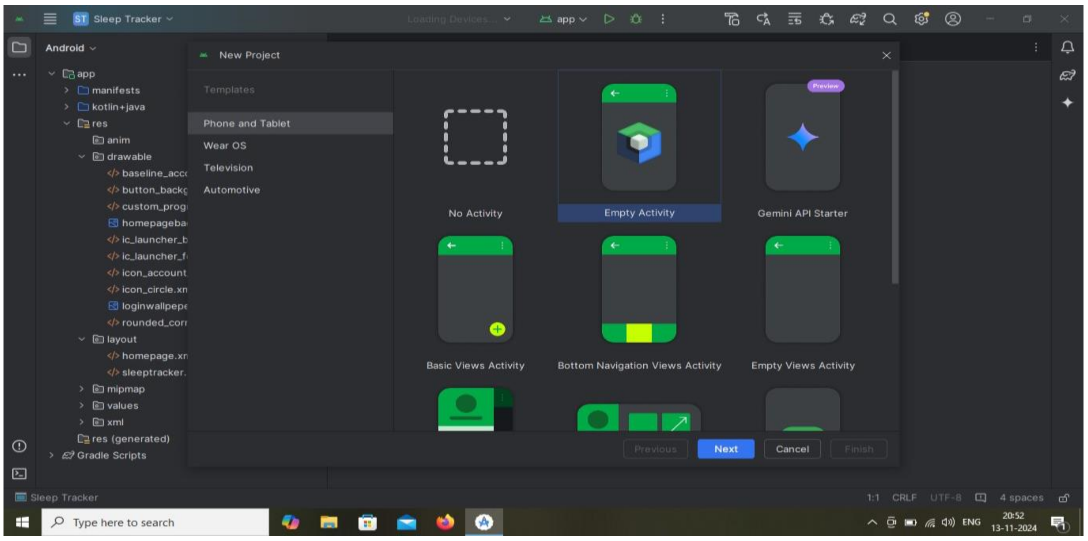
- Empty Activity (for a blank app).
- Basic Activity (with a toolbar and floating action button).
- Navigation Drawer Activity (with a side navigation menu).
- Fullscreen Activity (for apps that use the whole screen).

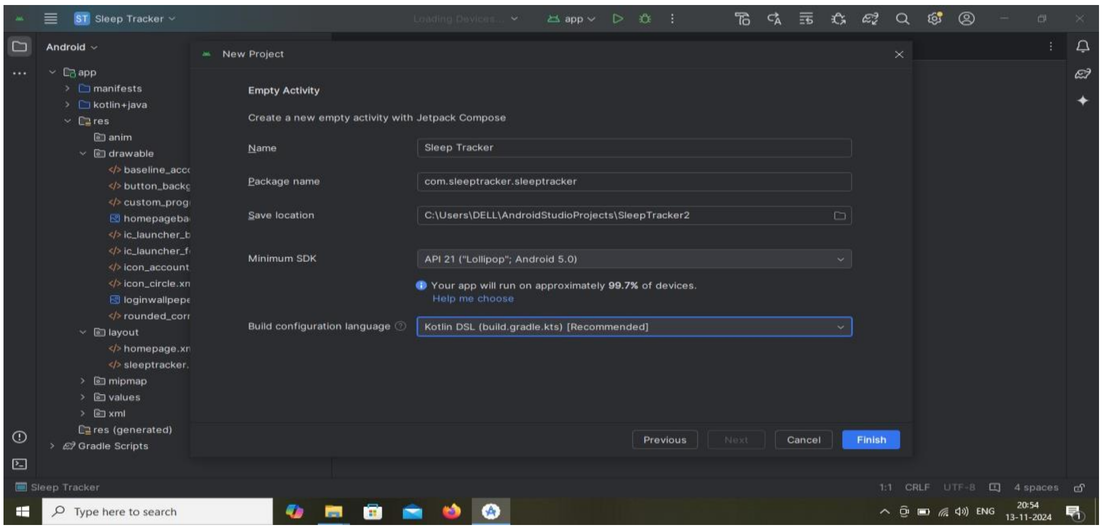
Choose "Empty Activity" for a simple start.

4. **Configure Your Project**
   - Name: Enter your app's name (e.g., "My First App").
   - Package Name: A unique identifier (usually in reverse domain format, like `com.example. myfirstapp`).
   - Save Location: Choose a location on your computer to store the project.
   - Language: Choose between Java or Kotlin. (Kotlin is now the preferred language for Android development.)
   - Minimum API Level: Select the lowest version of Android your app will support. Android Studio recommends an API level based on your target audience.
5. **Finish**
   - Click Finish to create your project. Android Studio will generate the necessary files and open your new project.
6. **Start Coding**
   - After the project is created, you'll see:
   - MainActivity.java/Kotlin: The main entry point for your app.
   - activity_main.xml: The layout file for the main activity (UI design).
   - You can now edit the code and design for your new Android app.
7. **Run Your App**
   - To test your app, you can either:
   - Use an Android Emulator by creating a virtual device (AVD).
   - Connect an Android device via USB and enable developer mode.

Click the green Run button (a play icon) in the top toolbar to build and run your app

Screenshots:





New Project

Empty Activity

Create a new empty activity with Jetpack Compose

Name: Sleep Tracker

Package name: com.sleeptracker.sleeptracker

Save location: C:\Users\DELL\AndroidStudioProjects\SleepTracker2

Minimum SDK: API 21 ("Lollipop"; Android 5.0)

Your app will run on approximately 99.7% of devices.
Help me choose

Build configuration language: Kotlin DSL (build.gradle.kts) [Recommended]

Add requirements dependencies:

1. Project-Level build.gradle (root-level)

This file is where you configure project-wide settings, dependencies for Gradle itself, and repositories that will be used for all modules.

2. App-Level build.gradle (module-level)

This file contains the configurations for the Android application module. It's where you specify your dependencies, Android-specific build options, and signing configurations.

3. Sync the Project

- Click on the Sync Now button that appears in the top right corner of Android Studio.
- Go to File > Sync Project with Gradle Files to sync your changes. This will fetch the necessary dependencies and sync the project.

4. Optional: Adding Repositories

- You can add other repositories if you're using custom libraries. For example, you might add a repository in your all projects section.

5. Optional: Adding Repositories

You can add other repositories if you're using custom libraries. For example, you might add a repository in your allprojects section

Make sure you include essential dependencies for Android development:

- AndroidX Libraries: Most modern Android libraries are part of AndroidX (e.g., androidx.appcompat, androidx.core).

- Google Libraries: If you're using Firebase, Google Play services, or other Google libraries, you'll need to include them as well.

6.Gradel Version and Wrapper

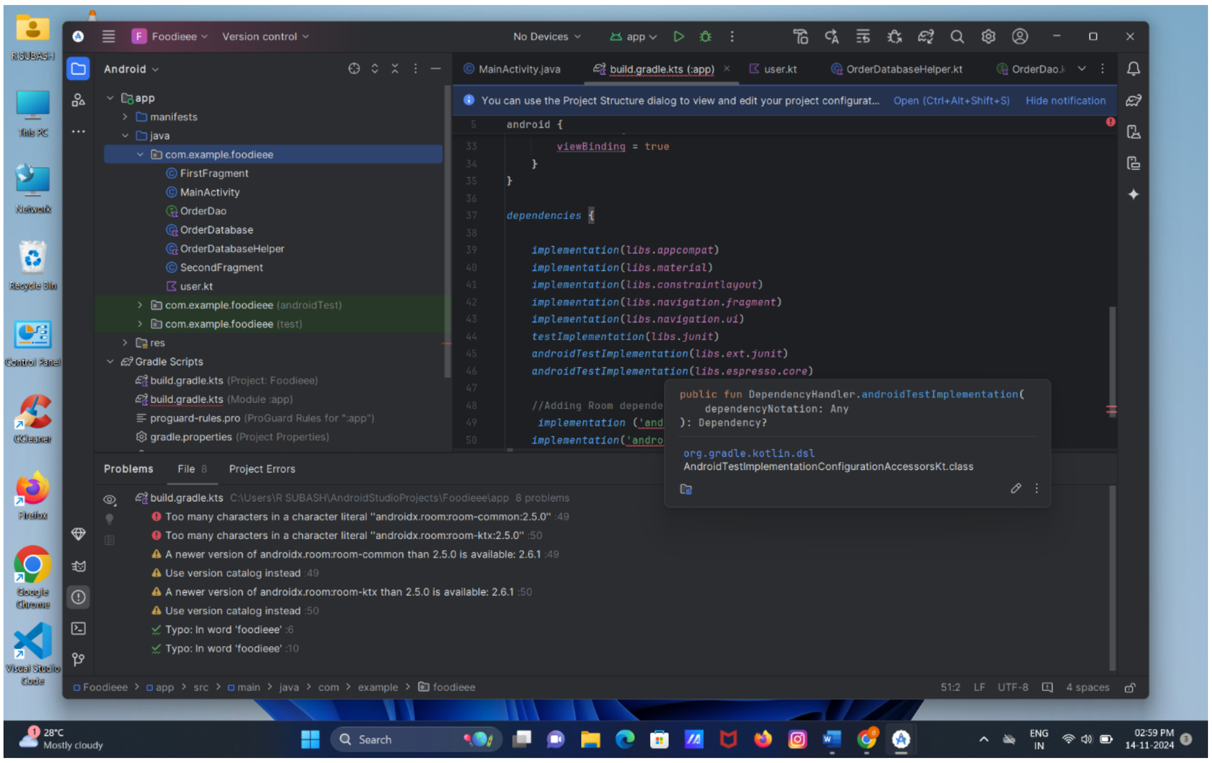
- In case you need to update the version of Gradle or the Android Gradle Plugin, modify the gradle-wrapper.properties file in the gradle/wrapper directory. You can update the version like this:

Final Notes:

- After modifying Gradle files, always perform a **Gradle sync** to make sure everything is set up properly.

- Ensure that all libraries and versions in the dependencies are compatible with each other

Screenshots:



You can use the Project Structure dialog to view and edit your project configurat... Open (Ctrl+Alt+Shift+S) Hide notification

```
android {
        viewBinding = true
    }
}

dependencies {

    implementation(libs.appcompat)
    implementation(libs.material)
    implementation(libs.constraintlayout)
    implementation(libs.navigation.fragment)
    implementation(libs.navigation.ui)
    testImplementation(libs.junit)
    androidTestImplementation(libs.ext.junit)
    androidTestImplementation(libs.espresso.core)

    //Adding Room depende
     implementation ('and
    implementation('andro
```

```
public fun DependencyHandler.androidTestImplementation(
    dependencyNotation: Any
): Dependency?

org.gradle.kotlin.dsl
AndroidTestImplementationConfigurationAccessorsKt.class
```

```
build.gradle.kts C:\Users\R SUBASH\AndroidStudioProjects\Foodieee\app 8 problems
  Too many characters in a character literal "androidx.room:room-common:2.5.0" :49
  Too many characters in a character literal "androidx.room:room-ktx:2.5.0" :50
  A newer version of androidx.room:room-common than 2.5.0 is available: 2.6.1 :49
  Use version catalog instead :49
  A newer version of androidx.room:room-ktx than 2.5.0 is available: 2.6.1 :50
  Use version catalog instead :50
  Typo: In word 'foodieee' :6
  Typo: In word 'foodieee' :10
```

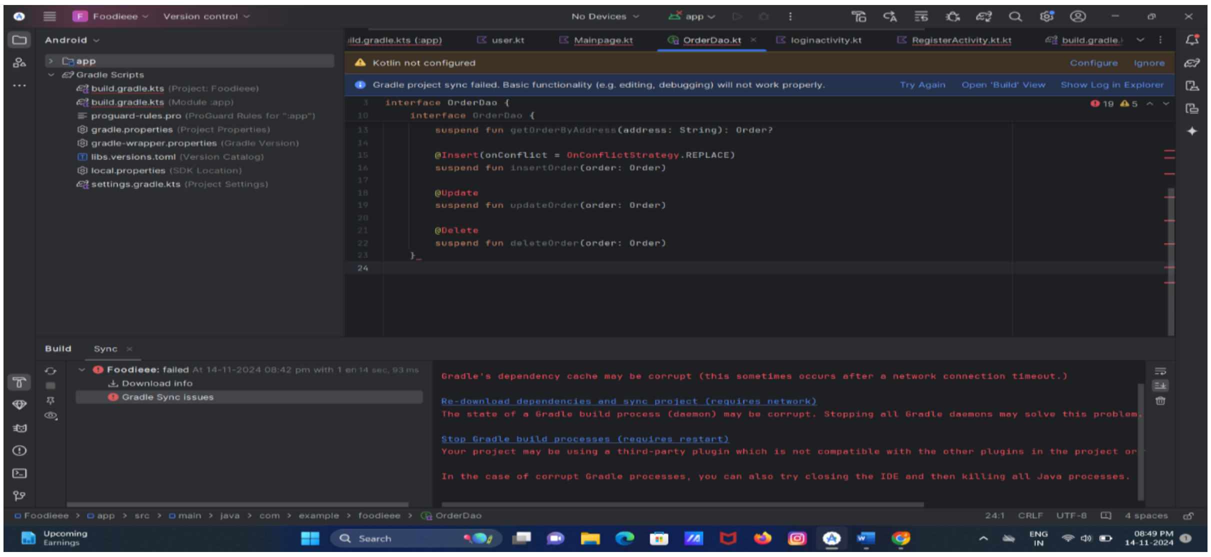
Creating the database classes

1.Database 1(create user data class)

Add Room dependencies to your project

In your build.gradle file (app module), add the following dependencies for Room

Screenshots:



Kotlin not configured Configure Ignore

Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not work properly. Try Again Open 'Build' View Show Log in Explorer

```
interface OrderDao {
    interface OrderDao {
        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)
    }
```

```
Foodieee: failed At 14-11-2024 08:42 pm with 1 en 14 sec, 93 ms
  Download info
  Gradle Sync issues

Gradle's dependency cache may be corrupt (this sometimes occurs after a network connection timeout.)

Re-download dependencies and sync project (requires network)
The state of a Gradle build process (daemon) may be corrupt. Stopping all Gradle daemons may solve this problem.

Stop Gradle build processes (requires restart)
Your project may be using a third-party plugin which is not compatible with the other plugins in the project or

In the case of corrupt Gradle processes, you can also try closing the IDE and then killing all Java processes.
```

Create an userdao interface

Step 1:

Add Room dependencies to build.gradle

First, we need to add the necessary Room dependencies to your project.

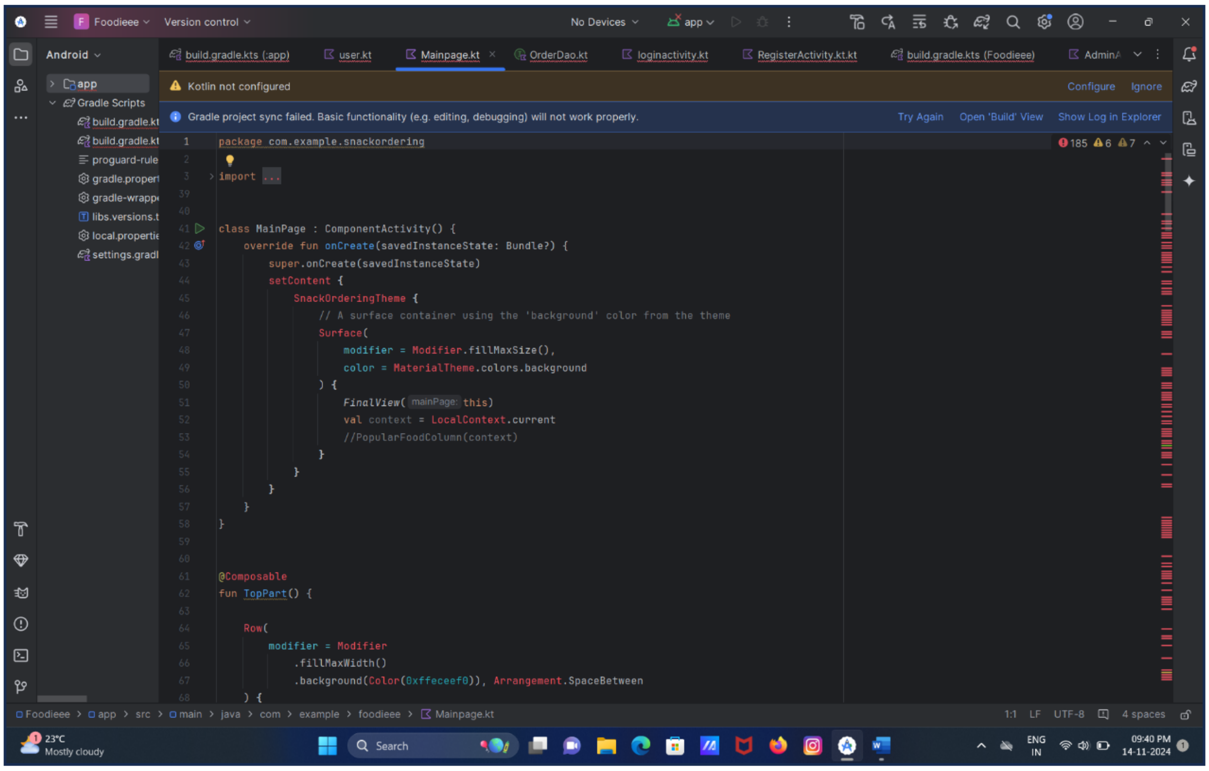
- Open the build.gradle file (app level).
- Add the following dependencies

Step 2:

Create the User data class

The User data class will represent the User entity in the database. Here's how you can define it:

Screenshots:



Kotlin not configured

Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not work properly.

```
package com.example.snackordering

import ...


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( mainPage: this)
                    val context = LocalContext.current
                    //PopularFoodColumn(context)
                }
            }
        }
    }
}


@Composable
fun TopPart() {

    Row(
        modifier = Modifier
            .fillMaxWidth()
            .background(Color(0xffeceef0)), Arrangement.SpaceBetween
    ) {
```

Foodieee > app > src > main > java > com > example > foodieee > Mainpage.kt

Create an user database class

Step 1:

Add Room dependencies to build.gradle

In your app/build.gradle file, add the following dependencies for Room:

Step 2:

Create the User Entity Class

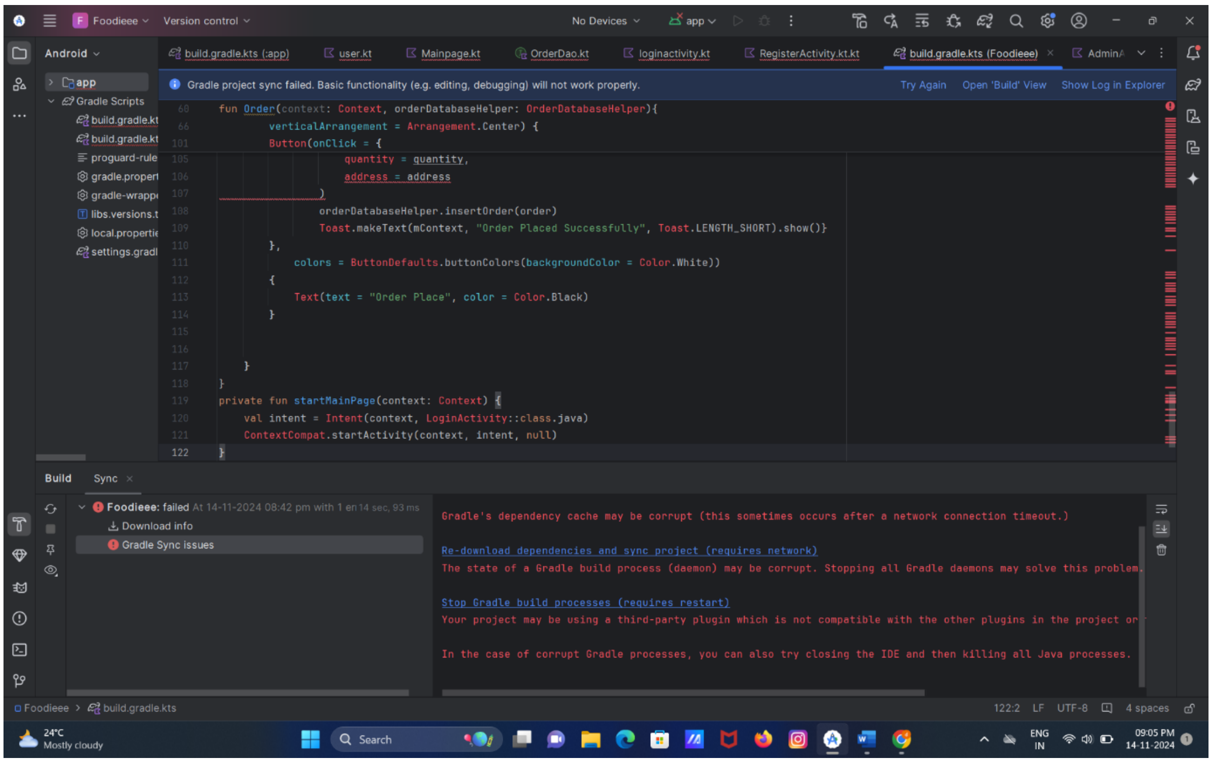
The User class represents the data you want to store in the database. Each User will be stored as a row in a users table.

Step 3:

Create the UserDao Interface

The UserDao (Data Access Object) defines the methods to interact with the database. These methods will be used to insert, update, delete, and query users.

Screenshots:



Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not work properly.

```
fun Order(context: Context, orderDatabaseHelper: OrderDatabaseHelper){
        verticalArrangement = Arrangement.Center) {
        Button(onClick = {
                    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)
}
```

Foodieee: failed At 14-11-2024 08:42 pm with 1 er 14 sec, 93 ms

```
Gradle's dependency cache may be corrupt (this sometimes occurs after a network connection timeout.)

Re-download dependencies and sync project (requires network)
The state of a Gradle build process (daemon) may be corrupt. Stopping all Gradle daemons may solve this problem.

Stop Gradle build processes (requires restart)
Your project may be using a third-party plugin which is not compatible with the other plugins in the project or

In the case of corrupt Gradle processes, you can also try closing the IDE and then killing all Java processes.
```

Create a userdatabase helper

Steps:

- Create a User Database Helper class.
- Define a User table schema.
- Add methods for database operations (Create, Read, Update, Delete).

Here’s an example implementation:

Step 1:

Create a UserDatabaseHelper class

Create a new Java or Kotlin class called UserDatabaseHelper (or any name you prefer).

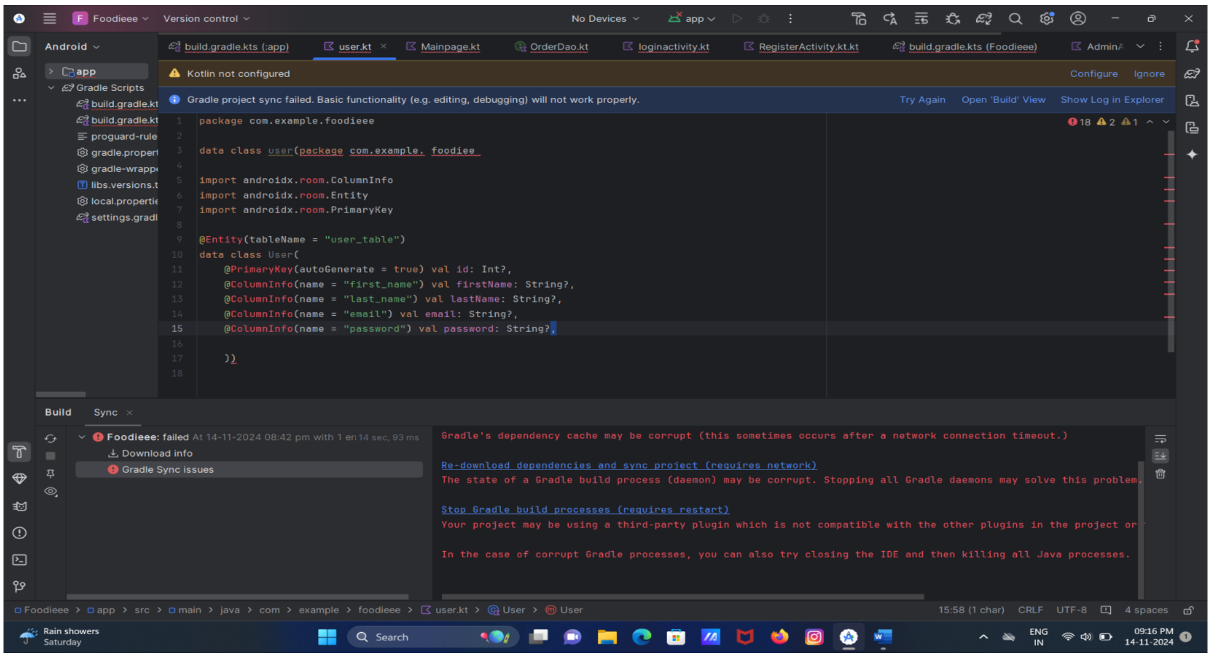
In Java

Step 2:

Using the UserDatabaseHelper class

Once you've created the helper class, you can use it in your activities or fragments to interact with the database. Here's an example of how you can use it to add a user and retrieve users.

Screenshots:



Kotlin not configured

Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not work properly.

```
package com.example.foodieee

data class user(package com.example. foodiee

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?,

    ))
```

```
Foodieee: failed At 14-11-2024 08:42 pm with 1 en 14 sec, 93 ms
  Download info
  Gradle Sync issues
```

```
Gradle's dependency cache may be corrupt (this sometimes occurs after a network connection timeout.)

Re-download dependencies and sync project (requires network)
The state of a Gradle build process (daemon) may be corrupt. Stopping all Gradle daemons may solve this problem.

Stop Gradle build processes (requires restart)
Your project may be using a third-party plugin which is not compatible with the other plugins in the project or

In the case of corrupt Gradle processes, you can also try closing the IDE and then killing all Java processes.
```

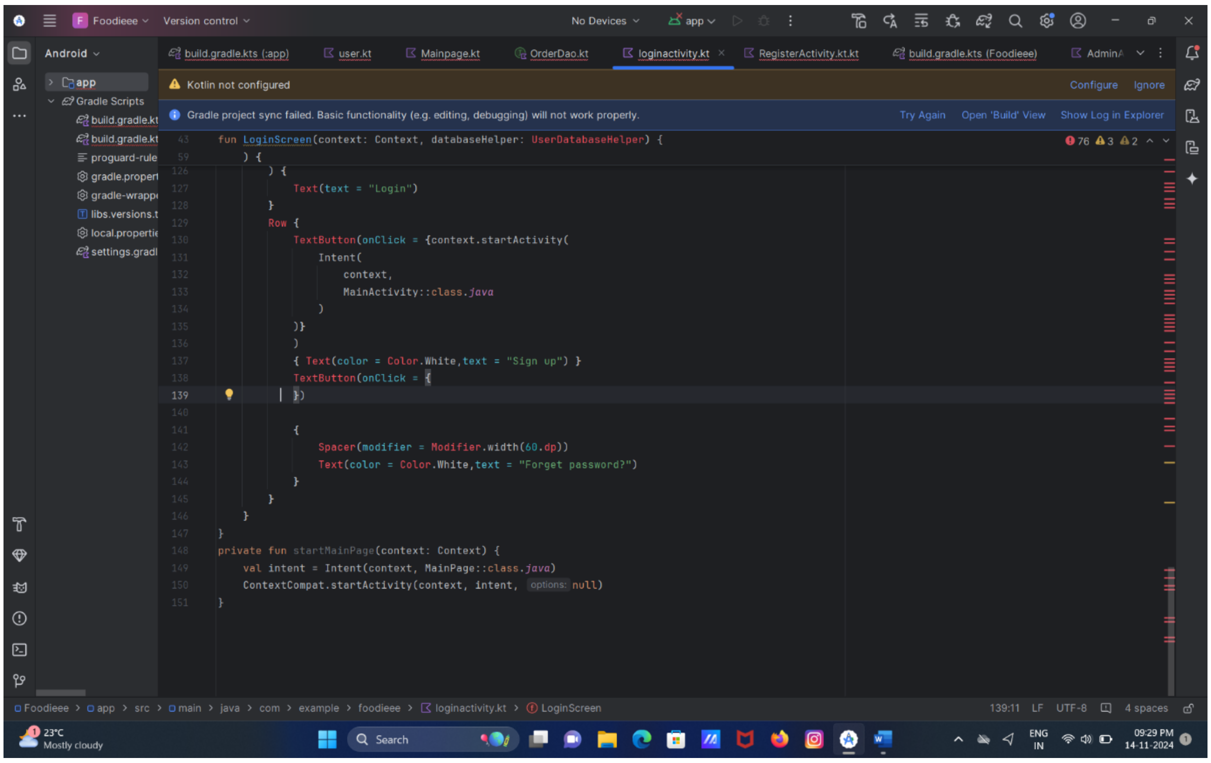
BULIDING APPLICATION UI AND CONNECTING TO DATABASE

Creating a login activity.kt with database

Steps:

- Create the Login Activity.kt file.
- Create the Login UI (Layout XML file).
- Connect the Login Activity to the User Database Helper.
- Handle user login (check credentials from the database)

Screenshots:



Kotlin not configured

Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not work properly.

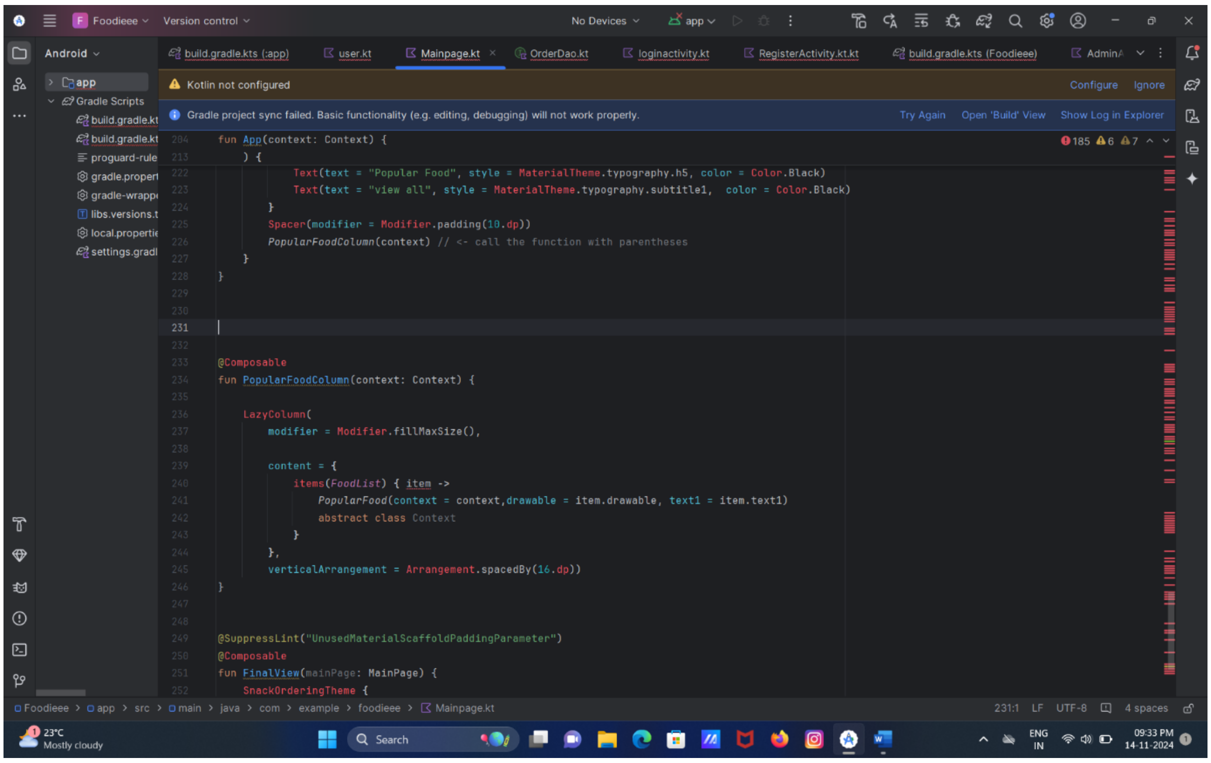
```
fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {
    ) {
        ) {
            Text(text = "Login")
        }
        Row {
            TextButton(onClick = {context.startActivity(
                Intent(
                    context,
                    MainActivity::class.java
                )
            )}
            )
            { 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, options: null)
}
```

Creating a mainactivity.kt with database

Steps:

- Create the MainActivity.kt file.
- Design the UI for the MainActivity.
- Interact with the database in MainActivity.kt

Screenshots:



Kotlin not configured

Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not work properly.

```kotlin
fun App(context: Context) {
    ) {
            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 {
```

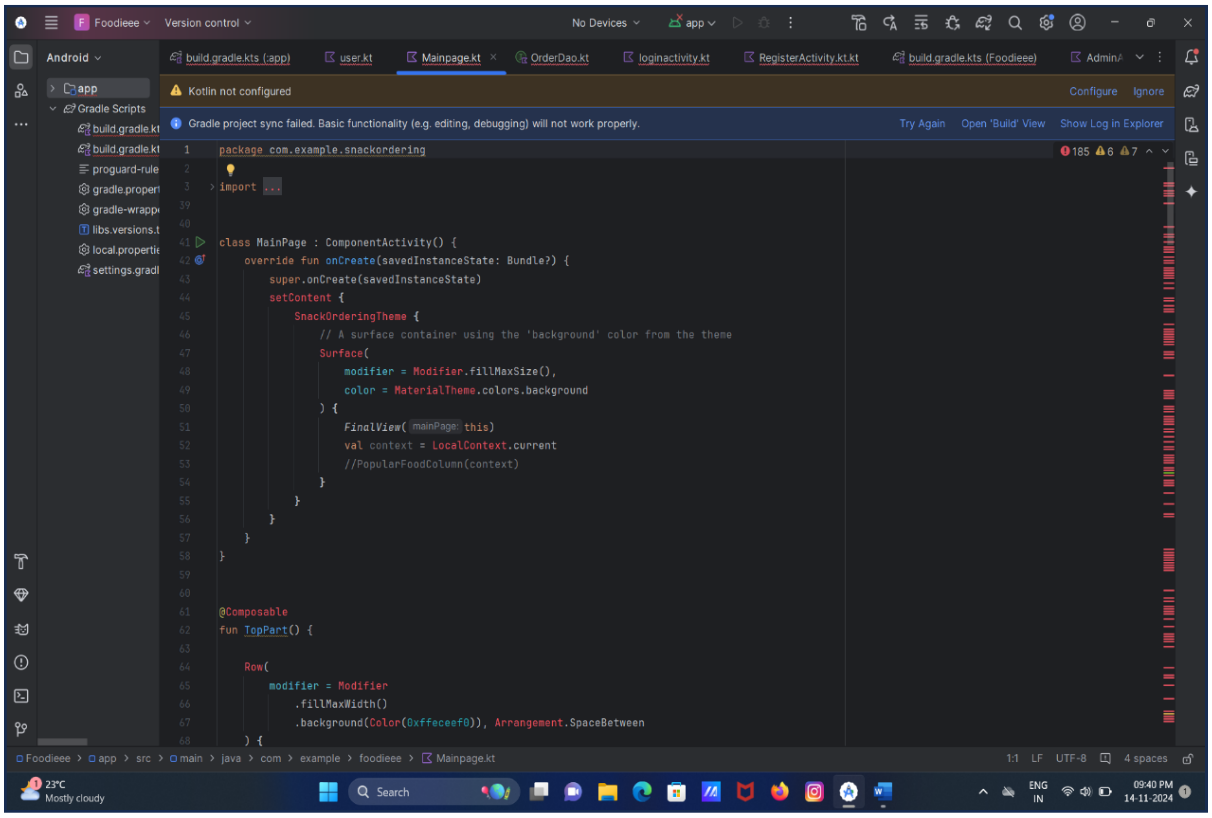
Creating a mainpage.kt file

steps:

- Open Project Explorer: In the Android Studio, open the Project view (usually on the left side).
- Navigate to the Package: Locate the package where you want to place MainPage.kt. Typically, you'll find it under app > java > com.example.yourprojectname.
- Create New Kotlin File:

    - Right-click on the desired package folder.
    - Select New > Kotlin File/Class.
- Name the File: In the pop-up window:
    - Enter MainPage as the file name.
    - Ensure the Kind is set to Class or File (choose Class if this will be an Activity, File if it contains functions only).
    - Click OK.
- Define the Class (if it’s an Activity):
    - If MainPage.kt will be an Activity, make it extend AppCompatActivity:

Screenshots:



Kotlin not configured

Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not work properly.

```kotlin
package com.example.snackordering

import ...


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( mainPage: this)
                    val context = LocalContext.current
                    //PopularFoodColumn(context)
                }
            }
        }
    }
}


@Composable
fun TopPart() {

    Row(
        modifier = Modifier
            .fillMaxWidth()
            .background(Color(0xffeceef0)), Arrangement.SpaceBetween
    ) {
```

Creating a target activity.kt

steps to follow:

- Open Project Explorer: Open the Project view in Android Studio.
- Navigate to the Package:
    - Find the package where you want to add TargetActivity.kt (usually under app > java > com.example.yourprojectname).
- Create New Activity:
    - Right-click on the package folder.
    - Select New > Activity > Empty Activity.
- Configure the New Activity:
    - In the pop-up window, set the Activity Name to TargetActivity.
    - The Layout Name will be auto-filled as activity_target.
    - Click Finish.
- TargetActivity.kt File:
    - Android Studio will automatically create the TargetActivity.kt file, which looks like this by default

Creating admin activity.kt

- Open Project Explorer: Open the Project view in Android Studio.
- Navigate to the Package:
    - Find the package where you want to add AdminActivity.kt (usually under app > java > com.example.yourprojectname).
- Create New Activity:
    - Right-click on the package folder.
    - Select New > Activity > Empty Activity.
- Configure the New Activity:

- In the pop-up window:
    - Set the Activity Name to AdminActivity.
    - The Layout Name will be automatically set to activity_admin.
- Click Finish.

 AdminActivity.kt File:

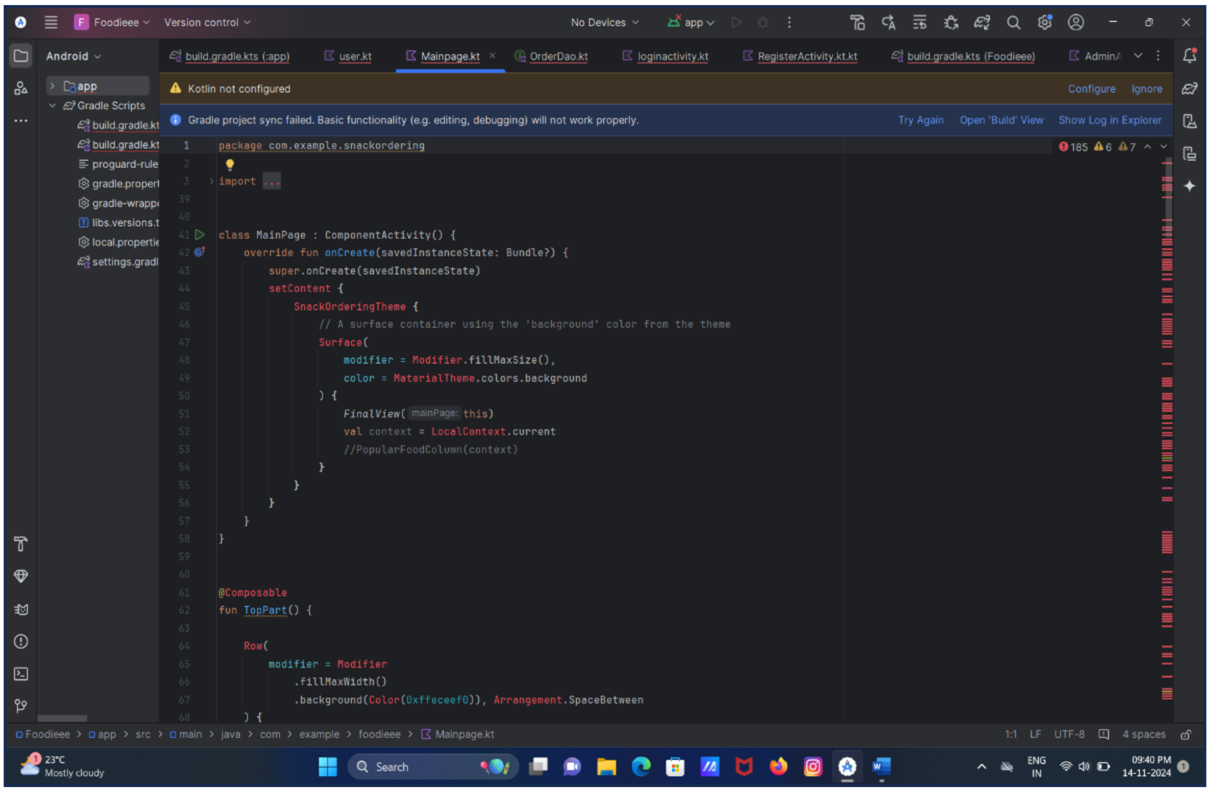
- Android Studio will create the AdminActivity.kt file with a basic setup

Modifying AndroidManifest.XML

Steps to Modify AndroidManifest.xml

- Locate the Manifest File:
    - Open your project in Android Studio.
    - In the Project view, navigate to app > manifests > AndroidManifest.xml. This is where the file is located by default.
- Understanding the Structure: The AndroidManifest.xml file typically has the following structure

Screenshots:



Kotlin not configured

Gradle project sync failed. Basic functionality (e.g. editing, debugging) will not work properly.

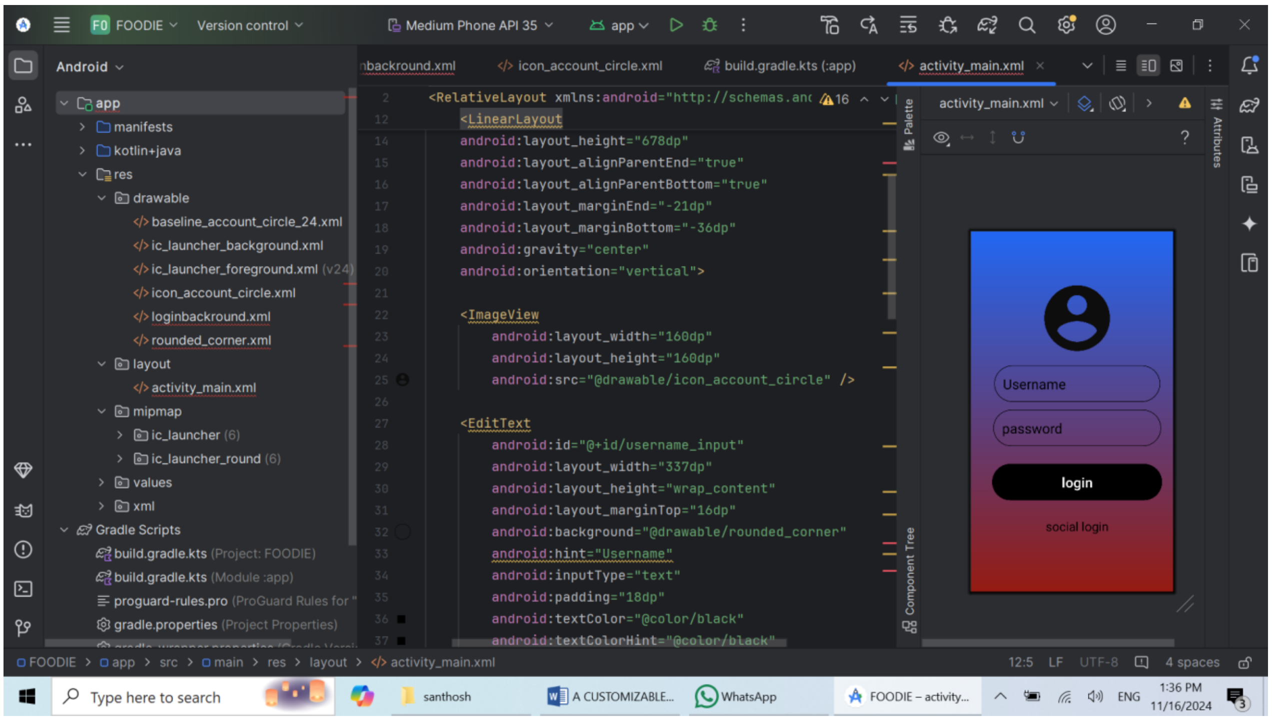
```kotlin
package com.example.snackordering

import ...


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( mainPage: this)
                    val context = LocalContext.current
                    //PopularFoodColumn(context)
                }
            }
        }
    }
}


@Composable
fun TopPart() {

    Row(
        modifier = Modifier
            .fillMaxWidth()
            .background(Color(0xffeceef0)), Arrangement.SpaceBetween
    ) {
```

RUNNING THE APPLICATIONS:

LOGIN ACTIVITY:

```
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.ContentScaleimport

 androidx.compose.ui.res.painterResource import

 androidx.compose.ui.text.font.FontFamilyimport

 androidx.compose.ui.text.font.FontWeightimport androidx.compose.ui.unit.dp

 import   androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

 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

                themeSurface(
                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.isNotE
    mpty()) {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.ja
                        va
                    )
            )
            //onLoginSuccess()
          }
            if (user != null && user.password ==
              "admin") {error = "Successfully log in"
              context.startActivity(
                Intent(
                  context,
                  AdminActivity::clas
                  s.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.ja
                    va
                  )
                )}
                )
                { 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)
```

Screenshort:



```
<RelativeLayout xmlns:android="http://schemas.and
    <LinearLayout
    android:layout_height="678dp"
    android:layout_alignParentEnd="true"
    android:layout_alignParentBottom="true"
    android:layout_marginEnd="-21dp"
    android:layout_marginBottom="-36dp"
    android:gravity="center"
    android:orientation="vertical">

    <ImageView
        android:layout_width="160dp"
        android:layout_height="160dp"
        android:src="@drawable/icon_account_circle" />

    <EditText
        android:id="@+id/username_input"
        android:layout_width="337dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:background="@drawable/rounded_corner"
        android:hint="Username"
        android:inputType="text"
        android:padding="18dp"
        android:textColor="@color/black"
        android:textColorHint="@color/black"
```

CONCLUSION:

In conclusion, the development of a customizable snack ordering and delivery app addresses the growing demand for convenience and personalization in the foodservice industry. By providing users with an intuitive interface that allows them to easily customize their snack orders, select from a wide range of options, and track their deliveries in real-time, this app enhances the overall user experience. The app's seamless integration of payment systems, diverse snack choices, and user-centric features such as order history, preferences, and loyalty rewards, positions it as a comprehensive solution for on-the-go snack lovers.

Moreover, with real-time tracking, notifications, and reliable delivery partnerships, the app ensures that customers receive their orders promptly and accurately. This not only improves customer satisfaction but also fosters brand loyalty. By embracing technological advancements and catering to evolving consumer needs, this snack ordering and delivery app has the potential to disrupt the snack industry and create a lasting impact on how consumers interactwith food delivery services. With ongoing updates and future scalability, this platform can adapt to a growing market and further enhance the snack consumption experience.