**A Flexible Email Client App**

|  |  |
| --- | --- |
| **Team Members** | **NM Id** |
| R.Chandhar | 01DA9CD11A2317A8CE7411956DCCB0A0 |
| E.Dhasaravi | 06AE072CC7147E180DD4732944F8CE56 |
| J.Madhan | E3011AE5BF0354EAE7145CEFA0688F93 |
| R.Vignesh | 7F7D045064E713383D04BF4169782CAC |

**Description:**

Adaptive Mail app is a sample project that demonstrates how to use the Android Compose UI toolkit to build a conversational UI. The app simulates a messaging interface, allowing the user to send and receive messages, and view a history of previous messages. It showcases some of the key features of the Compose UI toolkit, data management, and user interactions.

**Required Initial Steps:**

**Step 1.:Download Android Studio: - Visit the official Android Studio download page:**

* [https://developer.android.com/studio](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). o 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.

**Step3: First Launch and Setup:**

* 1. Choose UI Theme\*\*: Select between `Light` or `Dark` theme.
  2. 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:**

* 1. -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).
  2. 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:**

* 1. Go to `File` → `New` → `New Project`.
  2. Choose a template (e.g., `Empty Activity`).
  3. Configure the project (e.g., name, package name).
  4. Click `Finish`.

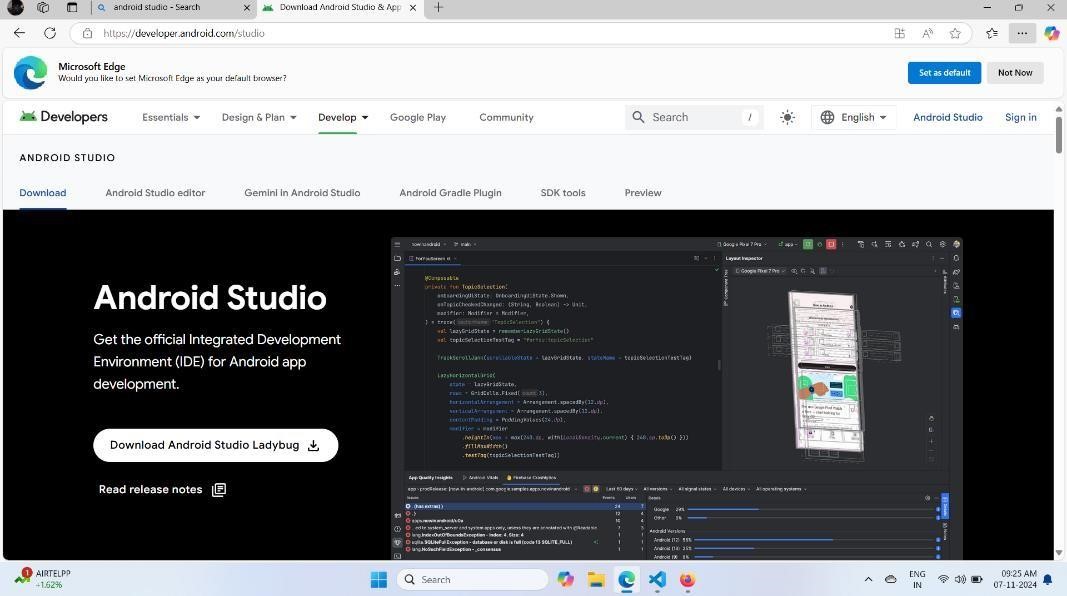
**Step6: Install Android Device/SDK Drivers (If needed):**

* 1. 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`).

* 1. You may need to install specific drivers for your device (especially on Windows).

**Screen Shorts:**



**Creating New Project:**

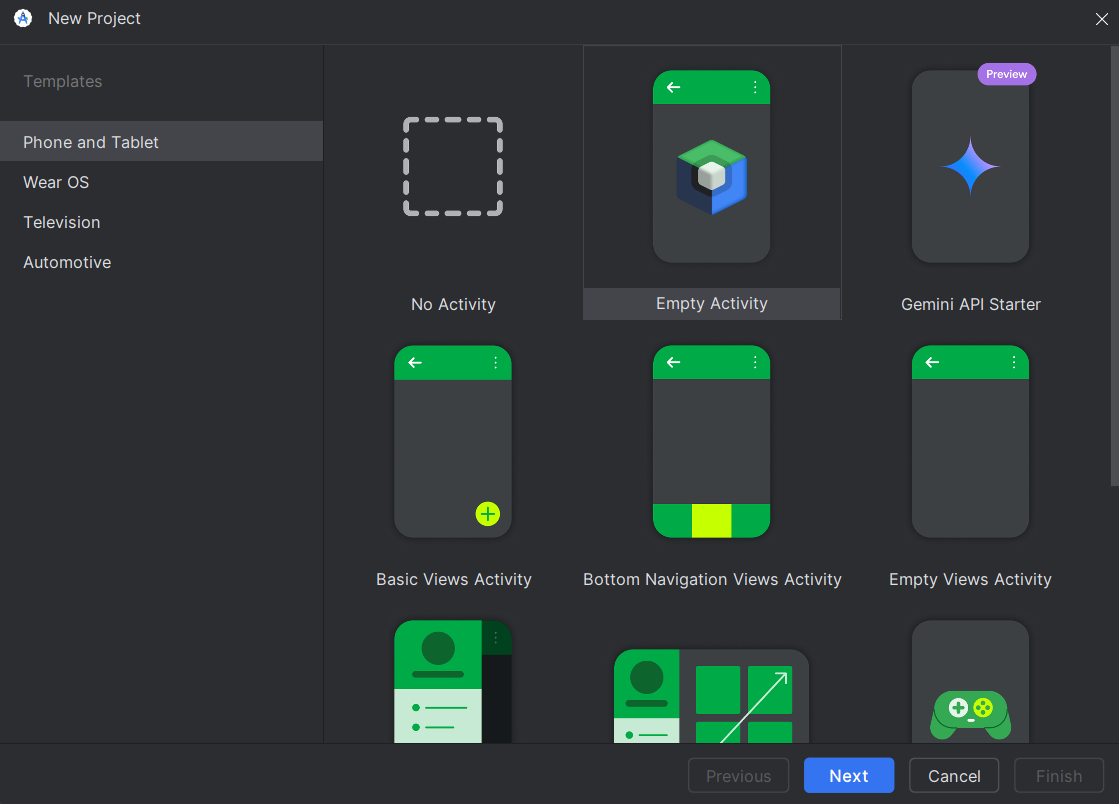
**Step1: Open Android Studio:**

Launch Android Studio on your computer.

**Step2:Start a New Project:**

On the welcome screen, click "Start a new Android Studio project".

**Screen Shots:**

****

**Step3:Configure Your Project:**

1. **Name:** Enter your app's name.
2. **Package Name:** Enter a unique identifier (e.g., `com.example.myapp`).
3. **Language:** Choose between Java or Kotlin.
4. **Minimum SDK:** Choose the lowest Android version your app will support.

**Step4:Choose a Template:**

Select a project template (e.g., “Empty Activity”for a simple starting point), then click “Next”.

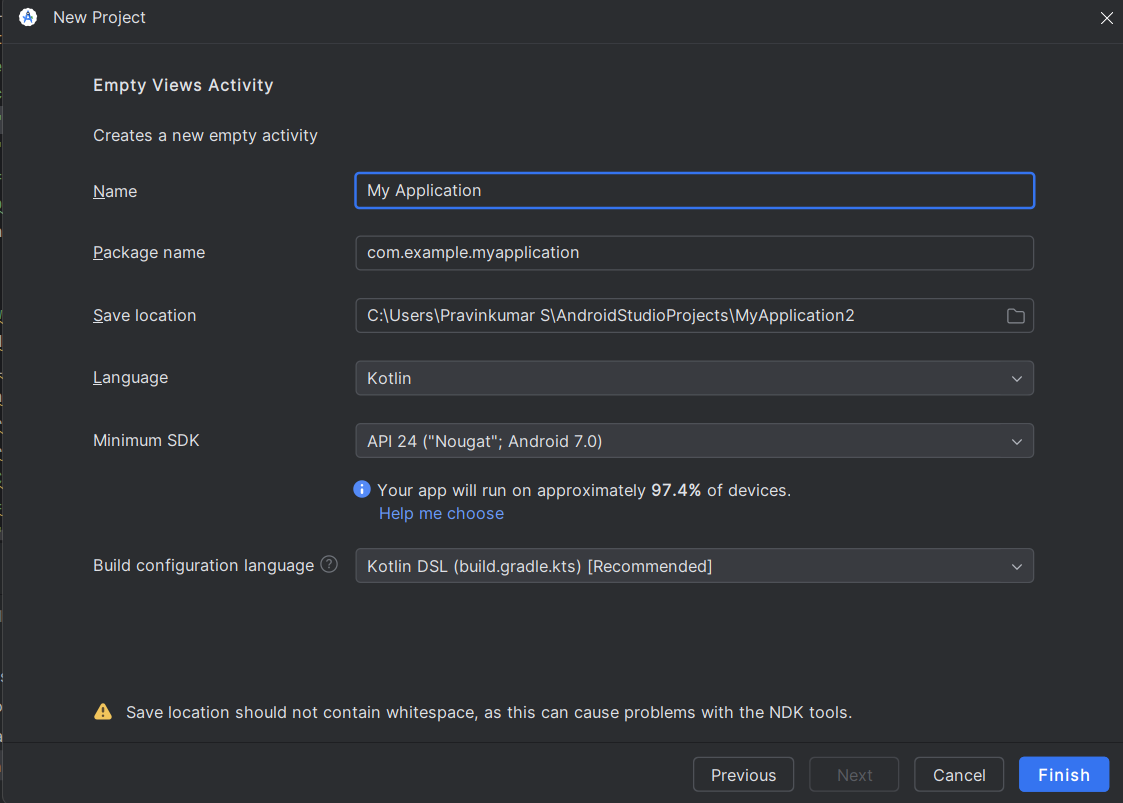
**Step5: Configure Activity:**

1. **Activity Name:** Default is `MainActivity` (you can change it).
2. **Layout Name:** Default is `activity\_main.xml`.
3. **-Click “Finish”.**

**Step6:Run the App:**

Once the project is set up, click the “Run”button to launch the app on an emulator or a physical device.

**Screen Shots:**

****

**Creating the database classes**

**Adding Required Dependencies:**

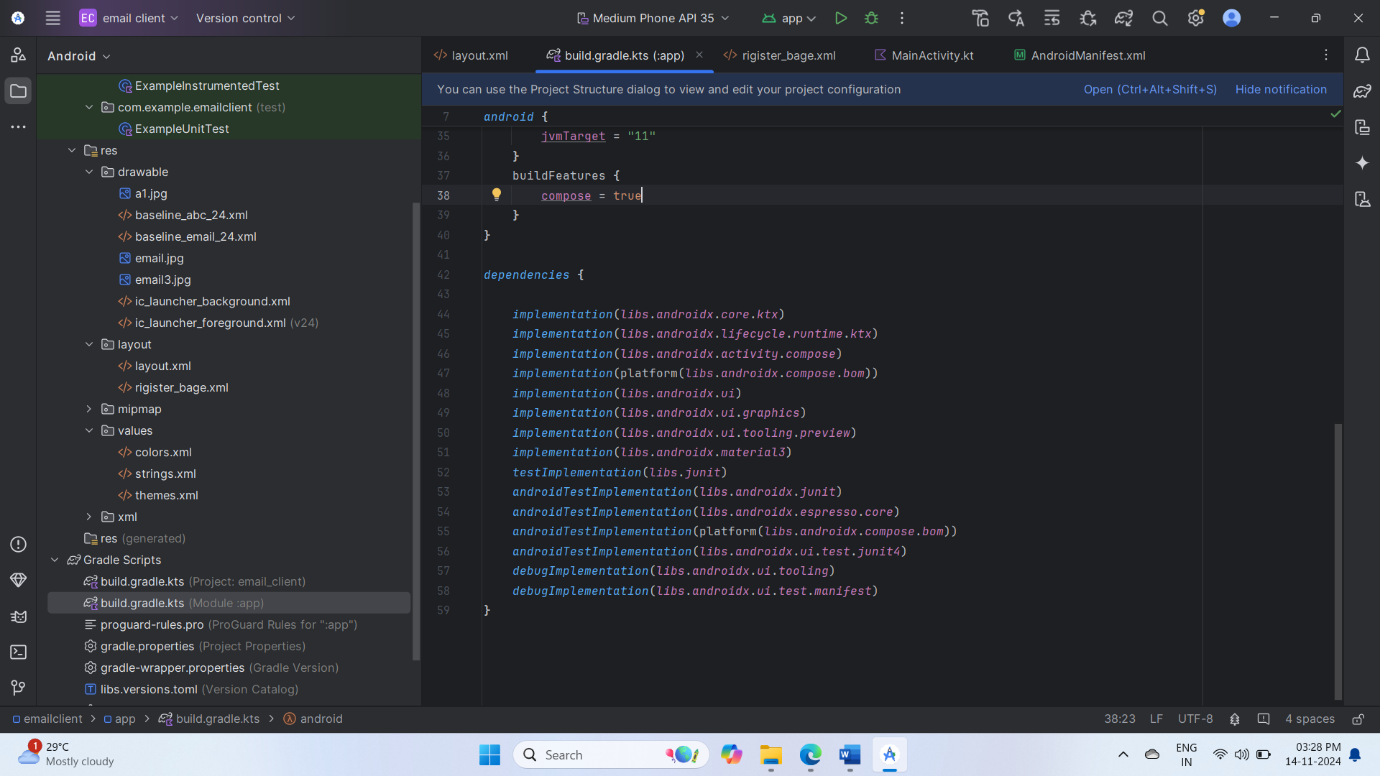
**Step 1: Open the `build.gradle (Project)` File:**

1. Open your “Android Studio” project.
2. In the “Project View”, navigate to the `build.gradle` file at the \*\*Project Level\*\* (usually the top `build.gradle` file).
3. Verify that you have the “Google”and “MavenCentral”repositories included in the `repositories` block. Ensure it looks like this:

**Step 2: Open the `build.gradle (Module: app)` File:**

1. Now, navigate to the `build.gradle` file at the “Module Level”(usually `app/build.gradle`).
2. Inside the `dependencies` block, you’ll add the Room dependencies.

**Screen Shots:**

****

**Step 3: Add Room Dependencies**

1. `room-runtime`: The Room persistence library.
2. `room-compiler`: A processor that generates code for your Room entities.
3. `room-ktx`: A Kotlin extension library for using Room with Kotlin.

Add the following dependencies in the `dependencies` block:

gradle

dependencies {

implementation "androidx.room:room-runtime:2.5.0" // Adjust version if needed

kapt "androidx.room:room-compiler:2.5.0" // For annotation processing, use kapt in Kotlin

implementation "androidx.room:room-ktx:2.5.0" // Optional for Kotlin

**}**

**Creating the database classes:**

**Step 1: Set Up Room Database in Android**

First, you'll need to add the Room dependencies to your `build.gradle` file:

**1. Add dependencies:**

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

`gradle

dependencies {

def room\_version = "2.5.0" // Use the latest version

implementation "androidx.room:room-runtime:$room\_version"

annotationProcessor "androidx.room:room-compiler:$room\_version" // For Java users

kapt "androidx.room:room-compiler:$room\_version" // For Kotlin users

implementation "androidx.room:room-ktx:$room\_version" // For Kotlin extensions

}

**Step 2: Create the Entity Classes**

Room uses “Entity” classes to define tables in the database. You will need to create two entities: one for user registration/login and one for the survey responses.

**1. User Entity (for registration and login):**

Create a `User` class that represents the user registration data.

java

@Entity(tableName = "users")

public class User {

@PrimaryKey(autoGenerate = true)

private int id;

@NonNull

@ColumnInfo(name = "username")

private String username;

@NonNull

@ColumnInfo(name = "email")

private String email;

@NonNull

@ColumnInfo(name = "password")

private String password; // Store hashed password

@ColumnInfo(name = "created\_at")

private long createdAt;

@ColumnInfo(name = "updated\_at")

private long updatedAt;

// Getters and setters

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getUsername() { return username; }

public void setUsername(String username) { this.username = username; }

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

public String getPassword() { return password; }

public void setPassword(String password) { this.password = password; }

public long getCreatedAt() { return createdAt; }

public void setCreatedAt(long createdAt) { this.createdAt = createdAt; }

public long getUpdatedAt() { return updatedAt; }

public void setUpdatedAt(long updatedAt) { this.updatedAt = updatedAt; }

}

```

**2. Survey Entity (for admin to view survey responses):**

Create a `Survey` class that will represent the survey details submitted by users.

```java

@Entity(tableName = "surveys")

public class Survey {

@PrimaryKey(autoGenerate = true)

private int id;

@ForeignKey(entity = User.class,

parentColumns = "id",

childColumns = "user\_id",

onDelete = ForeignKey.CASCADE)

private int userId;

@NonNull

@ColumnInfo(name = "survey\_data")

private String surveyData; // JSON or plain text survey responses

@ColumnInfo(name = "created\_at")

private long createdAt;

@ColumnInfo(name = "updated\_at")

private long updatedAt;

// Getters and setters

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public int getUserId() { return userId; }

public void setUserId(int userId) { this.userId = userId; }

public String getSurveyData() { return surveyData; }

public void setSurveyData(String surveyData) { this.surveyData = surveyData; }

public long getCreatedAt() { return createdAt; }

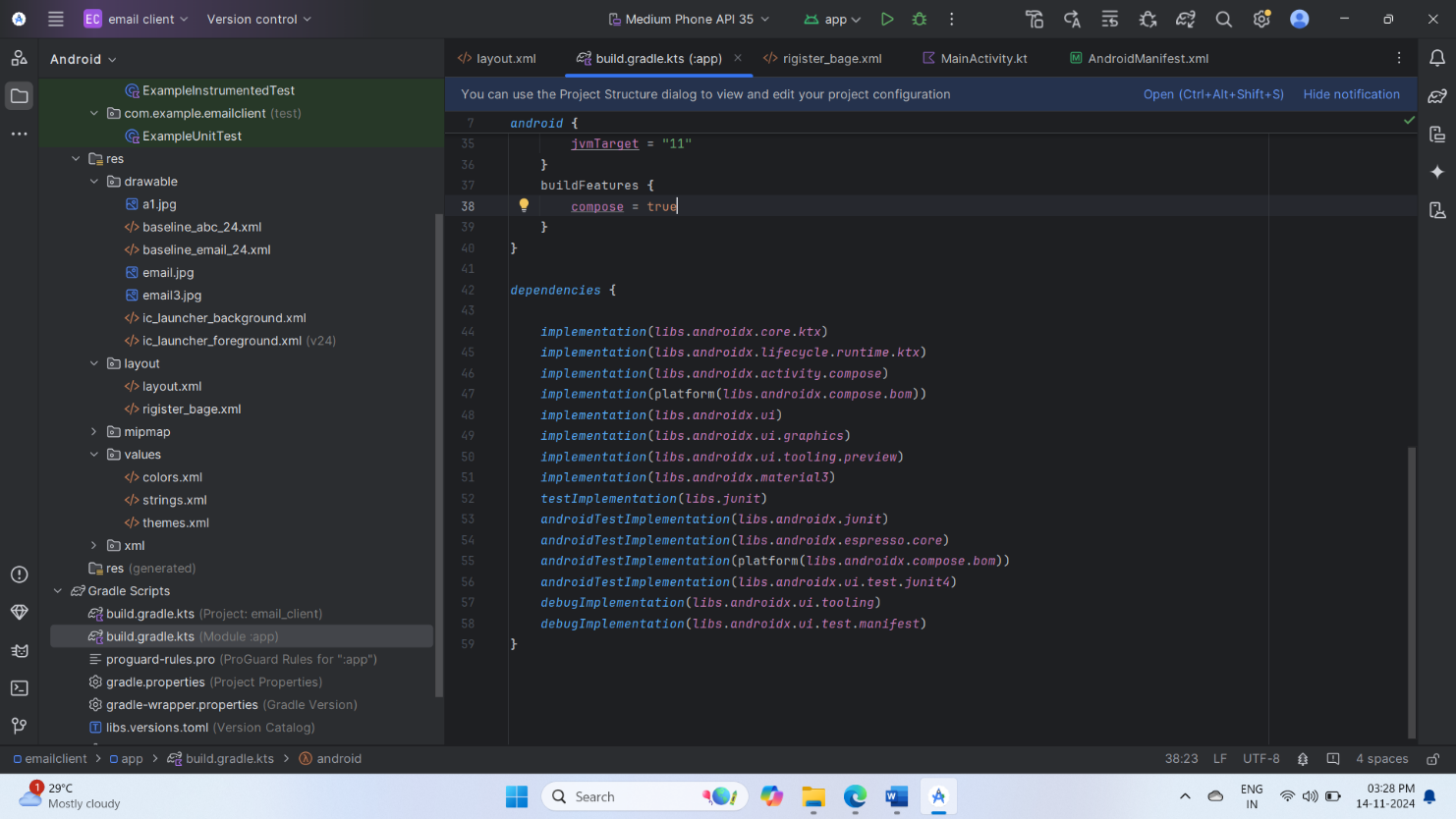
public void setCreatedAt(long createdAt) { this.createdAt = createdAt; }

public long getUpdatedAt() { return updatedAt; }

public void setUpdatedAt(long updatedAt) { this.updatedAt = updatedAt; }

}

**Screen Shots:**



**1. User DAO:**

The `UserDao` interface will include methods for inserting, updating, and querying user data.

```java

@Dao

public interface UserDao {

@Insert

long insertUser(User user);

@Update

int updateUser(User user);

@Query("SELECT \* FROM users WHERE username = :username AND password = :password LIMIT 1")

User getUserByUsernameAndPassword(String username, String password);

@Query("SELECT \* FROM users WHERE email = :email LIMIT 1")

User getUserByEmail(String email);

@Query("SELECT \* FROM users WHERE id = :userId LIMIT 1")

User getUserById(int userId);

}

**Building application UI and connecting to database:**

**Step 1: Setting Up the Project**:

1. Open Android Studio: and create a new project.

- Select Empty ‘Activity’for simplicity.

- Set up the name, package, and other details for your app.

2. Once the project is set up, you’ll have a basic structure with an `activity\_main.xml` layout and a `MainActivity.java` or `MainActivity.kt` file.

**Step 2: Designing the UI:**

Example `activity\_main.xml`:

xml

<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<EditText

android:id="@+id/editTextData"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:hint="Enter Data"

android:inputType="text"/>

<Button

android:id="@+id/buttonSave"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Save"

android:layout\_below="@id/editTextData"

android:layout\_marginTop="20dp"/>

<ListView

android:id="@+id/listViewData"

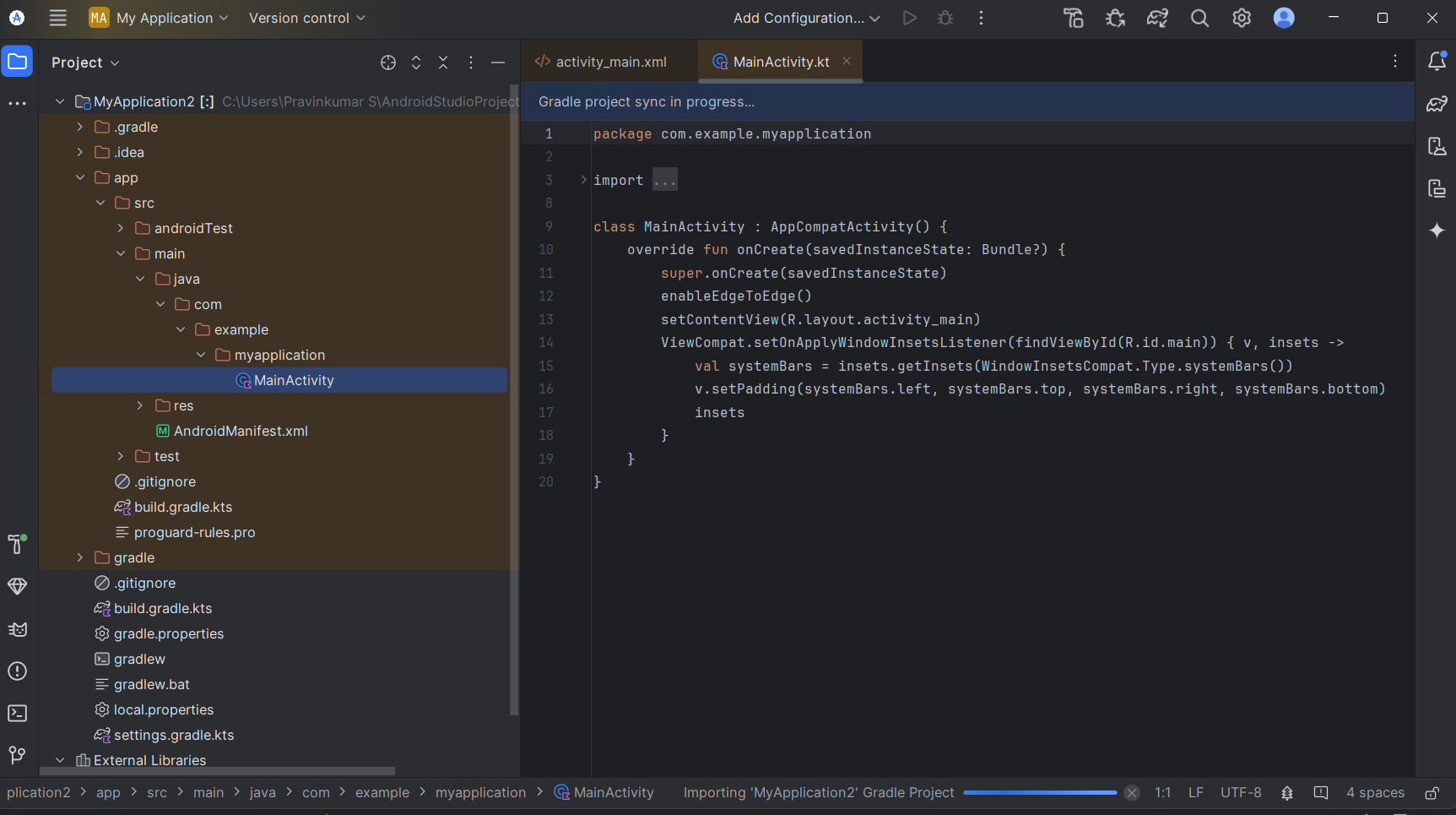
android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_below="@id/buttonSave"

android:layout\_marginTop="20dp"/>

</RelativeLayout>



Step 3: Setting Up SQLite Database

Now, we’ll set up an SQLite database to store and retrieve data.

1. \*\*Create a Database Helper Class\*\*

This class will manage database creation, connection, and CRUD operations (Create, Read, Update, Delete).

#### Example `DBHelper.java`:

```java

package com.example.myapp;

import android.content.Context;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

public class DBHelper extends SQLiteOpenHelper {

private static final String DATABASE\_NAME = "myDatabase.db";

private static final int DATABASE\_VERSION = 1;

public static final String TABLE\_NAME = "data\_table";

public static final String COLUMN\_ID = "\_id";

public static final String COLUMN\_DATA = "data";

// SQL to create the database table

private static final String TABLE\_CREATE =

"CREATE TABLE " + TABLE\_NAME + " (" +

COLUMN\_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +

COLUMN\_DATA + " TEXT NOT NULL);";

public DBHelper(Context context) {

super(context, DATABASE\_NAME, null, DATABASE\_VERSION);

}

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL(TABLE\_CREATE);

}

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

db.execSQL("DROP TABLE IF EXISTS " + TABLE\_NAME);

onCreate(db);

}

}

```

**2. Create the Database Manager Class:**

```java

package com.example.myapp;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import java.util.ArrayList;

public class DatabaseManager {

private SQLiteDatabase database;

private DBHelper dbHelper;

public DatabaseManager(Context context) {

dbHelper = new DBHelper(context);

}

public void open() throws SQLException {

database = dbHelper.getWritableDatabase();

}

public void close() {

dbHelper.close();

}

// Insert data into the table

public void insertData(String data) {

ContentValues values = new ContentValues();

values.put(DBHelper.COLUMN\_DATA, data);

database.insert(DBHelper.TABLE\_NAME, null, values);

}

// Fetch all data from the table

public ArrayList<String> getAllData() {

ArrayList<String> dataList = new ArrayList<>();

Cursor cursor = database.query(DBHelper.TABLE\_NAME,

new String[]{DBHelper.COLUMN\_ID, DBHelper.COLUMN\_DATA},

null, null, null, null, null);

if (cursor != null) {

cursor.moveToFirst();

while (!cursor.isAfterLast()) {

String data = cursor.getString(cursor.getColumnIndex(DBHelper.COLUMN\_DATA));

dataList.add(data);

cursor.moveToNext();

}

cursor.close();

}

return dataList;

}

}

**Step 5: Running the Application**

- Build and run your app on an emulator or device.

- You should be able to enter data into the `EditText`, click the "Save" button to store it in the SQLite database, and see the data displayed in the `ListView`.

**Modifying AndroidManifest.xml**

**Step 1: Locate `AndroidManifest.xml:**

1. In the \*\*Project\*\* window (on the left side), expand the `app` directory.

2. Navigate to `app > src > main > AndroidManifest.xml`.

**Step 2: Modify the Manifest:**

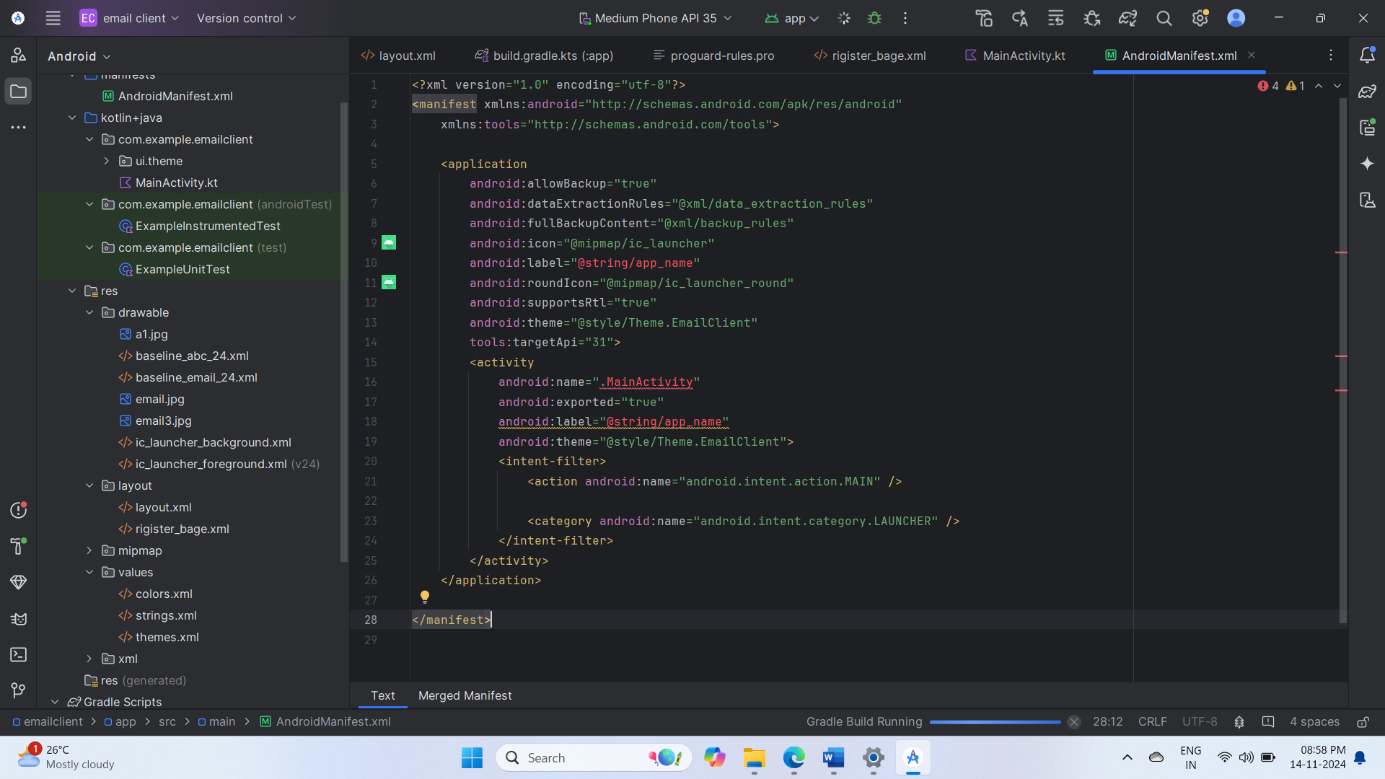
1. Open `AndroidManifest.xml`. It will contain essential information about your application, including components like activities, services, permissions, and more.

2. Make any necessary changes. For example, you can modify:

- Application-level settings (e.g., `android:icon`, `android:label`, etc.)

- Permissions\*(e.g., `android.permission.INTERNET`)

-Activity declarations (e.g., adding `android:theme`, `android:screenOrientation`, etc.)



**Step 4: Save Changes**

After making the necessary changes, simply press \*\*Ctrl+S\*\* (Windows/Linux) or \*\*Cmd+S\*\* (Mac) to save the file.

**Step 5: Sync Project**

Once you've saved the file, sync your project to ensure all changes are applied. You can do this by clicking the \*\*Sync Now\*\* button in the notification bar that appears or by selecting File > Sync Project with Gradle Files

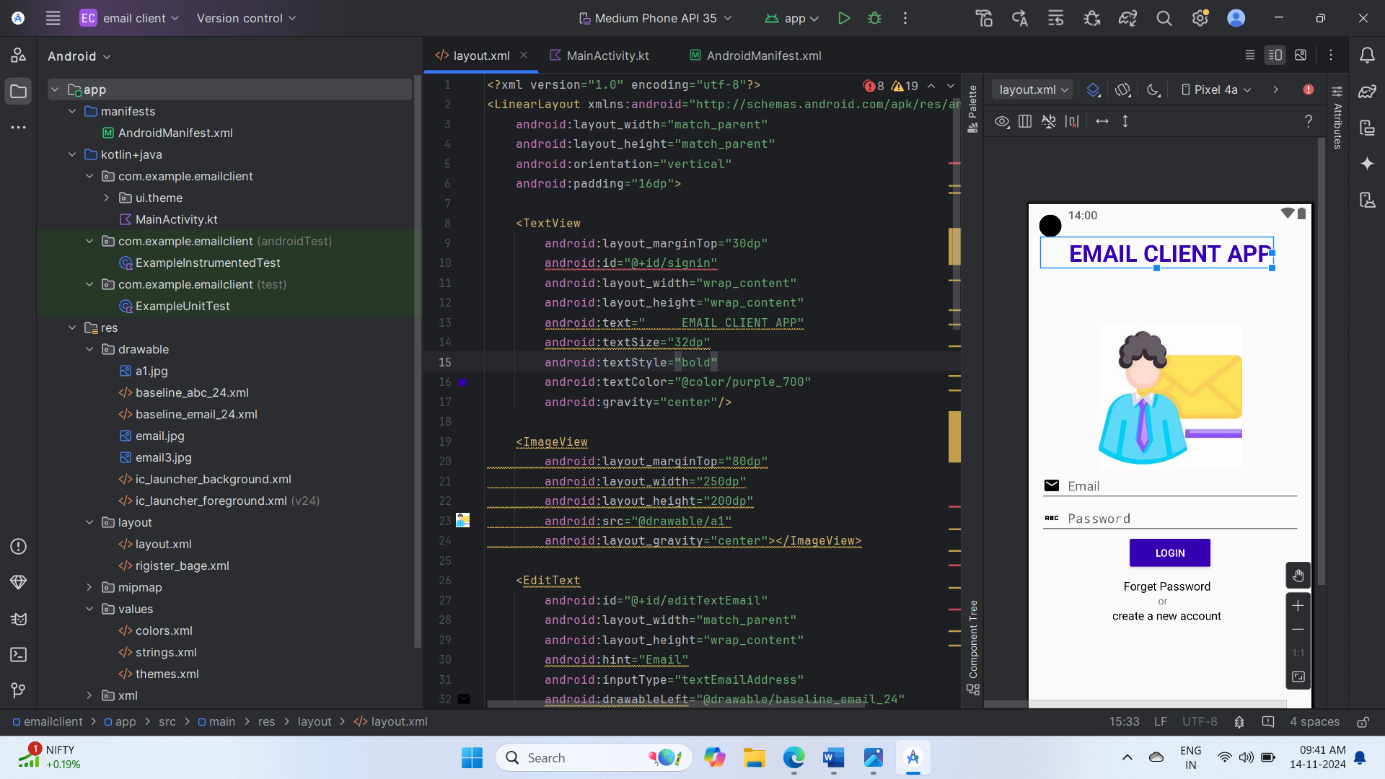
**Run The Application in Mobile:**

**#Step 1: Set up the Android Project in Android Studio\*\***

* + - * Open Android Studio\*\* and create a new project or open an existing email client app project.
      * Select the project template based on your needs (e.g., "Empty Activity" if you're building from scratch).
      * Make sure to configure the `minSdkVersion` and `targetSdkVersion` in the `build.gradle` file based on your target device's Android version.
      * Add required permissions in the `AndroidManifest.xml` file, such as `INTERNET` permission for network access.

**Step 2: Connect Your Android Device:**

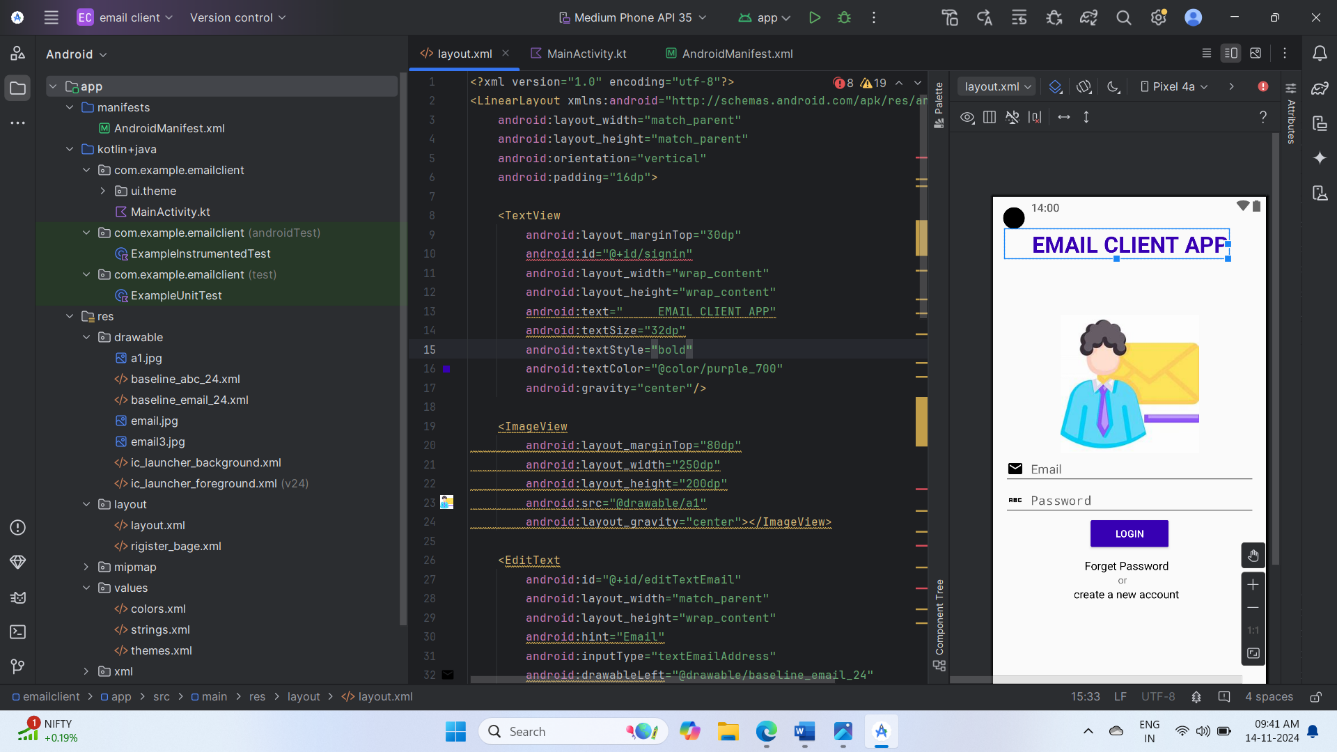
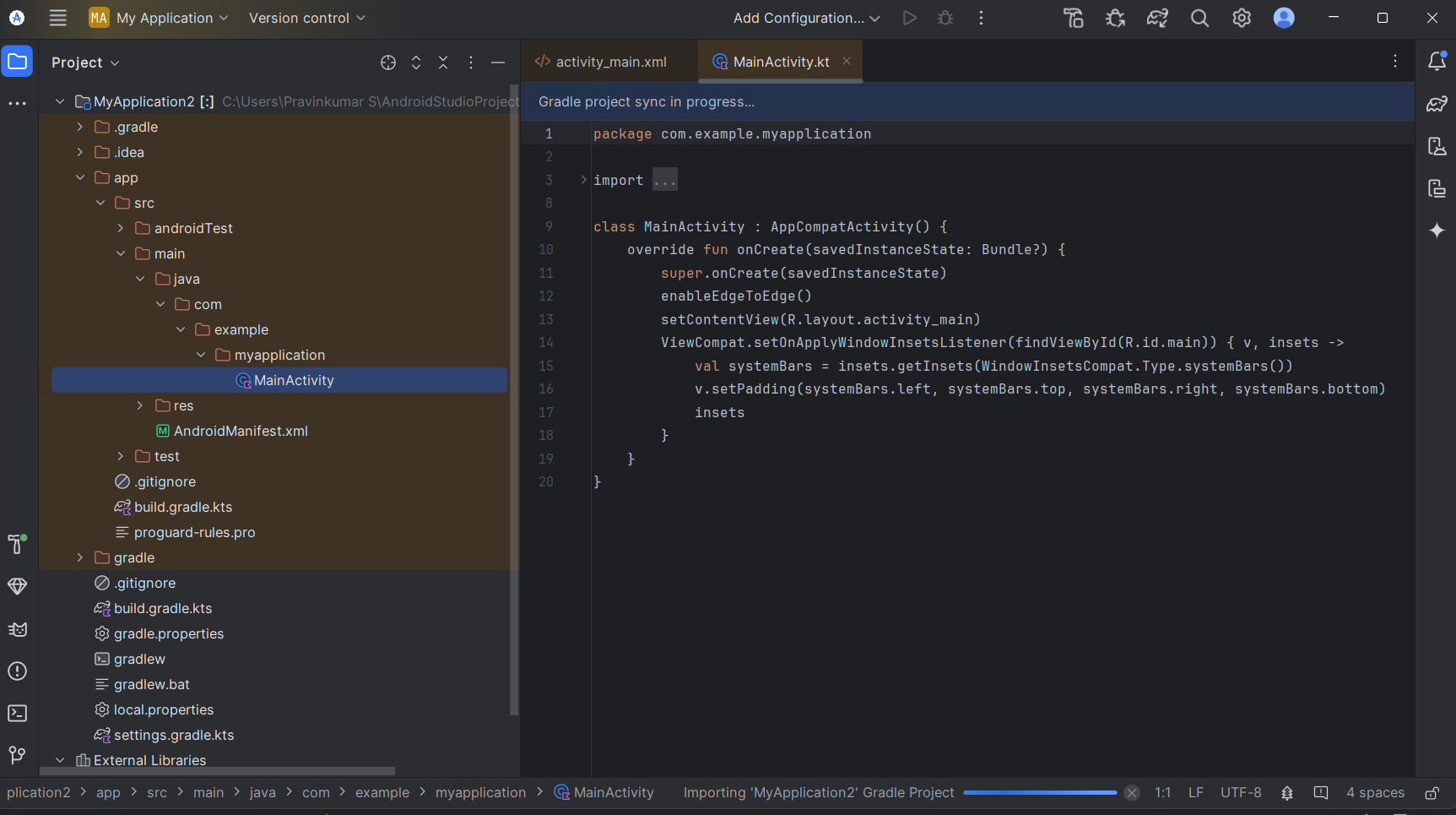
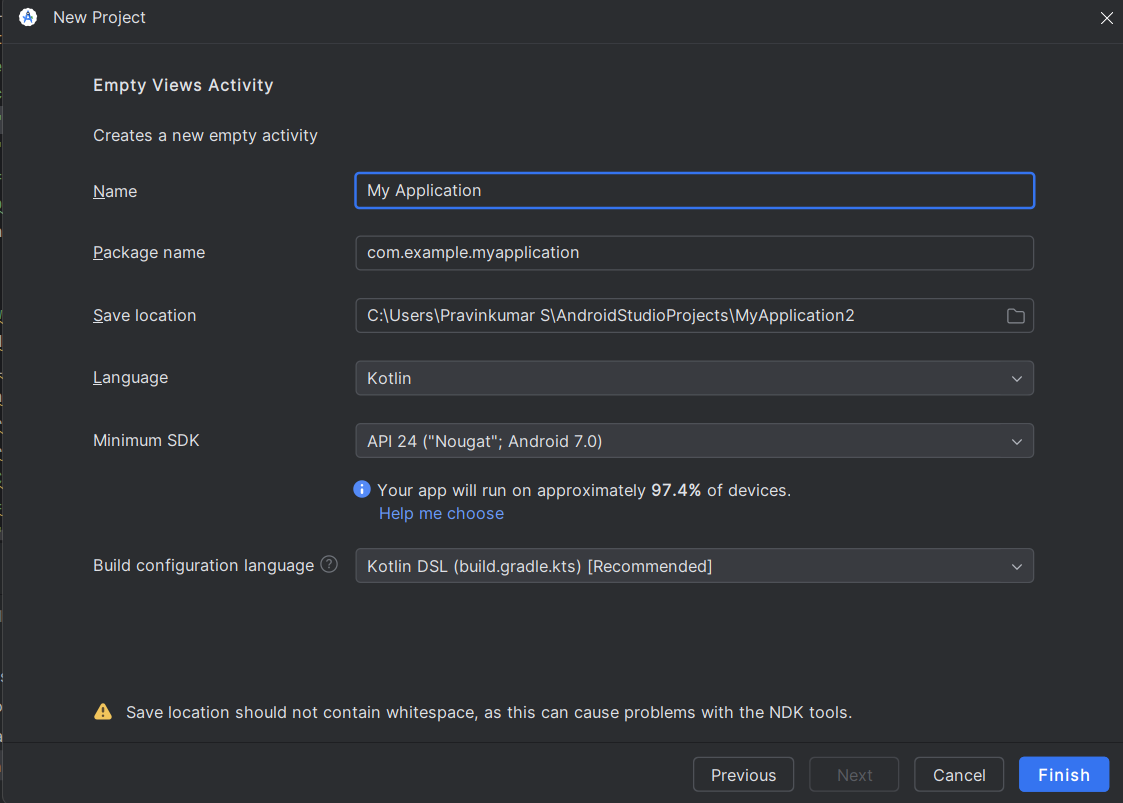
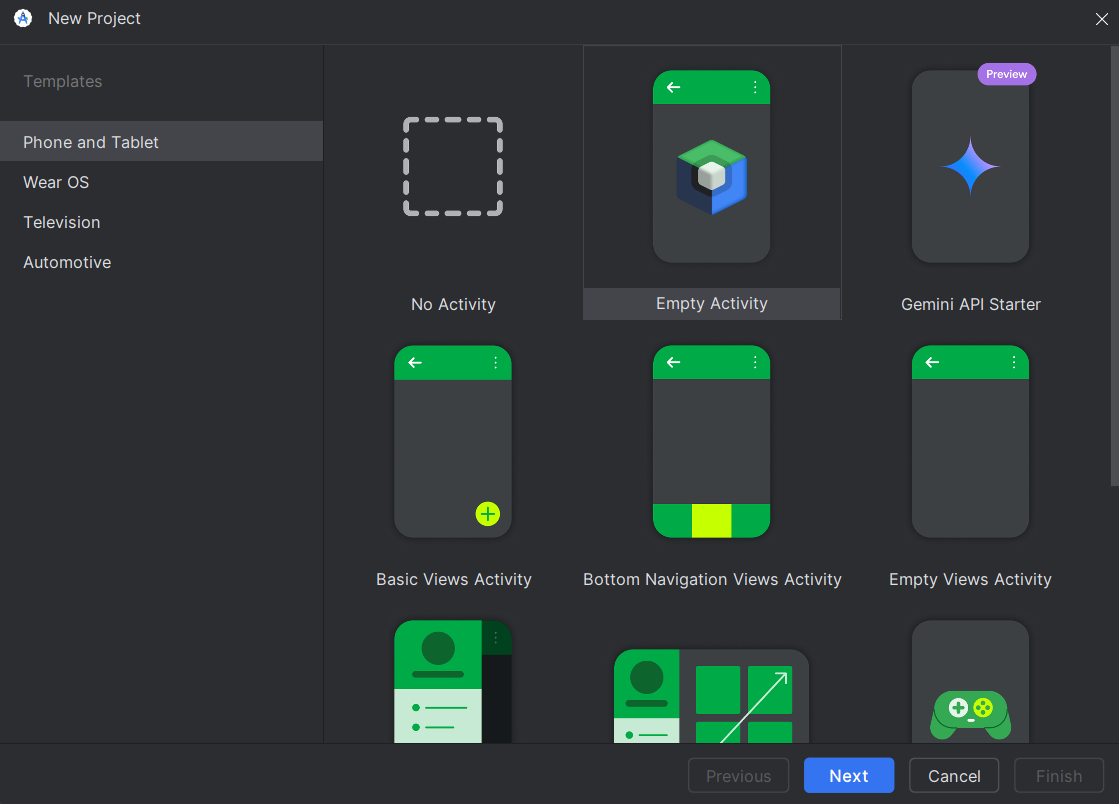
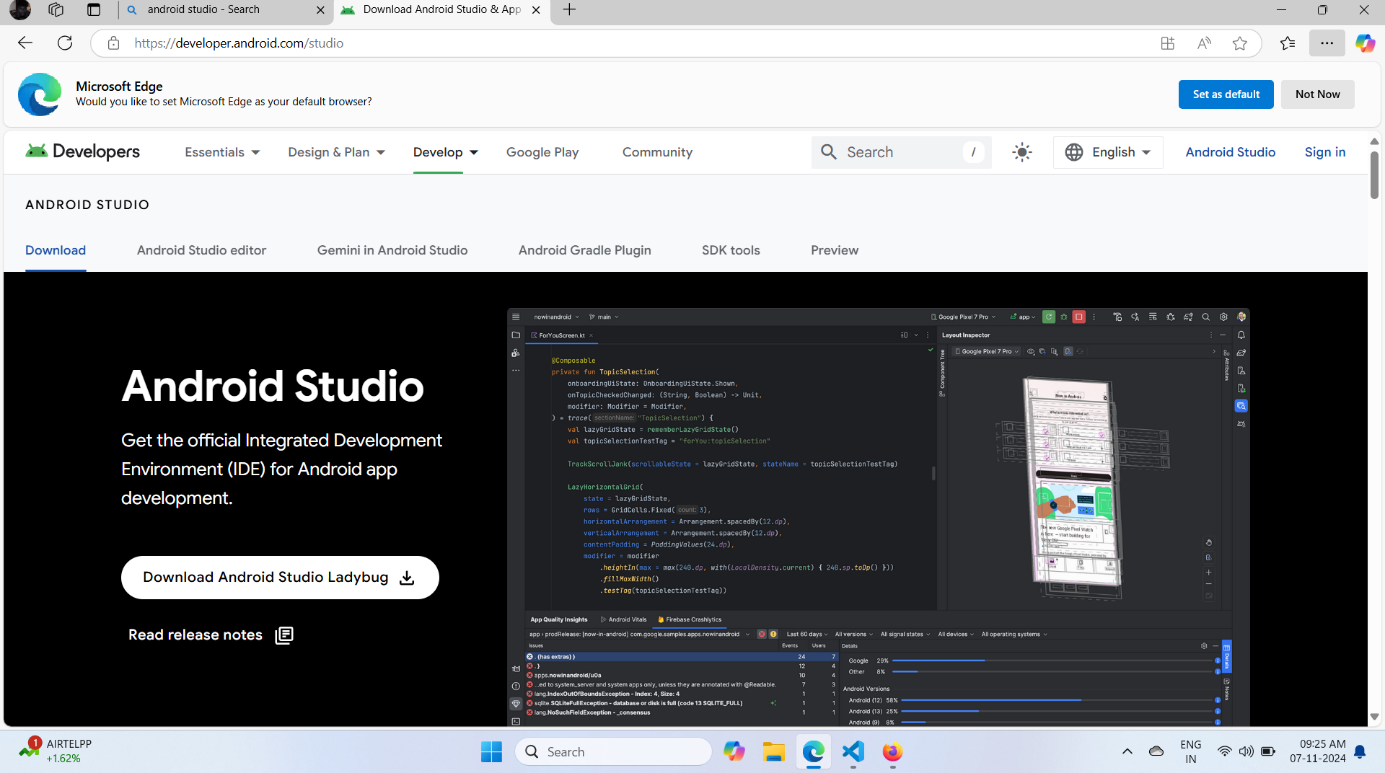
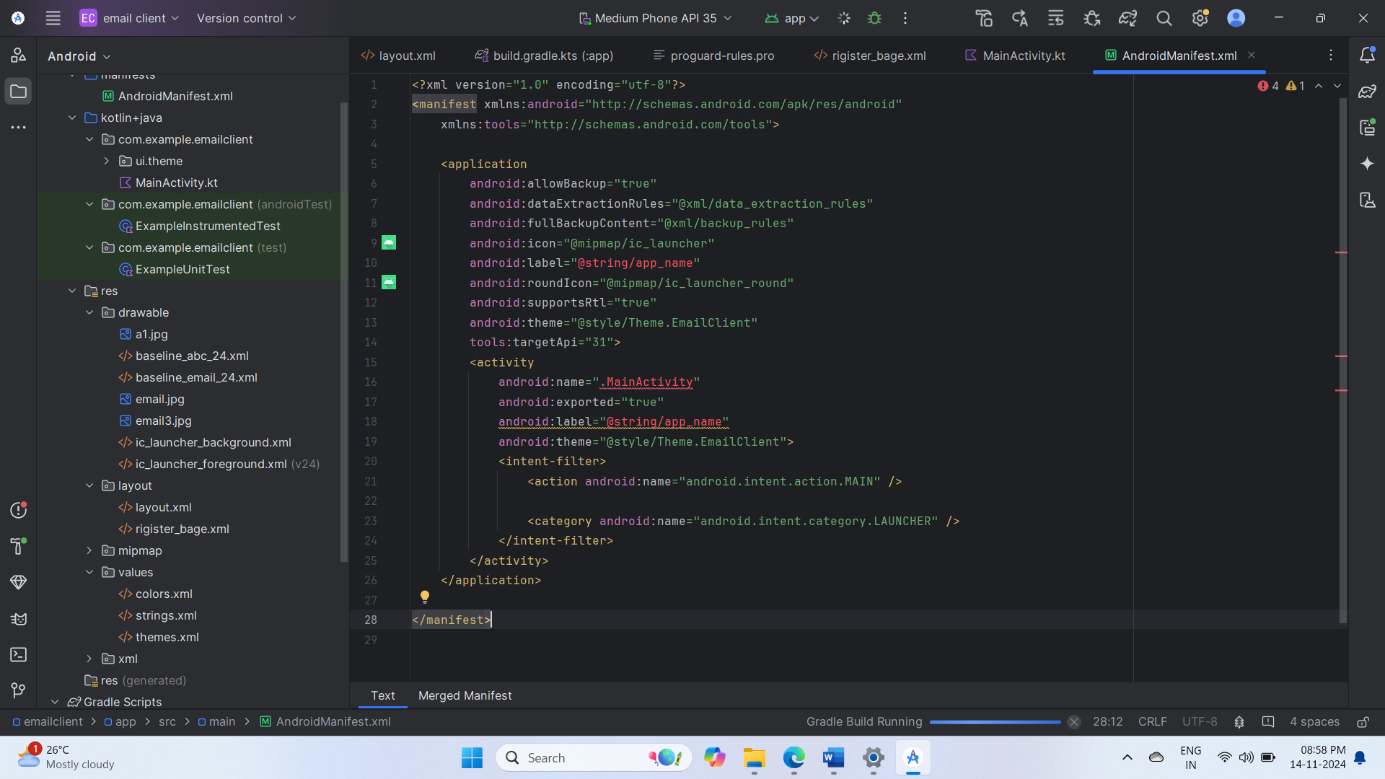
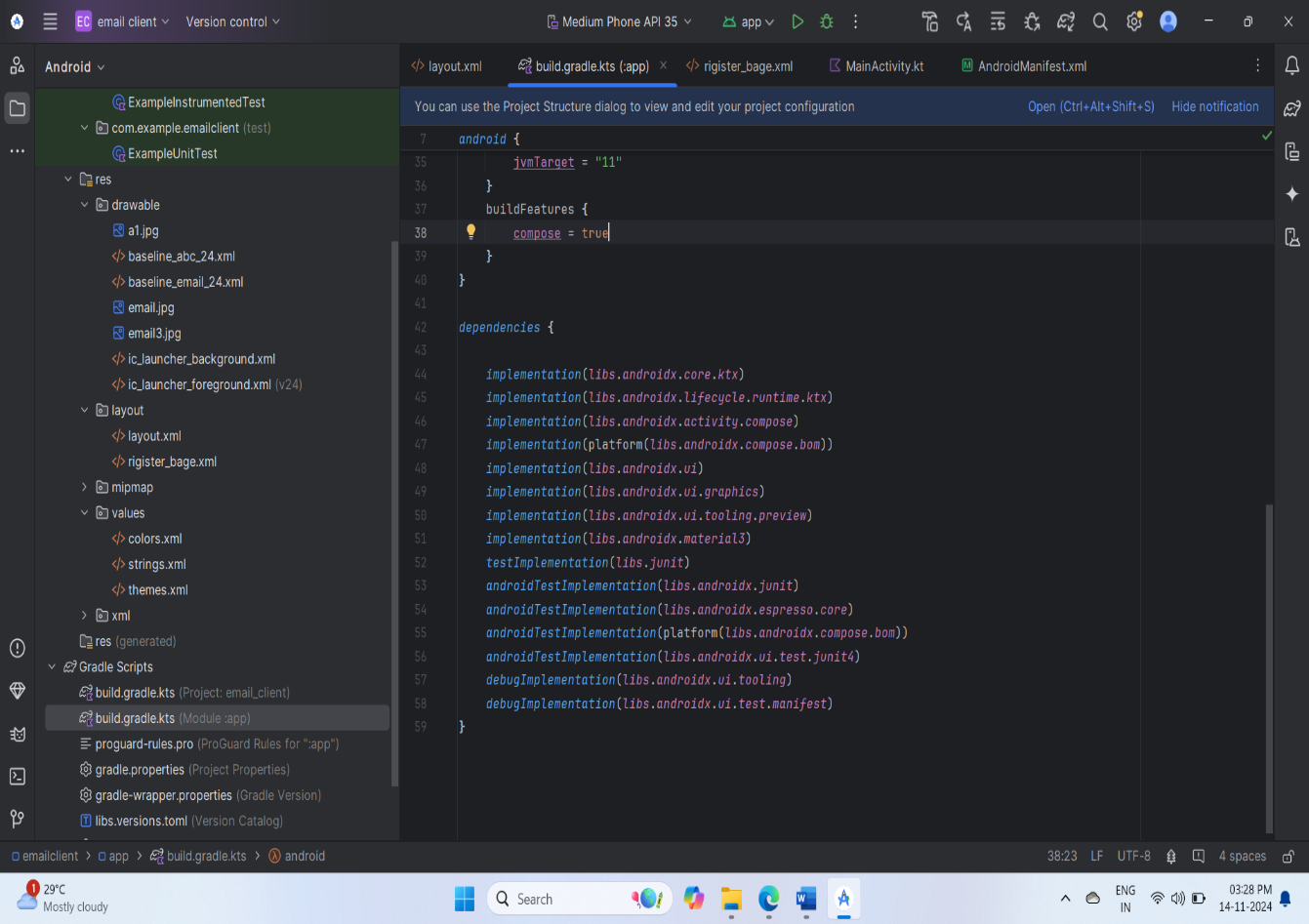
* + - * Enable Developer Options on your Android phone by going to `Settings > About phone > Tap 'Build number' 7 times`.
      * In `Developer Options`, enable `USB debugging`.
      * Connect your Android device to your computer using a USB cable.
      * In Android Studio, click on the device dropdown (next to the "Run" button) and select your connected device.
      * If your device doesn’t appear, make sure you have the proper drivers installed for your device.
      * You can also use an emulator if you don't have a physical device.



**Step 3: Run the App on Your Device:**

* + - * Click on the \*\*Run\*\* button (the green triangle) in Android Studio or press `Shift + F10`.
      * Android Studio will compile the app and install it on your connected Android device.
      * After the app is installed, the email client should launch on your device.

**Screen Shots:**

**g**

**Main activity.kt:**

package com.example.emailapplication

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.background

import androidx.compose.foundation.layout.\*

import androidx.compose.material.\*

import androidx.compose.runtime.Composable

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.FontWeight

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

import androidx.core.content.ContextCompat.startActivity

import com.example.emailapplication.ui.theme.EmailApplicationTheme

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

// A surface container using the 'background' color from the theme

Surface(

modifier = Modifier.fillMaxSize().background(Color.White),

) {

Email(this)

}

}

}

}

@Composable

fun Email(context: Context) {

Text(

text = "Home Screen",

modifier = Modifier.padding(top = 74.dp, start = 100.dp, bottom = 24.dp),

color = Color.Black,

fontWeight = FontWeight.Bold,

fontSize = 32.sp

)

Column(

horizontalAlignment = Alignment.CenterHorizontally,

verticalArrangement = Arrangement.Center

) {

Image(

painterResource(id = R.drawable.home\_screen), contentDescription = ""

)

Button(onClick = {

context.startActivity(

Intent(

context,

SendMailActivity::class.java

)

)

},

colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFadbef4))

) {

Text(

text = "Send Email",

modifier = Modifier.padding(10.dp),

color = Color.Black,

fontSize = 15.sp

)

}

Spacer(modifier = Modifier.height(20.dp))

Button(onClick = {

context.startActivity(

Intent(

context,

ViewMailActivity::class.java

)

)

},

colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFadbef4))

) {

Text(

text = "View Emails",

modifier = Modifier.padding(10.dp),

color = Color.Black,

fontSize = 15.sp

)

}

}

}

**Main Activity.Xml:**

package com.example.emailapplication

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.background

import androidx.compose.foundation.layout.\*

import androidx.compose.material.\*

import androidx.compose.runtime.Composable

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.FontWeight

import androidx.compose.ui.tooling.preview.Preview

import androidx.compose.ui.unit.dp

import androidx.compose.ui.unit.sp

import androidx.core.content.ContextCompat

import androidx.core.content.ContextCompat.startActivity

import com.example.emailapplication.ui.theme.EmailApplicationTheme

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

// A surface container using the 'background' color from the theme

Surface(

modifier = Modifier.fillMaxSize().background(Color.White),

) {

Email(this)

}

}

}

}

@Composable

fun Email(context: Context) {

Text(

text = "Home Screen",

modifier = Modifier.padding(top = 74.dp, start = 100.dp, bottom = 24.dp),

color = Color.Black,

fontWeight = FontWeight.Bold,

fontSize = 32.sp

)

Column(

horizontalAlignment = Alignment.CenterHorizontally,

verticalArrangement = Arrangement.Center

) {

Image(

painterResource(id = R.drawable.home\_screen), contentDescription = ""

)

Button(onClick = {

context.startActivity(

Intent(

context,

SendMailActivity::class.java

)

)

},

colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFadbef4))

) {

Text(

text = "Send Email",

modifier = Modifier.padding(10.dp),

color = Color.Black,

fontSize = 15.sp

)

}

Spacer(modifier = Modifier.height(20.dp))

Button(onClick = {

context.startActivity(

Intent(

context,

ViewMailActivity::class.java

)

)

},

colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFadbef4))

) {

Text(

text = "View Emails",

modifier = Modifier.padding(10.dp),

color = Color.Black,

fontSize = 15.sp

)

}

}

}

**Conclusion:**

**Creating an Email Client App in Android Studio**

Building an email client app in Android Studio involves several key steps that integrate UI design, networking, and working with email protocols (SMTP, IMAP, or POP3). By following the outlined process, you can create a fully functional email client that allows users to send and receive emails on their Android devices.