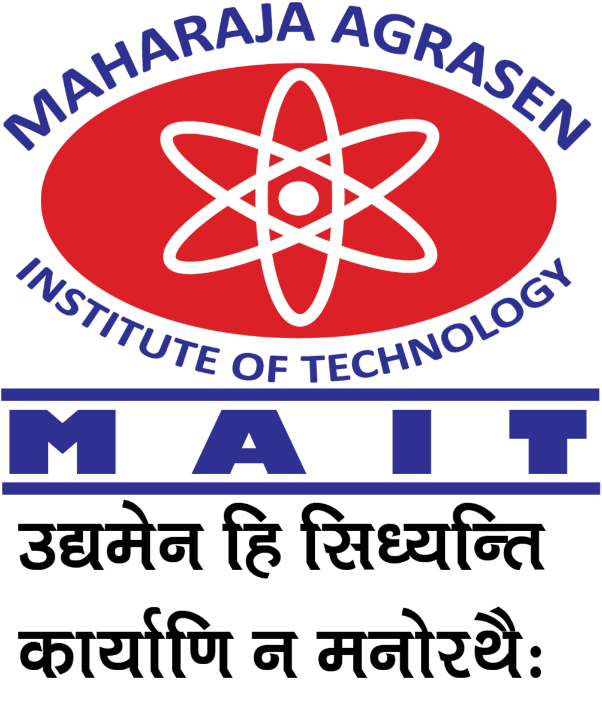
MOBILE ARCHITECTURE & PROGRAMMING LAB

ETCT 402

Faculty Name: **Mr. Ashish Sharma** Student Name: **Aaditya Mishra**

Enrolment No: **00114812720**

Semester : **8th Semester**



**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY, PSP AREA, SECTOR – 22, ROHINI, NEW DELHI – 110086**



**INDEX**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S.NO | DATE | NAME OF EXPERIEMENT | RUBRICS | | | | | | | SIGN |
| R1 | R2 | | R3 | R4 | R5 | |
| 1. |  | Setting up the Development Environment  ● Install Android Studio and set up the Android development environment.  ● Create a basic "Hello World" app for Android and platforms. |  |  |  | |  | |  |  |
| 2. |  | User Interface Design and Development  ● Design a simple user interface using XML for an Android app. |  |  |  | |  | |  |  |
| 3. |  | Database Integration in Mobile Apps  ● Create a SQLite database and perform basic CRUD operations in an Android app. |  |  |  | |  | |  |  |
| 4. |  | Web Services and API Consumption  ● Retrieve data from a RESTful API and display it in an Android app. |  |  |  | |  | |  |  |
| 5. |  | Location-Based Services and Mapping  ● Implement location tracking and display user location on a map in an Android app using Google Maps API. |  |  |  | |  | |  |  |
| 6. |  | Cross-Platform App Development with React Native or Flutter  ● Develop a simple mobile app using React Native, utilizing components and navigation. |  |  |  | |  | |  |  |
| 7. |  | Testing and Debugging  ● Perform unit testing on key functionalities of an Android app using JUnit and Android Testing frameworks. |  |  |  | |  | |  |  |
| 8. |  | App Deployment and Performance Optimization  ● Package and deploy an Android app to Google Play Store. |  |  |  | |  | |  |  |

**EXPERIMENT -1**

**AIM:**- Setting up the Development Environment

● Install Android Studio and set up the Android development environment.

● Create a basic "Hello World" app for Android platforms.

**PROCEDURE:-**

1) IP scanner- whois:

Whois is a widely used Internet record listing that identifies who owns a domain and how to

get in contact with them. Here for example we have searched for domain name

https://www.charusat.ac.in/

It displays domain information like the register date, update date, registrar provider etc..

1) IP scanner- whois:

Whois is a widely used Internet record listing that identifies who owns a domain and how to

get in contact with them. Here for example we have searched for domain name

https://www.charusat.ac.in/

It displays domain information like the register date, update date, registrar provider etc..

1) IP scanner- whois:

Whois is a widely used Internet record listing that identifies who owns a domain and how to

get in contact with them. Here for example we have searched for domain name

https://www.charusat.ac.in/

It displays domain information like the register date, update date, registrar provider etc..

1) IP scanner- whois:

Whois is a widely used Internet record listing that identifies who owns a domain and how to

get in contact with them. Here for example we have searched for domain name

https://www.charusat.ac.in/

It displays domain information like the register date, update date, registrar provider etc..

1) IP scanner- whois:

Whois is a widely used Internet record listing that identifies who owns a domain and how to

get in contact with them. Here for example we have searched for domain name

https://www.charusat.ac.in/

It displays domain information like the register date, update date, registrar provider etc..

1) IP scanner- whois:

Whois is a widely used Internet record listing that identifies who owns a domain and how to

get in contact with them. Here for example we have searched for domain name

https://www.charusat.ac.in/

It displays domain information like the register date, update date, registrar provider etc..

1) IP scanner- whois:

Whois is a widely used Internet record listing that identifies who owns a domain and how to

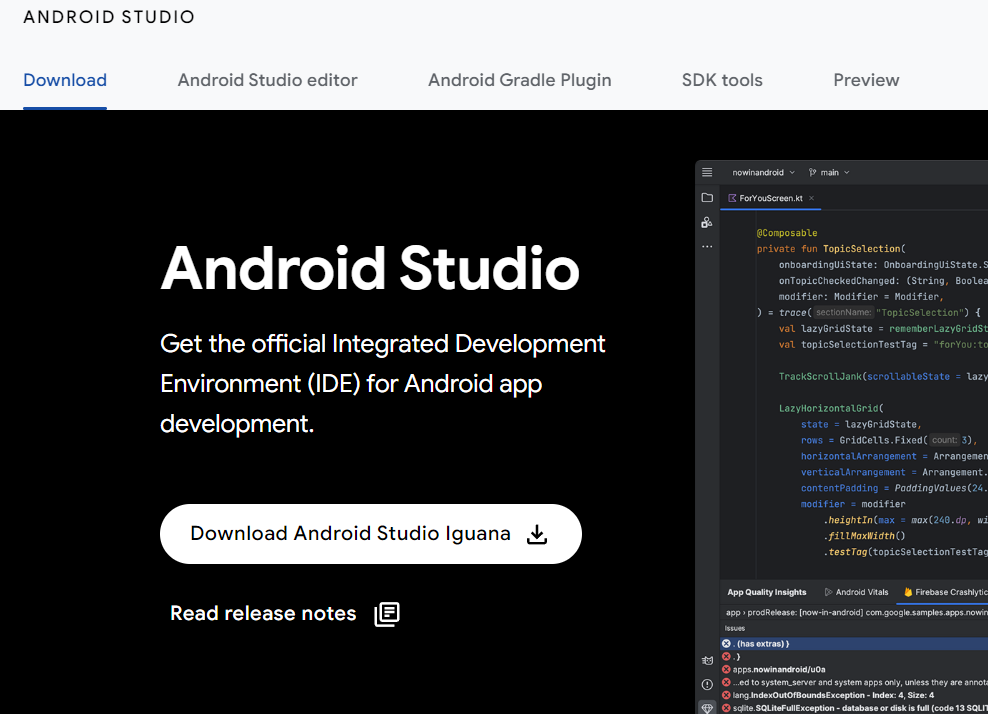
get in contact with them. Here for example we have searched for domain name

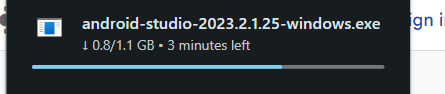
https://www.charusat.ac.in/

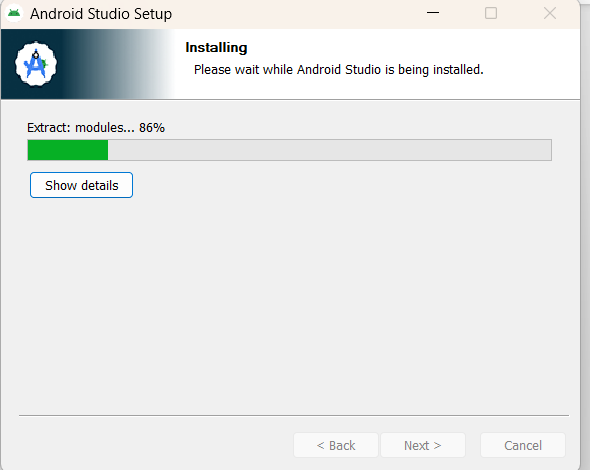
It displays domain information like the register date, update date, registrar provider etc..

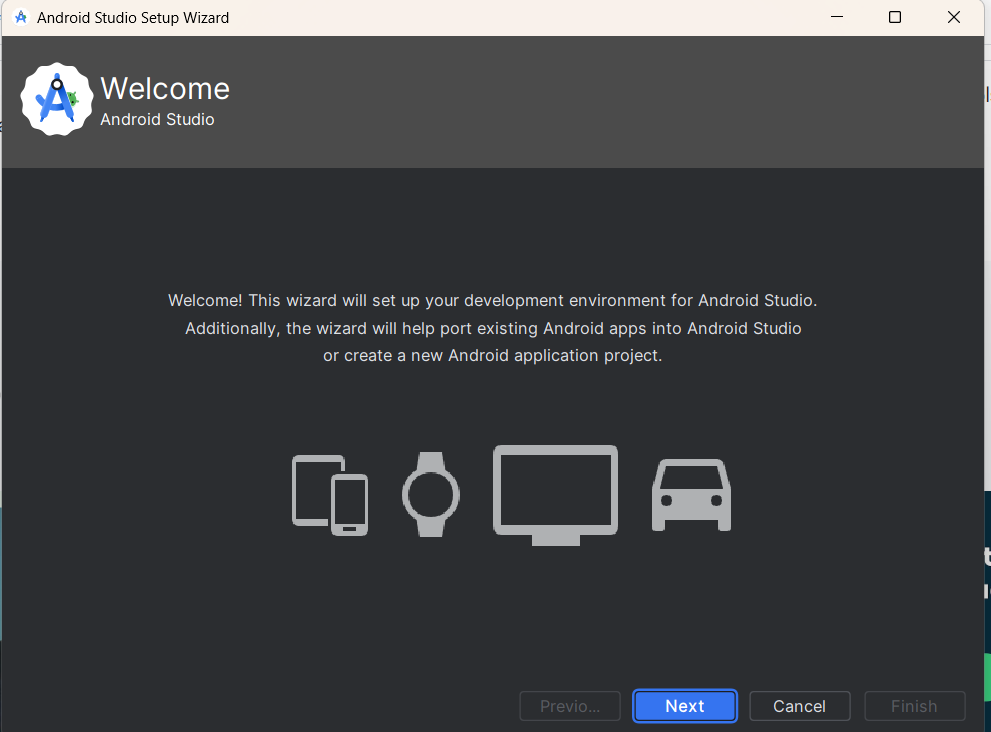
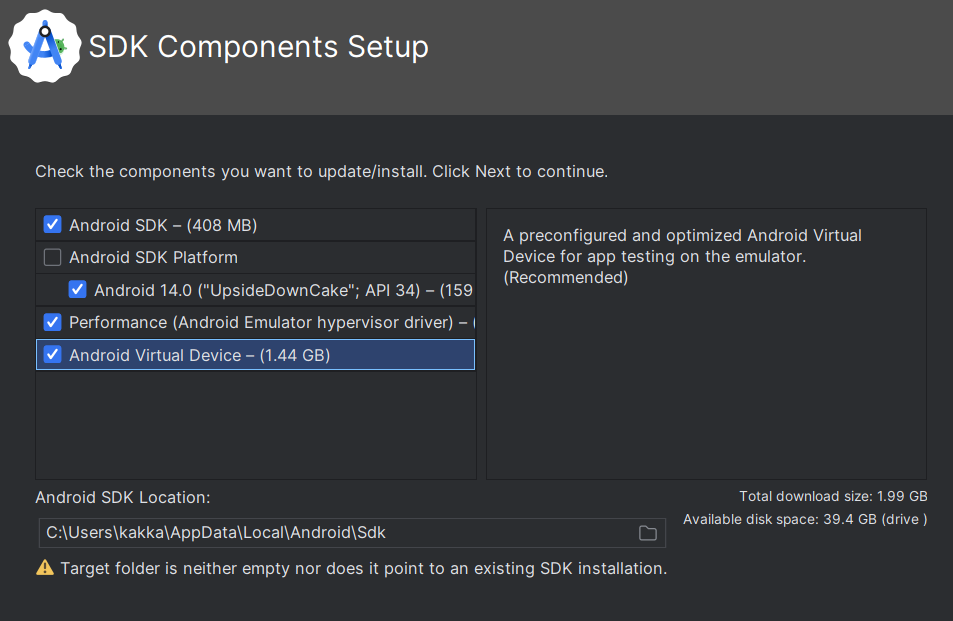
**Step-1)**

Download Android Studio from the official website: <https://developer.android.com/studio> Run the downloaded installer and follow the on-screen instructions. Once the installation is complete, launch Android Studio.

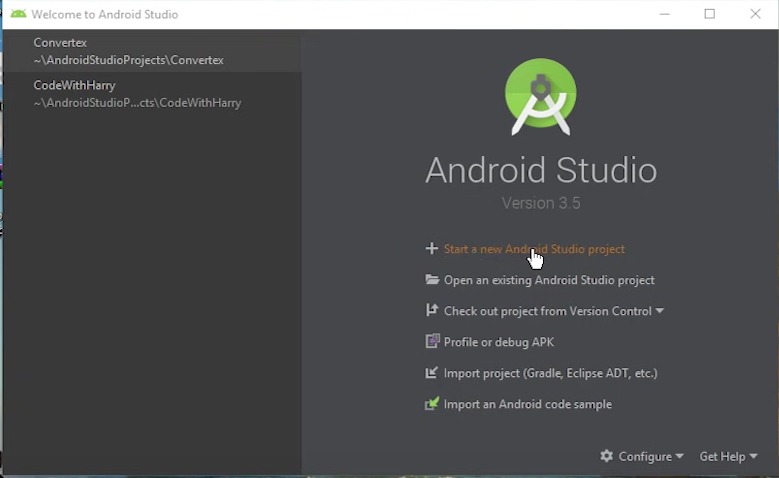






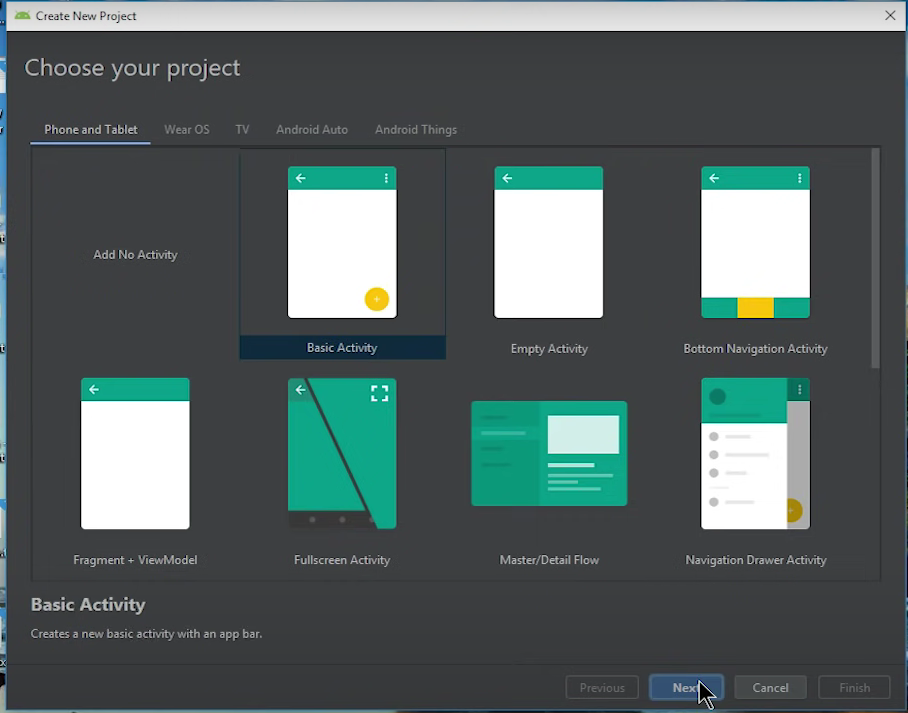


**Step-2)**

Once the installation is complete, launch Android Studio. In Android Studio, click on "Start a new Android Studio project" or select "File" > "New" > "New Project."

**Step-3)**

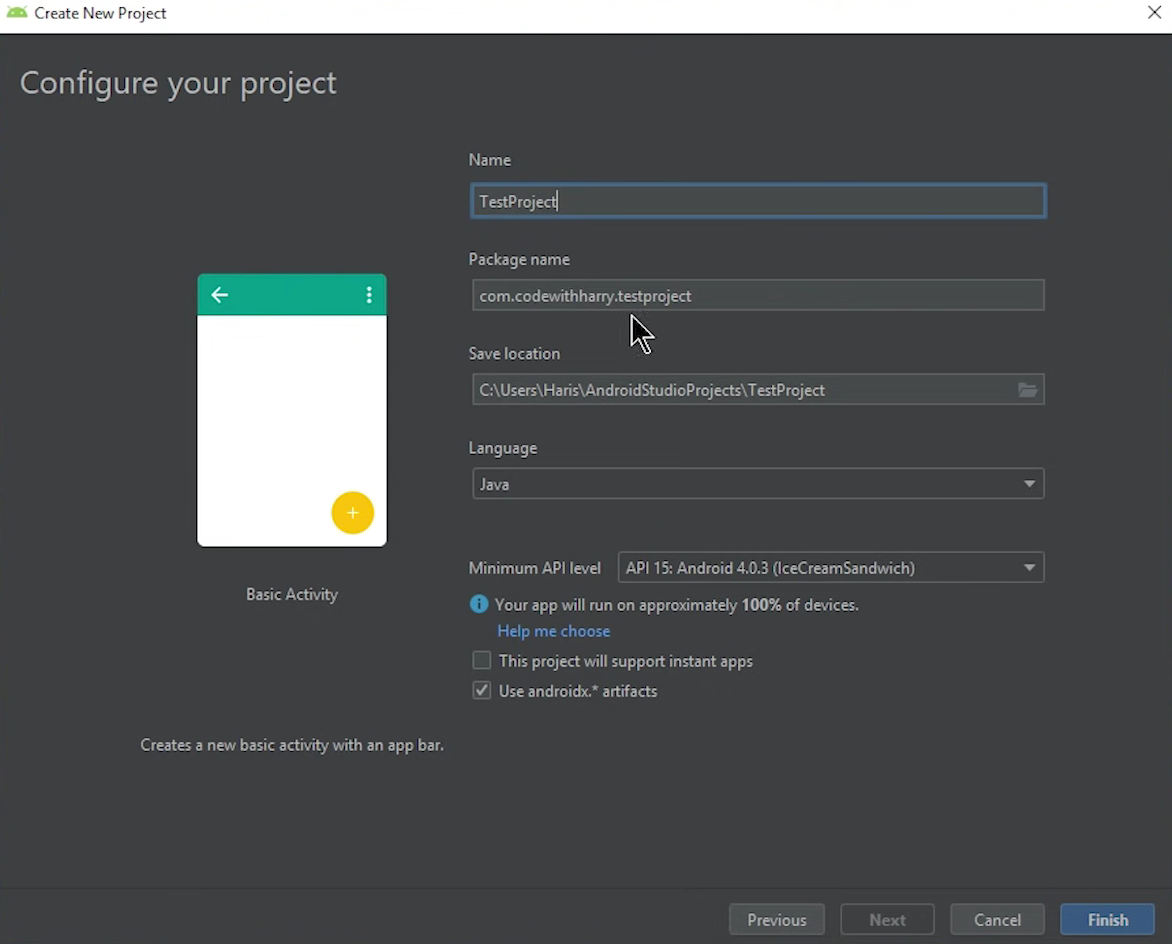
Choose "Empty Activity" or "Basic Activity" template for simplicity and click "Next."



**Step-4)**

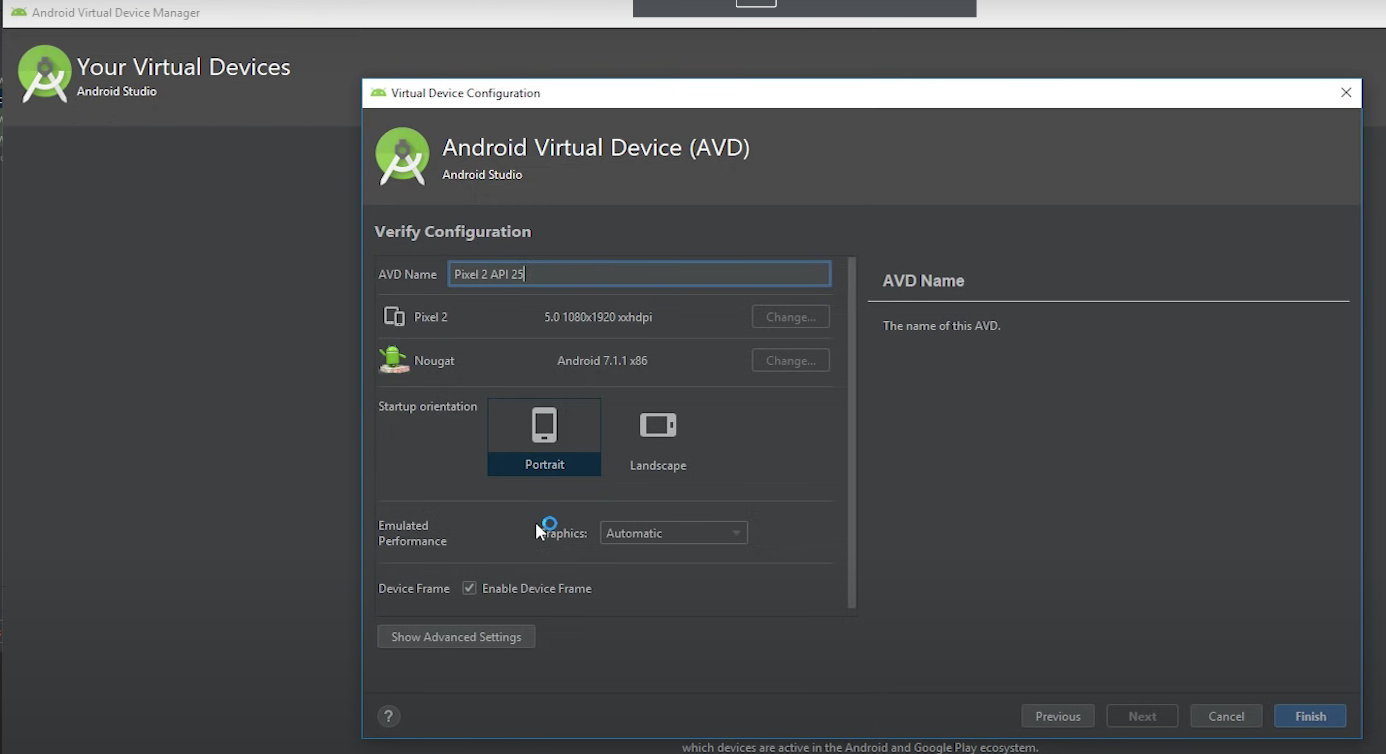
Enter the name of your app and package name, choose the language (Java or Kotlin), and set the minimum SDK version.

Click "Finish" to create the project.



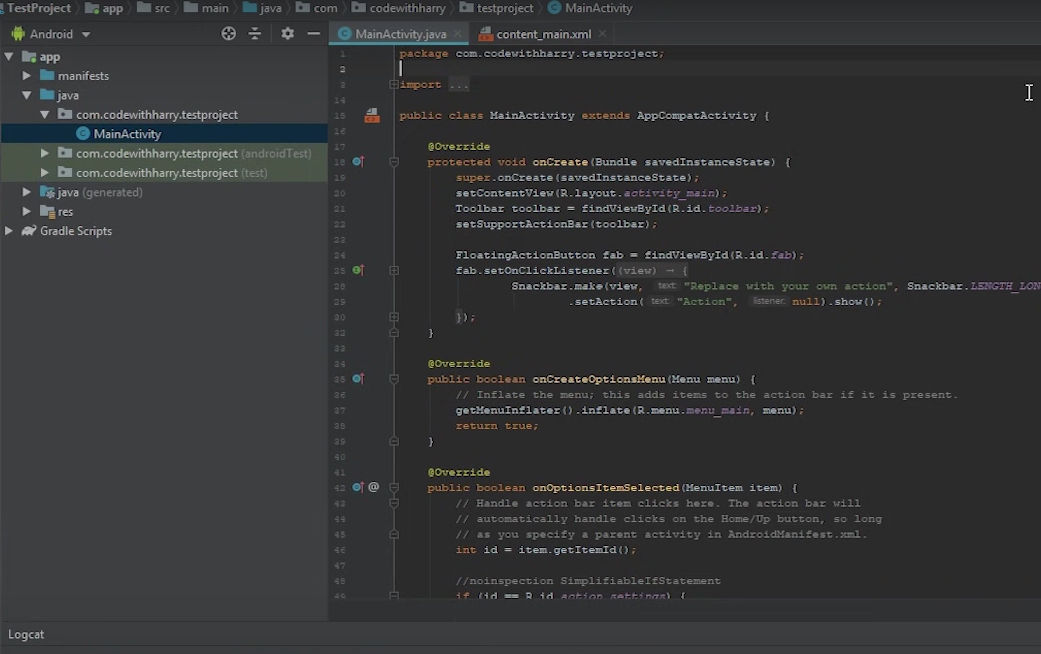
**Step-5)**

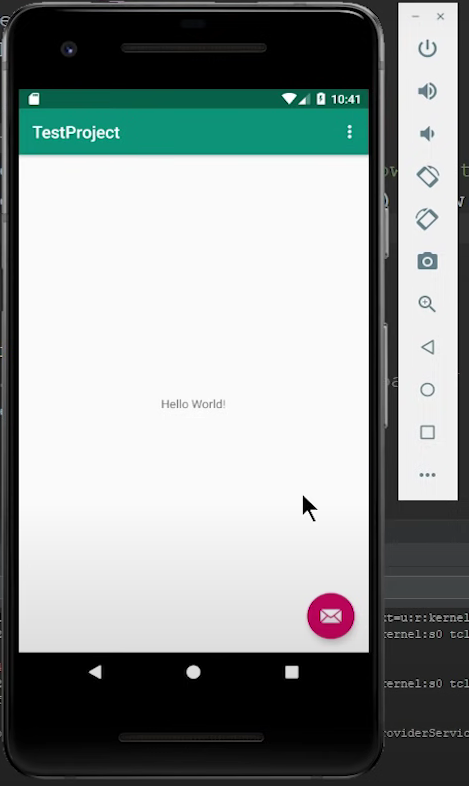
Connect an Android device to your computer or launch an Android Virtual Device (AVD) from the AVD Manager.



**Step-6)**

Open the activity\_main.xml layout file located in the res/layout directory.Replace the existing XML code with the following code to create a TextView displaying "Hello World!"Open the MainActivity.java file located in the java/<package\_name> directory.Replace the contents of the onCreate() method with the following code to set the content view to the layout file.





**Step-7)**

Click on the green play button (Run 'app') in the toolbar or select "Run" > "Run 'app'" from the menu.Select the target device or emulator and click "OK" to install and run the app.The "Hello World" app should launch on the device or emulator, displaying the text "Hello World!" on the screen.

**CODE:-**

1. **activity\_main.xml layout file:**

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

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

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello World!"

android:textSize="24sp"

android:layout\_centerInParent="true"/>

</RelativeLayout>

1. **MainActivity.java file:**

@Override

protected void onCreate(Bundle savedInstanceState) {

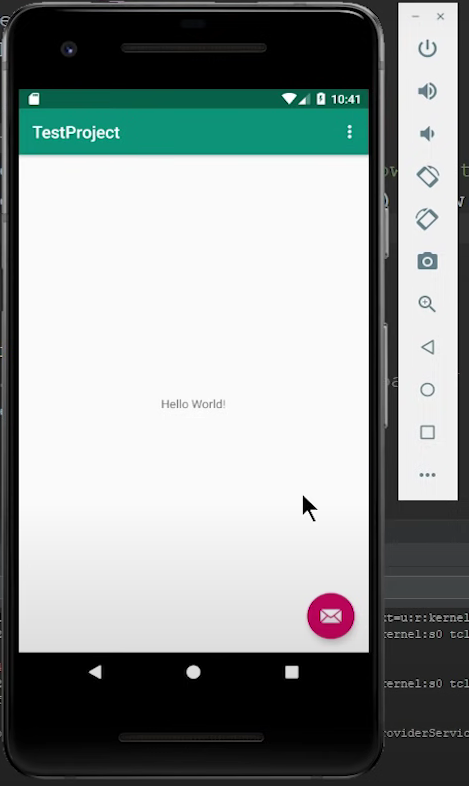
super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

**RESULT:-**

You've successfully set up the Android development environment, installed Android Studio, and created a basic "Hello World" app for Android.



**VIVA VOICE QUESTION**

**Question 1: What is Android Studio, and why is it the recommended IDE for Android development?**

Answer: Android Studio is the official integrated development environment (IDE) for Android app development, provided by Google. It offers a comprehensive set of tools for building, testing, and debugging Android applications. Android Studio is preferred for Android development because it provides features like code completion, visual layout editor, emulator support, and seamless integration with other Android development tools.

**Question 2: How do you install Android Studio on a Windows machine?**

Answer: To install Android Studio on a Windows machine, follow these steps:

* Download the Android Studio installer from the official website.
* Run the installer and follow the on-screen instructions.
* Choose the components you want to install, including the Android SDK and virtual device support.
* Select the installation location and click "Next."
* After the installation is complete, launch Android Studio and complete the setup wizard.
* Finally, update the SDK components and configure the emulator for testing.

**Question 3: What are the main components of the Android development environment setup?**

Answer: The main components of the Android development environment setup include:

* Android Studio: The primary IDE for Android development.
* Android SDK: Contains libraries, tools, and APIs necessary for Android app development.
* Emulator or Physical Device: Used for testing and debugging Android apps.
* Java Development Kit (JDK): Required for compiling Java code used in Android development.
* Additional tools and libraries: Optional components like Google Play services, support libraries, and third-party dependencies.

**EXPERIMENT-2**

**AIM:**- User Interface Design and Development

● Design a simple user interface using XML for an Android app.

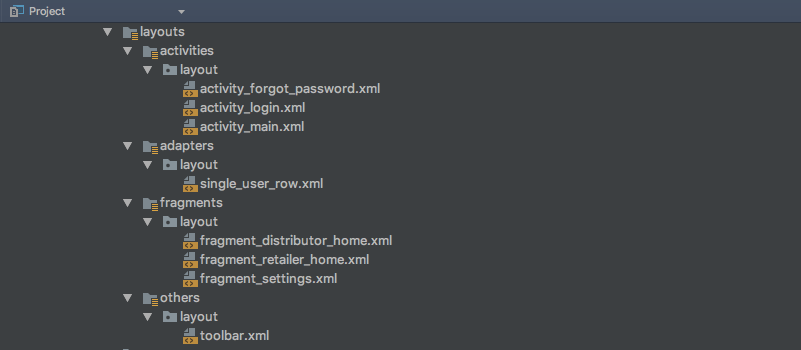
**PROCEDURE:-**

**Step-1)** **Open Android Studio:** Launch Android Studio on your computer.

**Step-2)** **Navigate to Layout Folder:** Locate the project explorer and find the res folder within your Android project. Expand the res folder, and then expand the layout folder.

**Step-3)** **Create a New XML Layout File:** Right-click on the layout folder and select New > XML resource file.

**Step-4)** **Name the XML File:** Enter a suitable name for your XML layout file, such as activity\_login.xml, in the dialog box that appears and click OK.



**Step-5)** **Open the XML File for Design:** After creating the XML file, it will open in the Android Studio editor. This is where you'll define the visual layout of your screen using XML markup.

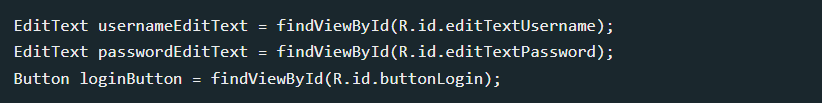
**Step-6)** **Define the Layout Structure:** Inside the XML file, start by defining the root layout container. For example, use a **LinearLayout** with vertical orientation to stack elements on top of each other.



In this , we use **EditText** for username and password input fields, and a **Button** for the login action.

**Step- 7) Save the XML Layout File:** Ensure to save the XML file by pressing **Ctrl + S** (Windows/Linux) or **Cmd + S** (Mac).

**Step- 8) Connect XML with Java Code:** To make the login screen functional, connect the XML elements to Java code in your activity file (**LoginActivity.java**).

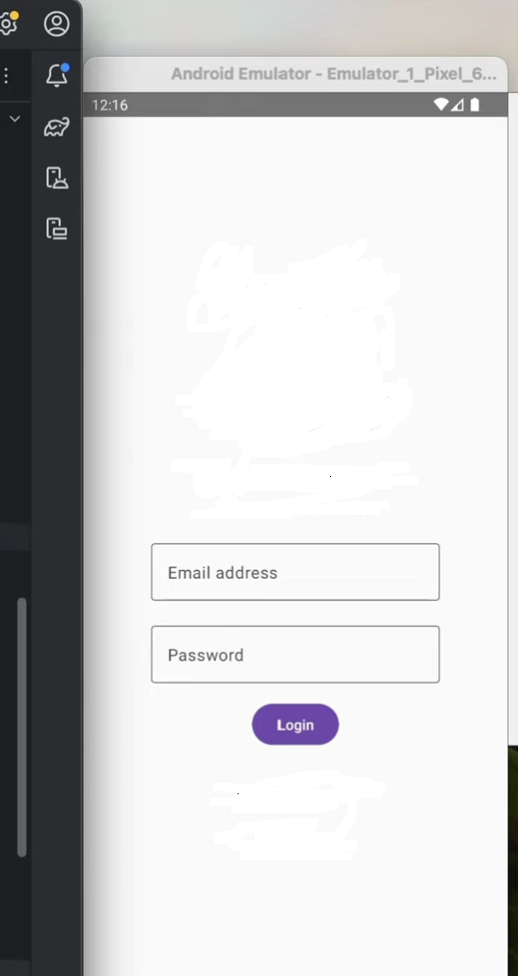


Replace **R.id.editTextUsername**, **R.id.editTextPassword**, and **R.id.buttonLogin** with the actual IDs assigned to these views in your XML layout file (**activity\_login.xml**).

**Step -9) Build and Run:** Finally, build and run your Android app to see the designed login screen in action on an emulator or physical device.

**RESULT:** –

Following these steps will allow us to create a basic user interface layout using XML for your Android app's login screen and integrate it with Java code to handle user interactions and functionality.



**VIVA VOICE QUESTION**

**Question 1: What is XML, and why is it used in Android app development?**

Answer: XML stands for eXtensible Markup Language. In Android app development, XML is used to define the layout and structure of user interfaces. It provides a way to separate the presentation of an app from the application logic. XML files are used to describe the visual elements of an app's user interface, such as buttons, text fields, and layouts.

**Question 2: What are some common layout elements used in designing Android user interfaces?**

Answer: Common layout elements used in Android UI design include:

* Linear Layout: Arranges child views in a single row or column.
* Relative Layout: Positions child views relative to each other or to the parent layout.
* Constraint Layout: Allows you to create complex layouts with flexible constraints between views.
* Frame Layout: Places child views on top of each other, with the last added view appearing at the top.

**Question 3: Explain the concept of "resource qualifiers" in Android XML layouts**.

Answer: Resource qualifiers allow you to specify alternative resources for different device configurations such as screen size, orientation, and language. By using resource qualifiers in XML layout files, you can provide different layouts optimized for various device configurations. For example, you can create separate layout files for landscape and portrait orientations or for different screen sizes.

**Question 4: How do you add UI elements such as buttons, text fields, and images to an XML layout file?**

Answer: To add UI elements to an XML layout file, you can use XML tags representing different view elements. For example:

<Button>: Defines a clickable button.

<TextView>: Displays text to the user.

<ImageView>: Displays an image.

You can set attributes such as android:id, android:text, android:src, etc., to customize the appearance and behavior of these UI elements.

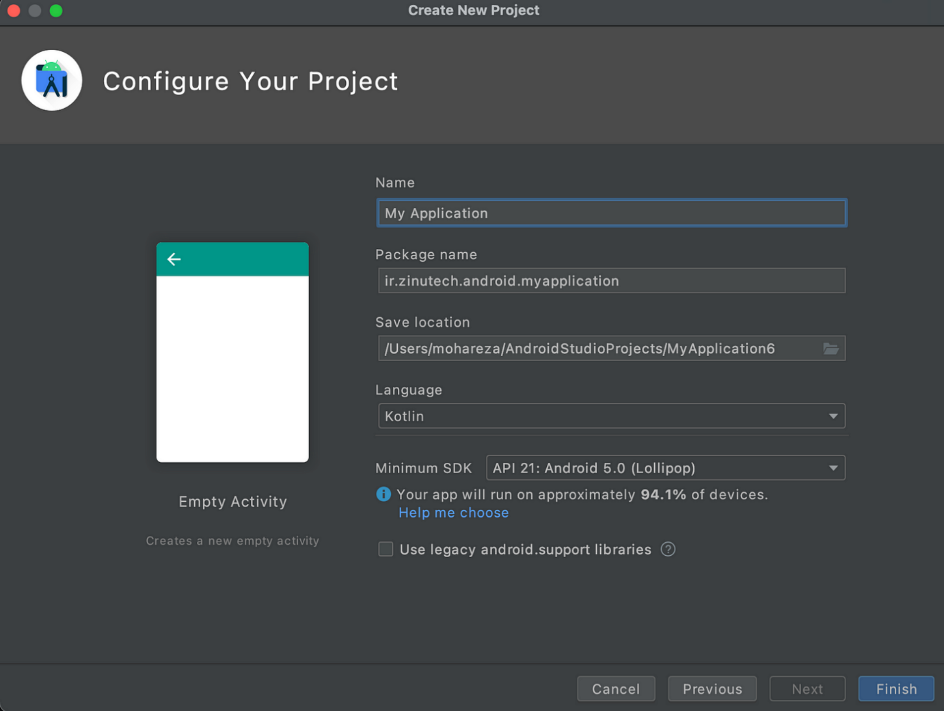
**EXPERIMENT-3**

**AIM:**- Database Integration in Mobile Apps

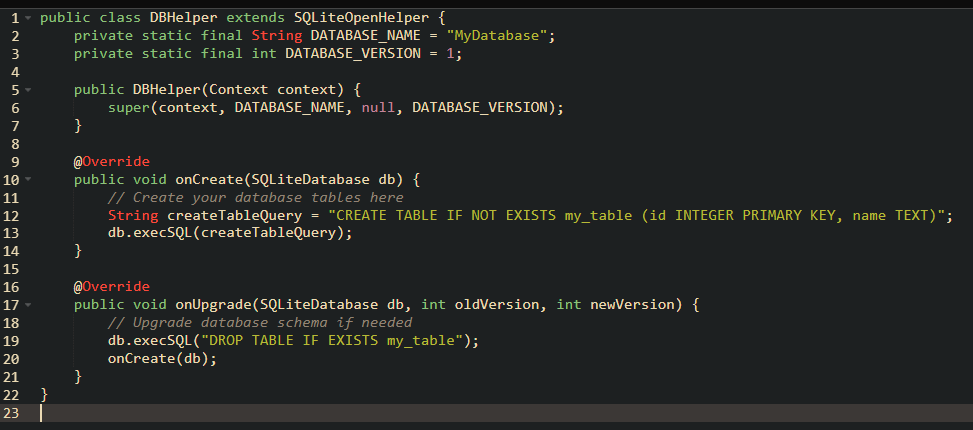
● Create a SQLite database and perform basic CRUD operations in an Android app.

**PROCEDURE:**

**Step-1)** **Setting Up the Project:** Create a new Android project in Android Studio, starting with an empty activity template, by configuring project details and clicking Finish.

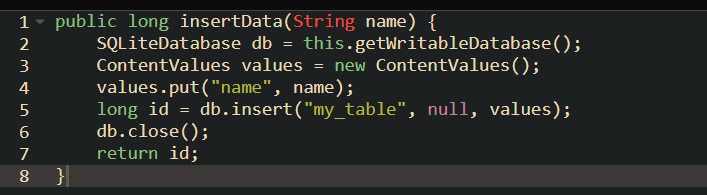


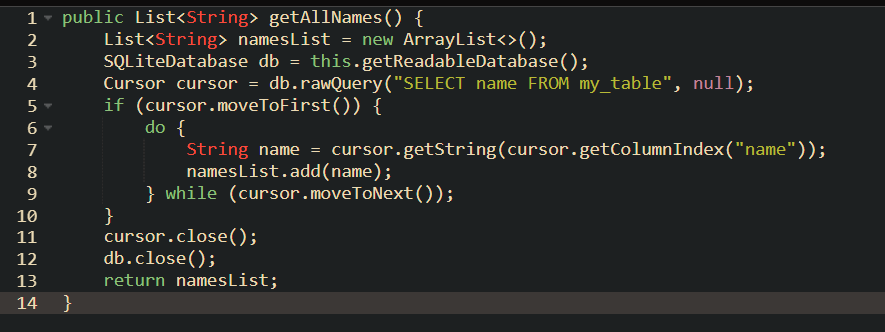
**Step-2)** **Adding SQLite Support:** Add SQLite dependency by opening the build.gradle (Module: app) file and including the specified dependency under the dependencies block.

**Step-3)** **Creating Database Helper Class:** Create a DBHelper Java class extending SQLiteOpenHelper, implementing onCreate() and onUpgrade() methods for database setup and versioning.

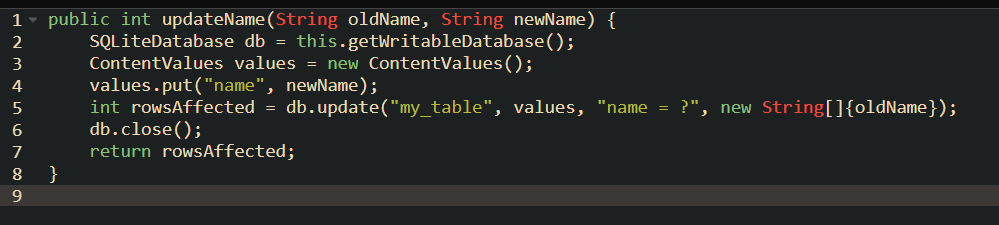
**Step-4)** **Performing CRUD Operations:**

4.1) **Inserting Data:** Inside **DBHelper** class, create a method to insert data into the database.

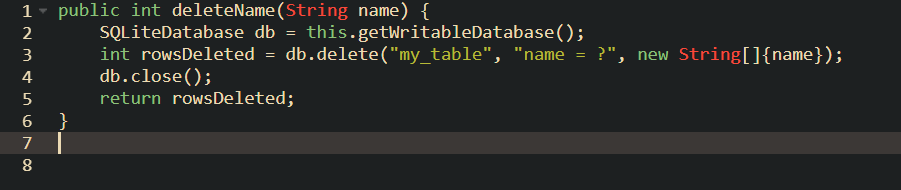


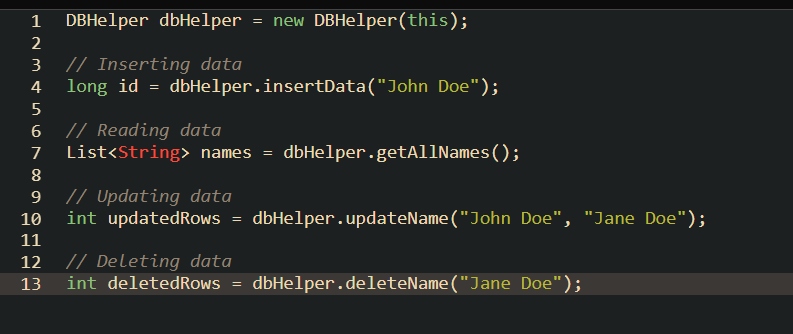
4.2) **Reading Data:** Create a method to retrieve all data from the database.

4.3) **Updating Data:** Implement a method to update existing data in the database.



4.4) **Deleting Data:** Add a method to delete data based on a specific condition.

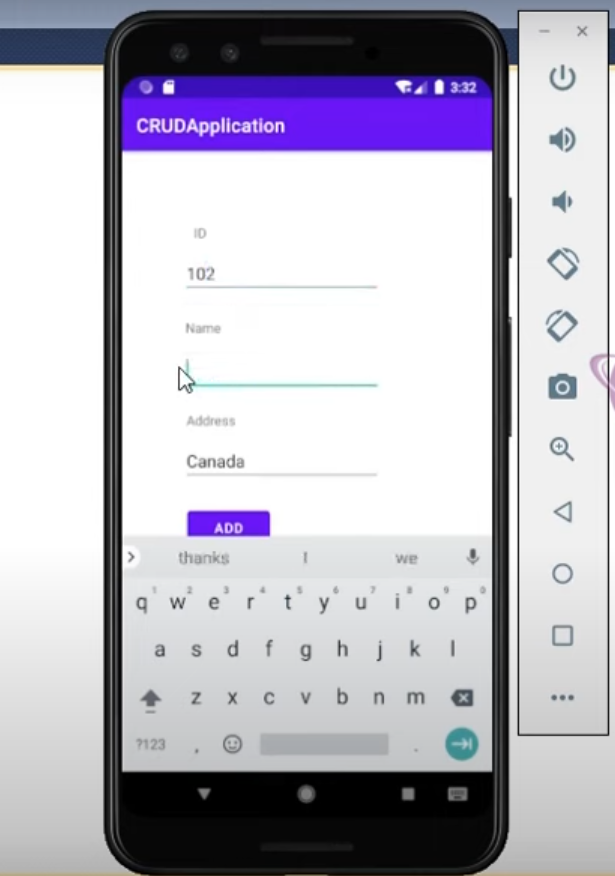


**Step-5)** **Using Database Operations in Activity:** Instantiate DBHelper within MainActivity to utilize its methods for database operations.

**Step-6)** **Testing the App:** Test the implemented SQLite operations by building and running the app on an emulator or device. Verify functionality including data insertion, retrieval, updating, and deletion to ensure expected results.

**RESULT:**

By following these steps, you'll have successfully integrated SQLite database operations into your Android app using a **DBHelper** class. This allows you to perform basic CRUD (Create, Read, Update, Delete) operations on a local SQLite database from within your app.



**VIVA VOICE QUESTION**

**Question 1: What is SQLite, and why is it commonly used in Android app development?**

Answer: SQLite is a lightweight, embedded relational database management system (RDBMS) that is widely used in Android app development. It provides a simple and efficient way to store and manage structured data locally on the device. SQLite databases are self-contained, serverless, and require minimal setup, making them ideal for mobile applications.

**Question 2: What are the primary database operations that can be performed using SQLite in an Android app?**

Answer: The primary database operations in SQLite include:

* Create: Creating a new database or table.
* Read: Retrieving data from the database using SELECT queries.
* Update: Modifying existing data in the database.
* Delete: Removing data or records from the database.

Additionally, SQLite supports operations like INSERT, ALTER, and DROP for manipulating data and schema.

**Question 3: How do you create a SQLite database in an Android app?**

Answer: To create a SQLite database in an Android app, follow these steps:

* Define a subclass of SQLiteOpenHelper to manage database creation and version management.
* Override the onCreate() method to execute SQL statements for creating tables and initializing the database schema.
* Optionally, override the onUpgrade() method to handle database schema upgrades when the database version changes.

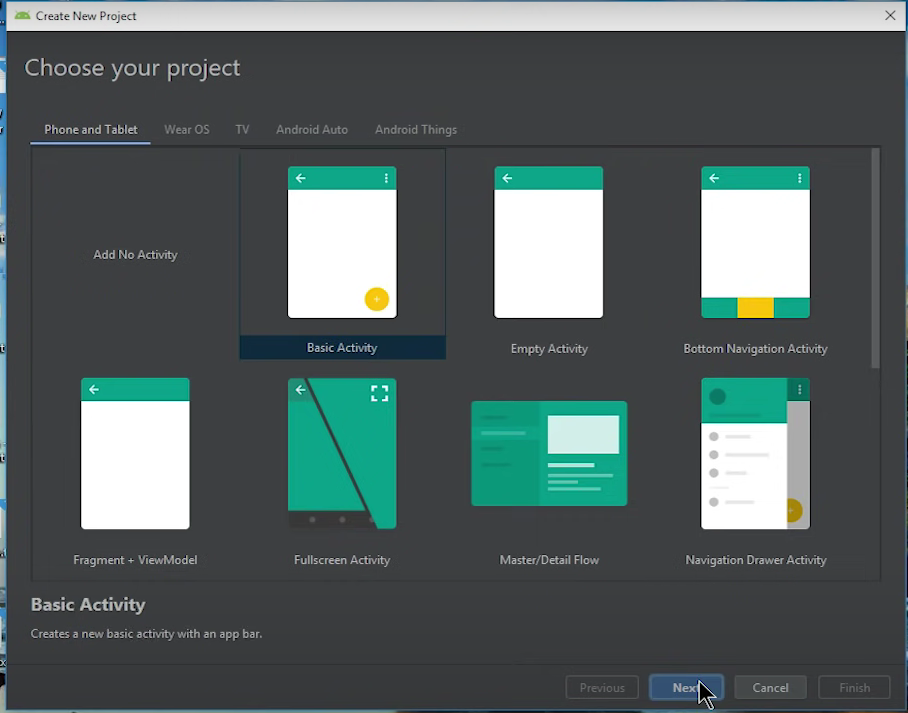
**EXPERIMENT-4**

**AIM:**- Web Services and API Consumption

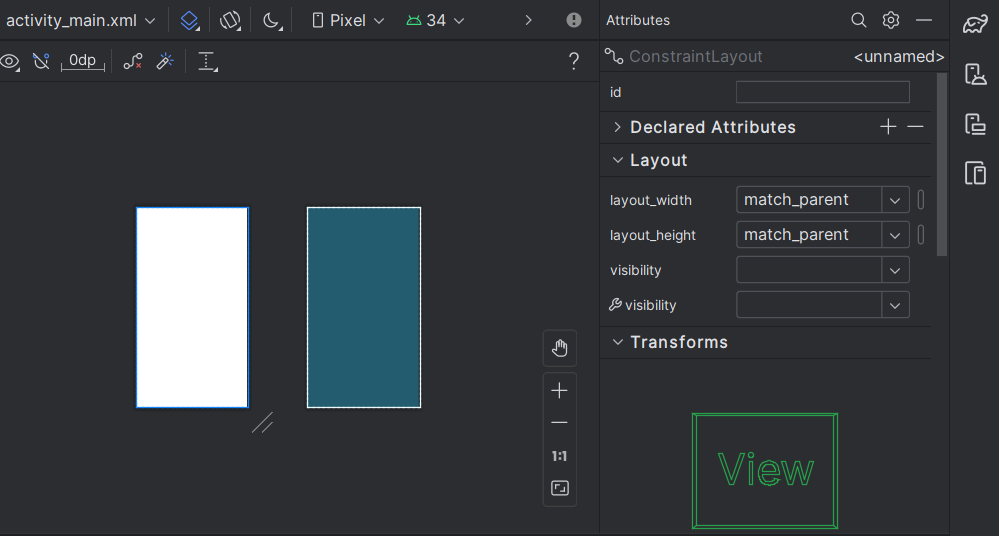
● Retrieve data from a RESTful API and display it in an Android app.

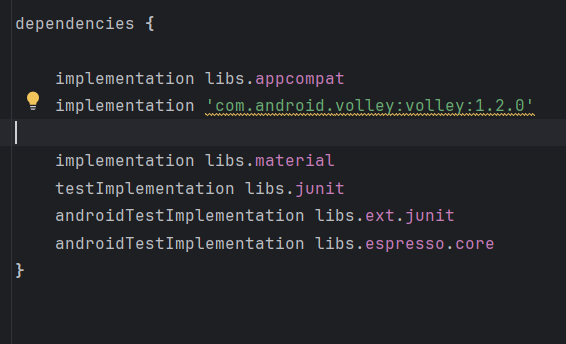
**PROCEDURE:-**

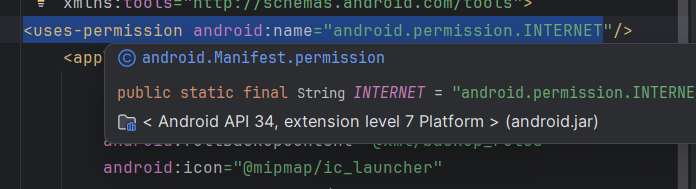
**Step-1)** Open a new project in Android Studio.

**Step-2)** choose basic activity project .

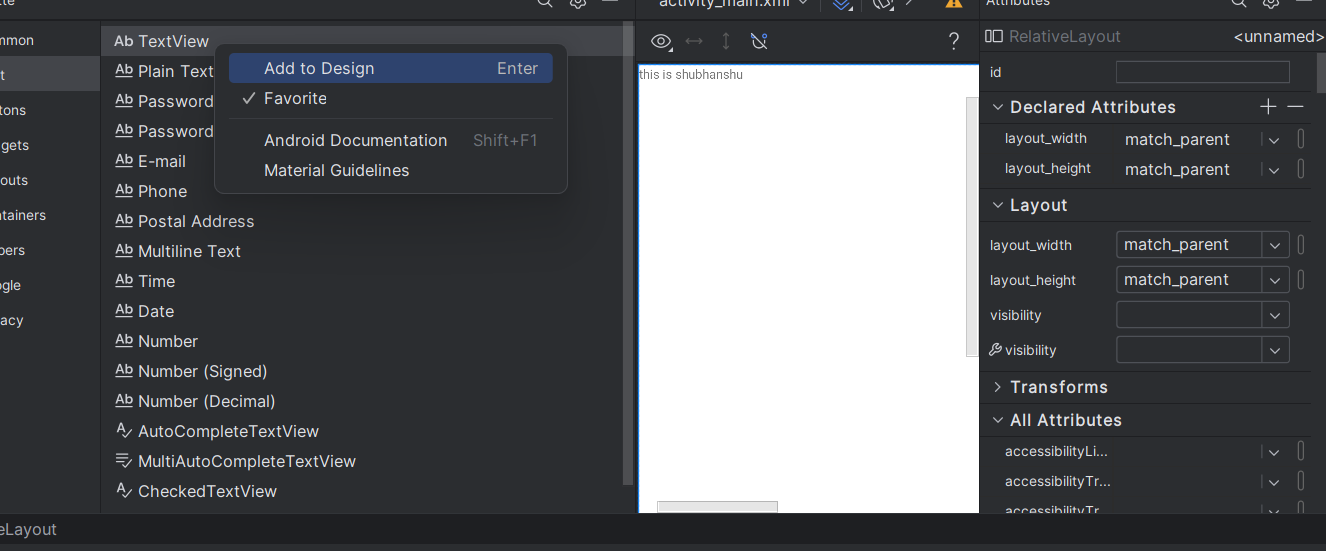
**Step-3)** Create activity file :

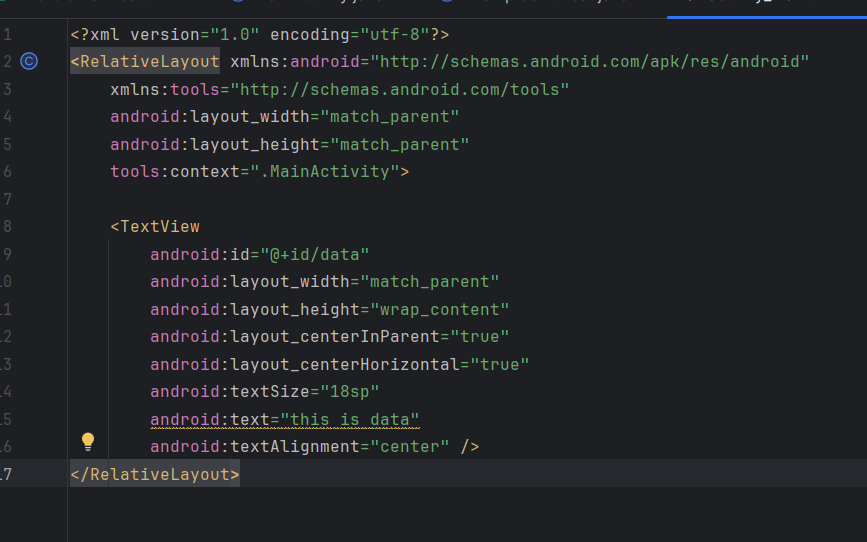
**Step-4)** Create activity main .xml file in layout

**Step-5)** In guild scripts folder Go to build gradle and add dependency for volley and click on sync changes

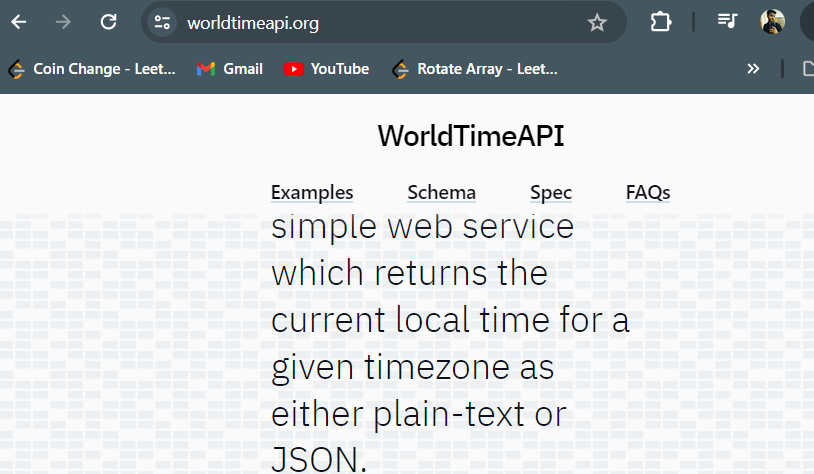
**Step-6)** Add the <uses-permission android:name="android.permission.INTERNET"/> line to the AndroidManifest.xml file. This permission enables your app to create network sockets and communicate over the network.

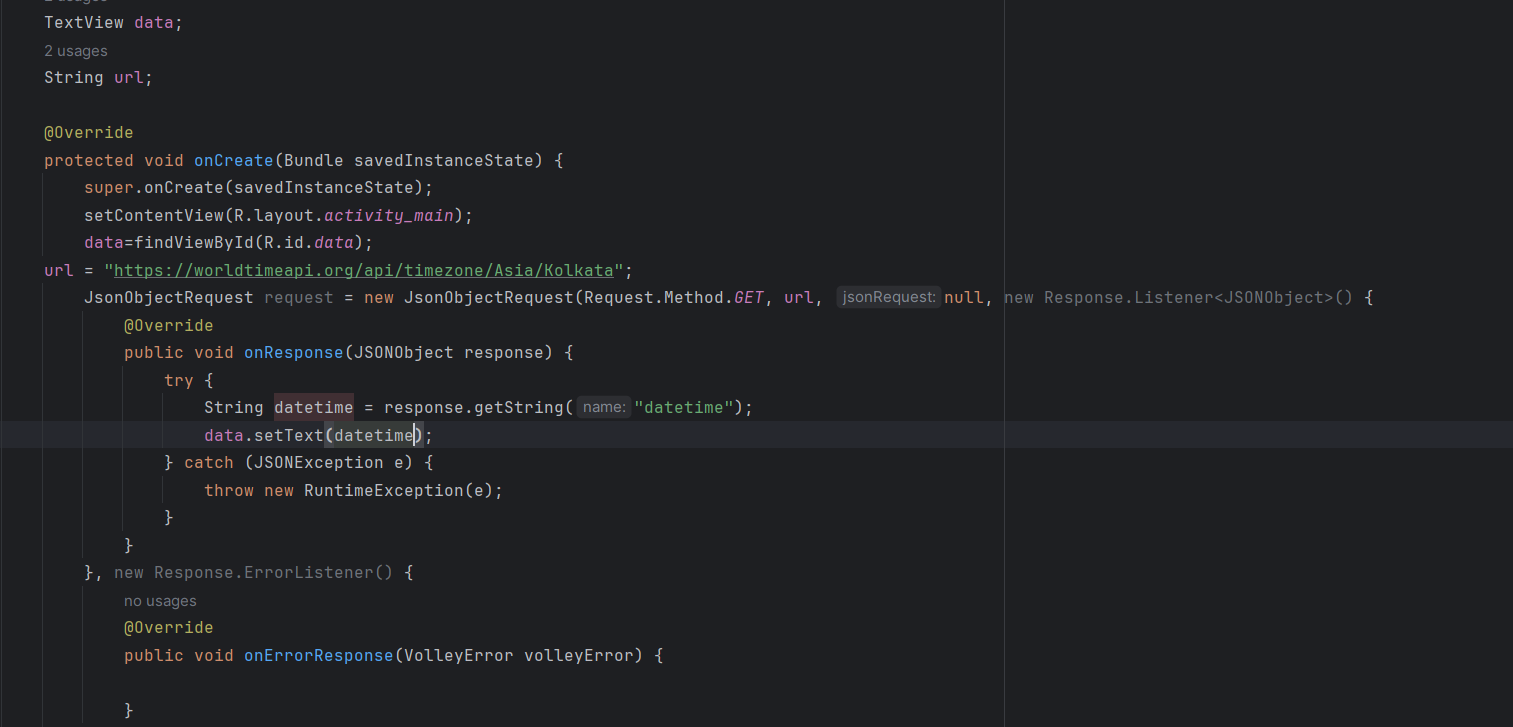
**Step-7)** Navigate to the activity\_main.xml file and include a TextView component. Set its height and width attributes accordingly.



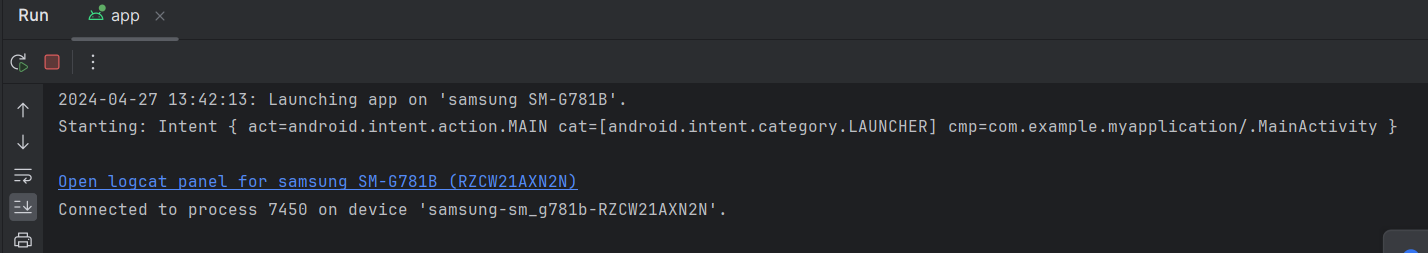


**Step-8)** Now, copy the API URL string from any open-source web service. For instance, we can use the World Time API:



**Step-9)** Create a JSONObject request body that accepts a request from a URL. Surround it with a try-catch block and include proper exception handling.

**Step-10)** Finally, click on the "Run" button to start the project. Gradle will initiate the build process.





**RESULT:**

As we can see the current time and date can be fetched for the desired location and time zone successfully using a restful API web service.

**VIVA VOICE QUESTION**

**Question 1: What is a RESTful API, and why is it commonly used in mobile app development?**

Answer: A RESTful API (Representational State Transfer) is an architectural style for designing networked applications. It uses standard HTTP methods like GET, POST, PUT, and DELETE to perform CRUD operations on resources. RESTful APIs are commonly used in mobile app development because they provide a flexible and scalable way to interact with server-side data and services.

**Question 2: What are the different methods available for making network requests in Android apps?**

Answer: There are several methods for making network requests in Android apps, including:

* HttpURL Connection: Standard Java library for making HTTP requests.
* Http Client: Deprecated library for making HTTP requests, replaced by HttpURLConnection or third-party libraries.
* Retrofit: A type-safe HTTP client for Android and Java that simplifies network requests by abstracting away the HTTP implementation.
* Volley: A networking library provided by Google that makes it easier to manage network requests and responses asynchronously.

**Question 3: How do you handle asynchronous network requests in Android apps to prevent blocking the main UI thread?**

Answer: Use AsyncTask or background threads to perform network operations asynchronously.

Alternatively, use libraries like Retrofit or Volley, which handle asynchronous operations internally.

Ensure that UI updates are performed on the main UI thread using methods like runOnUiThread or AsyncTask's onPostExecute method.

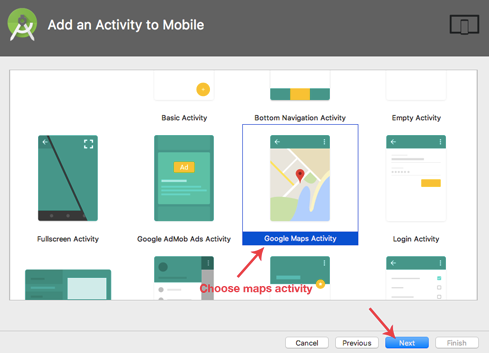
**EXPERIMENT-5**

**AIM:** - Location-Based Services and Mapping

● Implement location tracking and display user location on a map in an Android app using Google Maps API.

**PROCEDURE:-**

**Step-1)** **Setup Google Maps API:** Create a new project on the Google Cloud Console, enable the Google Maps SDK for Android, and generate an API key.



**Step-2)** **Set Up Project Dependencies:** In your app-level `build.gradle` file, add the dependency for Google Maps:

```gradle

Implementation'com.google.android.gms:play-services-maps:18.0.0'

```

**Step-3)** **Add Permissions:** - Ensure your app has the necessary permissions in the `AndroidManifest.xml` file:

```xml

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

```

**Step-4)** **Layout File** - Create a layout file (`activity\_main.xml`) with a `MapView` element to display the map

```xml

<com.google.android.gms.maps.MapView

android:id="@+id/mapView"

android:layout\_width="match\_parent"

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

```

**Step-5)** **Initialize GoogleMap:** - In your activity or fragment, initialize the `MapView` and `GoogleMap` object:

```

lateinit var mapView: MapView

lateinit var googleMap: GoogleMap

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

mapView = findViewById(R.id.mapView)

mapView.onCreate(savedInstanceState)

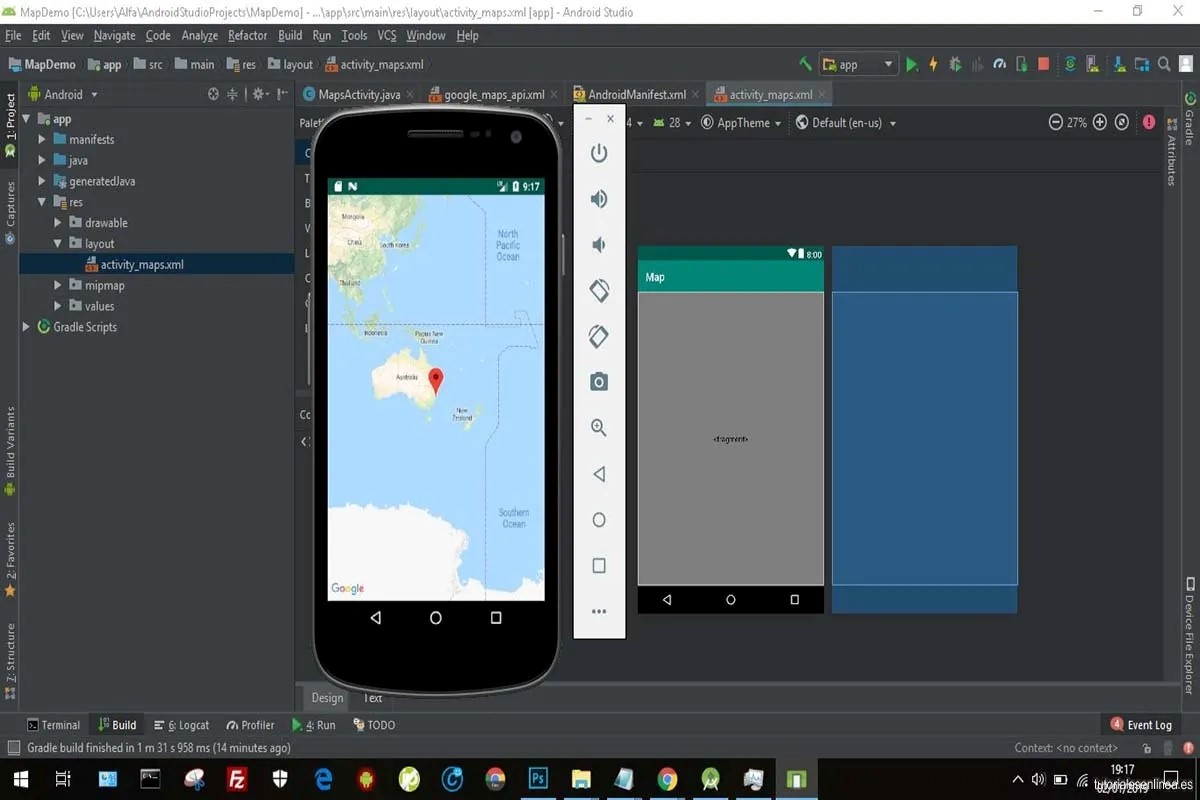
mapView.getMapAsync { map ->

googleMap = map

googleMap.isMyLocationEnabled = true

}

}

 ```

**Step-6)** **Handle Lifecycle Events::** - Forward lifecycle events to the `MapView`

```

override fun onResume() {

super.onResume()

mapView.onResume()

}

override fun onPause() {

super.onPause()

mapView.onPause()

}

override fun onDestroy() {

super.onDestroy()

mapView.onDestroy()

}

override fun onLowMemory() {

super.onLowMemory()

mapView.onLowMemory()

}

```

**Step-7)** **Location Tracking:** - - Use the `FusedLocationProviderClient` to request and receive updates about the device's location:

```

valfusedLocationClient= locationServices.getFusedLocationProviderClient(this)

fusedLocationClient.lastLocation.addOnSuccessListener { location ->

// Use the location object

if (location != null) {

val currentLatLng = LatLng(location.latitude, location.longitude)

// Move camera to current location

googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(currentLatLng, 15f))

}

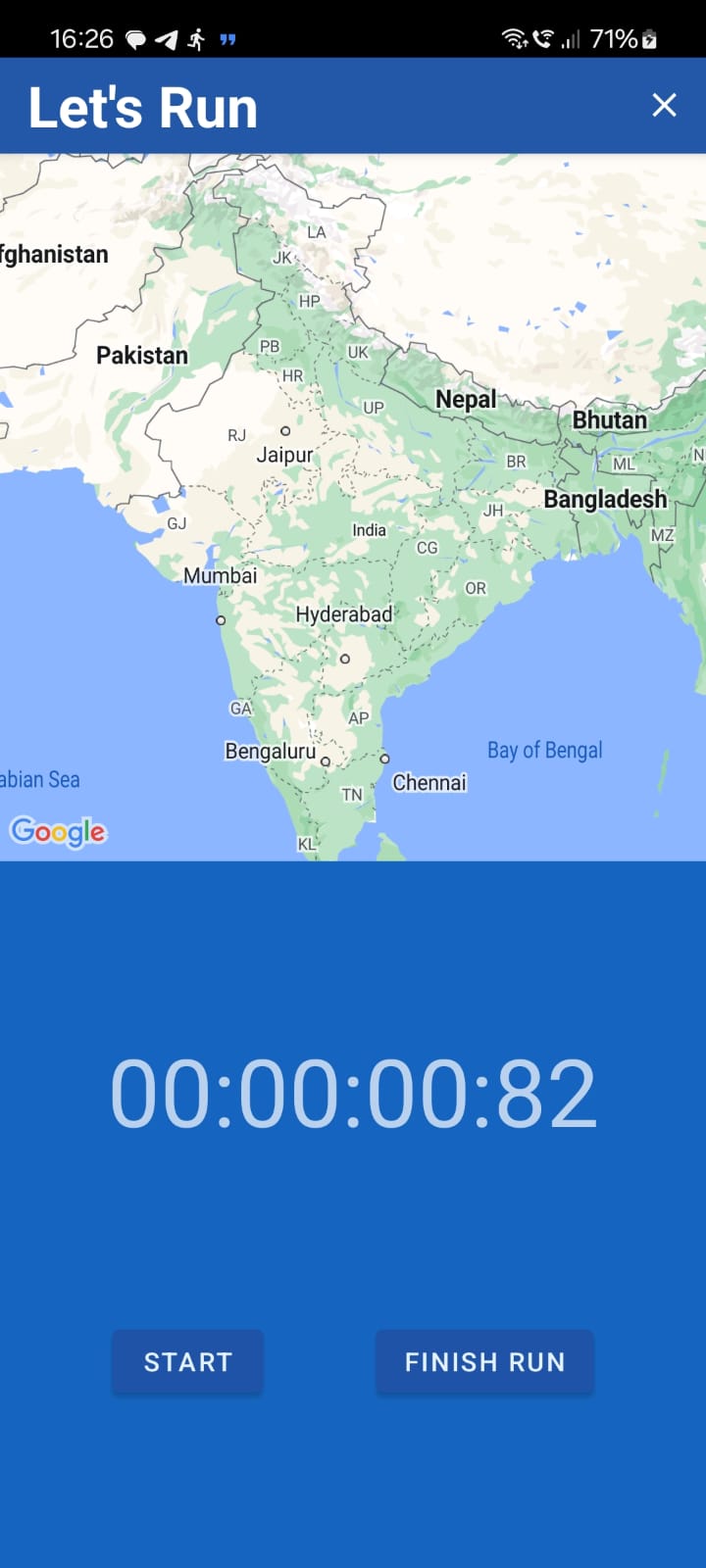
}

```

**Step-8)** **Run Your App:** - Ensure that your device or emulator has Google Play Services installed, then run your app to test the location tracking functionality.

**RESULT:**

By following these steps, you can implement location tracking and display the user's location on a map in an Android app using the Google Maps API. Experiment with additional features like adding markers, drawing routes, or handling location updates to enhance the functionality further.



**VIVA VOICE QUESTION**

**Question 1: What are location-based services, and why are they important in mobile app development?**

Answer: Location-based services involve using location data to provide relevant information or functionality to users based on their current location. These services enable apps to offer features such as mapping, navigation, local search, and location-based notifications. Location-based services are essential in mobile app development as they enhance user experience, improve engagement, and enable a wide range of location-aware applications.

**Question 2: What is the Google Maps API, and how does it enable mapping functionality in Android apps?**

Answer: The Google Maps API is a set of APIs provided by Google that allows developers to integrate mapping and location-based services into their applications. It provides access to various features such as displaying maps, adding markers, drawing routes, and obtaining user location. In Android app development, the Google Maps Android API allows developers to embed Google Maps directly within their apps and customize the map's appearance and behaviour.

**Question 3: What are some common features and functionalities provided by the Google Maps API for Android apps?**

Answer:

* Map Display: Embed interactive maps within the app's UI and customize map styles, layers, and controls.
* Markers and Info Windows: Add markers to the map to indicate points of interest and display additional information in info windows.
* Polylines and Polygons: Draw lines and shapes on the map to represent routes, boundaries, or areas of interest.
* Geocoding and Reverse Geocoding: Convert between geographic coordinates and human-readable addresses, and vice versa.
* Place Autocomplete: Provide autocomplete suggestions for places based on user input in search or address fields.
* Routing and Directions: Calculate routes and provide turn-by-turn directions for driving, walking, cycling, or transit navigation.

**EXPERIMENT-6**

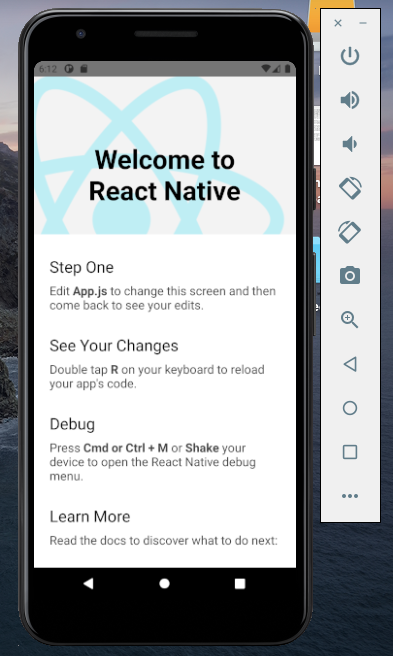
**AIM:**- Cross-Platform App Development with React Native or Flutter

● Develop a simple mobile app using React Native, utilizing components and navigation.

**PROCEDURE:-**

**Step-1)** **Setting Up Development Environment:**

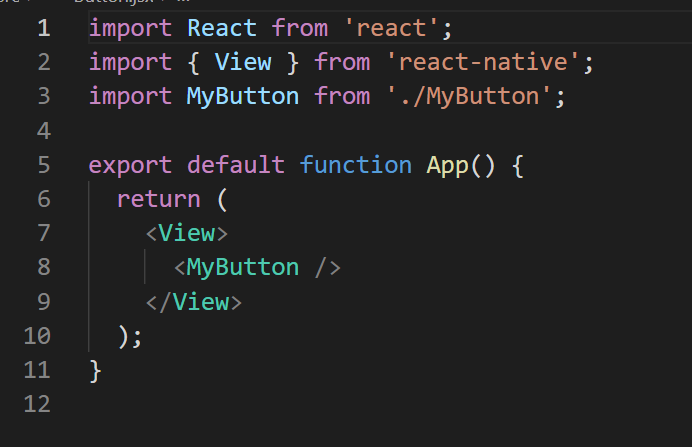
* 1. Install Node.js by visiting the official website, downloading the latest version, following OS-specific installation instructions, and verifying with "node -v" in the terminal.
  2. Install JDK 17 from the Oracle website, then verify the installation by running "java -version" in the terminal.
  3. Download and install Android Studio from the official website, launch it to complete the initial setup, then open the SDK Manager to install necessary components like Android SDK, Build Tools, and Platform Tools.
  4. Configure environment variables by setting up ANDROID\_HOME to point to your Android SDK location (%LOCALAPPDATA%\Android\Sdk) and adding %LOCALAPPDATA%\Android\Sdk\platform-tools to the Path variable.



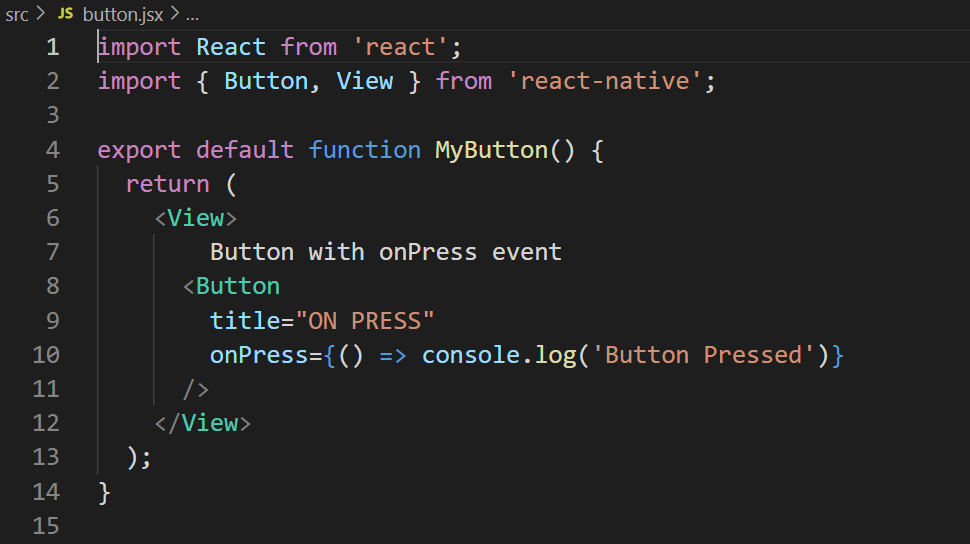
**Step-2)** **Installing React Native CLI and Creating a Project:**

1. Install React Native CLI globally by running "npm install -g react-native-cli" in the terminal.
2. Navigate to your desired project directory, then run "npx react-native init ProjectName" to create a new React Native project.

**Step-3)** **Create and Use React Native Components:** Create a new component file (e.g., MyButton.js) to define a reusable component using functional components with JSX syntax. Then, import and utilize the component in the main app file (e.g., App.js).

**App.jsx**

**Button.jsx**



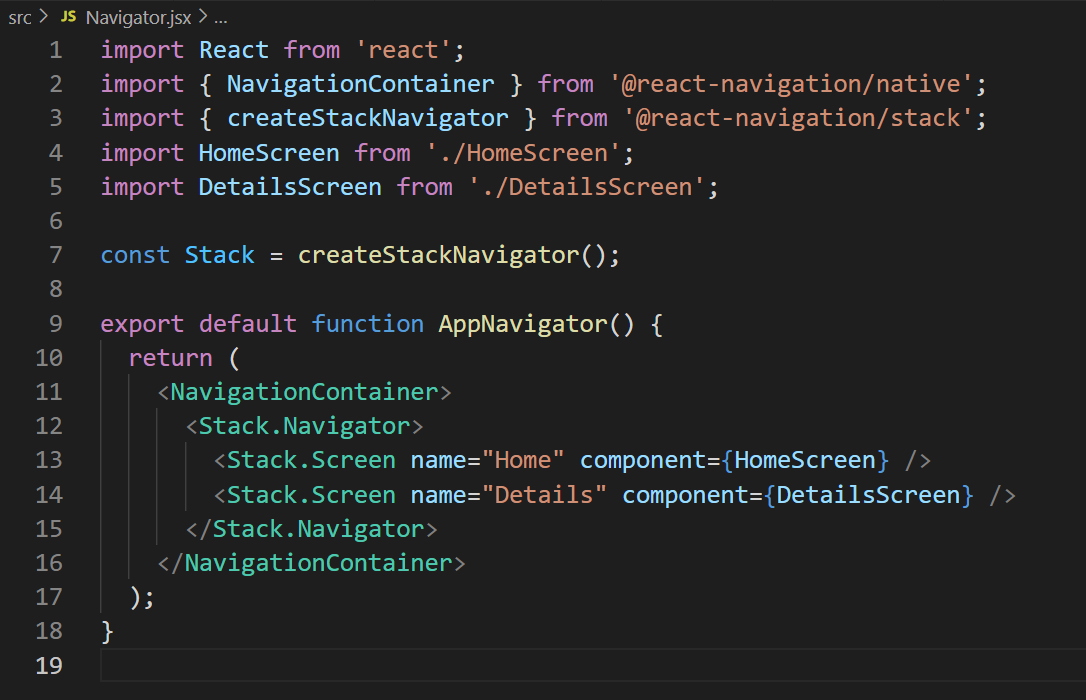
**Step-4)** **Implementing Navigation in React Native:**

1. Install the React Navigation library using npm with the following commands:

npm install @react-navigation/native

npm install react-native-reanimated react-native-gesture-handler react-native-screens react-native-safe-area-context @react-native-community/masked-view

1. Set up the navigation stack by creating a new navigation file (e.g., AppNavigator.js) to define navigation routes and configuring the navigation stack using createStackNavigator from React Navigation.



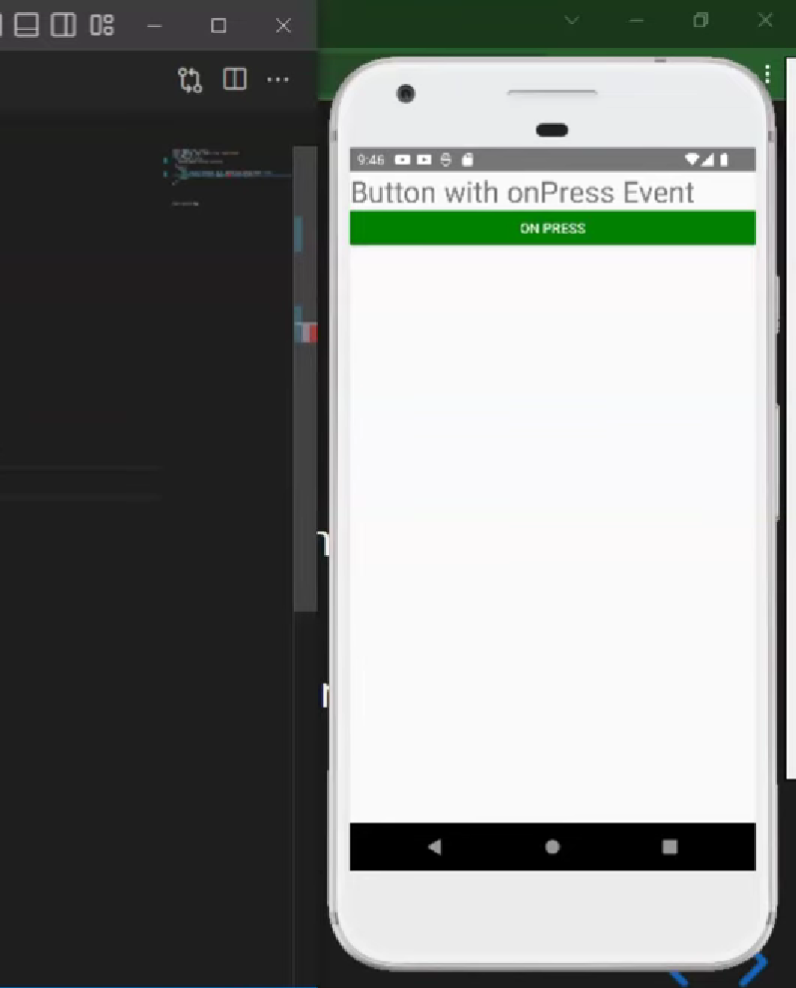
1. Import and use **AppNavigator** in the main app component (**App.js**).

**Step-5)** **Running and Testing the React Native App:**

1. To run your React Native app, set up an Android emulator or connect a device with debugging enabled. Start the Metro bundler with npm start, then run npm run android to deploy on Android (or use iOS command for iOS). This compiles your code and launches the app for testing.
2. To verify your React Native app's execution, keep an eye on the terminal for build progress and errors. Once launched, ensure the app appears on the emulator or device. Test navigation and component functionality to confirm proper integration. This process ensures your app is running smoothly and functions as intended on the target platform.

**RESULT:**

Upon completion of this experiment, you will have developed a simple React Native mobile app with integrated components and basic navigation. This experiment covers the essential steps for setting up the development environment, creating components, implementing navigation using React Navigation, and running the app on an emulator or physical device.



**VIVA VOICE QUESTION**

**Question 1: What is React Native, and how does it differ from traditional native app development?**

Answer: React Native is an open-source framework developed by Facebook for building mobile applications using JavaScript and React. Unlike traditional native app development, which involves writing separate codebases for iOS and Android using platform-specific languages like Swift or Java, React Native allows developers to write code once and deploy it across multiple platforms.

**Question 2: Explain the concept of components in React Native and their role in app development.**

Answer: Components are the building blocks of React Native applications. They are reusable, self-contained UI elements that encapsulate a piece of the app's UI and logic. Components can be simple, like a button or text input, or complex, like a navigation bar or list view. By composing components together, developers can create rich, interactive user interfaces for their apps.

**Question 3: What is navigation in the context of React Native app development, and why is it important?**

Answer: Navigation refers to the process of moving between different screens or views within a mobile app. In React Native, navigation is crucial for creating a seamless user experience and allowing users to navigate between different parts of the app. Proper navigation implementation ensures that users can easily find and access the content or features they need, improving engagement and usability.

**Question 4: Explain the difference between stack navigation and tab navigation in React Native.**

Answer:

Stack Navigation: Stack navigation manages navigation using a stack data structure, where each screen is pushed onto the stack when navigated to and popped off the stack when navigated away from. It is commonly used for hierarchical navigation flows, such as navigating between different levels of a nested menu.

Tab Navigation: Tab navigation displays multiple screens as tabs at the bottom or top of the screen, allowing users to switch between them by tapping on the tabs. It is suitable for presenting different sections or categories of content within the app and provides a consistent way to access commonly used features.

**EXPERIMENT-7**

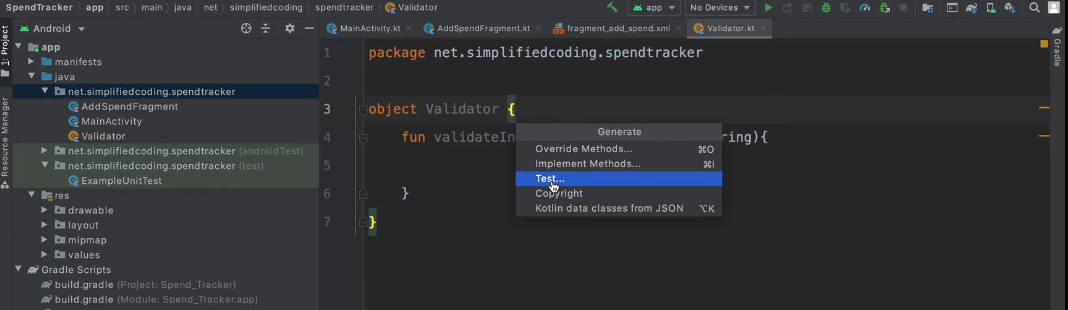
**AIM:**- Testing and Debugging

● Perform unit testing on key functionalities of an Android app using JUnit and Android Testing frameworks.

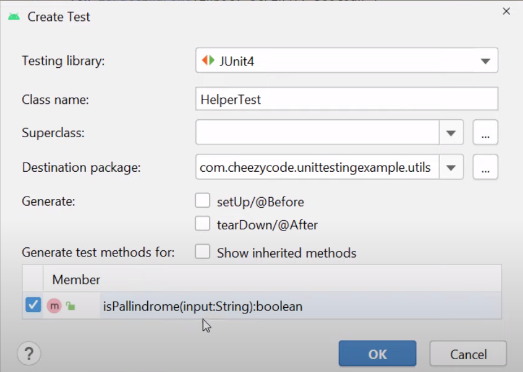
**PROCEDURE:**-

Unit testing is an essential aspect of Android app development to ensure that individual components of the app work as expected. Here's a basic guide on how you can perform unit testing on key functionalities of an Android app using JUnit and Android Testing frameworks in Kotlin:

**Step-1)** **Setting Up Development Environment:** Make sure you have the necessary dependencies in your build.gradle file for testing, including JUnit and AndroidX testing libraries.



**Step-2)** **Write Test Cases :** Identify the key functionalities or methods in your Android app that you want to test.Create test classes in the src/test directory for unit tests and src/androidTest for Android tests. Write test methods using JUnit framework to test individual functionalities.



**Step-3)** **Example :** Let's say you have a simple calculator app and you want to test the addition functionality.

import org.junit.Test

import org.junit.Assert.assertEquals

class CalculatorTest {

@Test

fun testAddition() {

val calculator = Calculator()

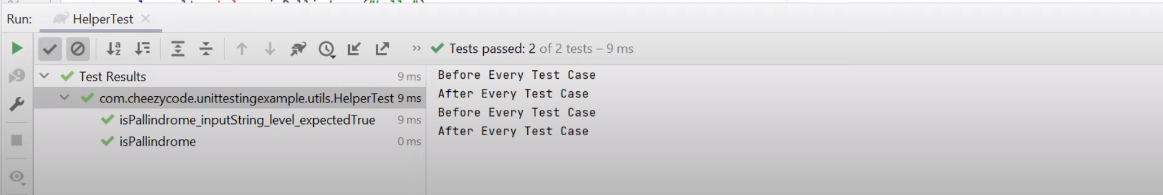
val result = calculator.add(2, 3)

assertEquals(5, result)

}

}

**Step-4)** **Run Tests:** You can run the tests either through Android Studio's built-in test runner or via command line using Gradle commands.

In Android Studio, right-click on the test class or method and select "Run" or "Debug".

**Step-5)** **Analyze Results:** After running the tests, check the test results in the test runner window. Ensure that all tests pass. If any test fails, debug and fix the issues in your code.

**Step-6)** **Mocking (Optional):** If your app interacts with external dependencies such as APIs or databases, you may need to mock these dependencies in your tests using frameworks like Mockito.

**Step-7)** **Testing UI (Android Tests):** For testing UI components, you can use Espresso or other UI testing frameworks provided by AndroidX. Write test cases to simulate user interactions and verify the expected behavior of UI components.

**Step-8)** **Continuous Integration:** Integrate unit tests into your continuous integration pipeline to ensure that tests are run automatically with each code change

**VIVA VOICE QUESTION**

**Question 1: What is the importance of testing and debugging in Android app development?**

Answer: Testing and debugging are critical processes in Android app development to ensure that the app functions correctly, meets user requirements, and delivers a high-quality user experience. Testing helps identify and fix bugs, improve app performance, and validate that new features or changes work as expected before releasing the app to users.

**Question 2: What is JUnit, and how is it used for unit testing in Android app development?**

Answer: JUnit is a popular open-source unit testing framework for Java applications. In Android app development, JUnit is commonly used to write and execute unit tests for testing individual Java classes and methods. Developers can write test cases using JUnit's assertion methods to verify the expected behavior of the code under test and run the tests using tools like Android Studio's built-in test runner or Gradle.

**Question 3: What are some common types of tests performed on Android apps using JUnit and Android Testing frameworks?**

Answer:

* Unit Tests: Test individual units or components of the app in isolation, such as methods or classes, using JUnit.
* Integration Tests: Test the interaction between multiple components or modules of the app to ensure they work together correctly.
* UI Tests: Test the app's user interface and interaction flows to validate that the UI behaves as expected and responds to user input correctly.
* Instrumentation Tests: Test the app's behavior on a device or emulator, including interactions with the Android framework and system components, using Android Testing frameworks like Espresso or UI Automator.

**Question 4:How do you write a unit test for a method or function in an Android app using JUnit?**

Answer:

* Define a test class for the target class or component, annotated with @RunWith(JUnit4.class) or @ExtendWith(MockitoExtension.class).
* Write test methods annotated with @Test to define individual test cases, each testing a specific behavior or scenario.
* Use JUnit's assertion methods like assertEquals, assertTrue, or assertNotNull to validate the expected behavior of the code under test.
* Use setup and teardown methods annotated with @Before and @After to set up preconditions and clean up resources before and after each test method.

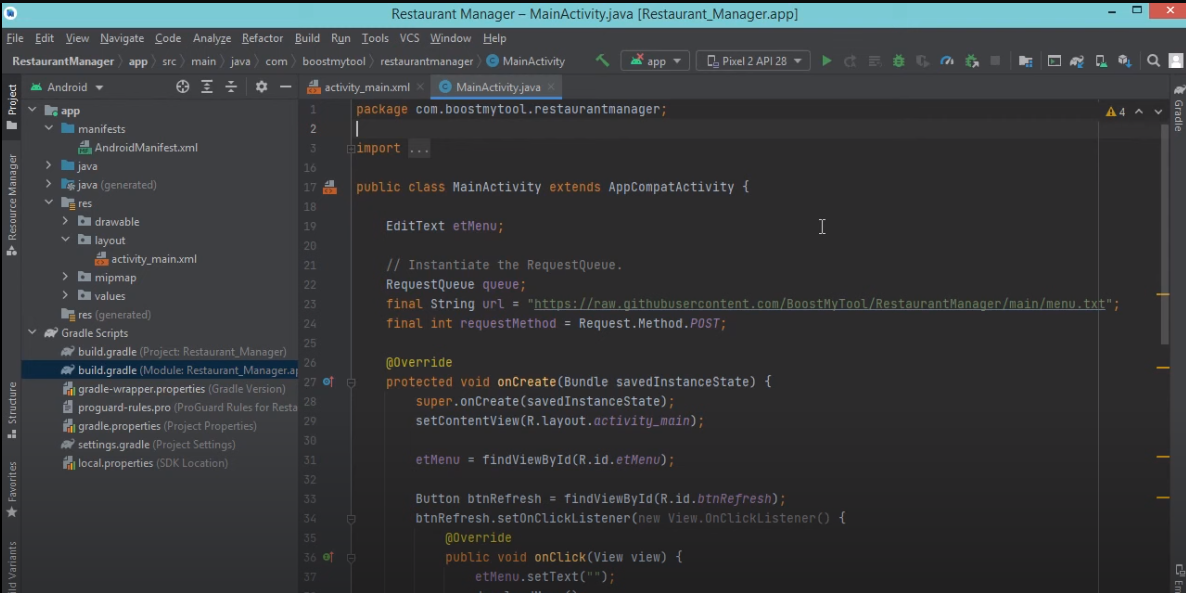
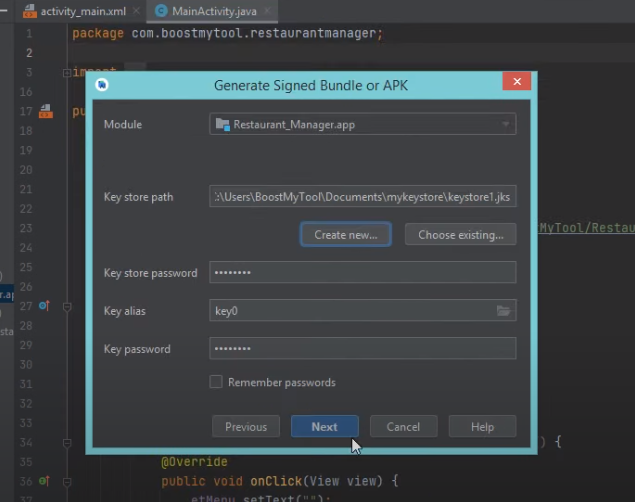
**EXPERIMENT- 8**

**AIM:**- – App Deployment and Performance Optimization

● Package and deploy an Android app to Google Play Store.

**PROCEDURE:-**

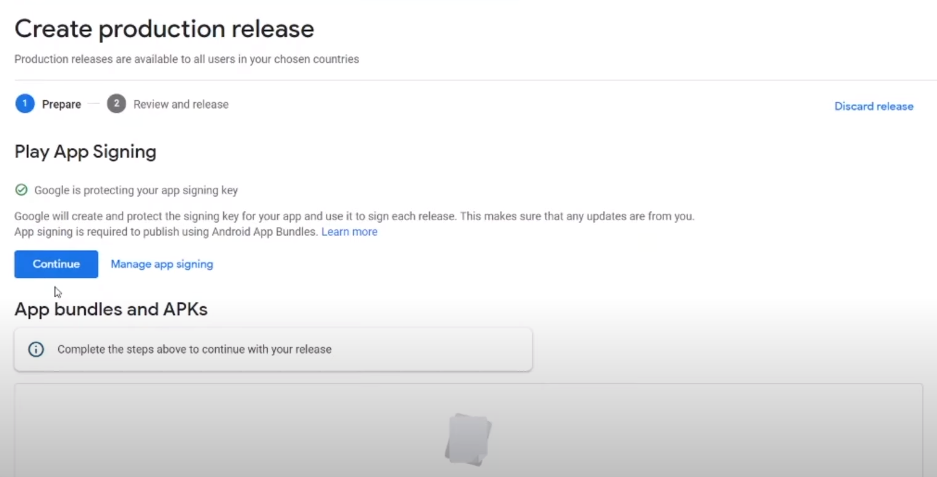
Deploying an Android app to the Google Play Store involves several steps, including preparing your app for release, generating a signed APK, creating a Google Play Developer account, and finally uploading your app to the Play Console.

**Step-1)** **Prepare Your App for Release:** Ensure your app is thoroughly tested, optimized for performance (minimizing resource usage, optimizing UI responsiveness, and handling memory efficiently), updated with final assets, and compliant with Google Play's policies and guidelines.

**Step-2)** **Generate a Signed APK:** In Android Studio, navigate to Build > Generate Signed Bundle / APK, select "APK," choose or create a keystore file, provide necessary information (key alias, passwords), select the build type (typically "release"), and finally click "Finish" to generate the signed APK.

Top of Form

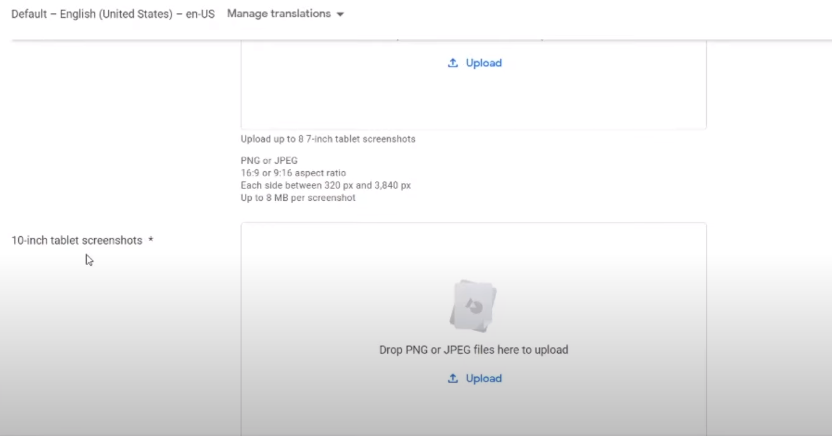
**Step-3)** **Create a Google Play Developer Account:** Visit the Google Play Console website (<https://play.google.com/console>), sign in with your Google account, create a developer account if needed, and complete the one-time registration fee payment (current fee as of my last update: $25).

**Step-4)** **Set Up Your App Listing:** In the Google Play Console, select "Create Application," input your app details (title, description, screenshots, categorization), upload the icon and feature graphic, and furnish additional information as requested, like content rating and pricing.

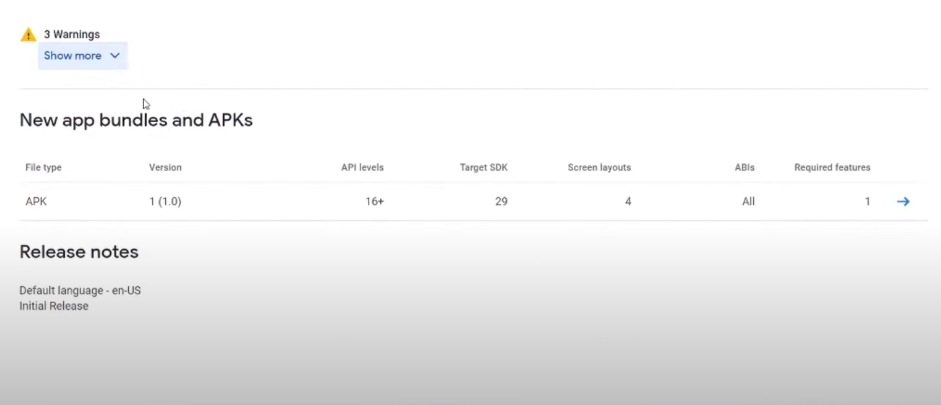
**Step-5)** **Upload Your APK:** In the Google Play Console, go to "App releases" > "Production," then click "Create Release" and follow instructions to upload your signed APK. Utilize features like staged rollouts or app bundles for optimized delivery if desired.



**Step-6)** **Prepare Store Listing and Content Rating:** Complete the store listing details, including the app's description, screenshots, and promotional graphics. Additionally, fill out the content rating questionnaire based on your app's content.



**Step-7)** **Review and Publish Your App:** Review all app listing details for accuracy and compliance with Google Play policies. Then, click "Review" and "Start Rollout to Production" to submit your app for review. Google will assess your app against their guidelines, a process taking from a few hours to several days. Upon approval, your app will be published to the Google Play Store, accessible for users to find and install.Top of Form



**VIVA VOICE QUESTION**

**Question 1: What is the significance of app deployment in the context of Android app development?**

Answer: App deployment refers to the process of preparing and distributing an Android app for installation and use by end-users. It is a crucial step in the app development lifecycle as it allows developers to make their apps available to a wider audience through distribution channels like app stores. Deploying an app to the Google Play Store enables users to discover, download, and install the app on their devices.

**Question 2: What is the purpose of signing an APK file, and why is it necessary for app deployment?**

Answer: Signing an APK file involves digitally signing the app's package with a cryptographic key to ensure its authenticity and integrity. It is necessary for app deployment to verify that the APK has not been tampered with or modified since it was signed by the developer. Signing the APK also allows users and devices to trust the app's source and origin, enabling secure installation and updates from the Google Play Store.

**Question 3: What are some common performance metrics and benchmarks used to evaluate Android app performance?**

Answer:

* App Launch Time: The time taken for the app to launch and become responsive after the user taps its icon.
* UI Responsiveness: The smoothness and responsiveness of the app's user interface, measured in terms of frame rate and touch response time.
* Memory Usage: The amount of memory consumed by the app during normal operation, including heap memory, native memory, and other resources.
* Battery Consumption: The impact of the app on device battery life, measured by monitoring CPU usage, network activity, and background processes.
* Network Performance: The speed and reliability of network requests and data transfer within the app, including latency, throughput, and error rates.

**Question 4: How do you monitor and analyze app performance after deployment to the Google Play Store?**

Answer:

* Use Google Play Console's built-in performance monitoring tools to track key performance metrics, including app startup time, crash rate, and ANR (Application Not Responding) rate.
* Collect and analyze user feedback and reviews to identify performance issues and prioritize improvements.
* Monitor app analytics and usage metrics using tools like Firebase Analytics or Google Analytics to gain insights into user behavior, engagement, and satisfaction.
* Use crash reporting and error tracking tools like Firebase Crashlytics to identify and diagnose performance-related crashes and exceptions in the app.