

|  |
| --- |
| **BATCH AND ROLL NO: R6 - 42257** |
| **EXPERIMENT NO.9** |
| **TITLE:** Design a mobile app to store data using internal or external storage. |
| **DATE OF PERFORMANCE** |
| **DATE OF CHECKING** |

**Title:** Design a mobile app to store data using internal or external storage.

**Requirements:**

1 Android studio

**Theory:**

Designing a mobile app to store data using internal or external storage involves creating an application that allows users to store and manage data on their mobile devices. Internal storage refers to the built-in storage on a device, while external storage can include removable SD cards or cloud-based storage solutions. To implement storage functionality in a mobile app, developers typically use platform-specific APIs and libraries, such as Android's SharedPreferences, SQLite database, or File API, or iOS's CoreData or File Manager API. These APIs provide a way to read, write, and manage data stored on the device or in the cloud.

When designing the user interface for a data storage app, developers need to consider how users will interact with the app and manage their data. This can involve creating screens and views that allow users to view, edit, and delete data.

Here are the general steps to design a mobile app to store data using internal or external storage:

1. Define the requirements for your app, including the types of data to be stored, the app's user interface, and any functionality or features that are needed.
2. Choose a development environment, such as Android Studio or Xcode, and create a new project.
3. Determine the appropriate storage method for your app, such as internal storage, external storage, or a cloud-based storage solution.
4. Implement the storage functionality using the appropriate API, such as Android's SharedPreferences, SQLite database, or File API, or iOS's CoreData or File Manager API.
5. Design the user interface for your app, including any screens or views needed to display and manage the stored data.
6. Implement the UI using the appropriate UI components, such as TextViews, EditTexts, and RecyclerViews, and link the UI to the storage functionality using appropriate callbacks or data bindings.
7. Test the app on a physical device or emulator, and make any necessary adjustments to the UI or functionality.



1. Optimize the app's performance and stability, using tools such as Android Profiler or Xcode's Instruments.
2. Add any additional features or functionality as needed, based on user feedback or evolving requirements.
3. Publish the app to the Google Play Store or Apple App Store, following the appropriate guidelines and requirements.

**Code:**

**import android.content.Context; import android.os.Bundle; import android.os.Environment; import android.view.View; import android.widget.Button;**

**import androidx.appcompat.app.AppCompatActivity; import java.io.File;**

**import java.io.FileOutputStream; import java.io.IOException;**

**import java.text.SimpleDateFormat; import java.util.Date;**

**import java.util.Locale;**

**public class MainActivity extends AppCompatActivity { private Button mSaveButton;**

**@Override**

**protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_main);**

**mSaveButton = findViewById(R.id.save\_button); mSaveButton.setOnClickListener(new View.OnClickListener() {**

**@Override**

**public void onClick(View view) { saveDateToFile();**

**}**

**});**

**}**

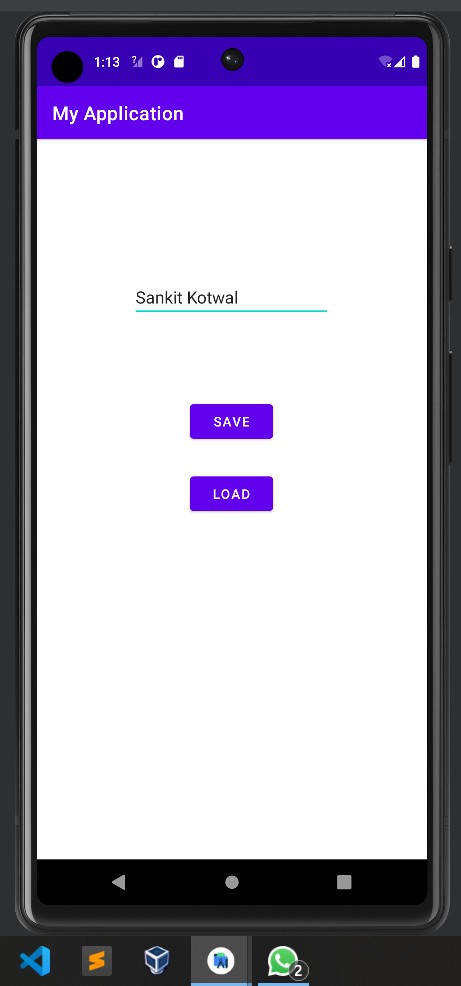
**private void saveDateToFile() {**

**String currentDate = new SimpleDateFormat("dd-MM-yyyy", Locale.getDefault()).format(new Date());**

**String fileName = "date.txt"; String fileContents = currentDate;**

**try {**

**// Get the internal storage directory File fileDirectory = getFilesDir();**



**// Create a new file**

**File file = new File(fileDirectory, fileName);**

**// Write the file contents to the file**

**FileOutputStream outputStream = new FileOutputStream(file); outputStream.write(fileContents.getBytes()); outputStream.close();**

**// Show a toast message indicating that the file was saved successfully Toast.makeText(this, "Date saved to file", Toast.LENGTH\_SHORT).show();**

**} catch (IOException e) { e.printStackTrace();**

**}**

**}**

**}**

**Output:**



**CONCLUSION:**

**……………………………………………………………………………………………**

**……………………………………………………………………………………………**

**……………………………………………………………………………………………**