MOBILE APPLICATION DEVELOPMENT

MODEL PAPER 2

SECTION - A

1. What is Android Emulator?

Android emulators are virtual devices that simulate the behaviour of real Android devices for testing and debugging apps they allow developers to test and debug applications on various device configurations without needing physical devices they provide a virtual testing environmental ensure app compatibility performance and functionality across different device configurations.

2. Define a Fragment in Android.

Fragments in Android are modular and reusable UI components that represent a portion of a user interface or behaviour with an activity in in simple terms fragments are like mini activities that can be embedded within an activity they were introduced to address the challenges of creating flexible and responsive user interfaces for different screen sizes and orientations fragments are used to build dynamic and multiplayer layouts that can adapt to various device configurations it is especially used for applications that run on a variety of screen sizes such as Smartphones and Tablets.

3. What is a ViewGroup in Android?

Wave group is a special type of view that can contain other views including other view groups and define the layout properties wave group extends the functionality of waves by providing the way to arrange and control the positioning of multiple child waves within a single container

4. What is a RecyclerView in Android?

Recycler view is a flexible and efficient component introduced in Android as a more advanced and versatile version of the list wave and gridview it is used to display large sets of data in a

scrollable layout the recycler view is designed to handle long list of data efficiently with features like we recycling and view holder to improve performance and memory usage

5. What is Content Resolver?

The content resolve in Android provides and abstract layer for accessing content providers the content resolve class in Android serves as the primary interface for interacting with content providers it's main purpose is to facilitate communication between Android applications and content providers

6. What is SQLite?

Sq Lite is a lightweight serverless self contained and open source relational database management system that is embedded within the application it is widely used in mobile and embedded devices due to its simplicity speed reliability and small food print sqlite data bases are stored as a single file on the disc making the easy to manage and deploy

Section-B

7. Explain Key Mobile Application Services.

1. Push Notifications: Push notifications are essential for engaging users and delivering timely information. They allow apps to send messages to users even when the app is not actively in

use Example: Push notifications are extensively used by social media platforms like Facebook and Instagram. These apps send notifications to users about new messages, likes, comments, and

other interactions to ensure users are engaged and informed.

2. In-App Messaging: In-app messaging enables users to communicate within the app to enhance engagement and interaction among users

Example: Messaging apps like WhatsApp and Messenger provide in app messaging services that allow wers to send text, voice, image, and video messages to their contacts. It creates a seamless communication experience within the app

3. Analytics and Tracking: Analytics services help developers track user behavior, app usage. and performance metrics to optimize the app's performance and user experience.

Example: Google Analytics for Firebase is a powerful tool that tracks user engagement retention, demographics, and in-app events Developers use this data to gain insights into user behavior and make data-driven decisions to improve then apps

4. User Authentication User authentication services verify user identities to provide secure access to app features and data.

Example: Google Sign In and Facebook Login are popular authentication services that allow users to sign in to various apps using their Google or Facebook credentials. It simplifies the login process and enhancing security

5. Payment Processing: Payment processing services enable secure in-app purchases. subscriptions, and financial transactions

Example: Apple Pay and Google Pay are widely used payment processing services that allow users to make purchases within apps securely and conveniently, enhancing the overall user experience

6. Cloud Storage: Cloud storage services enable apps to store data on remote servers to facilitate data synchronization and backup across multiple devices.

Example: Google Drive and iCloud offer cloud storage solutions that allow users to store and sync documents, photos and other files across their devices to ensure data accessibility and backup

7. Location Services: Location services empower apps to provide location-based functionalities such as navigation, geotagging, and location-based notifications

Example: Apps like Google Maps and Uber utilize location services to offer navigation, real time location tracking, and personalized services based on the user's location to enhance user experience and convenience

8. Social Media Integration Social media integration services enable apps to connect with social platforms, allowing users to share content and interact with their social networks seamlessly

Example: Social media apps like Instagram and TikTok integrate with platforms like Facebook and Twitter, enabling users to share photos, videos, and updates directly from the app to their social media accounts, enhancing user engagement and reach

9. Cloud Messaging: Cloud messaging services facilitate sending messages and notifications to users across different devices and platforms.

Example: Firebase Cloud Messaging (FCM) is a robust cloud messaging service that enables developers to send notifications and messages to users on iOS, Android, and web applications, ensuring effective communication and engagement:

10. Data Synchronization: Data synchronization services ensure that user data remains consistent and up-to-date across multiple devices and platforms

Example: Apps like Evernote and Microsoft OneNote utilize data synchronization to keep

notes and documents updated across smartphones, tablets, and computers, allowing users to access their data seamlessly across devices

11. Voice Recognition: Voice recognition services allow apps to convert spoken language into text, enabling voice commands and voice-activated features

Example: Siri (Apple) and Google Assistant use voice recognition to perform tasks and provide information based on user voice commands

12. Offline Access: Offline access services enable users to access certain app features and content without an active internet connection, enhancing usability in low-connectivity environments

Example: Apps like Google Maps allow users to download maps for offline use enabling navigation and location based services even when users are offline or have limited connectivity, ensuring uninterrupted access to essential features

13. Personalization and Customization: Personalization and customization services tailor the

app experience to individual user preferences, providing a personalized and engaging user experience

Example: Music streaming apps like Spotify use personalization algorithms to recommend music based on user listening habits, preferences, and history, creating a customized music experience for each user and enhancing user engagement.

14 Augmented Reality (AR) Integration AR integration services enable apps to overlay digital content and interactive experiences onto the real world, enhancing user engagement and interactivity

Example: Apps like Pokémon GO leverage All technology to superimpose virtual creatures and gameplay elements onto the real world environment, creating an immersive and interactive gaming experience for users

15. Offline Data Sync: Offline data syne services synchronize app data with backend servers when the device to back online, ensuring data consistency and seamless user experience across devices.

Example: Apps like Microsoft OneDrive use offline data sync to automatically update files and documents stored on the cloud when the device reconnects to the internet, allowing users to access the latest data seamlessly acruss devices.

16. Biometric Authentication: Biometric authentication services utilize unique biological characteristics such as fingerprints, facial recognition, or iris scans to verify user identities securely and conveniently

Example: Apps like banking applications and mobile wallets integrate biometric authentication features such as Touch ID (fingerprint recognition) or Face ID (facial recognition) on mobile devices to provide users with a secure and seamless login experience, enhancing security and uter convenience

17. App Distribution and Deployment: App distribution and deployment services provide platforms for distributing apps to users, managing updates, and tracking app performance.

Example: Apple App Store and Google Play Store are the primary platforms for distributing 105 and Android apps, respectively

8. How to Use ProgressBar in Android?

9. Explain TimePicker along with its attributes and features. Give an example

"TimePicker' is a Ut component that allows users to select a specific time. It can be displayed in either 12 hour or 24-hour formats.

Example Scenario: Setting an alarm, scheduling a meeting, or selecting a reminder time

Features of Time Picker

1. Time Selection: Allows users to select a specific time of day

2. 12-Hour and 24-Hour Formats: Can be configured to display time in either 12-hour (AM/PM) or 24-hour format using the is24Hour View attribute

3. OnTimeChangedListener: Allows developers to handle time changes and perform action when the selected time changes.

Code

itimPicker

android:id="@+id/timePicker"

android:layout width="wrap_content"

android:layout_height="wrap content

android:timePickerMode"clock" />

Explanation

This Tuneficker element is defined with a unique identifier+id/timelicker allowing it to be referenced in corresponding Java or Kotlin code. The layout, width and layout bright attributes are set to wrap content. meaning the Time Ficker will resize to fit its content. The android timePicker Mode="clock" attribute specifies tut the Time Picker should use a clock interface for time selection, providing a visually intuitive way for users be gick a time

Using Time Picker in Activity Code

Code

Java

TimePicker timePicker findViewById(R.id.timePicker);

int tour timePicker.getHour();

int minute timePicker, getMinute(),

Explanation

Tha rade snippet demonstrates how to interact with a TimePicker' view in an Android activity. The Time Picker first located in the layout using 'findViewById(R.id.timePicker) and assigned to the variable timePicker The currently selected hour and minute are then retrieved from the TimePicker' using the 'getHout() and petMinute() methods, respectively, and stored in the integer variables 'hour' and 'minute. This allows the application to access and use the selected time for various purposes.

10. How can a custom theme be created and applied to an activity in Android?

Step 2: Applying a Custom Theme:

Designing User Interface (Sudoku Game App)

If a predefined theme doesn't meet the requirements, a custom theme can be created by defining styles in 'res/values/styles.xml

Define a Custom Theme:

Code

'res/values/styles zml

<style name "CustomDialogTheme" parent "Theme AppCompat Dialog

cresources>

<ites name="android:windowBackground">@color/background</item

<item name="android:textColor">@color/text color</item>

<!-- Add more style attributes as needed->

</style>

</resources>

Explanation

CustomDialog Theme. A custom theme that extends an existing theme (Theme AppCompat Dialog) Additional style attributes can be defined to customize the look and feet of the dialog

Apply the Custom Theme:

Update the About activity declaration to include the custom theme in the AndroidManifest

mi

Code

AndroidManifest.xml

activity android:name" About

android:label="@string/about title"

android:theme="@style/CustomDialog Theme">

/activity>

Explanation

android:theme: Applies the custom theme to the activity The @style/CustomDialog Theme reference points to the custom theme defined in styles.xml

Output

The About bes will now appear as a dialog, making it visually distinct from the rest of the applicanon Thes myle change helps in emphasizing the informational nature of the About box and makes it louk

integrated with the overall app design ensers click the About button, the About activity is displayed as a dialog This provides a clear fred presentation of the Infor thation about Sudoku, with the ability to scroll through the test if The dining can be dismissed by pressing the Rack button, returning the user to the main screen.

11. Explain the basic steps to send an SMS using "SmsManager' in an Android application.

1 Declare Necessary Permissions:

Ensure that the SEND SMS permission is declared in the AndroidManifest.xml file to send SMS messages

Additionally, declare the 'RECEIVE SMS' permission if the application needs to receive and process incoming SMS messages

2. Design a Simple User Interface (UI):

Create a user interface that allows users to input the phone number and message they want to send via SMS.

Include appropriate input fields and a button to trigger the SMS sending process

1. Implement Runtime Permission Checks:

Check for and request the SEND SMS' permission at runtime on devices running Android 6.0 (API level 23) and above.
Handle permission responses to ensure that the application has the necessary permissions to send SMS messages.

4. Use SmsManager to Send the SMS:

Obtain an instance of 'SmsManager' using 'SmsManager getDefault()

Use SmsManager methods like 'sendTextMessage()' to send the SMS with the specified phone number and message content.

Handle any exceptions that may occur during the SMS sending process.

5. Optional: Handle SMS Delivery Reports:

If needed, set up delivery intents when sending SMS messages to receive delivery reports indicating the status of message delivery

• Process the delivery reports to track the status of SMS message delivery.

12. Outline the basic steps involved in consuming web services using HTTP in an Android application.

1. Add Internet Permission: Ensure that the AndroidManifest.xml file includes the necessary permission to access the internet. Add the following permission within the "<manifest> tag cuses-permission android:name="android.permission. INTERNET" />

2. Make HTTP Requests:

Create an AsyncTask or use a background thread to perform network operations asynchronously and avoid blocking the main Ul thread.

Use 'HttpURLConnection to establish a connection to the web service endpoint and configure the request method (GET, POST, PUT, DELETE)

Set request headers, parameters, and the request body as needed.

Send the request to the server and receive the response

3. Handle Response:

When handling responses from web services in an Android application, developers need to consider various scenarios based on the format of the response data

Process the data returned in the response and update the Ul or perform any necessary operations based on the received data

Text Response: For plain text responses, assign the content to a TextView for display

Image Response: Decode image data and display it in an ImageView

XML Response: Parse XML data to extract relevant information from the response.

JSON Response: Parse JSON data to extract required fields efficiently

4. Update UI: Ensure that any UI updates based on the web service response are performed on the main Ul thread using methods like 'runOnUiThread()'

5. Error Handling:

Implement error handling mechanisms to manage network errors, timeouts, and exceptions that may occur during the HTTP request.

Display appropriate error messages or take corrective actions based on the type of error encountered.

Section-C

13. Explain the Steps in Publishing Android Application.

Step 1. Preparing An App for Release

Before publishing an app, several steps must be completed to ensure it is ready for release. Th

involves

1. Testing Ensure the app is thoroughly tested on various devices and screen sizes to confom It works correctly under different conditions 2. Optimization: Optimize the app for performance, reduce app size, and improve startup times

For example. Use ProGuard to shrink and obfuscate code. 3. Security: Verify that the app is secure and free from vulnerabilities that could compromise

user data 4. Compliance: Ensure the app complies with all Google Play Store policies and guidelines

5. Update Version Information: In the build gradle kts file, update the version code and version name. These values help track the app's versions and manage updates

android (

defaultconfig (versionCode 2 versioniane "1.1"

// Increment this for each release

// Update to reflect the new version

Version Code: An integer value that must be incremented with each release. It is used to determine whether one version is more recent than another.

Version Name: A string that represents the release version of the app. This is visible to users

6. Check Permissions Ensure all necessary permissions are included in the AndroidManifest. xml file. Permissions are essential for accessing device features and data

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

package com.example.myapp">

cuses permission android:name="android.permission. INTERNET"/>

<uses-permission android:name="android.permission.ACCESS_FINE LOCATION"/>

</manifest>

Common Permissions:

INTERNET: Needed for network access

ACCESS FINE LOCATION: Required for precise location data

READ EXTERNAL STORAGE and WRITE EXTERNAL STORAGE: Needed for accessing and modifying external storage

Ensure Permissions: Only request permissions that are necessary for the app's functionality to maintain user trust and comply with privacy regulations.

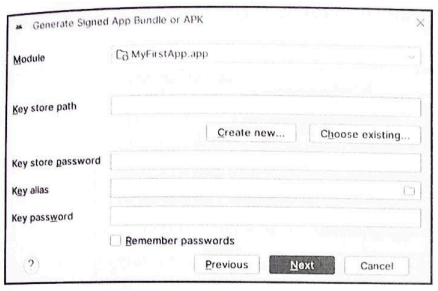
Step 2: Generating a Signed APK

To publish an app on the Google Play Store, we need to generate a signed APK (Android Package) file. This file is signed with a private key to ensure it comes from a trusted source.

To generate a signed APK, follow these steps:

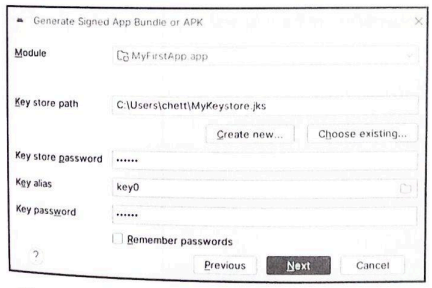
(a) In Android Studio, go to Build > Generate Signed Bundle / APK.

(b) Select "APK" and click "Next."

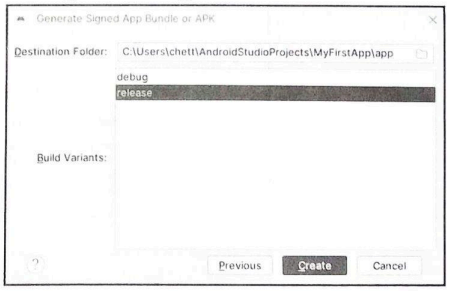


(c) If No Keystore Exists: Click Create new... and fill in the required details (location, passwords, alias).

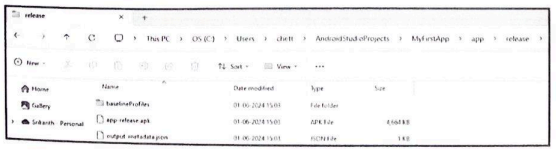
(d) Select keystore file, enter the keystore password, and the alias and click Next.



(e) Select the build type (e.g., release) and click "Finish." Android Studio will generate a signed APK file.



(f) The signed APK file will created as shown below.

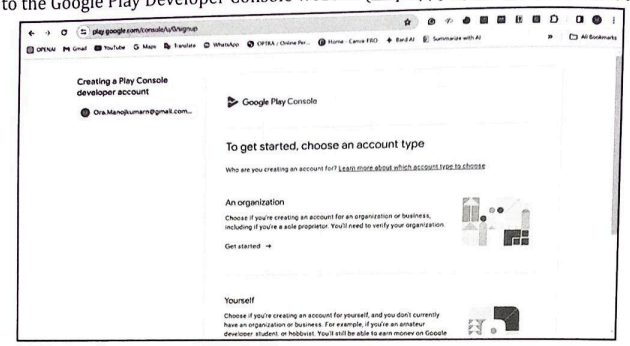


Step 3: Creating a Google Play Developer Account

Before publishing an app on the Google Play Store, we need to create a Google Play Developer accou
This requires a one-time registration fee.

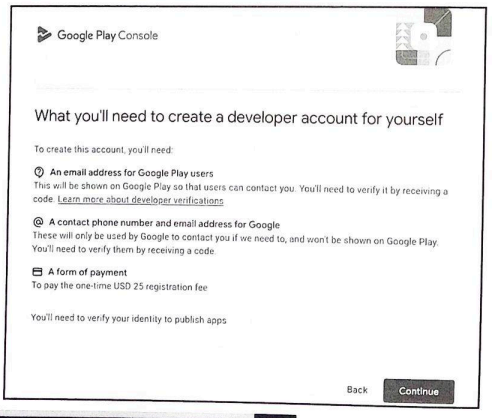
To create a Google Play Developer account, follow these steps:

(a) Go to the Google Play Developer Console website (https://play.google.com/apps/publish/)



(b) Sign in with a Google account or create a new one.

(c) Follow the on-screen instructions to complete the registration process and pay the registratio fee.



Step 4: Uploading an App to the Google Play Console

Once we have a signed APK file and a Google Play Developer account, we can upload an app to the Google Play Console. This is the platform where we manage app's listing on the Google Play Store.

To upload an app, follow these steps:

Log in to the Google Play Console.

Click on "Create Application and enter the details of an app, including the title.

description, and screenshots

Upload signed APK file and provide any additional information requested.

(d) Set the pricing and distribution options for an app

(e) Click "Save" and then "Publish to submit an app to the Google Play Store.

Pavithra.S

Once an app is published on the Google Play Store, we can promote it to attract users. Consider using app store optimization (ASO) techniques to improve app's visibility in the store We can also promote an app through social media, online advertising, and other channels to reach a wider audience.

14. a) Explain the Features and Functionalities of an Activity

1. User Interface (UI):

Activities are used to present a visual interface to the user, typically defined in XML layout files.

Developers can customize the Ul elements such as buttons, text fields, images, etc. ta create interactive screens

2. Lifecycle Management:

Activities have a well-defined lifecycle consisting of methods like "onCreate()', 'onStart 'onResume(), 'onPause(), 'onStop(), 'onDestroy(), etc.

Developers can override these lifecycle methods to perform specific actions at different stages of the activity lifecycle.

3. Navigation:

Activities can be started, paused, resumed, stopped, and destroyed based on user interactions and system events

Developers can navigate between activities using explicit intents to switch from one screen to another.

4. Intent Handling:

Activities can receive intents to perform actions like opening a new activity, sharing data or responding to system events

• Developers can extract data from intents to customize the behavior of the activity

5. Back Stack

Activities are managed in a back stack, allowing users to navigate back through the history of screens using the device's back button

Developers can control the back stack behavior by defining the parent activity or setting flags in the intent

6. Communication:

Activities can communicate with other components such as services, broadcast receivers, and content providers using intents

Developers can pass data between activities using intent extras or by starting activities for results.

7. Configuration Changes:

• Activities handle configuration changes like screen orientation, language, and keyboard availability by default.

Developers can manage configuration changes by overriding methods like 'onConfigurationChanged()' or handling them using resources

8. Result Handling:

Artivities can return results to the calling activity using the 'startActivityForResult() method

Developers can set a result code and data to be sent back to the calling activity when the child activity finishes

9. Permissions:

Activities can request permissions from the user to access sensitive device features like camera, location, contacts, etc

• Developers can handle permission requests and responses to ensure the app functions correctly and securely

10. Task Affinity:

Activities can have a task affinity that defines how they interact with other activities in the same task.

Developers can set task affinity in the manifest file to control the behavior of activities within the task

These features make activities a versatile component in Android development, allowing developers to create dynamic and interactive applications with multiple screens and user interactions.

b) How to Create an Intent? Explain with example.

The basic syntax for creating an Intent to start a new activity is:

Intent intent

new Intent(CurrentActivity.this, TargetActivity.class);

This line creates a new Intent object. The first parameter is the context of the current activity (CurrentActivity this), and the second parameter is the class of the activity to start (TargetActivity class)

Intent: This is a class in the Android framework used to define an action to be performed, such as starting another activity

CurrentActivity this: This parameter specifies the current context, usually the activity from which the intent is being created. It provides the necessary context for the intent to start the new activity

TargetActivity class: This parameter specifies the class of the activity that you want to navigate to. It tells the Android system which activity to launch.

15.a) Explain RecyclerView along with its attributes and features. Give an example

Key Concepts of RecyclerView

1. ViewHolder Pattern:

View ViewHolder pattern is used to minimize the number of calls to findViewByte The Holder holds references to the Ul components of a single item view, which makes teay to recycle views efficiently

2. Adapter:

The adapter in RecyclerView' binds the data to the item views. It creates new "ViewHolder objects as needed and binds data to them.

3. LayoutManager:

The "LayoutManager' is responsible for measuring and positioning item views within a 'RecyclerView". It determines how items are laid out, such as in a linear list or a grid

Key Features of RecyclerView

1. Efficient View Recycling:

RecyclerView efficiently recycles views that are no longer visible, reducing the number of views created and enhancing performance, especially when dealing with large data sets.

2. ViewHolder Pattern:

The View Holder pattern is used to minimize the number of calls to findViewByld. A ViewHolder

holds references to the Ul components of a single item view, making it easy to recycle views efficiently

3. Flexible Layout Management:

RecyclerView uses LayoutManager classes to manage the positioning and layout of items This allows for flexible layouts such as linear lists, grids, and staggered grids, enhancing the be adaptability of the user interface

4. Built-in Animations:

RecyclerView supports built in animations for adding and removing items, providing a smoother and more visually appealing user experience.

b) Explain DialogFragment along with its attributes and features. Give an example.

Key Features of DialogFragment

1. Encapsulation of Dialog:

DialogFragment encapsulates a dialog as its primary layout element. This encapsulation streamlines the process of displaying dialogs within a Fragment and promotes modular design and reusability of dialog-related functionality.

2. Lifecycle Management:

DialogFragment includes its own lifecycle methods, similar to those of an Activity or Fragment. These methods facilitate easier management of dialog-related operations, such as "onCreateDialog(), "onStart()', 'onResume(), 'onPause()', onStop(), and onDestroyView()

3. Built-in Dialog Handling:

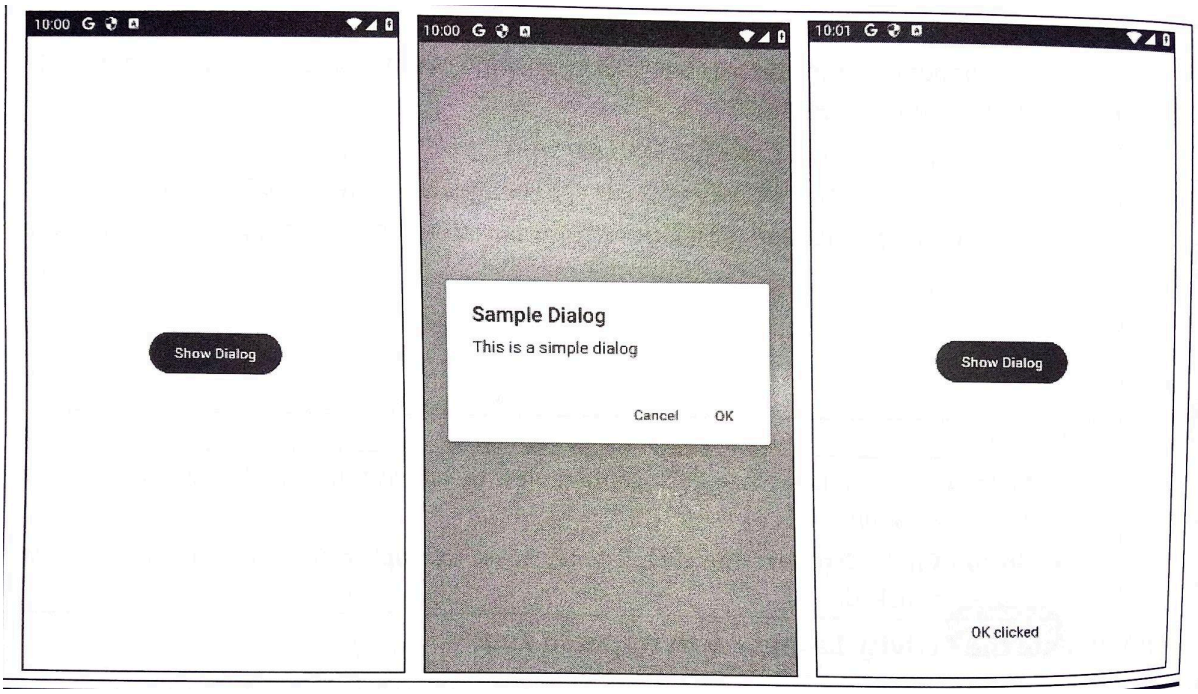
DialogFragment offers built in support for creating and managing dialogs, including custom dialogs. This feature simplifies the implementation of dialog interactions within Android applications.

4. Adapter Integration:

'DialogFragment' can integrate with adapters to bind data to the dialog. This integration simplifies the process of populating the dialog with data and updating it when the underlying dataset changes.

5. Modular and Reusable Components:

"DialogFragment' facilitates the creation of modular and reusable components that display dialogs within various parts of an application. By encapsulating dialog functionality within a Fragment, 'DialogFragment enhances code reusability and maintainability.



16.a) Explain Resizing and Repositioning Views to Adapt Display Orientation

Resizing and repositioning views is a technique used in Android development to dynamically a the size and position of UI elements based on the current screen orientation or available screen spac This ensures that the layout remains functional and aesthetically pleasing regardless of how the holds the device.

?f What is Resizing and Repositioning Views?

Retizing refers to changing the dimensions of a view (width and height) while repositioning inve changing its location on the screen. These adjustments are often necessary to make the best use of the available screen space and to ensure that all important elements are visible and accessible

How It Works?

Resizing and repositioning views typically involve using layout attributes that respond to changes

in screen size and orientation. Layout managers such as 'ConstraintLayout' or 'LinearLayour w

weights can be used to define flexible, responsive layouts. These layouts can adjust the dimensions

and positions of views dynamically The below example uses ConstraintLayout to demonstrate resizing and repositioning views. Tha example will adjust the size and position of an 'EditText' and a 'Button' to make better use of the available screen space in different orientations.

Code

Resizing and Repositioning Views

<androidx.constraintlayout.widget Constraint Layout

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

xmlns:app="http://schemas.android.com/apk/res-auto"

android:layout width="match_parent"

android:layout_height="match parent"

android:padding="15dp">

<EditText

android:id="@+id/username"

android:layout_width="0dp"

android:layout height="wrap content"

android: hint "Username

app: Layout constraintTop totopOf›"parent"

app:layout_constraintStart toStartOf="parent" app:layout constraintEnd to EndOf"parent"

android:layout marginTop="32dp"

android:layout marginBottom="16dp"/>

EditText

android:id="@+id/password"

android:layout_width="0dp"

android:layout height="wrap_content"

android:hint="Password"

android: Input Type="textPassword"

app layout constraintTop_toBottomOf="@id/username"

app:layout constraintStart_toStartOf="parent" app:layout constraint End_toEndOf="parent"

android:layout marginBottom="16dp"/>

Button

android:id="@+id/login_button

android:layout_width="0dp"

android: layout, height="wrap content"

android:text="Login"

app:layout constraintTop_toBottomOf›"@id/password"

app:layout_constraintStart toStartOf="parent"

app:layout_constraintend toEndOf="parent"/>

androidx.constraintlayout.widget.ConstraintLayout>

Explanation

1. ConstraintLayout:

Padding: Added padding to ensure content isn't flush with the screen edges

Width & Height: Set to match parent' to fill the entire screen.

2. EditText for Username:

Width: Set to Odp to allow dynamic resizing based on constraints

Height: Set to wrap content to fit the content

Constraints:

app layout constraintTop to TopOf="parent" Anchors the top of the "EditText' to the top of the parent

app layout constraintStart toStartOf="parent" and app layout constraintEnd toEndOf="parent" Ensures the 'EditTest stretches across the width of the parent.

Margins: Adds top and bottom margins for spacing.
3. EditText for Password:

Width: Set to Odp' to allow dynamic resizing based on constraints.

Height: Set to wrap content to fit the content.

Constraints

app layout constraint Top toBottomOf="@id/username" Anchors the top of the EditText to the bottom of the username EditText

app layout, constraintStart, toStartOf="parent" and app layout constraintEnd toEndOf"parent" Ensures the 'EditText" stretches across the width of the parent Margins: Adds a bottem margin for spacing

4. Button for Login:

Width: Set to "Odp' to allow dynamic resizing based on constraints.

Height: Set to wrap content' to fit the content.

Constraints:

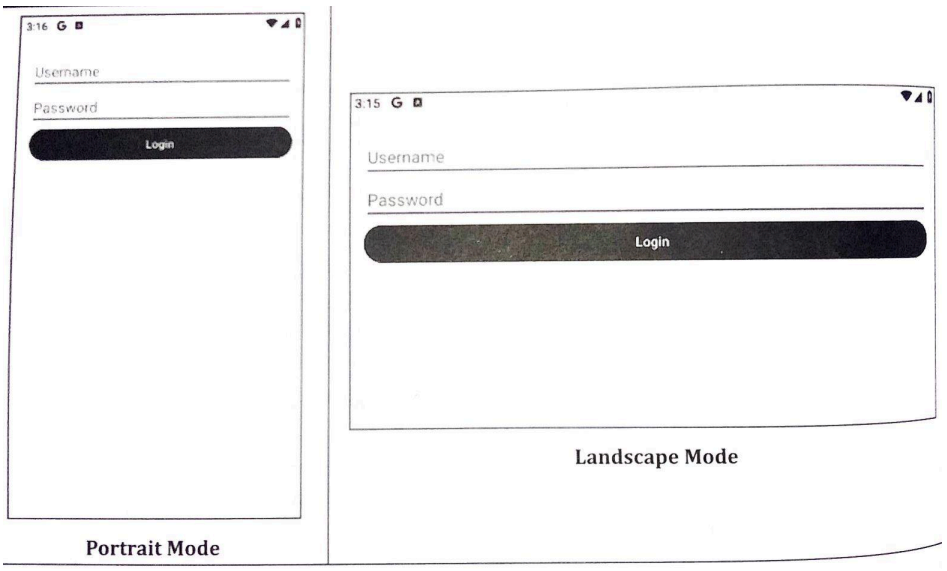
app-layout constraintTop toBottomOf="@id/password": Anchors the top of the "Tuttor to the bottom of the password 'EditText

app layout constraintStart_toStartOf="parent" and app:layout_constraintEnd toEndOf="parent" Ensures the "Button" stretches across the width of the parent.

5. How It Works:

Portrait Mode: The layout is vertically aligned, with the username and password fields and login button stretching across the width of the screen This ensures a clean, centered appearance

Landscape Mode: The same constraints ensure that the views stretch horizontally to use the available space efficiently. The vertical alignment is maintained to provide a consistent user experience.



b) Explain the process of creating a view programmatically in Android.

17. Describe the step-by-step implementation of data binding in an Android application

This application demonstrates live data binding in Android. It includes two EditText fields for the user's name and phone number. As the user types, the entered data is displayed in a ListView below.
Sar feat fields The Coal is to bind data directly from the Ul components to the View Model using

A's Data Binding Library

1 Creating the Project:

1. Open Android Studio,

2. Create a new project by selecting "Empty Views Activity" 3. Name the project "livedatabindingexample"

2. Enable Data Binding in Gradle

Enable Data Binding in the app's 'build gradle' file by adding the following

Code

Code

androld (

buildFeatures {

databinding

true

Esplanation

más Sync the project by clicking "Sync Now" at the top of the window.

: Define the Layout File:

Create and define the layout file 'activity main.xml" with EditText fields for name and phone number and a ListView for displaying the data.

Code

activity main.xml

chxml version="1.0" encoding="utf-8"?>

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

<data>

cvariable

name="viewModel"

type="com, example. livedatabindingexample.UserViewModel" />

</data>

<LinearLayout

android: layout width="match parent"

```
android:layout height="match_parent"

android:orientation="vertical"

android:padding="15dp">

<EditText

android:id="@+id/editTextName"
android:layout width="match_parent"

android:layout_height="wrap_content"

android: hint "Enter Name"

android:text="@{viewModel.name)" />

<EditText

android:id="@+id/editTextPhone"

android:layout width="match_parent"

android:layout_height="wrap_content"

android:hint="Enter Phone Number"

android:text="@(viewModel.phone)" />

<ListView
```

android:id="@+id/listView"

android:layout width="match_parent"

android:layout_height="0dp"

android:layout weight="1"

app:items="@{viewModel.userlist)" />

</LinearLayout>

</layout>

Explanation

1. Data Binding Setup:

The layout file begins with the <layout> tag which enables data binding for this layout. Inside this tag, the <data> element declares a variable named 'viewModel of type 'com example livedatabindingexample User ViewModel. This sets up a link between the UI and the

View Model

2. EditText for Name:

An EditText' component is defined for entering the name, with an 'android id of @+id/ editTextName' The text property is bound to the 'name' property of the ViewModel using {viewModel.name)': This two way binding ensures that any changes in the EditTest update the ViewModel and vice versa

3. EditText for Phone Number:

Similar to the name field, another 'EditText" component is defined for entering the phone number, with an android:id" of "@+id/editTextPhone its text property is hound to the 'phone' property of the ViewModel using '@[viewModel.phone)', ensuring two-way data synchronization.

4. ListView for Displaying Users:

A ListView' is defined to display a list of users. The app items' attribute is bound to the "userList' property of the ViewModel. This requires a custom BindingAdapter to convert the userlist' data into a format that the ListView can display, The ListView's layout attributes are set to make it expand to fill the remaining space in the layout.
sup 4 Create a Binding Adapter:

Putting SQL to Work

8:29

Greate a binding adapter to bind the list of users to the ListView.

Code

BindingAdapters java

package com.example.livedatabindingexample,

import android.widget. ArrayAdapter Import android.widget.ListView;

import androidx.databinding.BindingAdapter,

import java.util.List;

public class BindingAdapters { @BindingAdapter("items")

public static void setItems (ListView listView, List<String> items) {

ArrayAdapter<String> adapter = new ArrayAdapter<>(listView.getContext(), android.R.layout.simple_list_item 1, itens); listView.setAdapter(adapter);

Explanation

1. Purpose of @BindingAdapter":

The @BindingAdapter annotation is used to create custom binding logic in data binding. It allows the definition of how specific attributes should be set or modified when bound to data In this case, it is used to define how a 'ListView should be populated with a list of items.

2. Creating the 'setitems' Method:

The 'setitems method is a static method annotated with @BindingAdapter("items" This means that whenever the 'items' attribute is used in a layout file, this method will be called to handle the binding

3. Parameters:

The method takes two parameters:

"ListView list View The 'ListView instance where the data will be set

List<String> items: The list of strings to be displayed in the ListView

4. Setting Up the Adapter:

Inside the 'setitems' method, an ArrayAdapter<String> is created. The ArrayAdapter is a common adapter used to convert an array of strings into views (in this case, text views) that are displayed in the 'ListView

The ArrayAdapter is initialized with:

"listView.getContext() The context of the 'ListView'

android R.layout simple list item, 1 A predefined layout resource for a simple list item

(just a single TextView).
5. Binding the Adapter to the ListView:

Finally the setAdapter method of the 'ListView is called to set the ArrayAdapter created in the previous step. This bunds the list of strings to the ListView, causing the "LntView display the items

Step 5: Create the ViewModel

Create a ViewModel to manage the data for the UI.

Code

Uner View Model java

package com.example livedatabindingexample,

Import androidx.databinding BaseObservable;

```
import android.databinding Bindable;

import java.util.Arraylist;

import java.util.List;

public class UserViewModel extends BaseObservable (

private String name=""

private String phone";

private List<String> userlist new ArrayList<>();

@Bindable

public String getName() {

return name;

1

public void setName(String name) ( this name name, notifyPropertyChanged(BR.name);

updateUserlist();

@Bindable

public String get Phone() {

return phone;
```

public void setPhone(String phone) (this phone phone; notifyPropertyChanged(BR phone);
updateliserList(),
@Bindable

Putting SQL to Work

8.31

public List<String> getUserList() {

return userList;

private void updateUserList() {

userList.clear(),

if (Iname isEmpty() && !phone. IsEmpty()) {

userlist.add(name phone);

notifyPropertyChanged(BR.userlist);

Explanation

1. Extending laseObservable': The UserViewModel class extends BaseObservable which is a key component in the data binding framework. This extension allows the class to notify the data binding system about changes in its properties, enabling the UI to react to data changes dynamically and automatically

2. Observable Properties: The class defines observable properties 'name' phone, and user List These properties are annotated with Bindable, making them bindable to the Ul components The Bindable annotation signals the data binding library to generate a corresponding entry in the BR class, facilitating property change notifications

3. Updating 'userList': When the name or phone properties are updated the 'updatetiserList method is called. This method updates the 'userlist by clearing it and adding a new entry that combines name and phone, provided neither is empty This ensures that the list displayed in the Ul reflects the most recent data entered by the uter

4. Property Change Notifications: The notify PropertyChanged(BR.name) and notify Property Changed [BR. phone) methods are used to inform the data hinding system that the respective properties have changed. This triggers the data hinding library to update any UI components bound to these properties, ensuring the Ul stays in sync with the underlying data model without requiring explicit updates in the activity or fragment code

ep 6 Update MainActivity:

Update MainActivity' to set up data binding

Code

MainActivity.java

package com.example livedatabindingexample;

Import android.os.Bundle;

lsport androidx appcompat.app.AppCompatActivity; Ssport androidx,databinding DataBindingUt11;
import com.example. livedatabindingexample.databinding ActivityMainBindinga

public class MainActivity extends AppCompatActivity(@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

ActivityMainBinding binding DataBindingUtil.setContentView(this,

UserViewModel viewModel new UserViewModel();

R.layout.activity_main);

binding.setViewModel (viewModel);

binding.setLifecycleOwner(this);

)

Explanation

1. Initializing Data Binding and Setting the ViewModel:

In MainActivity, data binding is initialized with DataBindingUtil.setContentView(this R.layout.activity main): This replaces the traditional 'setContentView and returns a binding object that is tied to the layout. The UserViewModel is instantiated and set as the binding's view model with binding setViewModellviewModel),, linking the layout directly to the view model s data

2. Setting Up the Binding:

The method DataBindingUtil.setContentView not only inflates the layout but also initializes the binding object. This object allows direct access to the Ul components and enables data binding expressions in the XML layout to work. It effectively bridges the layout and the view model, allowing Ul components to automatically reflect changes in the view model's data

3. Making Binding Lifecycle-Aware:

By calling binding setLifecycleOwner(this);', the binding becomes aware of the lifecycle of the activity. This ensures that the data binding framework can handle lifecycle events (like activity pause, resume, or destroy) seamlessly It allows the binding to observe LiveData properties and update the Ul components accordingly, ensuring consistent and up-to-date Ul throughout the lifecycle changes.

Step 7: Clean and Rebuild the Project:

1. Clean the Project: Go to Build -> 'Clean Project

2. Rebuild the Project: Go to "Build' -> 'Rebuild Project'

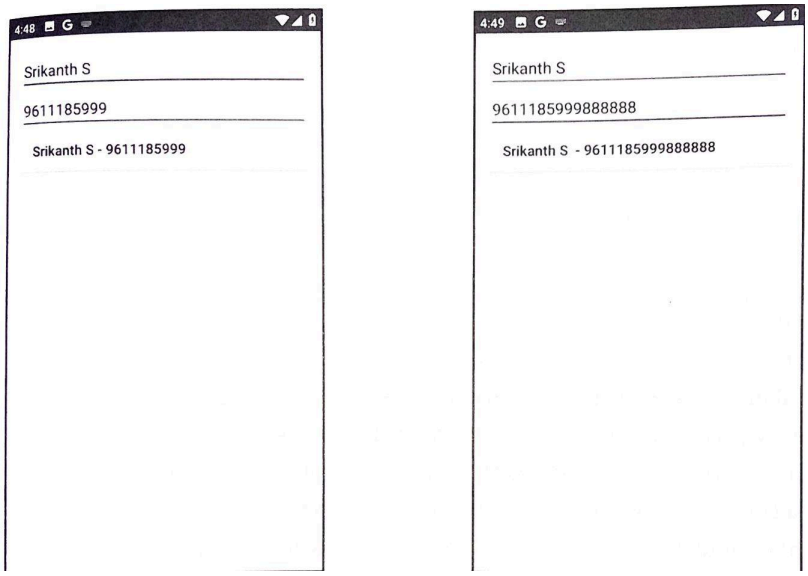
Output

When the application runs:

The user can enter a name and phone number in the EditText fields.

As • the user types, the name and phone number are displayed in the ListView below.

This demonstrates live data binding, where changes in the ViewModel automatically update the Ul.



18. Explain the Step-by-Step Implementation of Consuming JSON Web Services.

Step 1: Create a New Android Project:

Create an Android project by selecting "Empty Views Activity", naming the project "JSONDownloader", and choosing Java as the language

Step 2: Add Internet Permissions:

Ensure that the AndroidManifest.xml file includes the necessary permission to access the internet. Add the following permission within the <manifest> tag <uses-permission android:name="android.permission.INTERNET" />

Step 3: Design the Layout:

Define the user interface in activity main.xml This layout includes a TextView to display the

parsed JSON data and a Button to initiate the download

acstets malu.sm)

csml version="1.0" encoding="utf-8"?> RelativeLayout

xmlns:android="http://schemas.android.com/apk/res/android" seins
tools="http://schemas.android.com/tools"

android layout width="match_parent" android: layout height="match parent"

tools contexts MainActivity"s

<Button

android id="@+id/button_download android, layout width="wrap content"

android: layout height="wrap_content"

android:text="Download JSON

android:layout centerHorizontal="true"

android:layout marginTop="50dp" />

<TextView

android:id="@+id/text_view"

android:layout width="match_parent"

android: layout height="wrap_content"

android:layout below "@id/button_download"

android:layout_marginTop="20dp"

android:padding="16dp"

android:textSize-"16sp" />

</RelativeLayout>

To use the OpenWeatherMap API, sign up and get an API key

Step 4: Get an API Key:

1. Go to the [OpenWeather Map Sign Up Page](https://home.openweathermap.org/users/ sign up)

2. Create an account by providing an email and creating a password

3. Once the email is verified and logged in, navigate to the API keys section.

4. Generate a new API key and copy it for use in the application.

Step 5: Implement the MainActivity:

In MainActivityjava, implement the logic to download and parse the JSON data un

AsyncTask and display the relevant information in the "TextView

Code

MainActivity.java

package com.example.jsondownloader,

```
import android.os.AsyncTask,

Import android.os.Bundle;
import android.widget.Button,

import android.widget.TextView; import android.widget.Toast;

sport android appcompat.app.AppCompatActivity,

taport org.json 350Narray;

Japort org.json. 350NObject,

ssport java 10 BufferedReader,

import java 10. InputStreamReader;

import java.net.HttpURLConnection;

import java.net.URL;

public class MainActivity extends AppCompatActivity {

private TextView textView;

private Button buttonDownload;

@Override

protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);

textView findViewById(R.id.text_view),

buttonDownload findViewById(R.id.button download);

buttonDownload.setOnClickListener(v new Download350NTask()
execute("https://api.openweathermap.org/data/2.5/ weather?q-Bangalore&appid=YOUR_API
KEY));

private class Download]SONTask extends AsyncTask<String, Void, String> { @Override

protected String dolnBackground(String... urls) (

String urlString urls[0];

StringBuilder result new StringBuilder();

try {

URL url new URL(urlString), HttpURLConnection ur Connection (HttpURLConnection)

url.openConnection();

BufferedReader reader new BufferedReader(new
InputStreamReader(urlConnection.getInputStream())); String line, while
((linereader.readLine()) != null) ( result.append(line);
reader.close();
```

```
urlConnection disconnect();

)catch (Exception e) (

e.printStackTrace();

return "Error:e.getMessage();

return result.toString();

@Override

protected void onPostExecute(String result) {

if (result startsWith("Error:")) (

Toast.makeText(MainActivity, this, result,

Toast LENGTH SHORT).show();

} else {

try 1

JSONObject jsonObject new 350NObject(result),

JSONArray weatherArray jsonObject.getJSONArray("weather"),

JSONObject weather String description weatherArray.getJSONObject(0); weather
getString("description");
```

350NObject main jsonObject getJSONObject("main").

double temperaturekelvin main.getDouble("temp");

double temperatureCelsius temperaturekelvin- 273.15,

String cityNane jsonObject.getString("name"),

String displayText = "City:"cityName "In"

"Temperature: String.format("%.2f", temperature(elsist)

+""C\n" + "Weather: description:

TextView.setText(displayText);

) catch (Exception e) {

e.printStackTrace();

Toast.makeText(MainActivity, this, "Error parsing 3500 data" Toast.LENGTH SHORT).show();

. Setup and Layout:

The layout file (activity main.xml) contains a Button with the 10 battut download and TextView with the ID test view The Button is ued to trigger the dawnload process and the TestView displays the parsed JSON data

2. MainActivity

In the MainActivity class, references to the Button and TestView are obtened using findViewById()

An OnClickListener is set on the Button to start the AryncTack for enleading the ISON data frons a specified URL, when clicked

The URL https://aptopemewathermap.org/data/25/ weather ig-Bangalore&appad YOUR API KEY is used as an example. This URL return JSON data containing weather information for Bangalore

1. AsyncTask for Downloading and Parsing JSON:

The Download SONTask is an inner class that estends AsyncTask It performs network operations on a background thread to prevent blocking the Uf thread

dolnBackground():

This method runs on a background thread and takes a URL string as a parameter

It creates a URL object from the given URL string and opens an HttpURLConnection to connect to the URL

A BufferedReader is used to read the 150N data fres the URL's input stream line by line

The ISON data is appended to a StringBuilder

If an exception occors jeg network error). It is caught and an error message is returned

The downloaded 150N data is returned as a string

onPostExecute():

This method runs in the Ul thread after the background operation is completed.

it checks if the result string starts with "Error" If so, a toast message is shown to display the error

if no error occurred the JSON data is parsed using the SONObject and JSONArray classes to extract specific information feg. weather description, temperature in Celsius, city name)

The extrarted data is formatted into a string and set to the TextView to display it to the user

Output

When the app is run, pressing the "Download JSON" button will start the download and parsing process. Once the 15ON data is downloaded and parsed, the weather information for Bangalore Including the temperature in Celsius, will be displayed in the TextView If there is an error during the download or parsing process, an error message will be shown using a "Toast.