RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

SkyCast

MOBILE APPLICATION DEVELOPMENT

PROJECT REPORT

Submitted by

AADHITHYA K (221701001)

in partial fulfilment for the course

CD19606 Mobile Application Development

for the degree of

BACHELOR OF ENGINEERING in

COMPUTER SCIENCE AND DESIGN

RAJALAKSHMI ENGEERING COLLEGE

RAJALAKSHMI NAGAR

THANDALAM

CHENNAI-602 105

APRIL 2025

RAJALAKSHMI ENGINEERING COLLEGE

CHENNAI – 602105

BONAFIDE CERTIFICATE

Certified that this project report **"SkyCast"** is the bonafide work of **AADHITHYA K (221701001)** who carried out the project work for the subject CD19606 – Mobile Application Development under my Supervision.

SIGNATURE	**SIGNATURE**
Prof. S .UMA MAHESHWAR RAO	**Mr. R. VIJAYKUMAR**
Head of the Department	**Supervisor**
Professor and Head	Assistant Professor
Computer Science and Design	Computer Science and Design
Rajalakshmi Engineering College	Rajalakshmi Engineering College
Chennai-602105	Chennai-602105

Submitted to Project and Viva Voce Examination for the subject

CD19606-Mobile Application Development held on_____.

ABSTRACT

SkyCast is a global weather application designed to provide accurate and real-time weather updates for any location across the world. Leveraging the OpenWeather API, the app fetches live meteorological data including temperature, humidity, wind speed, atmospheric pressure, and weather conditions. With a focus on user experience, SkyCast features a clean, intuitive UI adorned with aesthetic icons and smooth navigation to ensure an engaging interaction.

SkyCast aims to make weather tracking simple and beautiful. Users can search for any city worldwide and instantly access detailed weather information. The app is ideal for travelers, commuters, or anyone needing quick and reliable weather updates. With scalability and future enhancements in mind, SkyCast is structured to integrate features like multi-day forecasts, radar maps, and weather alerts.

A key highlight of SkyCast is its minimalist and user-friendly interface, crafted with attention to design aesthetics and usability. The app features clean layouts, elegant icons, and intuitive navigation, offering users a seamless experience across various devices. The visual representation of weather conditions through animated icons and color-coded elements enhances the clarity and engagement of the app's interface.

ACKNOWLEDGEMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavour to put forth this report. Our sincere thanks to our Chairman **Mr. S. Meganathan, B.E, F.I.E.,** our Vice Chairman **Mr. Abhay Shankar Meganathan, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) Thangam Meganathan,Ph.D.,** for providing us with the requisite infrastructure and sincere endeavouring in educating us in their premier institution.

Our sincere thanks to **Dr. S. N. Murugesan, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to our **Prof. Uma Maheshwar Rao** Associate Professor and Head of the Department of Computer Science and Design for his guidance and encouragement throughout the project work. We convey our sincere thanks to our internal guide and Project Coordinator, **Mr. R. Vijay Kumar.,** Department of Computer Science and Design, Rajalakshmi Engineering College for his valuable guidance throughout the course of the project.

AADHITHYA K (221701001)

Table Of Contents

List Of Figures

CHAPTER 1

INTRODUCTION

In today's fast-paced world, having access to accurate and timely weather information is essential for planning daily activities, travel, events, and ensuring personal safety. Traditional weather apps often compromise either on data reliability or user experience. To bridge this gap, SkyCast was developed — a modern, global weather application that merges precision forecasting with a visually refined user interface.

SkyCast is designed to provide users with comprehensive weather updates for any location worldwide, powered by the trusted OpenWeather API. Whether it’s current temperature, humidity, wind speed, or atmospheric conditions, SkyCast delivers all essential weather metrics in real time. The app stands out with its clean, elegant UI, featuring beautifully designed icons and intuitive layouts that enhance user engagement and make weather data easy to interpret at a glance.

SkyCast caters to a broad range of users — from daily commuters and students to frequent travelers and outdoor enthusiasts — by offering location-aware forecasts, global search functionality, and a responsive, accessible design. The app is built with flexibility in mind, allowing room for future enhancements such as multi-day forecasts, weather animations, radar maps, and emergency alerts.

By combining reliable data, sleek design, and user-centric features, SkyCast reimagines the way people interact with weather applications, turning routine weather checks into a smooth, informative, and enjoyable experience.

CHAPTER 2

ANDROID STUDIO

2.1. Introduction to Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development, provided by Google. It is built on IntelliJ IDEA and provides powerful tools to design, code, test, and debug Android apps efficiently.

Key Features:

- Code Editor: Smart code completion, real-time error checking, and code refactoring.
- Layout Editor: Drag-and-drop interface builder for XML layouts.
- Android Virtual Device (AVD) Manager: Lets you run and test apps on emulated Android devices.
- Gradle Build System: Automates the build process and handles dependencies.
- Logcat & Profiler: Helps in debugging and performance tracking.

Project Structure:

- manifests/: Contains the AndroidManifest.xml (app configuration).
- java/: Holds your Kotlin or Java source code.
- res/ (resources): Layouts, drawables (images), strings, styles, etc.
- build.gradle files: Manage dependencies and configurations.
- .idea/ & gradle/: Internal settings and scripts for Android Studio.

2.2. Kotlin for Android Development

Kotlin is the preferred programming language for Android development, officially supported by Google. It is concise, expressive, null-safe, and interoperable with Java.

Kotlin Basics:

- Variables:
  o Immutable: val name = "John" o Mutable: var
  age = 25
- Functions: fun greet(name: String): String { return

"Hello, $name"

}

- Classes & Objects: class User(val name: String, var age: Int)

Activity Lifecycle in Kotlin:

Activities are screens in an app. The main entry point is the MainActivity.kt.

Key lifecycle methods:

- onCreate(): Called when the activity is created (UI setup done here) • onStart(), onResume(), onPause(), onStop(), onDestroy()

```
class MainActivity : AppCompatActivity() {     override fun
onCreate(savedInstanceState:Bundle?){
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
   }
}
```

Why Kotlin for Android:

- Reduces boilerplate code
- More readable and safer
- Backed by JetBrains and Google

2.3. User Interface Design in Android

In Android, the UI (User Interface) is built using XML layouts that define how the app looks and interacts with users.

Layout Types:

1. LinearLayout – Aligns views in a single row or column xml

```
<LinearLayout orientation="vertical">
   <TextView android:text="Welcome" />
```

<Button android:text="Click Me" />

</LinearLayout>

2. ConstraintLayout – Flexible layout for complex UIs using constraints

   (Recommended for modern apps)

3. RelativeLayout – Positions elements relative to each other

4. FrameLayout – Displays one view at a time (useful for fragments)
   Common UI Widgets:

   - TextView: Displays text
   - EditText: Accepts user input
   - Button: Triggers an action
   - ImageView: Shows an image
   - RecyclerView: Displays lists

Material Design:

- Follow Google's design guidelines for consistent UI/UX
- Use components like CardView, BottomNavigationView, FloatingActionButton

Styling the UI:

- colors.xml, styles.xml, themes.xml in res/values folder
- Use custom fonts, padding, and margin to enhance appearance

Example linking XML to Kotlin: val button = findViewById<Button>(R.id.myButton) button.setOnClickListener {

```
    Toast.makeText(this, "Button clicked!",
Toast.LENGTH_SHORT).show()
}
```

2.4. Intents and Activities

In Android, an Activity represents a single screen (like a page). You can use Intents to navigate between activities or perform actions like opening the camera or browser. Types of Intents:

- Explicit Intent – Start a specific component (Activity, Service) val intent = Intent(this, SecondActivity::class.java) startActivity(intent)
- Implicit Intent – Request an action (e.g., send SMS, open URL) val intent=Intent(Intent.ACTION_VIEW)intent.data=Uri.parse("https://www.go ogle.com") startActivity(intent)

Passing Data Between Activities: val intent = Intent(this,SecondActivity::class.java) intent.putExtra("username","John") startActivity(intent)

And retrieve it in SecondActivity:

val name = intent.getStringExtra("username")

2.5. Fragments and Navigation Components

A Fragment is a reusable UI component inside an Activity. It helps build flexible and modular UIs, especially on tablets or multi-pane screens.

Using a Fragment:

```
class HomeFragment : Fragment(R.layout.fragment_home) { override fun
onViewCreated(view:View,savedInstanceState:Bundle?){
super.onViewCreated(view, savedInstanceState)
        // Handle UI logic here
    }
}
```

Navigation Component (Recommended):

- Manages fragment transitions using a navigation graph
- Use NavHostFragment and NavController for safe and clean navigation Benefits:
- Cleaner back-stack handling
- Type-safe argument passing using SafeArgs
- Easier UI testing.

2.6. Networking and APIs

Networking is essential for connecting Android applications to web services, fetching data from remote servers, and enabling real-time communication. Android provides multiple tools and libraries for handling network operations.

2.6.1 HTTP Requests

HTTP requests are used to communicate with remote servers, fetch data, or send data from the Android application to the server.

Basic Steps for HTTP Requests:

1. Create a URL: The URL to the API or web service you want to interact with.
2. Open a connection: Use HttpURLConnection or a third-party library like Retrofit or Volley to open a connection to the server.
3. Handle responses: After the request is made, handle the response (e.g., data in JSON or XML format).

Common Methods for HTTP Requests:

- GET: Fetch data from the server.
- POST: Send data to the server.
- PUT: Update data on the server.
- DELETE: Remove data from the server.

2.6.2 Retrofit and Volley Both Retrofit and Volley are popular libraries that make network communication easier.

Retrofit:

Retrofit is a type-safe HTTP client for Android. It simplifies interacting with RESTful APIs by converting API responses into Java objects.

- Setup Retrofit:
  - Add Retrofit dependencies to the project's build.gradle file.

implementation 'com.squareup.retrofit2:retrofit:2.9.0' implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

  - Create a Retrofit interface for defining API endpoints.

```
public interface ApiService {
    @GET("users/{user}")
    Call<User> getUser(@Path("user") String user);
}
```

- Set up Retrofit instance and call API:

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.example.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build();
ApiService apiService = retrofit.create(ApiService.class);
Call<User> call = apiService.getUser("john_doe");
```

2.7. Multimedia

Multimedia in Android refers to handling audio, video, and images within an app. It involves interacting with media files, using the camera, and playing sounds or videos in your app.

2.7.1 Camera and Image Capture

Capturing images or videos is a common feature in many Android apps.

Android provides built-in support for accessing the device's camera.

Steps to Capture Images:

1. Declare Permissions:
    - In your AndroidManifest.xml, add the required permissions:

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

<uses-permission

android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

2. Open the Camera:

o Use Intent to launch the camera application or directly access the camera using the Camera2 API.

Intent(MediaStore.ACTION_IMAGE_CAPTURE);

startActivityForResult(cameraIntent, CAMERA_REQUEST);

3. Handle Captured Image:

o When the camera returns a result, handle it in the onActivityResult() method: @Override

protected void onActivityResult(int requestCode, int resultCode, Intent data)

{ super.onActivityResult(requestCode, resultCode, data); if (requestCode == CAMERA_REQUEST && resultCode ==

RESULT_OK) {

Bitmapphoto=(Bitmap)data.getExtras().get("data");

imageView.setImageBitmap(photo); // Show the captured image

}

}

Advanced: For higher control and custom settings, you can use Camera2 API for modern devices.

2.7.2 Audio & Video

Playing and recording audio and video are common features in Android applications.

Playing Audio:

- Use MediaPlayer to play audio from resources or external files.

MediaPlayer mediaPlayer = MediaPlayer.create(this,

R.raw.sample_audio); mediaPlayer.start();

Recording Audio:

- Use AudioRecord to record audio from the device's microphone.

AudioRecord recorder = new

AudioRecord(MediaRecorder.AudioSource.MIC, 44100,
AudioFormat.CHANNEL_IN_MONO,

AudioFormat.ENCODING_PCM_16BIT,bufferSize); recorder.startRecording();

Playing Video:

- Use VideoView for playing videos from resources or a URL.

VideoView videoView=findViewById(R.id.videoView);

videoView.setVideoURI(Uri.parse("android.resource://" + getPackageName()

+ "/" + R.raw.sample_video)); videoView.start();

Recording Video:

- You can use MediaRecorder for video recording. MediaRecorder

mediaRecorder=newMediaRecorder();

mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA)

; mediaRecorder.setOutputFile(outputFile); mediaRecorder.prepare();

mediaRecorder.start();

2.8. Sensors and Location

Android devices are equipped with several sensors and location services that allow you to interact with the device environment and track location-based data.

2.8.1 GPS (Location Services)

Location-based features are widely used in many apps. Android provides two primary ways to get location data:

- GPS (Global Positioning System)
- Network-based location (Wi-Fi, cell towers) Steps to Get GPS Location:

1. Declare Permissions:
    - In AndroidManifest.xml, add:

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

2. Using LocationManager:
    - LocationManager is a service that allows you to access GPS or network-based location.

```
LocationManager locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
```

```
LocationListener locationListener = new LocationListener() {
    @Override      public void onLocationChanged(Location
location) {         double latitude = location.getLatitude();
double longitude
= location.getLongitude();
    }
};
locationManager.requestLocationUpdates(LocationManager.GPS_PROVI
DER, 0, 0, locationListener);
```

3. Converting Coordinates to Human-Readable Address:
    - You can convert latitude and longitude to a physical address using Geocoder.

```
Geocoder geocoder = new Geocoder(this, Locale.getDefault());
List<Address> addresses = geocoder.getFromLocation(latitude, longitude,
1);
String address = addresses.get(0).getAddressLine(0);
```

4. FusedLocationProviderClient (Google Play Services):
    - For more accurate and battery-efficient location services, you can use FusedLocationProviderClient from Google Play Services.

```
FusedLocationProviderClient fusedLocationClient =
LocationServices.getFusedLocationProviderClient(this);
fusedLocationClient.getLastLocation()
    .addOnSuccessListener(this, new OnSuccessListener<Location>() {
```

```
@Override public void onSuccess(Location location) { if (location != null) { double latitude = location.getLatitude(); double longitude = location.getLongitude();
}
}
});
```

2.8.2 Sensors

Android devices are equipped with various sensors such as accelerometer, gyroscope, magnetometer, etc. These sensors allow you to detect device orientation, movement, and other physical properties.

Common Sensors:

- Accelerometer: Measures the device's acceleration.
- Gyroscope: Measures the rotation of the device.
- Magnetometer: Measures magnetic field (compass).
- Proximity Sensor: Detects how close the device is to your face.
- Light Sensor: Detects ambient light.

Using Sensors:

1. Declare Sensor Permissions (if needed):
    - Some sensors, like the proximity sensor, may require specific permissions. 2. Get Sensor Manager:

SensorManager sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

3. Register and Use Sensors: o For example, using the accelerometer:

Sensor accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sensorManager.registerListener(sensorEventListener, accelerometer, SensorManager.SENSOR_DELAY_UI);

4. SensorEventListener:

o Implement SensorEventListener to receive sensor updates.

```
SensorEventListener sensorEventListener = new SensorEventListener() {
  @Override public void onSensorChanged(SensorEvent event) { if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
float x = event.values[0]; float y = event.values[1]; float z = event.values[2];
    }
  }
  @Override public void onAccuracyChanged(Sensor sensor, int accuracy) {} };
```

5. Unregister the Listener:

o Always unregister the sensor listener when it's no longer needed to save battery.

```
sensorManager.unregisterListener(sensorEventListener);
```

2.9. Background Tasks

Background tasks are operations that run without blocking the main user interface (UI) thread. In Android, it's important to handle long-running operations, like network requests, file operations, or database updates, in the background to keep the app responsive.

2.9.1 AsyncTask

AsyncTask is a helper class for performing background operations and publishing results on the UI thread. It is useful for short tasks like downloading data or updating the UI after background work.

Basic AsyncTask Usage:

1. DoInBackground: This method is executed on a background thread.
2. onPostExecute: This method is executed on the UI thread after the background task is finished.

Example: private class MyAsyncTask extends AsyncTask<Void, Void, String> { @Override protected String doInBackground(Void... voids) {

// Perform background work (e.g., network request) return "Task Completed";

}

@Override protected void onPostExecute(String result) {

// Update the UI with the result textView.setText(result);

}

}

// Execute AsyncTask new

MyAsyncTask().execute();

Note: AsyncTask is deprecated in Android API level 30 (Android 11). It's recommended to use ExecutorService or WorkManager instead.

2.9.2 Services

Services run in the background to perform long-running operations, even if the app's UI is not visible. Services don't have a UI and are perfect for tasks like music playback or data synchronization.

Types of Services:

- Started Service: Initiated by startService(), runs in the background and can continue even after the component that started it is destroyed.
- Bound Service: Allows interaction between the service and one or more components (like activities).

Example of a simple Started Service: public class

MyService extends Service {

@Override public int onStartCommand(Intent intent, int

flags, int startId) {

// Perform background task here return

START_STICKY;

}

```
    @Override      public IBinder onBind(Intent
intent) {         return null;
    }
}
```

```
// Start the service
Intent intent = new Intent(this, MyService.class); startService(intent);
```

2.9.3 WorkManager

WorkManager is a modern API for handling background tasks that need to run reliably, even if the app is terminated or the device is rebooted. It's suitable for tasks like syncing data or uploading files in the background.

Using WorkManager:

1. Add the dependency in build.gradle:

```
implementation "androidx.work:work-runtime:2.7.0"
```

2. Create a Worker class to define the task:

```
public class MyWorker
    extends Worker {
    @NonNull     @Override     public
Result doWork() {
        // Perform background work here (e.g., data upload)         return
Result.success();
    }
}
```

3. Schedule the work:

```
OneTimeWorkRequest uploadWorkRequest = new
OneTimeWorkRequest.Builder(MyWorker.class).build();
WorkManager.getInstance(context).enqueue(uploadWorkRequest);
```

2.9.4 AlarmManager

AlarmManager allows you to schedule tasks to be executed at a later time, even if your app is not running. It's often used for scheduling periodic tasks, like reminders or background sync.

Example:

```
AlarmManageralarmManager=(AlarmManager)
getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(this, MyReceiver.class);
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent,
0); long triggerAtMillis = System.currentTimeMillis() + 60000; // Trigger after
1 minute  alarmManager.set(AlarmManager.RTC_WAKEUP,
triggerAtMillis,  pendingIntent);
```

2. 10.UI Enhancements and Animations

UI enhancements and animations are crucial for improving user experience by providing visual feedback, smooth transitions, and making the app more

engaging. Android offers several ways to enhance UI with animations and transitions.

2. 10.1Transitions

Transitions in Android help create smooth animations between UI changes. They are useful for moving from one screen or view to another, for example, when switching between activities or fragments.

Basic Transition Example:

1. Set up a simple transition: o In res/transition/ folder, create a fade_in.xml file for a fade-in transition:

<fade xmlns:android=”http://schemas.android.com/apk/res/android” />

2. Apply Transition:

o In your activity or fragment, set the transition like so:

Transition fadeTransition = TransitionInflater.from(this).inflateTransition(R.transition.fade_in);

getWindow().setEnterTransition(fadeTransition);

2.10.2 View Animations

View animations are used to animate individual UI elements like buttons, text fields, and images. They include transformations such as fading, scaling, rotation, and moving views.

Example of a Simple View Animation:

3. XML Animation (e.g., res/anim/rotate.xml):

<rotate xmlns:android=”http://schemas.android.com/apk/res/android” android:fromDegrees=”0”android:toDegrees=”360” android:duration=”1000” />

4. Applying Animation:

```
Animation rotateAnimation = AnimationUtils.loadAnimation(this, R.anim.rotate);
imageView.startAnimation(rotateAnimation);
```

2.10.3 Property Animations

Property animations allow you to animate properties of views, such as position, scale, rotation, and alpha. Unlike view animations, property animations don’t rely on the view hierarchy and are more flexible.

Example of Property Animation:

```
ObjectAnimator animator = ObjectAnimator.ofFloat(imageView, “25ranslation”, 0f, 500f); animator.setDuration(1000); // Duration in milliseconds animator.start();
```

In this example, 25ranslation is the property being animated from 0f to 500f (moving the view horizontally).

2.10.4 Animating Layout Changes

Android allows you to animate changes in the layout itself (e.g., resizing views, adding or removing views dynamically). This can be done using LayoutTransition.

Example of Layout Animation:

```
ViewGroup layout = findViewById(R.id.layout); LayoutTransition transition
        =       new  LayoutTransition();
layout.setLayoutTransition(transition);
```

This allows changes like adding/removing views to trigger animations automatically.

2.10.5 RecyclerView Animations

In RecyclerView, animations can be used when adding, removing, or updating items. You can create custom item animations or use built-in ones.

Example of Default Item Animation:

```
RecyclerView recyclerView = findViewById(R.id.recyclerView);
recyclerView.setLayoutManager(new LinearLayoutManager(this));
recyclerView.setAdapter(adapter);
```

This automatically adds default animations when items are added or removed from the list.

CHAPTER 3

SkyCast

3.1. Introduction

SkyCast is a sleek, user-friendly weather application that provides real-time weather updates for locations across the globe. With beautifully designed icons, accurate forecasts, and an intuitive interface, SkyCast transforms weather tracking into a smooth and enjoyable experience. It aims to keep users informed with essential weather data anytime, anywhere.

Target Audience:

SkyCast is designed for a wide range of users including daily commuters, travelers, event planners, and weather enthusiasts. It caters to anyone who requires quick, reliable, and visually engaging weather updates on the go.

Key Features:

- **Real-Time Weather Updates** – Get accurate temperature, humidity, wind speed, and more using the OpenWeather API.
- **Location Search** – Search for any city or use GPS to fetch weather for the current location.
- **Clean & Modern UI** – Simplified, elegant interface with weather-themed icons and responsive layouts.
- **Current Weather Details** – Displays current conditions like cloudiness, pressure, and sunrise/sunset times.
- **Weather Icons & Animations** – Visually rich elements to represent weather conditions like sunny, rainy, or snowy.
- **Scalable Design** – Built to support future features like forecasts, maps, and severe weather alerts.

3.2. App Design and User Interface (UI) Layout

Structure:

- Home Screen with current location weather.
- Search Screen for entering city names manually.
- Detailed Weather Screen showing temperature, wind, humidity, pressure, and weather description.
- Settings Screen (future scope) for unit preferences and themes.

UI Elements:

- TextInputLayout for location search.
- CardView to display weather data in well-structured containers.
- Material Icons for weather representation (sun, cloud, rain, etc.).
- ProgressBar or Lottie Animation for loading state.
- ConstraintLayout for flexible, responsive design.

3.3. App Architecture and Components

MVVM Pattern:

- Model – Handles weather data parsing and representation.
- View – Activities and Fragments show weather data and respond to user input.
- ViewModel – Manages UI-related data and survives configuration changes.
- LiveData – Observes and reacts to weather data changes.

Repository Pattern:

- Serves as the abstraction layer between the ViewModel and the OpenWeather API.

3.4. Expense Tracking and Reminder Functionality

- Weather Fetching via OpenWeather API
- City Search Functionality with real-time API request
- Unit Conversion Support (°C/°F and km/h/mph – future feature)
- Smooth error handling with messages for failed API calls or invalid locations
- Dynamic UI Update based on the weather response

3.5. Database and Offline Support

- Local Caching (future scope) using Room or SharedPreferences for last viewed location and weather data
- Offline Support to display last known data when there's no internet connection
- Settings Persistence for user preferences like units or themes

3.6. Reports and Enhancements (Future Scope)

- Multi-day Forecast (e.g., 3-day or 7-day)
- Weather Trend Visualization using graphs/charts
- Integration with Google Maps for map-based weather search
- PDF Export for weather logs or travel planning
- Widgets for Home Screen weather previews

3.7. Notifications and Reminders (Optional/Future)

- Daily Weather Summary Notification
- Severe Weather Alerts using push notifications
- Custom Reminder System for travel or events

3.8. App Performance and Optimization

- Efficient API Requests using Retrofit and caching
- Lightweight UI optimized for all screen sizes
- Image and Icon Optimization for smooth rendering

- Use of Coroutines or WorkManager for background tasks like API polling or preloading

3.9. Testing and Debugging

- Unit Testing for weather data conversion and API responses
- UI Testing for flow from search to weather display
- Integration Testing for API calls
- Crash Reporting with Firebase Crashlytics

3.10. App Deployment and Monetization

- Deployment: Signed APK/AAB with Play Store listing and feature screenshots
- Monetization Options:

- AdMob Integration for banner or interstitial ads

- Premium Upgrade: Remove ads, add detailed forecasts, dark mode themes, or advanced charts

3.11. Security

- Secure API Key Handling using encrypted storage or backend proxy
- No unnecessary permissions (uses only location and internet)
- Code Obfuscation using ProGuard
- Safe Error Handling to prevent app crashes or data leaks

3.12 APPLICATION CODE AND FILE :

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.SkyCast"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
        <meta-data
            android:name="preloaded_fonts"
            android:resource="@array/preloaded_fonts" />
    </application>

</manifest>
```

MainActivity.kt

```
package com.example.skycast;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

import org.json.JSONException;
import org.json.JSONObject;

import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;

import java.io.IOException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class MainActivity extends AppCompatActivity {
    private TextView cityNameText, temperatureText, humidityText,
descriptionText, windText;
    private ImageView weatherIcon;
    private Button refreshButton;
    private EditText cityNameInput;
    private static final String API_KEY =
"f4345d93eae798dee7a86f6dab12c20a";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        cityNameText = findViewById(R.id.cityNameText);
```

```
        temperatureText = findViewById(R.id.temperatureText);
        humidityText = findViewById(R.id.humidityText);
        windText= findViewById(R.id.windText);
        descriptionText = findViewById(R.id.descriptionText);
        weatherIcon = findViewById(R.id.weatherIcon);
        refreshButton  = findViewById(R.id.fetchWeatherButton);
        cityNameInput = findViewById(R.id.cityNameInput);

        refreshButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String cityName = cityNameInput.getText().toString();
                if(!cityName.isEmpty())
                {
                    FetchWeatherData(cityName);
                }
                else
                {
                    cityNameInput.setError("Please enter a city name");
                }
            }
        });



        FetchWeatherData("Mumbai");
    }

    private void FetchWeatherData(String cityName) {

        String url =
"https://api.openweathermap.org/data/2.5/onecall?lat={19.07060}&lon={72.
8777}&exclude={part}&appid={f4345d93eae798dee7a86f6dab12c20a\n}}"
+ cityName + "&appid=" + API_KEY + "&units=metric";

        ExecutorService executorService =
Executors.newSingleThreadExecutor();
        executorService.execute(() ->
                {
```

```
                OkHttpClient client = new OkHttpClient();
                Request request = new Request.Builder().url(url).build();
                try {
                    Response response = client.newCall(request).execute();
                    String result = response.body().string();
                    runOnUiThread(() -> updateUI(result));
                } catch (IOException e)
                {
                    e.printStackTrace();;
                }
            }
        );
    }

    private void updateUI(String result)
    {
        if(result != null)
        {
            try {
                JSONObject jsonObject = new JSONObject(result);
                JSONObject main =  jsonObject.getJSONObject("main");
                double temperature = main.getDouble("temp");
                double humidity = main.getDouble("humidity");
                double windSpeed =
jsonObject.getJSONObject("wind").getDouble("speed");

                String description =
jsonObject.getJSONArray("weather").getJSONObject(0).getString("descripti
on");
                String iconCode =
jsonObject.getJSONArray("weather").getJSONObject(0).getString("icon");
                String resourceName = "ic_" + iconCode;
                int resId = getResources().getIdentifier(resourceName, "drawable",
getPackageName());
                weatherIcon.setImageResource(resId);

                cityNameText.setText(jsonObject.getString("name"));
                temperatureText.setText(String.format("%.0f°", temperature));
                humidityText.setText(String.format("%.0f%%", humidity));
```

```
            windText.setText(String.format("%.0f km/h", windSpeed));
            descriptionText.setText(description);
        } catch (JSONException e)
        {
            e.printStackTrace();
        }
    }
  }
}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@drawable/background"
    android:padding="10sp"
    >

    <TextView
        android:id="@+id/cityNameText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:fontFamily="@font/urbanist"
        android:text="City"
        android:textSize="36sp"
        android:textColor="@color/white"
        />

    <TextView
        android:id="@+id/temperatureText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dp"
    android:layout_marginBottom="10dp"
    android:fontFamily="@font/urbanist"
    android:text="25°"
    android:textColor="#FFBF00"
    android:textSize="60sp"
    android:textStyle="bold"
    android:layout_below="@id/cityNameText"/>

<LinearLayout
    android:id="@+id/detailsLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:gravity="center"
    android:layout_below="@id/temperatureText"
    android:background="@drawable/background2"
    >

    <LinearLayout
        android:id="@+id/humidityLayout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:layout_weight="0.5"
        android:orientation="vertical"
        >
        <ImageView
            android:id="@+id/humidityIcon"
            android:layout_width="35dp"
            android:layout_height="35dp"
            android:layout_gravity="center"
            android:src="@drawable/humidity"
            />

        <TextView
            android:id="@+id/humidityText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
        android:text="60%"
        android:textSize="16sp"
        android:textStyle="bold"
        android:layout_gravity="center"
        android:textColor="@color/white"
        android:fontFamily="@font/urbanist"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Humidity"
        android:textSize="16sp"
        android:layout_gravity="center"
        android:textColor="@color/white"
        android:fontFamily="@font/urbanist"
        />


</LinearLayout>


<LinearLayout
    android:id="@+id/windLayout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:layout_weight="0.5"
    android:orientation="vertical"
    >
    <ImageView
        android:id="@+id/windIcon"
        android:layout_width="35dp"
        android:layout_height="35dp"
        android:layout_gravity="center"
        android:src="@drawable/wind"
        />


    <TextView
        android:id="@+id/windText"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="10 km/h"
            android:textSize="16sp"
            android:textStyle="bold"
            android:layout_gravity="center"
            android:textColor="@color/white"
            android:fontFamily="@font/urbanist"
            />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Wind"
            android:textSize="16sp"
            android:layout_gravity="center"
            android:textColor="@color/white"
            android:fontFamily="@font/urbanist"
            />

    </LinearLayout>
</LinearLayout>

<ImageView
    android:id="@+id/weatherIcon"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:layout_below="@id/detailsLayout"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    android:elevation="12dp"
    android:src="@drawable/contrast"
    android:contentDescription="Weather Icon"
    />

<TextView
    android:id="@+id/descriptionText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/weatherIcon"
    android:textColor="@color/white"
```

```
        android:layout_centerHorizontal="true"
        android:fontFamily="@font/urbanist"
        android:text="Sunny"
        android:textSize="28sp"
        />


    <EditText
        android:id="@+id/cityNameInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/descriptionText"
        android:textColor="#EFEFEF"
        android:padding="12dp"
        android:textColorHint="#BFBFBF"
        android:gravity="center"
        android:fontFamily="@font/urbanist"
        android:textSize="24sp"
        android:layout_marginTop="20dp"
        android:hint="Enter City Name"
        />

    <Button
        android:id="@+id/fetchWeatherButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/cityNameInput"
        android:backgroundTint="#2B3A67"
        android:fontFamily="@font/urbanist"
        android:text="Change City"
        android:textColor="#FFF"
        android:textSize="20sp"

        />



</RelativeLayout>
```

CHAPTER 4

OUTPUT

Fig 1 : Output Screen

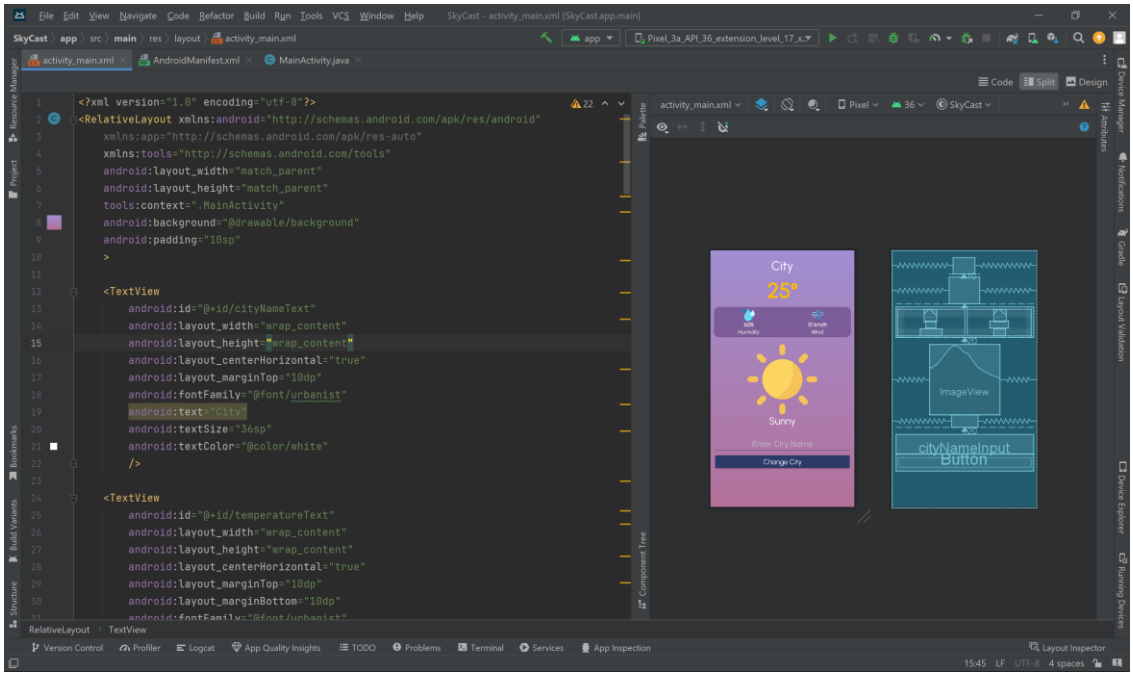

Fig 2: Design layout

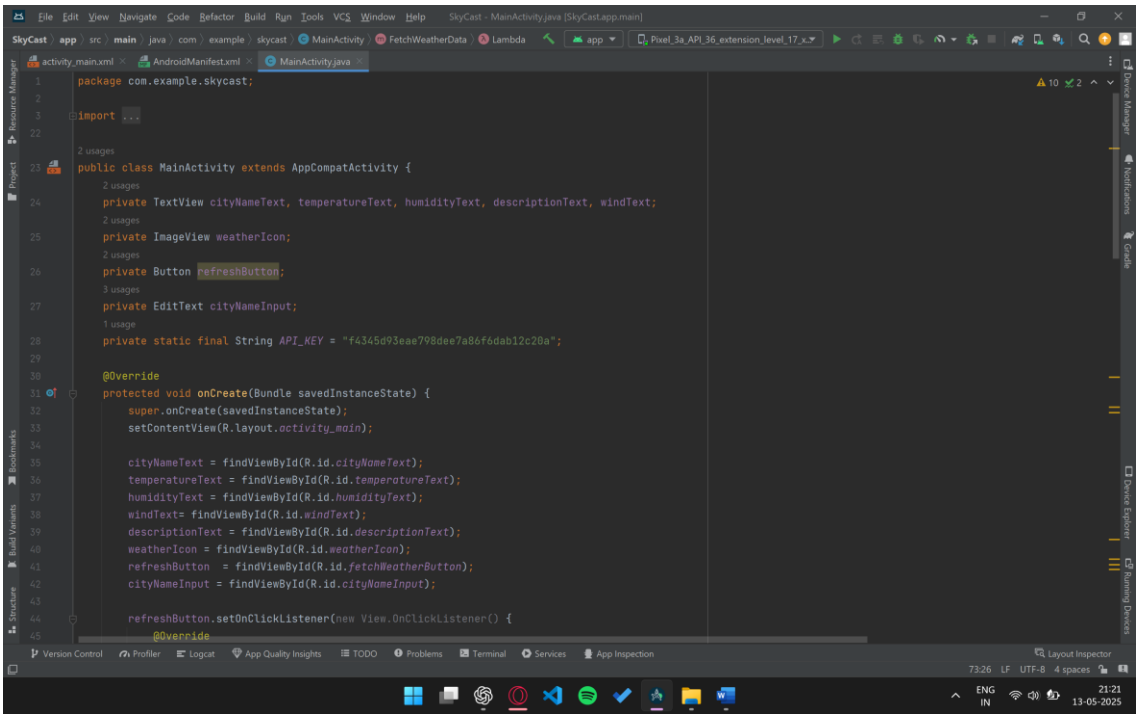

Fig 3: Main file

CHAPTER 5

CONCLUSION

QuizzHub successfully delivers an engaging and user-centric quiz experience through its clean interface, responsive design, and interactive features. By allowing users to personalize their journey, choose quiz topics, and receive realtime feedback, the app transforms traditional quizzing into a dynamic learning tool. The integration of visual cues and progress tracking enhances user motivation and retention.

Built with scalability in mind, QuizzHub lays the groundwork for future enhancements such as timed quizzes, performance analytics, and online leaderboards. As both a fun and educational platform, it holds potential for classroom use, self-assessment, and even gamified learning. In conclusion, QuizzHub stands out as a reliable and expandable solution for modern quiz-based applications, aligning well with the evolving needs of learners and tech-savvy users alike.

REFERENCES

1. Android Developers Documentation
   https://developer.android.com
   Official source for Android APIs, tools, and best practices used to develop the SkyCast app using Java.

2. Java Official Documentation
   https://docs.oracle.com/javase/8/docs/
   Core language documentation used for implementing logic, handling API requests, and managing app behavior in Java.

3. OpenWeather API Documentation
   https://openweathermap.org/api
   Primary weather data source for retrieving real-time weather information, forecasts, and geo-based data.

4. Material Design 3 Guidelines
   https://m3.material.io
   Google's design framework followed to ensure the SkyCast UI is consistent, modern, and user-friendly.

5. Stack Overflow Developer Community
   https://stackoverflow.com
   Used to solve implementation challenges, Java integration issues, and UI bugs during development.

6. YouTube Tutorials on Android Development in Java
   Various tutorials were referenced for implementing RecyclerViews, API calls with Retrofit, and creating responsive UIs in Java.

7. ChatGPT by OpenAI
   Served as a technical guide for real-time debugging, app architecture advice, Java logic corrections, UI/UX suggestions, and documentation writing.