**AGRI AIUTO ANDROID APP**

###### ***Submitted by***

**R.Yashwant(9915004107)**

**K.Hari Krishna(9915004109)**

**D.Hari Krishna(9915004219)**

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

****

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**KALASALINGAM UNIVERSITY**

**(Kalasalingam Academy of Research and Education)**

**KRISHNANKOIL 626 126**

Academic Year (2018-19)

**DECLARATION**

We affirm that the project work titled **AGRI AIUTO ANDROID APP** being submitted in partial fulfillment for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** is the original work carried out by us. It has not formed the part of any other project work submitted for award of any degree or diploma, either in this or any other University.

R.Yashwant

9915004107

K.Hari Krishna

9915004109

D.Hari Krishna

9915004219

**KALASALINGAM UNIVERSITY**

**(Kalasalingam Academy of Research and Education)**

**KRISHNANKOIL 626 126**

**BONAFIDE CERTIFICATE**

Certified that this project report “**AGRI AIUTO ANDROID APP**” is the bonafide work of “**R.Yashwant ,K.Hari Krishna, D.Hari Krishna**” who carried out the project work under my supervision.

**SUPERVISOR HEAD OF THE DEPARTMENT**

DR.S.Dhanasekaran Dr.R.Ramalakshmi

Assistant ProfessorComputer Science &Engineering

Computer Science & Engineering

Submitted for the Project Viva-voce examination held on.......................................

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ACKNOWLEDGEMENT**

We would like to express our thanks to our respected vice chancellor of our university **Dr.PNagRaj** for providing us an opportunity to improve our knowledge by doing this project. Secondly, We would like to thank the respected, **Co-ordinator Mr.M Raja** of who gave us an opportunity for completing the project. We would also like to express our thanks to our respected HOD of CSE department **Dr.R.RamaLakshmi** for providing us with all requirements for successfully completing the final year project and also for guiding us for our successful future.

A special thanks to our **Guide** respected Assistant professor (CSE department) **DR.S.Dhanasekaran**,whose contribution in stimulating suggestions and encouragement, helped us to do project especially in writing this report.

With respect and thankfulness,

R.Yashwant

K.Hari Krishna

D.Hari Krishna

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
| 1 | Introduction | **9** |
| 1.1 | Motivation | 9 |
| 1.1.1 | Problem Definiton | 9 |
| 1.1.2 | Objective of the project | 9 |
| 1.1.3 | Limitations of the project | 10 |
| 1.1.4 | Literature Survey | 10 |
| 1.1.5 | Existing System | 10 |
| 1.1.6 | Proposed System | 11 |
| 2 | Hardware and Software Requirement | 13 |
| 3 | UI Design | 14 |
| 3.1.1 | Navigation Drawer | 15 |
| 3.1.2 | Full Screen Activity | 16 |
| 3.1.3 | Basic Activity | 17 |
| 3.2 | Module Design And Organisation | 18 |
| 3.3 | UML Diagrams | 18 |
| 3.3.1 | Use Case Diagram | 20 |
| 3.3.2 | Sequence Diagram | 21 |
| 3.3.3 | Class Diagram | 23 |
| 3.3.4 | Output Screenshots And Source Code | 24 |
| 4 | Conclusion And References | 76 |

**LIST OF FIGURES**

**FIGURE PAGE NO**

Navigation Drawer 17

Use Case Diagram 21

Sequence Diagram 22

Class Diagram 24

Login Page 25

Registration Page 25

Language Selection 26

Module Selection 26

Crop Selection 27

**LIST OF ABBREVATIONS & NOMENCLATURE**

ABBREVATION MEANING

AGRI Agriculture

AIUTO Help

T&P Expert Profile

CV Module Page

FB FireBase

RDB Real Time Database

ALU  Arithmetic logic unit

AM Amplitude Modulation

API Application Programming Interface

DA Destination address

DB Database

DBA Database Administrator

DBMS Database management system

**ABSTRACT**

The "AGRI AIUTO" application delivers the detail information about which crop to grow in which season and which crop is suitable for that particular soil in which the farmer wishes to start. This application helps the farmer to choose the perfect pesticide in order to maintain a healthy crop and also provides the climatic conditions so that we can select the seasonal crop to get benefited. User sends their problem by uploading the image and any other user can respond through text or image. Now-a-days mobile being a common device, with this application we aim to make it friendly for farmers.

**CHAPTER 1**

**INTRODUCTION**

* 1. **Motivation**

Agriculture is the primary occupation of the larger part of Indian population. 65-70 % of Indian population is being depends on agriculture for their living. The challenging task for farmers is information management mainly in terms of the amount of data and the complexity of processes in precision farming. The data regarding farming are available from many sources like printed media, audio and visual aids, newspaper, TV, internet, mobile etc. but the formats and structures of data are dissimilar. So it‘s very hard for farmer to get the information and to understand the various information which are disseminate from various sources. Sometime many manual steps are required while processing data for transforming data from one format to another. New opportunities are shaped by smart phone technology for farmers. Farmers are capable with a low cost smart phone and the particular software to gain facilities which couldn‘t available on their hands before. In the days of financial crisis, farming is becoming more and more vigorous and much more important to be completed efficiently during the time period.

**1.1.1Problem Definiton**

Agriculture is the most healthful, useful and noble employment of man. We need agriculture to satisfy our hunger and thirst, good or bad people on this planet who earn their livelihood by agriculture as it is the backbone of Indian Economy.

The Agri Aiuto application is built by keeping in mind about farmer in mind and also the common man who want to grow crops for his daily needs.

**1.1.2 Objective Of The Project**

The objective of this project is to keep the farmer updated with all the information need by the farmers are about type of the soil , type of seed, required pesticide for the particular crop in all stage of its growth, fertilizer type, crop diseases and its selling.

This mobile application helps the farmer to understand the information provided through TextToSpeech Conversation and also makes them to interact with the experts and take suggestions from them.

**1.1.3 Limitations Of Project**

Agri aiuto provides the user access only to a limited amount of the data related to crops, pesticides and season.Mobile phones are less prone to malfunctioning when compared to Computers So, Cheaper Maintenance is the next added advantage for any system based on mobile phones.

**1.1.4 Literature Survey**

The progression in the agriculture production straight increases the Indian Economy and vice-versa is also true. Mobile apps in the arena of agriculture can be the best option to increase countries agriculture production. The inventions in technology in agriculture domain are not getting to the farmers; because of either most of them are illiterates or due to unawareness of the location from where they can have information. Hence, utmost of the farmers is being failed in acquisition of the possible production rate.

Today farmers are receiving diverse facts or information about faming like seeds, crop selection, crop processes weather, fertilizer, pesticides etc. from various resources which are distributed on many different locations according to its origin, its processors, producers or vendors. The data having different format and may have different specific contents can be heterogeneous in their structure and format. Therefore it is required to develop a system from where the required information is available to the farmer directly. **1.1.5** **Existing System**

**1.1.5.1 Plantix:**

Plantix is a free mobile application which offers farmers and gardeners the possibility to receive decision support directly on their smart phone. Due to image recognition, the app is able to identify the plant type - as well as the appearance of a possible disease, pest or nutrient deficiency**.**

**1.1.5.2 Disadvantages of Existing System:**

* Information about the pesticides and seasonal data is not provided.
* Information of Merchants and their availability is not provided.
* Request for more user details in order to access the application and does not involve experts for asking suggestions.
* Provided in only one language and provided information is not understood by the farmer.

**1.2 Proposed System**

This agriculture application App requires an Active Internet Connection for fetching the information. It provides user authentication using phone number. It provides the information about which crop to grow in which soil based upon the season to gain profit. The information about crops, pesticides and merchants is accessed from the Firebase cloud.

1. This application stores the list of crop information in the favourites for revisiting them using a special type of SQLite Database called Room Database. Room provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.
2. The Speech is the most common & primary mode of communication among human beings. It is the most natural and efficient form of exchanging information among humans. As the farmers are not graduates, we provide the additional feature of TextToSpeech so that they can understand the information accessed.
3. Agri Aiuto provides the information about the weather forecast of the current day, tomorrow and also for later 16 days .It also provides the temperature, rain and wind on the Google maps. The information about the weather is obtained from Weather API (Application Program Interface) i.e. from openweatherapi.com.
4. This agriculture application provides a feature of asking suggestions from experts for the problem that arise in plantation. It provides a group chat where the farmers or other individuals can upload their problem through images and experts can communicate with them by sending message. It also stores the details of experts like name, email, phone number, type of stream and their degree certificate which is stored in Firebase Database.

**CHAPTER 2**

**HARDWARE REQUIREMENT**

**Client side:**

|  |  |
| --- | --- |
| **RAM** | 1 GB |
| **Android API Level** | 20 + |
| **Processor** | 1.0 GHz |

**Server side:**

|  |  |
| --- | --- |
| **RAM** | 1 GB |
| **Harddisk** | 20 GB |
| **Processor** | 2.0 GHz |

**SOFTWARE REQUIREMENT**

**Client side:**

|  |  |
| --- | --- |
| Android | Sdk |
| Compiler | Compiler app |

**Server side:**

|  |  |
| --- | --- |
| Language | Kotlin |
| Database Server | Firebase |
| Operating System | Windows or any equivelent OS |

**Front End :**Java,Xml,Kotlin.

**Back End :** Firebase.

**CHAPTERS 3**

**3.1 UI DESIGN**

Android is a widely used OS made for smart phones and tablets. It is an open source project led by Google and it is released under Apache License. This permissive license helped this OS to be widely adopted and allows the manufacturers to freely modify and customize it. As matter of fact, despite Android being designed for smartphones and tablets, it is also used in TVs, cameras and so on. Moreover, Android has a very large community that extend its features and creates apps that cover almost all aspects.

All android applications, called apps, are built on Android UI framework. App interface is the first thing a user sees and interacts with. From the user perspective, this framework keeps the overall experience consistent for every app installed in our smartphone or tablets. At the same time, from the developer perspective, this framework provides some basic blocks that can be used to build complex and consistent user interface (API).

Android UI interface is divided in three different areas:

• Home screen

• All apps

• Recent screen

The home screen is the “landing” area when we power our phone on. This interface is highly customizable and themed. Using widgets we can create and personalize our “home” screen. All apps is the interface where the app installed are displayed, while recent screens are the list of last used apps. Since its born, Android has changed a lot in terms of its features and its interfaces. The growth of the smartphone power made possible creating ever more appealing apps. At the beginning, apps in Android did not have a consistent interface and well defined rules so every app had a different approach, navigation structure and buttons position. This caused user confusion and it was one of the most important missing features compared to the iOS

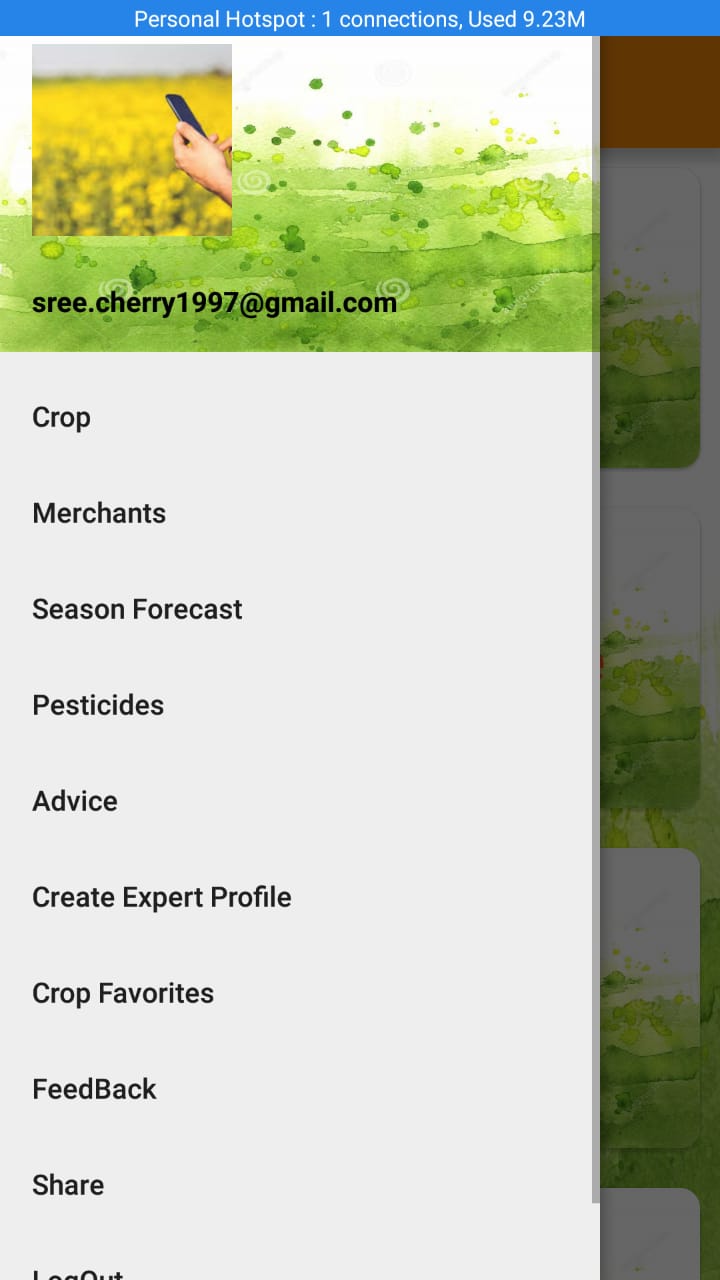
•View:- It is the base class for all visual components (control and widgets). All the controls present in an android app are derived from this class. A View is an object that draws something on a smartphone screen and enables an user to interact with it.

• Viewgroup :- A ViewGroup can contain one or more Views and defines how these Views are placed in the user interface (these are used along with Android Layout managers.)

•Fragments :- Introduced from API level 11, this component encapsulates a single piece of UI interface. They are very usefulwhen we have to create and optimize our app user interface for multiple devices or multiple screen size.

•Activities :- Usually an Android app consists of several activities that exchange data and information.

**3.1.1 NAVIGATION DRAWER**

* The navigation drawer is a panel that displays the app’s main navigation options on the left edge of the screen. It is hidden most of the time, but is revealed when the user swipes a finger from the left edge of the screen or, while at the top level of the app, the user touches the app icon in the action bar.The main content view must be the first child in the Drawer Layout because the XML order implies z-ordering and the drawer must be on top of the content.
* The main content view is set to match the parent view's width and height, because it represents the entire UI when the navigation drawer is hidden.
* The drawer view must specify its horizontal gravity with the android:layout\_gravity attribute. To support right-to-left (RTL) languages, specify the value with "start" instead of "left" (so the drawer appears on the right when the layout is RTL).
* The drawer view width fits the content and the height matches the parent view. The drawer width should be no more than 320dp so the user can always see a portion of the main content. 

**3.1.2 FULL SCREEN ACTIVITY**

The geometry of a view is that of a rectangle. A view has a location, expressed as a pair of *left* and *top* coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the pixel.

It is possible to retrieve the location of a view by invoking the methods [getLeft()](https://developer.android.com/reference/android/view/View.html#getLeft()) and [getTop()](https://developer.android.com/reference/android/view/View.html#getTop()). The former returns the left, or X, coordinate of the rectangle representing the view. The latter returns the top, or Y, coordinate of the rectangle representing the view. These methods both return the location of the view relative to its parent. For instance, when getLeft() returns 20, that means the view is located 20 pixels to the right of the left edge of its direct parent.

In addition, several convenience methods are offered to avoid unnecessary computations, namely [getRight()](https://developer.android.com/reference/android/view/View.html#getRight()) and [getBottom()](https://developer.android.com/reference/android/view/View.html#getBottom()). These methods return the coordinates of the right and bottom edges of the rectangle representing the view. For instance, calling [getRight()](https://developer.android.com/reference/android/view/View.html#getRight()) is similar to the following computation: getLeft() + getWidth() (see [Size](https://developer.android.com/reference/android/view/View.html#SizePaddingMargins) for more information about the width.)

**3.1.3 Basic Activity**

Android Studio provides code templates that follow the Android design and development best practices to get you on the right track to creating beautiful, functional apps. You can use templates to create new app modules, individual activities, or other specific Android project components.

Some templates provide starter code for common usage contexts, such as navigation drawers or login screens. You can choose from these app module and activity templates when you first [create your project](https://developer.android.com/studio/projects/create-project.html), when you [add a new app module within an existing project](https://developer.android.com/studio/projects/add-app-module.html), or when you add a new activity within an app module.

In addition to activities, you can also add other Android project components to an existing app using templates. These templates include both code components, such as services and fragments, and non-code components, such as folders and XML files.

This page discusses how to add Android project components like activities to your project and describes the commonly used activity templates available in Android Studio. Note that most templates depend on the [Android Support Library](https://developer.android.com/tools/support-library/features.html) to include user interface principles based on [material design](https://developer.android.com/design/material/index.html). The list of templates provided in Android Studio is constantly growing. Android Studio groups templates by the type of component that they add, such as an **Activity** or an **XML** file, as shown in figure 1.

To add an Android project component using a template, use the **Project** window. Right-click on the folder in which you want to add the new component, and select **New**. Based on what components can be added to the folder you clicked on, you then see a list of template types like those shown in figure 1.

When you select the template you want to add, a corresponding wizard window appears and asks for the component's configuration information, such as its name. After you enter the configuration information, Android Studio creates and opens the files for your new component. It also runs a Gradle build to sync your project.

**3.2** **Module design and organization**

**Admin Module:**

The Admin has to create an account or sign in the account. Admin adds the data related to crop, pesticides, weather forecast and merchants and store it in the cloud. Admin verifies whether the expert is real or fake which is the background manual task.

**User Module:**

The User has to sign in the account using their phone authentication and needed to verify using their gmail and password. otherwise the application cannot be accessed .While signing the account ,user has to enter his name ,email and mobile number. User can view the information related to crops, pesticides and merchants and make use of it. User can listen the information provided using TextToSpeech. User can see the weather details by searching for a particular city. If the user is a merchant he can also add his details along with the available merchant’s information. User can take suggestions for their crop growth by uploading the image of the problem raised in their crop and notify the user of their problem.

**Expert Module**:

The Expert has to sign in the account using their phone number and needed to verify the One Time Password (OTP) otherwise the application cannot be accessed. Expert need to give advice to the users by sending a message for their problem and notify the user about the message.

* 1. **UML DIAGRAMS**

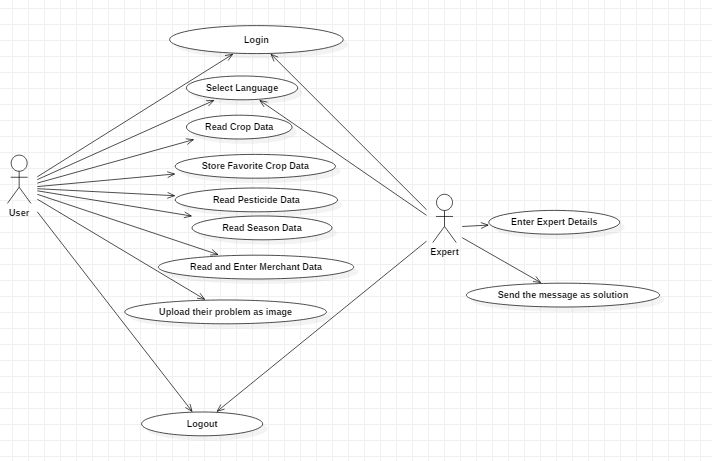
UML stands for Unified Modeling Language.UML is a standardized general purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

1. The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta- model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.
2. The Unified Modeling Language is a standard language for specifying , visualization, constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.
3. The Unified Modeling Language represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

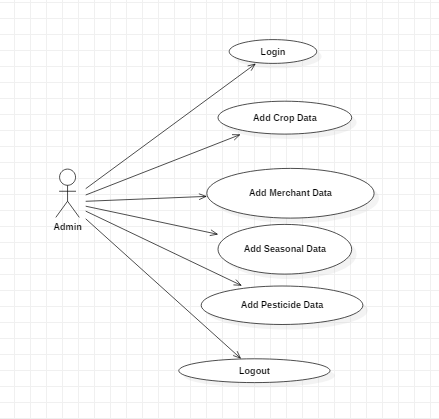
The Unified Modeling Language is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**3.3.1 Use case Diagram:**

A Use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

****

Use case Diagram for User and Expert

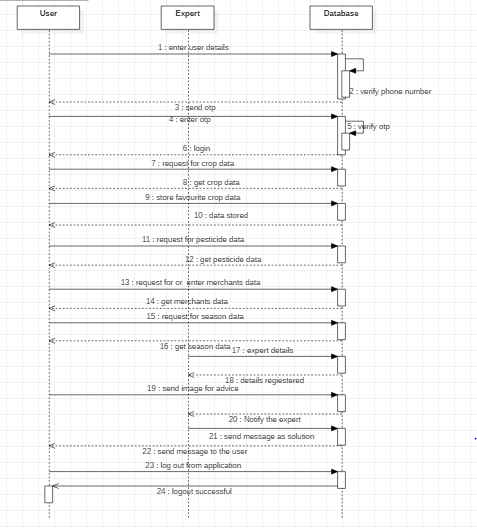


Use case Diagram for Admin

**3.3.2 Sequence Diagram**

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

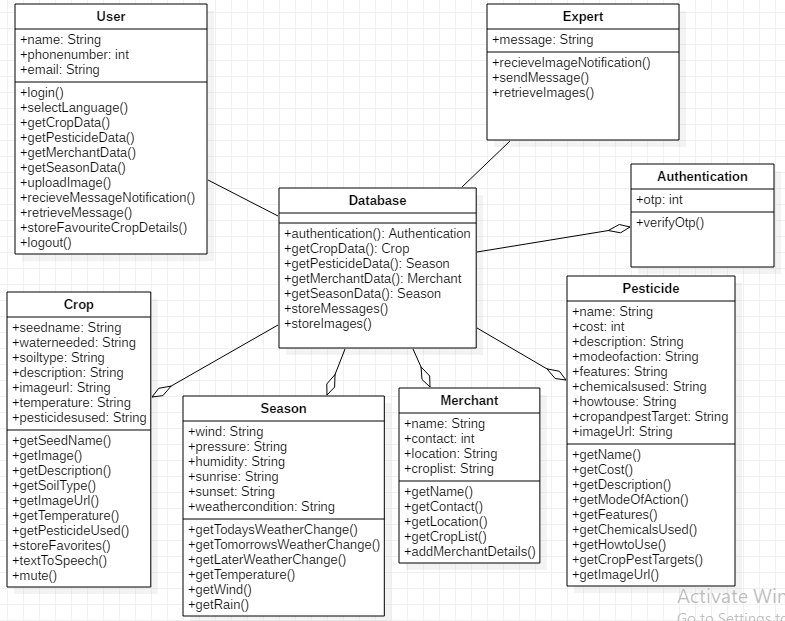


**3.3.3 Class Diagram**

In software engineering, a **class diagram** in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

In the diagram, classes are represented with boxes which contain three parts:

* The top part contains the name of the class
* The middle part contains the attributes of the class
* The bottom part gives the methods or operations the class can take or undertake.

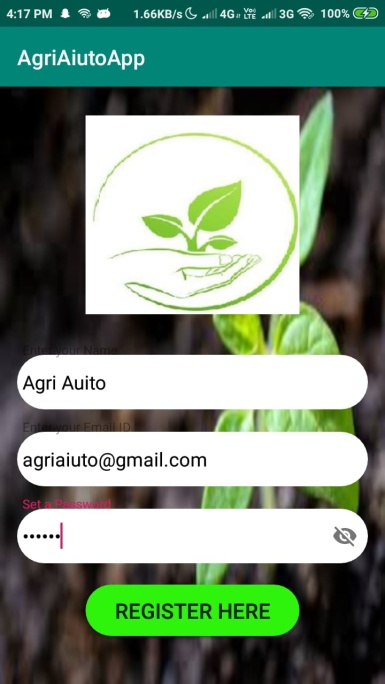


Class Diagram

**3.3.4 Output Screen Shots And Source Code**

****

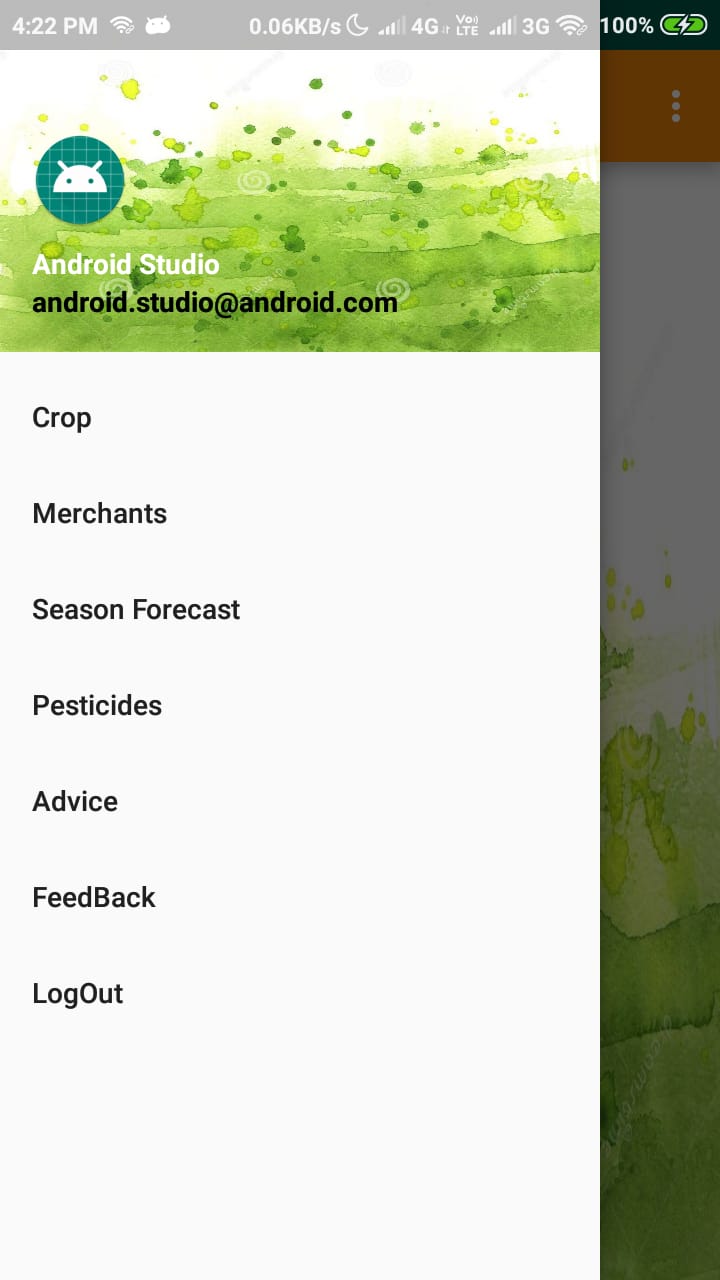
Login Page



Registration Page

****

Language Selection



Module Selection



Crop Selection

**Source Code**

**Map Activity.java**

**package com.example.agriaiuto.activities;**

**import android.content.SharedPreferences;**

**import android.os.Bundle;**

**import android.preference.PreferenceManager;**

**import android.support.annotation.IdRes;**

**import android.support.v7.app.AppCompatActivity;**

**import android.webkit.WebView;**

**import com.roughike.bottombar.BottomBar;**

**import com.roughike.bottombar.OnMenuTabClickListener;**

**import com.example.agriaiuto.R;**

**public class MapActivity extends AppCompatActivity {**

**private BottomBar mBottomBar;**

**@Override**

**protected void onCreate(Bundle savedInstanceState) {**

**SharedPreferencesprefs=PreferenceManager.getDefaultSharedPreferences(this);**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.activity\_map);**

**SharedPreferencessp=PreferenceManager.getDefaultSharedPreferences(this);**

**StringapiKey=sp.getString("apiKey", getResources().getString(R.string.apiKey));**

**final WebView webView = (WebView) findViewById(R.id.webView)**

**webView.getSettings().setJavaScriptEnabled(true);**

**webView.loadUrl("file:///android\_asset/map.html?lat=" + prefs.getFloat("latitude", 0) + "&lon=" + prefs.getFloat("longitude", 0) + "&appid=" + apiKey);**

**mBottomBar = BottomBar.attach(this, savedInstanceState);**

**mBottomBar.setItems(R.menu.menu\_map\_bottom);**

**mBottomBar.setOnMenuTabClickListener(new OnMenuTabClickListener() {**

**@Override**

**public void onMenuTabSelected(@IdRes int menuItemId) {**

**if (menuItemId == R.id.map\_rain) {**

**webView.loadUrl("javascript:map.removeLayer(windLayer);map.removeLayer(tempLayer);map.addLayer(rainLayer);");**

**} else if (menuItemId == R.id.map\_wind) {**

**webView.loadUrl("javascript:map.removeLayer(rainLayer);map.removeLayer(tempLayer);map.addLayer(windLayer);");**

**} else if (menuItemId == R.id.map\_temperature) {**

**webView.loadUrl("javascript:map.removeLayer(windLayer);map.removeLayer(rainLayer);map.addLayer(tempLayer);");**

**}**

**}**

**@Override**

**public void onMenuTabReSelected(@IdRes int menuItemId) {**

**}**

**});**

**}**

**@Override**

**protected void onSaveInstanceState(Bundle outState) {**

**super.onSaveInstanceState(outState);**

**mBottomBar.onSaveInstanceState(outState);**

**}**

**}**

**Graph Activity.Java**

**package com.example.agriaiuto.activities;**

**import android.content.SharedPreferences;**

**import android.graphics.Color;**

**import android.graphics.DashPathEffect;**

**import android.graphics.Paint;**

**import android.os.Bundle;**

**import android.preference.PreferenceManager;**

**import android.support.design.widget.Snackbar;**

**import android.support.v7.app.AppCompatActivity;**

**import android.support.v7.widget.Toolbar;**

**import android.util.Log;**

**import com.db.chart.Tools;**

**import com.db.chart.model.LineSet;**

**import com.db.chart.view.ChartView;**

**import com.db.chart.view.LineChartView;**

**import org.json.JSONArray;**

**import org.json.JSONException;**

**import org.json.JSONObject;**

**import java.text.SimpleDateFormat;**

**import java.util.ArrayList;**

**import java.util.TimeZone;**

**import com.example.agriaiuto.R;**

**import com.example.agriaiuto.models.Weather;**

**import com.example.agriaiuto.tasks.ParseResult;**

**import com.example.agriaiuto.utils.UnitConvertor;**

**public class GraphActivity extends AppCompatActivity {**

**SharedPreferences sp;**

**int theme;**

**ArrayList<Weather> weatherList = new ArrayList<>();**

**float minTemp = 100000;**

**float maxTemp = 0;**

**float minRain = 100000;**

**float maxRain = 0;**

**float minPressure = 100000;**

**float maxPressure = 0;**

**@Override**

**protected void onCreate(Bundle savedInstanceState) {**

**SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);**

**setTheme(theme = getTheme(prefs.getString("theme", "fresh")));**

**boolean darkTheme = theme == R.style.AppTheme\_NoActionBar ;**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.activity\_graph);**

**Toolbar toolbar = (Toolbar) findViewById(R.id.graph\_toolbar);**

**setSupportActionBar(toolbar);**

**getSupportActionBar().setDisplayHomeAsUpEnabled(true);**

**if (darkTheme) {**

**toolbar.setPopupTheme(R.style.AppTheme\_PopupOverlay\_Dark);**

**}**

**sp = PreferenceManager.getDefaultSharedPreferences(GraphActivity.this);**

**String lastLongterm = sp.getString("lastLongterm", "");**

**if (parseLongTermJson(lastLongterm) == ParseResult.OK) {**

**temperatureGraph();**

**rainGraph();**

**pressureGraph();**

**} else {**

**Snackbar.make(findViewById(android.R.id.content), R.string.msg\_err\_parsing\_json, Snackbar.LENGTH\_LONG).show();**

**}**

**}**

**private void temperatureGraph() {**

**LineChartView lineChartView = (LineChartView) findViewById(R.id.graph\_temperature);**

**// Data**

**LineSet dataset = new LineSet();**

**for (int i = 0; i < weatherList.size(); i++) {**

**float temperature = UnitConvertor.convertTemperature(Float.parseFloat(weatherList.get(i).getTemperature()), sp);**

**if (temperature < minTemp) {**

**minTemp = temperature;**

**}**

**if (temperature > maxTemp) {**

**maxTemp = temperature;**

**}**

**dataset.addPoint(getDateLabel(weatherList.get(i), i), (float) temperature);**

**}**

**dataset.setSmooth(false);**

**dataset.setColor(Color.parseColor("#FF5722"));**

**dataset.setThickness(4);**

**lineChartView.addData(dataset);**

**// Grid**

**Paint paint = new Paint();**

**paint.setStyle(Paint.Style.STROKE);**

**paint.setAntiAlias(true);**

**paint.setColor(Color.parseColor("#333333"));**

**paint.setPathEffect(new DashPathEffect(new float[]{10, 10}, 0));**

**paint.setStrokeWidth(1);**

**lineChartView.setGrid(ChartView.GridType.HORIZONTAL, paint);**

**lineChartView.setBorderSpacing(Tools.fromDpToPx(10));**

**lineChartView.setAxisBorderValues((int) (Math.round(minTemp)) - 1, (int) (Math.round(maxTemp)) + 1);**

**lineChartView.setStep(2);**

**lineChartView.setXAxis(false);**

**lineChartView.setYAxis(false);**

**lineChartView.show();**

**}**

**private void rainGraph() {**

**LineChartView lineChartView = (LineChartView) findViewById(R.id.graph\_rain);**

**// Data**

**LineSet dataset = new LineSet();**

**for (int i = 0; i < weatherList.size(); i++) {**

**float rain = Float.parseFloat(weatherList.get(i).getRain());**

**if (rain < minRain) {**

**minRain = rain;**

**}**

**if (rain > maxRain) {**

**maxRain = rain;**

**}**

**dataset.addPoint(getDateLabel(weatherList.get(i), i), rain);**

**}**

**dataset.setSmooth(false);**

**dataset.setColor(Color.parseColor("#2196F3"));**

**dataset.setThickness(4);**

**lineChartView.addData(dataset);**

**// Grid**

**Paint paint = new Paint();**

**paint.setStyle(Paint.Style.STROKE);**

**paint.setAntiAlias(true);**

**paint.setColor(Color.parseColor("#333333"));**

**paint.setPathEffect(new DashPathEffect(new float[]{10, 10}, 0));**

**paint.setStrokeWidth(1);**

**lineChartView.setGrid(ChartView.GridType.HORIZONTAL, paint);**

**lineChartView.setBorderSpacing(Tools.fromDpToPx(10));**

**lineChartView.setAxisBorderValues(0, (int) (Math.round(maxRain)) + 1);**

**lineChartView.setStep(1);**

**lineChartView.setXAxis(false);**

**lineChartView.setYAxis(false);**

**lineChartView.show();**

**}**

**private void pressureGraph() {**

**LineChartView lineChartView = (LineChartView) findViewById(R.id.graph\_pressure);**

**// Data**

**LineSet dataset = new LineSet();**

**for (int i = 0; i < weatherList.size(); i++) {**

**float pressure = UnitConvertor.convertPressure(Float.parseFloat(weatherList.get(i).getPressure()), sp);**

**if (pressure < minPressure) {**

**minPressure = pressure;**

**}**

**if (pressure > maxPressure) {**

**maxPressure = pressure;**

**}**

**dataset.addPoint(getDateLabel(weatherList.get(i), i), pressure);**

**}**

**dataset.setSmooth(true);**

**dataset.setColor(Color.parseColor("#4CAF50"));**

**dataset.setThickness(4);**

**lineChartView.addData(dataset);**

**// Grid**

**Paint paint = new Paint();**

**paint.setStyle(Paint.Style.STROKE);**

**paint.setAntiAlias(true);**

**paint.setColor(Color.parseColor("#333333"));**

**paint.setPathEffect(new DashPathEffect(new float[]{10, 10}, 0));**

**paint.setStrokeWidth(1);**

**lineChartView.setGrid(ChartView.GridType.HORIZONTAL, paint);**

**lineChartView.setBorderSpacing(Tools.fromDpToPx(10));**

**lineChartView.setAxisBorderValues((int) minPressure - 1, (int) maxPressure + 1);**

**lineChartView.setStep(2);**

**lineChartView.setXAxis(false);**

**lineChartView.setYAxis(false);**

**lineChartView.show();**

**}**

**public ParseResult parseLongTermJson(String result) {**

**int i;**

**try {**

**JSONObject reader = new JSONObject(result);**

**final String code = reader.optString("cod");**

**if ("404".equals(code)) {**

**return ParseResult.CITY\_NOT\_FOUND;**

**}**

**JSONArray list = reader.getJSONArray("list");**

**for (i = 0; i < list.length(); i++) {**

**Weather weather = new Weather();**

**JSONObject listItem = list.getJSONObject(i);**

**JSONObject main = listItem.getJSONObject("main");**

**JSONObject windObj = listItem.optJSONObject("wind");**

**weather.setWind(windObj.getString("speed"));**

**weather.setPressure(main.getString("pressure"));**

**weather.setHumidity(main.getString("humidity"));**

**JSONObject rainObj = listItem.optJSONObject("rain");**

**JSONObject snowObj = listItem.optJSONObject("snow");**

**if (rainObj != null) {**

**weather.setRain(MainScreenActivity.getRainString(rainObj));**

**} else {**

**weather.setRain(MainScreenActivity.getRainString(snowObj));**

**}**

**weather.setDate(listItem.getString("dt"));**

**weather.setTemperature(main.getString("temp"));**

**weatherList.add(weather);**

**}**

**} catch (JSONException e) {**

**Log.e("JSONException Data", result);**

**e.printStackTrace();**

**return ParseResult.JSON\_EXCEPTION;**

**}**

**return ParseResult.OK;**

**}**

**String previous = "";**

**public String getDateLabel(Weather weather, int i) {**

**if ((i + 4) % 4 == 0) {**

**SimpleDateFormat resultFormat = new SimpleDateFormat("E");**

**resultFormat.setTimeZone(TimeZone.getDefault());**

**String output = resultFormat.format(weather.getDate());**

**if (!output.equals(previous)) {**

**previous = output;**

**return output;**

**} else {**

**return "";**

**}**

**} else {**

**return "";**

**}**

**}**

**private int getTheme(String themePref) {**

**switch (themePref) {**

**/\* case "dark":**

**return R.style.AppTheme\_NoActionBar\_Dark;**

**case "classic":**

**return R.style.AppTheme\_NoActionBar\_Classic;**

**case "classicdark":**

**return R.style.AppTheme\_NoActionBar\_Classic\_Dark;\*/**

**default:**

**return R.style.AppTheme\_NoActionBar;**

**}**

**}**

**}**

**MainScreen Activity.java**

**package com.example.agriaiuto.activities;**

**import android.Manifest;**

**import android.annotation.SuppressLint;**

**import android.app.ProgressDialog;**

**import android.content.Context;**

**import android.content.DialogInterface;**

**import android.content.Intent;**

**import android.content.SharedPreferences;**

**import android.content.pm.PackageManager;**

**import android.graphics.Typeface;**

**import android.location.Location;**

**import android.location.LocationListener;**

**import android.location.LocationManager;**

**import android.net.ConnectivityManager;**

**import android.net.NetworkInfo;**

**import android.os.Bundle;**

**import android.preference.PreferenceManager;**

**import android.provider.Settings;**

**import android.support.design.widget.Snackbar;**

**import android.support.design.widget.TabLayout;**

**import android.support.v4.app.ActivityCompat;**

**import android.support.v4.content.ContextCompat;**

**import android.support.v4.view.ViewPager;**

**import android.support.v7.app.AlertDialog;**

**import android.support.v7.app.AppCompatActivity;**

**import android.support.v7.widget.Toolbar;**

**import android.text.InputType;**

**import android.util.Log;**

**import android.view.Menu;**

**import android.view.MenuItem;**

**import android.view.View;**

**import android.widget.EditText;**

**import android.widget.TextView;**

**import org.json.JSONArray;**

**import org.json.JSONException;**

**import org.json.JSONObject;**

**import java.text.DateFormat;**

**import java.text.DecimalFormat;**

**import java.util.ArrayList;**

**import java.util.Calendar;**

**import java.util.Date;**

**import java.util.GregorianCalendar;**

**import java.util.HashMap;**

**import java.util.List;**

**import java.util.Map;**

**import com.example.agriaiuto.AlarmReceiver;**

**import com.example.agriaiuto.Constants;**

**import com.example.agriaiuto.Fragments.SeasonFragment;**

**import com.example.agriaiuto.R;**

**import com.example.agriaiuto.adapters.ViewPagerAdapter;**

**import com.example.agriaiuto.adapters.WeatherRecyclerAdapter;**

**import com.example.agriaiuto.Fragments.SeasonFragment;**

**import com.example.agriaiuto.models.Weather;**

**import com.example.agriaiuto.tasks.GenericRequestTask;**

**import com.example.agriaiuto.tasks.ParseResult;**

**import com.example.agriaiuto.tasks.TaskOutput;**

**import com.example.agriaiuto.utils.UnitConvertor;**

**import com.example.agriaiuto.widgets.AbstractWidgetProvider;**

**import com.example.agriaiuto.widgets.DashClockWeatherExtension;**

**public class MainScreenActivity extends AppCompatActivity implements LocationListener {**

**protected static final int MY\_PERMISSIONS\_ACCESS\_FINE\_LOCATION = 1;**

**// Time in milliseconds; only reload weather if last update is longer ago than this value**

**private static final int NO\_UPDATE\_REQUIRED\_THRESHOLD = 300000;**

**private static Map<String, Integer> speedUnits = new HashMap<>(3);**

**private static Map<String, Integer> pressUnits = new HashMap<>(3);**

**private static boolean mappingsInitialised = false;**

**Typeface weatherFont;**

**Weather todayWeather = new Weather();**

**TextView todayTemperature;**

**TextView todayDescription;**

**TextView todayWind;**

**TextView todayPressure;**

**TextView todayHumidity;**

**TextView todaySunrise;**

**TextView todaySunset;**

**TextView lastUpdate;**

**TextView todayIcon;**

**ViewPager viewPager;**

**TabLayout tabLayout;**

**View appView;**

**LocationManager locationManager;**

**ProgressDialog progressDialog;**

**int theme;**

**boolean destroyed = false;**

**private List<Weather> longTermWeather = new ArrayList<>();**

**private List<Weather> longTermTodayWeather = new ArrayList<>();**

**private List<Weather> longTermTomorrowWeather = new ArrayList<>();**

**public String recentCity = "";**

**@Override**

**protected void onCreate(Bundle savedInstanceState) {**

**// Initialize the associated SharedPreferences file with default values**

**PreferenceManager.setDefaultValues(this, R.xml.prefs, false);**

**SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);**

**setTheme(theme = getTheme(prefs.getString("theme", "fresh")));**

**boolean darkTheme = theme == R.style.AppTheme\_NoActionBar;**

**// Initiate activity**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.activity\_scrolling);**

**appView = findViewById(R.id.viewApp);**

**progressDialog = new ProgressDialog(MainScreenActivity.this);**

**// Load toolbar**

**Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);**

**setSupportActionBar(toolbar);**

**if (darkTheme) {**

**toolbar.setPopupTheme(R.style.AppTheme\_PopupOverlay\_Dark);**

**}**

**// Initialize textboxes**

**todayTemperature = (TextView) findViewById(R.id.todayTemperature);**

**todayDescription = (TextView) findViewById(R.id.todayDescription);**

**todayWind = (TextView) findViewById(R.id.todayWind);**

**todayPressure = (TextView) findViewById(R.id.todayPressure);**

**todayHumidity = (TextView) findViewById(R.id.todayHumidity);**

**todaySunrise = (TextView) findViewById(R.id.todaySunrise);**

**todaySunset = (TextView) findViewById(R.id.todaySunset);**

**lastUpdate = (TextView) findViewById(R.id.lastUpdate);**

**todayIcon = (TextView) findViewById(R.id.todayIcon);**

**weatherFont = Typeface.createFromAsset(this.getAssets(), "fonts/weather.ttf");**

**todayIcon.setTypeface(weatherFont);**

**// Initialize viewPager**

**viewPager = (ViewPager) findViewById(R.id.viewPager);**

**tabLayout = (TabLayout) findViewById(R.id.tabs);**

**destroyed = false;**

**initMappings();**

**// Preload data from cache**

**preloadWeather();**

**updateLastUpdateTime();**

**// Set autoupdater**

**AlarmReceiver.setRecurringAlarm(this);**

**}**

**public WeatherRecyclerAdapter getAdapter(int id) {**

**WeatherRecyclerAdapter weatherRecyclerAdapter;**

**if (id == 0) {**

**weatherRecyclerAdapter = new WeatherRecyclerAdapter(this, longTermTodayWeather);**

**} else if (id == 1) {**

**weatherRecyclerAdapter = new WeatherRecyclerAdapter(this, longTermTomorrowWeather);**

**} else {**

**weatherRecyclerAdapter = new WeatherRecyclerAdapter(this, longTermWeather);**

**} return weatherRecyclerAdapter;**

**}**

**@Override**

**public void onResume() {**

**super.onResume();**

**if (getTheme(PreferenceManager.getDefaultSharedPreferences(this).getString("theme", "fresh")) != theme) {**

**// Restart activity to apply theme**

**overridePendingTransition(0, 0);**

**finish();**

**overridePendingTransition(0, 0);**

**startActivity(getIntent());**

**} else if (shouldUpdate() && isNetworkAvailable()) {**

**getTodayWeather();**

**getLongTermWeather();**

**}**

**}**

**@Override**

**protected void onDestroy() {**

**super.onDestroy();**

**destroyed = true;**

**if (locationManager != null) {**

**try {**

**locationManager.removeUpdates(MainScreenActivity.this);**

**} catch (SecurityException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**private void preloadWeather() {**

**SharedPreferences sp = PreferenceManager.getDefaultSharedPreferences(MainScreenActivity.this);**

**String lastToday = sp.getString("lastToday", "");**

**if (!lastToday.isEmpty()) {**

**new TodayWeatherTask(this, this, progressDialog).execute("cachedResponse", lastToday);**

**}**

**String lastLongterm = sp.getString("lastLongterm", "");**

**if (!lastLongterm.isEmpty()) {**

**new LongTermWeatherTask(this, this, progressDialog).execute("cachedResponse", lastLongterm);**

**}**

**}**

**private void getTodayWeather() {**

**new TodayWeatherTask(this, this, progressDialog).execute();**

**}**

**private void getLongTermWeather() {**

**new LongTermWeatherTask(this, this, progressDialog).execute();**

**}**

**@SuppressLint("RestrictedApi")**

**private void searchCities() {**

**AlertDialog.Builder alert = new AlertDialog.Builder(this);**

**alert.setTitle(this.getString(R.string.search\_title));**

**final EditText input = new EditText(this);**

**input.setInputType(InputType.TYPE\_CLASS\_TEXT);**

**input.setMaxLines(1);**

**input.setSingleLine(true);**

**alert.setView(input, 32, 0, 32, 0);**

**alert.setPositiveButton(R.string.dialog\_ok, new DialogInterface.OnClickListener() {**

**public void onClick(DialogInterface dialog, int whichButton) {**

**String result = input.getText().toString();**

**if (!result.isEmpty()) {**

**saveLocation(result);**

**}**

**}**

**});**

**alert.setNegativeButton(R.string.dialog\_cancel, new DialogInterface.OnClickListener() {**

**public void onClick(DialogInterface dialog, int whichButton) {**

**// Cancelled**

**}**

**});**

**alert.show();**

**}**

**private void saveLocation(String result) {**

**SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(MainScreenActivity.this);**

**recentCity = preferences.getString("city", Constants.DEFAULT\_CITY);**

**SharedPreferences.Editor editor = preferences.edit();**

**editor.putString("city", result);**

**editor.commit();**

**if (!recentCity.equals(result)) {**

**// New location, update weather**

**getTodayWeather();**

**getLongTermWeather();**

**} }**

**private String setWeatherIcon(int actualId, int hourOfDay) {**

**int id = actualId / 100;**

**String icon = "";**

**if (actualId == 800) {**

**if (hourOfDay >= 7 && hourOfDay < 20) {**

**icon = this.getString(R.string.weather\_sunny);**

**} else {**

**icon = this.getString(R.string.weather\_clear\_night);**

**}**

**} else {**

**switch (id) {**

**case 2:**

**icon = this.getString(R.string.weather\_thunder);**

**break;**

**case 3:**

**icon = this.getString(R.string.weather\_drizzle);**

**break;**

**case 7:**

**icon = this.getString(R.string.weather\_foggy);**

**break;**

**case 8:**

**icon = this.getString(R.string.weather\_cloudy);**

**break;**

**case 6:**

**icon = this.getString(R.string.weather\_snowy);**

**break;**

**case 5:**

**icon = this.getString(R.string.weather\_rainy);**

**break;**

**}**

**}**

**return icon;**

**} public static String getRainString(JSONObject rainObj) {**

**String rain = "0";**

**if (rainObj != null) {**

**rain = rainObj.optString("3h", "fail");**

**if ("fail".equals(rain)) {**

**rain = rainObj.optString("1h", "0");**

**}**

**}**

**return rain;**

**}**

**private ParseResult parseTodayJson(String result) {**

**try {**

**JSONObject reader = new JSONObject(result);**

**final String code = reader.optString("cod");**

**if ("404".equals(code)) {**

**return ParseResult.CITY\_NOT\_FOUND;**

**}**

**String city = reader.getString("name");**

**String country = "";**

**JSONObject countryObj = reader.optJSONObject("sys");**

**if (countryObj != null) {**

**country = countryObj.getString("country");**

**todayWeather.setSunrise(countryObj.getString("sunrise"));**

**todayWeather.setSunset(countryObj.getString("sunset"));**

**}**

**todayWeather.setCity(city);**

**todayWeather.setCountry(country);**

**JSONObject coordinates = reader.getJSONObject("coord");**

**if (coordinates != null) {**

**SharedPreferences sp = PreferenceManager.getDefaultSharedPreferences(this);**

**sp.edit().putFloat("latitude", (float) coordinates.getDouble("lon")).putFloat("longitude", (float) coordinates.getDouble("lat")).commit();**

**}**

**JSONObject main = reader.getJSONObject("main");**

**todayWeather.setTemperature(main.getString("temp"));**

**todayWeather.setDescription(reader.getJSONArray("weather").getJSONObject(0).getString("description"));**

**JSONObject windObj = reader.getJSONObject("wind");**

**todayWeather.setWind(windObj.getString("speed"));**

**if (windObj.has("deg")) {**

**todayWeather.setWindDirectionDegree(windObj.getDouble("deg"));**

**} else {**

**Log.e("parseTodayJson", "No wind direction available");**

**todayWeather.setWindDirectionDegree(null);**

**}**

**todayWeather.setPressure(main.getString("pressure"));**

**todayWeather.setHumidity(main.getString("humidity"));**

**JSONObject rainObj = reader.optJSONObject("rain");**

**String rain;**

**if (rainObj != null) {**

**rain = getRainString(rainObj);**

**} else {**

**JSONObject snowObj = reader.optJSONObject("snow");**

**if (snowObj != null) {**

**rain = getRainString(snowObj);**

**} else {**

**rain = "0";**

**} }**

**todayWeather.setRain(rain);**

**final String idString = reader.getJSONArray("weather").getJSONObject(0).getString("id");**

**todayWeather.setId(idString);**

**todayWeather.setIcon(setWeatherIcon(Integer.parseInt(idString), Calendar.getInstance().get(Calendar.HOUR\_OF\_DAY)));**

**SharedPreferences.Editor editor = PreferenceManager.getDefaultSharedPreferences(MainScreenActivity.this).edit();**

**editor.putString("lastToday", result);**

**editor.commit();**

**} catch (JSONException e) {**

**Log.e("JSONException Data", result);**

**e.printStackTrace();**

**return ParseResult.JSON\_EXCEPTION;**

**}**

**return ParseResult.OK;**

**}**

**private void updateTodayWeatherUI() {**

**try {**

**if (todayWeather.getCountry().isEmpty()) {**

**preloadWeather();**

**return;**

**}**

**} catch (Exception e) {**

**preloadWeather();**

**return;**

**}**

**String city = todayWeather.getCity();**

**String country = todayWeather.getCountry();**

**DateFormat timeFormat = android.text.format.DateFormat.getTimeFormat(getApplicationContext());**

**getSupportActionBar().setTitle(city + (country.isEmpty() ? "" : ", " + country));**

**SharedPreferences sp = PreferenceManager.getDefaultSharedPreferences(MainScreenActivity.this); // Temperature**

**float temperature = UnitConvertor.convertTemperature(Float.parseFloat(todayWeather.getTemperature()), sp);**

**if (sp.getBoolean("temperatureInteger", false)) {**

**temperature = Math.round(temperature);**

**}// Rain**

**double rain = Double.parseDouble(todayWeather.getRain());**

**String rainString = UnitConvertor.getRainString(rain, sp);**

**// Wind**

**double wind;**

**try {**

**wind = Double.parseDouble(todayWeather.getWind());**

**} catch (Exception e) {**

**e.printStackTrace();**

**wind = 0;**

**}**

**wind = UnitConvertor.convertWind(wind, sp);**

**// Pressure**

**double pressure = UnitConvertor.convertPressure((float) Double.parseDouble(todayWeather.getPressure()), sp);**

**todayTemperature.setText(new DecimalFormat("0.#").format(temperature) + " " + sp.getString("unit", "°C"));**

**todayDescription.setText(todayWeather.getDescription().substring(0, 1).toUpperCase() +**

**todayWeather.getDescription().substring(1) + rainString);**

**if (sp.getString("speedUnit", "m/s").equals("bft")) {**

**todayWind.setText(getString(R.string.wind) + ": " +**

**UnitConvertor.getBeaufortName((int) wind) +**

**(todayWeather.isWindDirectionAvailable() ? " " + getWindDirectionString(sp, this, todayWeather) : ""));**

**} else {**

**todayWind.setText(getString(R.string.wind) + ": " + new DecimalFormat("#.0").format(wind) + " " +**

**localize(sp, "speedUnit", "m/s") +**

**(todayWeather.isWindDirectionAvailable() ? " " + getWindDirectionString(sp, this, todayWeather) : ""));**

**}**

**todayPressure.setText(getString(R.string.pressure) + ": " + new DecimalFormat("#.0").format(pressure) + " " +**

**localize(sp, "pressureUnit", "hPa"));**

**todayHumidity.setText(getString(R.string.humidity) + ": " + todayWeather.getHumidity() + " %");**

**todaySunrise.setText(getString(R.string.sunrise) + ": " + timeFormat.format(todayWeather.getSunrise()));**

**todaySunset.setText(getString(R.string.sunset) + ": " + timeFormat.format(todayWeather.getSunset()));**

**todayIcon.setText(todayWeather.getIcon());**

**}**

**public ParseResult parseLongTermJson(String result) {**

**int i;**

**try {**

**JSONObject reader = new JSONObject(result);**

**final String code = reader.optString("cod");**

**if ("404".equals(code)) {**

**if (longTermWeather == null) {**

**longTermWeather = new ArrayList<>();**

**longTermTodayWeather = new ArrayList<>();**

**longTermTomorrowWeather = new ArrayList<>();**

**}**

**return ParseResult.CITY\_NOT\_FOUND;**

**}**

**longTermWeather = new ArrayList<>();**

**longTermTodayWeather = new ArrayList<>();**

**longTermTomorrowWeather = new ArrayList<>();**

**JSONArray list = reader.getJSONArray("list");**

**for (i = 0; i < list.length(); i++) {**

**Weather weather = new Weather();**

**JSONObject listItem = list.getJSONObject(i);**

**JSONObject main = listItem.getJSONObject("main");**

**weather.setDate(listItem.getString("dt")); weather.setTemperature(main.getString("temp"));**

**weather.setDescription(listItem.optJSONArray("weather").getJSONObject(0).getString("description"));**

**JSONObject windObj = listItem.optJSONObject("wind");**

**if (windObj != null) {**

**weather.setWind(windObj.getString("speed"));**

**weather.setWindDirectionDegree(windObj.getDouble("deg"));**

**}**

**weather.setPressure(main.getString("pressure"));**

**weather.setHumidity(main.getString("humidity"));**

**JSONObject rainObj = listItem.optJSONObject("rain");**

**String rain = "";**

**if (rainObj != null) {**

**rain = getRainString(rainObj);**

**} else {**

**JSONObject snowObj = listItem.optJSONObject("snow");**

**if (snowObj != null) {**

**rain = getRainString(snowObj);**

**} else {**

**rain = "0";**

**}**

**}**

**weather.setRain(rain);**

**final String idString = listItem.optJSONArray("weather").getJSONObject(0).getString("id");**

**weather.setId(idString);**

**final String dateMsString = listItem.getString("dt") + "000";**

**Calendar cal = Calendar.getInstance();**

**cal.setTimeInMillis(Long.parseLong(dateMsString));**

**weather.setIcon(setWeatherIcon(Integer.parseInt(idString), cal.get(Calendar.HOUR\_OF\_DAY)));**

**Calendar today = Calendar.getInstance();**

**if (cal.get(Calendar.DAY\_OF\_YEAR) == today.get(Calendar.DAY\_OF\_YEAR)) {**

**longTermTodayWeather.add(weather);**

**} else if (cal.get(Calendar.DAY\_OF\_YEAR) == today.get(Calendar.DAY\_OF\_YEAR) + 1) {**

**longTermTomorrowWeather.add(weather);**

**} else {**

**longTermWeather.add(weather);**

**}**

**}**

**SharedPreferences.Editor editor = PreferenceManager.getDefaultSharedPreferences(MainScreenActivity.this).edit();**

**editor.putString("lastLongterm", result);**

**editor.commit();**

**} catch (JSONException e) {**

**Log.e("JSONException Data", result);**

**e.printStackTrace();**

**return ParseResult.JSON\_EXCEPTION;**

**}**

**return ParseResult.OK;**

**}**

**private void updateLongTermWeatherUI() {**

**if (destroyed) {**

**return;**

**}**

**ViewPagerAdapter viewPagerAdapter = new ViewPagerAdapter(getSupportFragmentManager());**

**Bundle bundleToday = new Bundle();**

**bundleToday.putInt("day", 0);**

**SeasonFragment seasonFragmentToday = new SeasonFragment();**

**seasonFragmentToday.setArguments(bundleToday);**

**viewPagerAdapter.addFragment(seasonFragmentToday, getString(R.string.today));**

**Bundle bundleTomorrow = new Bundle();**

**bundleTomorrow.putInt("day", 1);**

**SeasonFragment seasonFragmentTomorrow = new SeasonFragment();**

**seasonFragmentTomorrow.setArguments(bundleTomorrow);**

**viewPagerAdapter.addFragment(seasonFragmentTomorrow, getString(R.string.tomorrow));**

**Bundle bundle = new Bundle();**

**bundle.putInt("day", 2);**

**SeasonFragment seasonFragment = new SeasonFragment();**

**seasonFragment.setArguments(bundle);**

**viewPagerAdapter.addFragment(seasonFragment, getString(R.string.later));**

**int currentPage = viewPager.getCurrentItem();**

**viewPagerAdapter.notifyDataSetChanged();**

**viewPager.setAdapter(viewPagerAdapter);**

**tabLayout.setupWithViewPager(viewPager);**

**if (currentPage == 0 && longTermTodayWeather.isEmpty()) {**

**currentPage = 1;**

**}**

**viewPager.setCurrentItem(currentPage, false);**

**}**

**private boolean isNetworkAvailable() {**

**ConnectivityManager connectivityManager = (ConnectivityManager) getSystemService(Context.CONNECTIVITY\_SERVICE);**

**NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();**

**return activeNetworkInfo != null && activeNetworkInfo.isConnected();**

**}**

**private boolean shouldUpdate() {**

**long lastUpdate = PreferenceManager.getDefaultSharedPreferences(this).getLong("lastUpdate", -1);**

**boolean cityChanged = PreferenceManager.getDefaultSharedPreferences(this).getBoolean("cityChanged", false);**

**// Update if never checked or last update is longer ago than specified threshold**

**return cityChanged || lastUpdate < 0 || (Calendar.getInstance().getTimeInMillis() - lastUpdate) > NO\_UPDATE\_REQUIRED\_THRESHOLD;**

**}**

**@Override**

**public boolean onCreateOptionsMenu(Menu menu) {**

**getMenuInflater().inflate(R.menu.menu\_main, menu);**

**return true;**

**}**

**@Override**

**public boolean onOptionsItemSelected(MenuItem item) {**

**int id = item.getItemId();**

**if (id == R.id.action\_refresh) {**

**if (isNetworkAvailable()) {**

**getTodayWeather();**

**getLongTermWeather();**

**} else {**

**Snackbar.make(appView, getString(R.string.msg\_connection\_not\_available), Snackbar.LENGTH\_LONG).show();**

**}**

**return true;**

**}**

**if (id == R.id.action\_map) {**

**Intent intent = new Intent(MainScreenActivity.this, MapActivity.class);**

**startActivity(intent);**

**}**

**if (id == R.id.action\_graphs) {**

**Intent intent = new Intent(MainScreenActivity.this, GraphActivity.class);**

**startActivity(intent);**

**}**

**if (id == R.id.action\_search) {**

**searchCities();**

**return true;**

**}**

**if (id == R.id.action\_location) {**

**getCityByLocation();**

**return true;**

**}**

**/\* if (id == R.id.action\_settings) {**

**Intent intent = new Intent(MainScreenActivity.this, SettingsActivity.class);**

**startActivity(intent);**

**}\*/**

**return super.onOptionsItemSelected(item);**

**}**

**public static void initMappings() {**

**if (mappingsInitialised)**

**return;**

**mappingsInitialised = true;**

**speedUnits.put("m/s", R.string.speed\_unit\_mps);**

**speedUnits.put("kph", R.string.speed\_unit\_kph);**

**speedUnits.put("mph", R.string.speed\_unit\_mph);**

**speedUnits.put("kn", R.string.speed\_unit\_kn);**

**pressUnits.put("hPa", R.string.pressure\_unit\_hpa);**

**pressUnits.put("kPa", R.string.pressure\_unit\_kpa);**

**pressUnits.put("mm Hg", R.string.pressure\_unit\_mmhg);**

**}**

**private String localize(SharedPreferences sp, String preferenceKey, String defaultValueKey) { return localize(sp, this, preferenceKey, defaultValueKey);**

**}**

**public static String localize(SharedPreferences sp, Context context, String preferenceKey, String defaultValueKey) {**

**String preferenceValue = sp.getString(preferenceKey, defaultValueKey);**

**String result = preferenceValue;**

**if ("speedUnit".equals(preferenceKey)) {**

**if (speedUnits.containsKey(preferenceValue)) {**

**result = context.getString(speedUnits.get(preferenceValue));**

**}**

**} else if ("pressureUnit".equals(preferenceKey)) {**

**if (pressUnits.containsKey(preferenceValue)) {**

**result = context.getString(pressUnits.get(preferenceValue));**

**}**

**}**

**return result;**

**}**

**public static String getWindDirectionString(SharedPreferences sp, Context context, Weather weather) {**

**try {**

**if (Double.parseDouble(weather.getWind()) != 0) {**

**String pref = sp.getString("windDirectionFormat", null);**

**if ("arrow".equals(pref)) {**

**return weather.getWindDirection(8).getArrow(context);**

**} else if ("abbr".equals(pref)) {**

**return weather.getWindDirection().getLocalizedString(context);**

**}**

**}**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**return "";**

**}**

**void getCityByLocation() {**

**locationManager = (LocationManager) getSystemService(LOCATION\_SERVICE);**

**if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS\_FINE\_LOCATION) != PackageManager.PERMISSION\_GRANTED) {**

**if (ActivityCompat.shouldShowRequestPermissionRationale(this,**

**Manifest.permission.ACCESS\_FINE\_LOCATION)) {**

**// Explanation not needed, since user requests this themmself**

**} else {**

**ActivityCompat.requestPermissions(this,**

**new String[]{Manifest.permission.ACCESS\_FINE\_LOCATION},**

**MY\_PERMISSIONS\_ACCESS\_FINE\_LOCATION);**

**}**

**} else if (locationManager.isProviderEnabled(LocationManager.NETWORK\_PROVIDER) ||**

**locationManager.isProviderEnabled(LocationManager.GPS\_PROVIDER)) {**

**progressDialog = new ProgressDialog(this);**

**progressDialog.setMessage(getString(R.string.getting\_location));**

**progressDialog.setCancelable(false);**

**progressDialog.setButton(DialogInterface.BUTTON\_NEGATIVE, getString(R.string.dialog\_cancel), new DialogInterface.OnClickListener() {**

**@Override**

**public void onClick(DialogInterface dialogInterface, int i) {**

**try {**

**locationManager.removeUpdates(MainScreenActivity.this);**

**} catch (SecurityException e) {**

**e.printStackTrace();**

**}**

**}**

**});**

**progressDialog.show();**

**if (locationManager.isProviderEnabled(LocationManager.NETWORK\_PROVIDER)) {**

**locationManager.requestLocationUpdates(LocationManager.NETWORK\_PROVIDER, 0, 0, this);**

**}**

**if (locationManager.isProviderEnabled(LocationManager.GPS\_PROVIDER)) {**

**locationManager.requestLocationUpdates(LocationManager.GPS\_PROVIDER, 0, 0, this);**

**}**

**} else {**

**showLocationSettingsDialog();**

**}**

**}**

**private void showLocationSettingsDialog() {**

**AlertDialog.Builder alertDialog = new AlertDialog.Builder(this);**

**alertDialog.setTitle(R.string.location\_settings);**

**alertDialog.setMessage(R.string.location\_settings\_message);**

**alertDialog.setPositiveButton(R.string.location\_settings\_button, new DialogInterface.OnClickListener() {**

**public void onClick(DialogInterface dialog, int which) {**

**Intent intent = new Intent(Settings.ACTION\_LOCATION\_SOURCE\_SETTINGS);**

**startActivity(intent);**

**}**

**});**

**alertDialog.setNegativeButton(R.string.dialog\_cancel, new DialogInterface.OnClickListener() {**

**public void onClick(DialogInterface dialog, int which) {**

**dialog.cancel();**

**}**

**});**

**alertDialog.show();**

**}**

**@Override**

**public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {**

**switch (requestCode) {**

**case MY\_PERMISSIONS\_ACCESS\_FINE\_LOCATION: {**

**// If request is cancelled, the result arrays are empty.**

**if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED) {**

**getCityByLocation();**

**}**

**return;**

**}**

**}**

**}**

**@Override**

**public void onLocationChanged(Location location) {**

**progressDialog.hide();**

**try {**

**locationManager.removeUpdates(this);**

**} catch (SecurityException e) {**

**Log.e("LocationManager", "Error while trying to stop listening for location updates. This is probably a permissions issue", e);**

**}**

**Log.i("LOCATION (" + location.getProvider().toUpperCase() + ")", location.getLatitude() + ", " + location.getLongitude());**

**double latitude = location.getLatitude();**

**double longitude = location.getLongitude();**

**new ProvideCityNameTask(this, this, progressDialog).execute("coords", Double.toString(latitude), Double.toString(longitude));**

**}**

**@Override**

**public void onStatusChanged(String provider, int status, Bundle extras) {**

**}**

**@Override**

**public void onProviderEnabled(String provider) {**

**}**

**@Override**

**public void onProviderDisabled(String provider) {**

**}**

**class TodayWeatherTask extends GenericRequestTask {**

**public TodayWeatherTask(Context context, MainScreenActivity activity, ProgressDialog progressDialog) {**

**super(context, activity, progressDialog);**

**}**

**@Override**

**protected void onPreExecute() {**

**loading = 0;**

**super.onPreExecute();**

**}**

**@Override**

**protected void onPostExecute(TaskOutput output) {**

**super.onPostExecute(output);**

**// Update widgets**

**AbstractWidgetProvider.updateWidgets(MainScreenActivity.this);**

**DashClockWeatherExtension.updateDashClock(MainScreenActivity.this);**

**}**

**@Override**

**protected ParseResult parseResponse(String response) {**

**return parseTodayJson(response);**

**}**

**@Override**

**protected String getAPIName() {**

**return "weather";**

**}**

**@Override**

**protected void updateMainUI() {**

**updateTodayWeatherUI();**

**updateLastUpdateTime();**

**}**

**}**

**class LongTermWeatherTask extends GenericRequestTask {**

**public LongTermWeatherTask(Context context, MainScreenActivity activity, ProgressDialog progressDialog) {**

**super(context, activity, progressDialog);**

**}**

**@Override**

**protected ParseResult parseResponse(String response) {**

**return parseLongTermJson(response);**

**}**

**@Override**

**protected String getAPIName() {**

**return "forecast";**

**}**

**@Override**

**protected void updateMainUI() {**

**updateLongTermWeatherUI();**

**}**

**}**

**class ProvideCityNameTask extends GenericRequestTask {**

**public ProvideCityNameTask(Context context, MainScreenActivity activity, ProgressDialog progressDialog) {**

**super(context, activity, progressDialog);**

**}**

**@Override**

**protected void onPreExecute() { /\*Nothing\*/ }**

**@Override**

**protected String getAPIName() {**

**return "weather";**

**}**

**@Override**

**protected ParseResult parseResponse(String response) {**

**Log.i("RESULT", response.toString());**

**try {**

**JSONObject reader = new JSONObject(response);**

**final String code = reader.optString("cod");**

**if ("404".equals(code)) {**

**Log.e("Geolocation", "No city found");**

**return ParseResult.CITY\_NOT\_FOUND;**

**}**

**String city = reader.getString("name");**

**String country = "";**

**JSONObject countryObj = reader.optJSONObject("sys");**

**if (countryObj != null) {**

**country = ", " + countryObj.getString("country");**

**}**

**saveLocation(city + country);**

**} catch (JSONException e) {**

**Log.e("JSONException Data", response);**

**e.printStackTrace();**

**return ParseResult.JSON\_EXCEPTION;**

**}**

**return ParseResult.OK;**

**}**

**@Override**

**protected void onPostExecute(TaskOutput output) {**

**/\* Handle possible errors only \*/**

**handleTaskOutput(output);**

**}**

**}**

**public static long saveLastUpdateTime(SharedPreferences sp) {**

**Calendar now = Calendar.getInstance();**

**sp.edit().putLong("lastUpdate", now.getTimeInMillis()).apply();**

**return now.getTimeInMillis();**

**}**

**private void updateLastUpdateTime() {**

**updateLastUpdateTime(**

**PreferenceManager.getDefaultSharedPreferences(this).getLong("lastUpdate", -1)**

**);**

**}**

**private void updateLastUpdateTime(long timeInMillis) {**

**if (timeInMillis < 0) {**

**// No time**

**lastUpdate.setText("");**

**} else {**

**lastUpdate.setText(getString(R.string.last\_update, formatTimeWithDayIfNotToday(this, timeInMillis)));**

**}**

**}**

**public static String formatTimeWithDayIfNotToday(Context context, long timeInMillis) {**

**Calendar now = Calendar.getInstance();**

**Calendar lastCheckedCal = new GregorianCalendar();**

**lastCheckedCal.setTimeInMillis(timeInMillis);**

**Date lastCheckedDate = new Date(timeInMillis);**

**String timeFormat = android.text.format.DateFormat.getTimeFormat(context).format(lastCheckedDate);**

**if (now.get(Calendar.YEAR) == lastCheckedCal.get(Calendar.YEAR) &&**

**now.get(Calendar.DAY\_OF\_YEAR) == lastCheckedCal.get(Calendar.DAY\_OF\_YEAR)) {**

**// Same day, only show time**

**return timeFormat;**

**} else {**

**return android.text.format.DateFormat.getDateFormat(context).format(lastCheckedDate) + " " + timeFormat;**

**}**

**}**

**private int getTheme(String themePref) {**

**switch (themePref) {**

**/\*case "dark":**

**return R.style.AppTheme\_NoActionBar\_Dark;**

**case "classic":**

**return R.style.AppTheme\_NoActionBar\_Classic;**

**case "classicdark":**

**return R.style.AppTheme\_NoActionBar\_Classic\_Dark;\*/**

**default:**

**return R.style.AppTheme\_NoActionBar;**

**}**

**}**

**}**

**CHAPTER 4**

**CONCLUSION**

Agri Aiuto Application helps the farmers to know the information related to the agriculture as it is the backbone of Indian Economy. It provides the information about the type of seeds, water required for plantation, and which soil the crop needs to be planted. The information accessed could be understood by the user through the TextToSpeech Conversion for crops. This application also provides weather forecast for making the farmers to plant the crop according to the seasonal conditions. Pesticide information is provided to have an idea about which one is needed to be used for which crop. Farmers can know the merchant detail so that they can sell their crops at their nearest location. Farmers can solve their agriculture related problems by contacting with the experts through messages.

**REFERENCES**

[1] Steinberger, G., Rothmund, M., & Auernhammer, H. (2009) ―Mobile farm equipment as a data source in an agricultural service architecture‖, Computers and Electronics in Agriculture, 65(2), 238-246.

[2] Patel, V. B., Thakkar, R. G., & Radadiya, B. L. (2014) ‖ An Android Application for Farmers to Disseminate Horticulture Information‖, International Journal of Computer Applications, 88(4)

[3] <https://ieeexplore.ieee.org/document/6965213/>

[5] <http://ieeexplore.ieee.org/abstract/document/8073690/>

[6] <http://ieeexplore.ieee.org/document/7358528/>