VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590 018

Mini Project Report

on

"Recipe Application"

Submitted in partial fulfillment of the requirement for the curriculum of the 6th semester

Bachelor of Engineering
In
Computer Science & Engineering

by

Chirag Radhakrishna (1VI19CS021)
Chiranjeevi PR (1VI19CS022)

Under the supervision of

Ms. Veena G
Assistant Professor

Dr. Kantharaju H C
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
VEMANA INSTITUTE OF TECHNOLOGY
Bengaluru-560034
2021-2022

Karnataka Reddyjana Sangha®

VEMANA INSTITUTE OF TECHNOLOGY

Koramangala, Bengaluru-34.

(Affiliated to Visvesvaraya Technological University, Belagavi)

Department of Computer Science & Engineering

Certificate

This is to certify that Mobile Application Development mini project work entitled **"Recipe Application"** is a bonafide work carried out by **Chirag Radhakrishna (1VI19CS021) and Chiranjeevi PR (1VI19CS022)** during the academic year 2021-22 in partial fulfillment of the award of **Bachelor of Engineering** in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belagavi. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The mini project report has been approved as it satisfies the academic requirements in respect of the mini project work prescribed for the said degree.

Guide	**Guide**	**HOD**
Ms. Veena G	Dr. Kantharaju H C	Dr. M Ramakrishna

External Viva

Name of the Examiners	Signature with date
1.____________________	1.____________________
2.____________________	2.____________________

ACKNOWLEDGEMENT

We sincerely thank **Visvesvaraya Technological University** for providing a platform to do the mini project.

Firstly, we would like to express our deep sense of gratitude to our institute "**VEMANA INSTITUTE OF TECHNOLOGY**" that provided us an opportunity to do a mini project entitled **"Recipe Application"**.

We thank **Dr. Vijayasimha Reddy. B.G,** Principal, Vemana Institute of Technology, Bengaluru for providing the necessary support.

We would like to place on record our regards to **Dr. M. Ramakrishna,** Head of the Department Computer Science and Engineering for his continued support.

We would like to thank our project guide **Ms. Veena G,** Assistant Professor, Dept. of CSE and **Dr. Kantharaju H C,** Associate Professor, Dept. of CSE for their continuous support and valuable guidance towards successful completion of the project.

We would be failing in our duty if we do not thank the faculty members, lab staffs, technicians and family members for their constant support and guidance.

Date: **Chirag Radhakrishna (1VI19CS021)**

Place: BANGALORE **Chiranjeevi PR (1VI19CS022)**

ABSTRACT

The advancement of technology has made our lives easier than ever before. Technology has had a profound impact on several fields/sectors and it is no surprise when we see how it has revolutionized the food industry, enabling amateur/simple home cooks to prepare different cuisines using everyday ingredients available at home. Our Recipe Application “ReciCheese” provides a step-by-step instruction guide to the user to recreate dishes of various cuisines, with the equipments and ingredients required at each step also mentioned. The user is also provided with the option of searching recipes by category or searching recipes of their choice.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Android OS-

Android is an Operating System (OS) and programming environment developed by Google for smartphones and other mobile device like tablets. It is developed by Open Handset Alliance. Open Handset Alliance are a consortium of developers. Android Operating System is commercially sponsored by Google.

Android tops the charts in terms of sales on smartphones and tablets since the early 2010's. Currently, it has more than three billion monthly active users.

The OS was unveiled on the first commercial Android Device, the HTC Dream, which was launched in September 2008. The latest version of Android OS is Android 12, which released on October 4, 2021.

Fig. 1.1 Android OS

Android Versions:

The version history of the Android mobile operating system began with the public release of the Android beta on November 5, 2007. The first commercial version, Android 1.0, was released on September 23, 2008.

At first, the Android versions were named using a confectionary(food) themed naming scheme. In 2019, this scheme was abandoned and a numeric naming scheme was initiated.

List of Android Versions-

a. Android 1.0
b. Android 1.1
c. Android Cupcake
d. Android Donut
e. Android Éclair
f. Android Froyo
g. Android Gingerbread
h. Android Honeycomb
i. Android Ice Cream Sandwich
j. Android Jelly Bean
k. Android KitKat
l. Android Lollipop
m. Android Marshmallow
n. Android Nougat
o. Android Oreo
p. Android Pie
q. Android 10
r. Android 11
s. Android 12
t. Android 12L
u. Android 13

Android Stack-

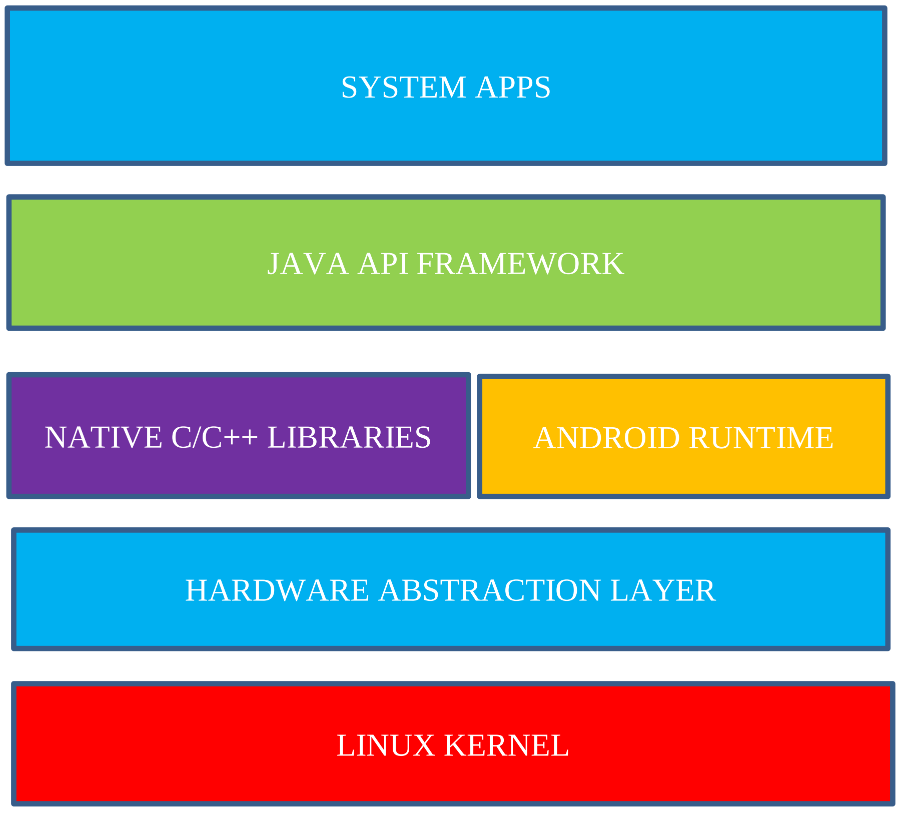


Fig. 1.2 Android Stack

Android is an open source, Linux-based software stack created for a wide array of devices. It comprises of the following components:

1. Linux Kernel: A key foundation piece of the Android Platform. It helps Android to take advantage of key security features.
2. Hardware Abstraction Layer [HAL]: provides a standard interface that helps sync hardware capacities of a device to the high-level Java API framework. Consists of library modules that help implement an interface for hardware components.
3. Android Runtime [ART]: is the managed runtime used by applications and some system services on Android.
4. Native C/C++ Libraries: Core Android components like HAL and ART are built from native code which are based on native libraries written in C/C++. The functionality of these libraries is made use of by the Java framework APIs.
5. Java API Framework: Form the essential components that are needed to develop android apps. Constitutes a rich modular system and services like:

Resource Manager: provides access to strings, layout files and vector assets.

Notification Manager: displays alerts on status bar.

Activity Manager: manages the lifecycle of the app.

6. System Apps: Android comes with a set of core apps for various activities like messaging, browsing, contacts etc.

1.2 Android Studio-

Android Studio is the official IDE for Google's Android Operating System. It is built on JetBrains' IntelliJ IDEA software.

The main purpose of Android Studio is for the development of Android applications.

Languages supported by Android Studio:

1.Kotlin

2.Java

3.C++

At present, Kotlin is the preferred language for Android development.

Features:

1. Visual Layout Editor: Helps create app designs by providing different layouts and views. A preview of the app layout can be seen using the preview window.
2. APK Analyzer: Helps reduce the size of your application by carrying out an inspection of the app contents.
3. Emulator: Helps install and run applications and also simulate different configurations and features.
4. Intelligent Code Editor: Provides code completion that helps write robust, efficient and productive code using either one of the Kotlin/Java/C++ languages.
5. Flexible Build System: Android Studio provides Gradle that helps customize your build to generate different builds for different devices.
6. Realtime Profilers: Provides information and statistics regarding the application's CPU, memory and network activity. Helps identify deficiencies in the performance of the app.

Once an app has been compiled with Android Studio, it can be published on the Google Play Store.

1.3 Project Introduction:

The food industry is one of the most important in the world. It generates billions of dollars in revenues per year. Different parts of the world offer different cuisines that constitute dishes that highlight the local/native produce. With the advancement of technology, the food industry has also improvised at a rapid rate. Recipes of popular dishes or recipes by famous cooks/Michelin star restaurants that were only available through cookbooks/magazines are now available at any instant of time through recipe applications. This has helped home cooks recreate some of their favourite dishes with ease by making use of any electronic device. Amateur cooks are also benefitted as they can easily learn to cook different dishes.

Our Recipe Application “ReciCheese”, is an application designed to provide users (home cooks) with a step-by-step instruction guide to recreate dishes that belong to various cuisines like Chinese, Continental, Mexican etc**.**

The user can select one of the dishes displayed on the screen, or can search for the dish of his/her choice or can select a dish based on a particular category like bread, beverage, soups, finger-food, snacks etc.
The user is also provided with a list of ingredients that constitute the dish, equipments required at each step, and can also get to know the preparation time, the number of servings and the number of people who have viewed the recipe.
This application is robust, user-friendly, yet easy and simple to use.

APPLICATION PROCESS OVERVIEW:

- On opening the application, the user can view a set of dishes that are generated in a random order.
- Information pertaining to the dish is provided which includes the name of the dish, the number of servings, the number of views and the preparation time.
- The user can also search dishes by making use of the search-box facility or can access the drop-down menu to view the different food categories like finger-food, drinks, soups etc and can select a particular category. On selection of a category, related dishes will be displayed.

- On selecting a particular dish, the user can see an image of the dish and the outlet/source that has published the recipe. He/she can also view a small description of the dish, the ingredients required and the step-by-step instructions to recreate the dish.
- For each step, the instructions and ingredients required at that particular step are also provided giving the user a well-detailed guide to recreate the dish.

E-Learning:
Since our application enables users (home cooks) to cook dishes by looking up the instruction guide, it can be classified under the category of electronic learning (E-Learning).

E-Learning is a flexible term used to describe a means of teaching through technology using any electronic device.

CHAPTER 2

System Requirement Specification

2.1 Minimum Hardware Requirements

- x86_64 CPU architecture.
- 8 GB RAM or more.
- 1280 x 800 minimum screen resolution.

2.2 Minimum Software Requirements

- 64-bit Microsoft Windows 8/10/11 or Mac OS 10.14 (Mojave) or higher.
- 8 GB of available disk space minimum to incorporate the IDE, Android SDK and Android Emulator

CHAPTER 3

DESIGN

The application has three major functionalities. The user can browse recipes according to a specific category, can browse a recipe for a specific dish or can pick a recipe from a randomly generated list of recipes.

Application Flow-Chart:

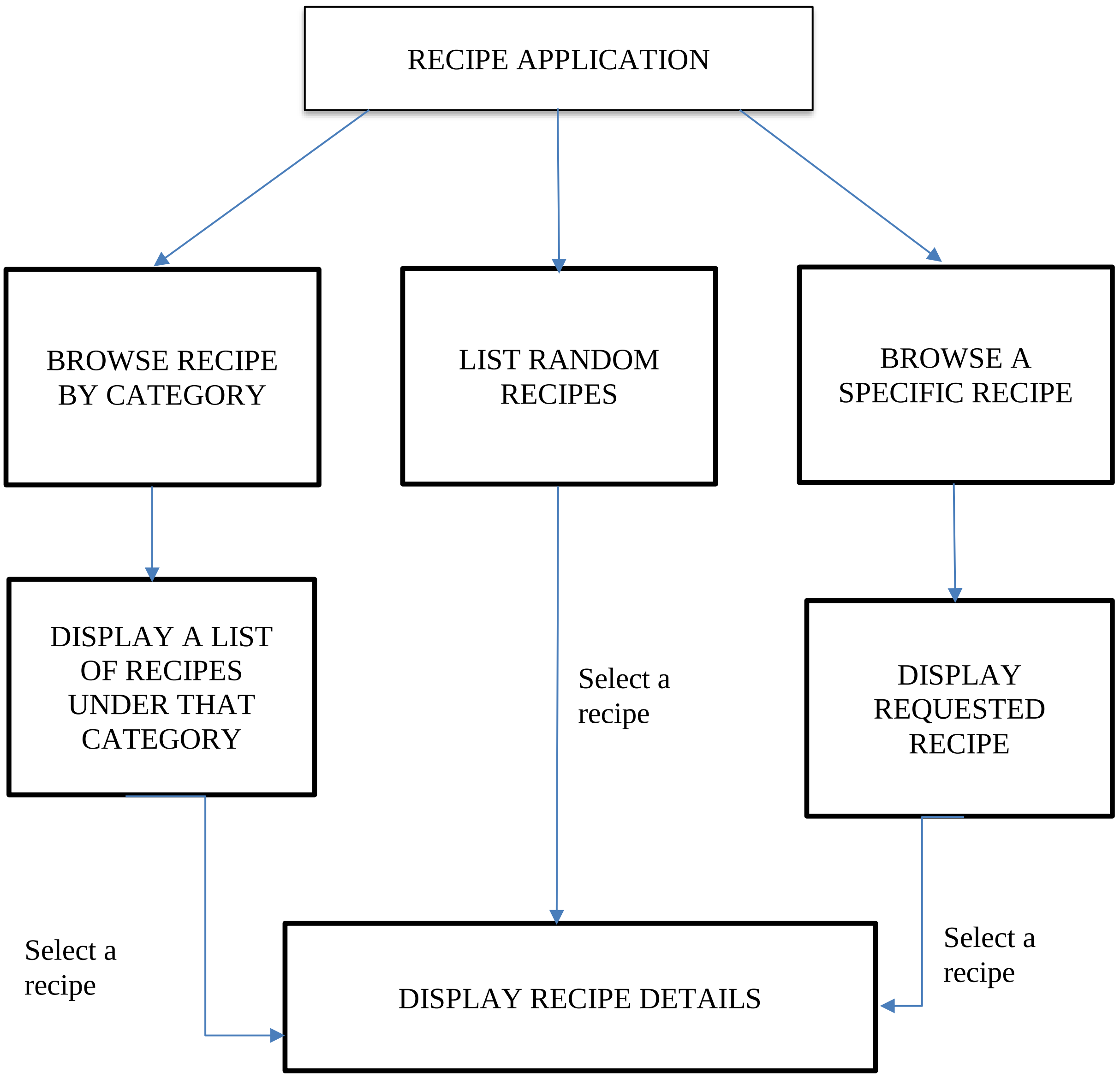


Fig. 3.1 App Flow-Chart

Our application design involves a total of 8 activities (Layout Resource Files). Each of these 8 activities have been designed using Extensible Markup Language (XML).

XML-

XML is a markup language that is similar to HTML, but does not make use of predefined tags. Tags can be designed accordingly for the purpose.

The user interface of our Android Application is designed using XML. All visual aspects have been specified using XML.

Table 3.1: List of activities

Activity Number	Activity
1.	activity_main.xml
2.	activity_recipe_details.xml
3.	list_instruction_step_items.xml
4.	list_instructions.xml
5.	list_instructions_steps.xml
6.	list_meal_ingredients.xml
7.	list_random_recipe.xml
8.	spinner_inner_text.xml

1. activity_main.xml:

This activity comprises the main interface of the application. It constitutes the application name, a search dialog box and a drop-down chooser for selecting recipes by category.
It also has 9 placeholders to contain 9 random recipe dishes.

The drop-down menu chooser has been implemented using a spinner element.

The search box has been implemented using a searchview element.

The placeholders for the recipe dishes have been implemented using recycler view element.

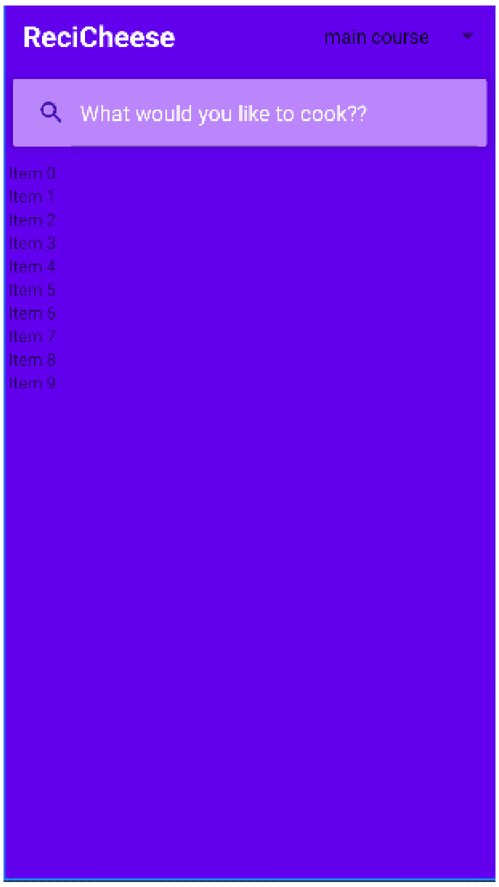


Fig. 3.2 activity_main.xml design

The above figure shows the design structure of the home page of the recipe application.

2. list_random_recipe.xml:

Acts as placeholder for the recipe items. Can contain a random recipe, or a specific recipe that the user has selected.

The previous activity comprised of placeholders for individual recipes. This activity constitutes the design for a recipe item. It provides the structure for the image and basic information like number of servings, views and preparation time.

The design is achieved by nesting textviews and imageviews within a linear layout. The respective icons for the number of servings, views and preparation time have been implemented using vector assets. The icons are then stored in the drawable folder and specified as the source for that particular imageview.

This is replicated for each individual recipe item.

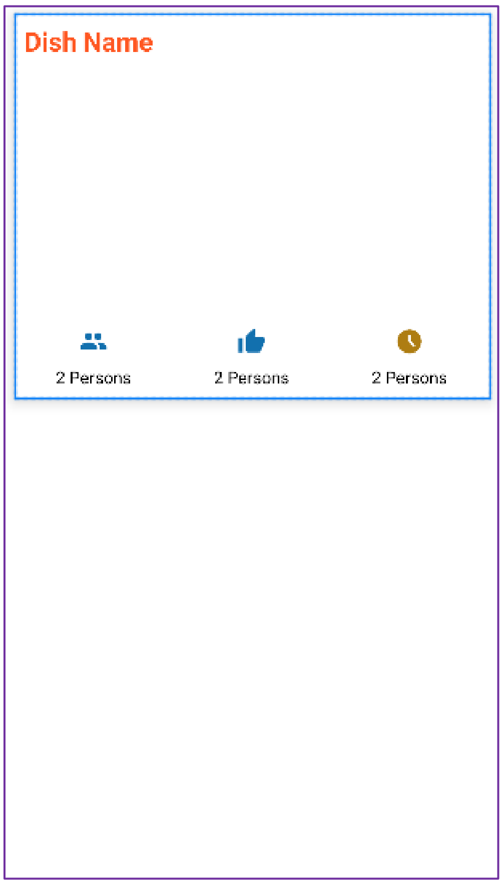


Fig. 3.3 list_random_recipe.xml design

3.spinner_inner_text.xml

The application comprises of a drop-down menu chooser that enables the user to filter recipe items based on categories like desserts, soups and finger-food to name a few. This chooser facility is rendered on the home page of the application. This activity creates the layout that specifies the elements that will be constituted within the spinner element that renders the chooser.

The menu-items are contained in the string.xml file.

Hence this activity will be constituted within the activity_main.xml.

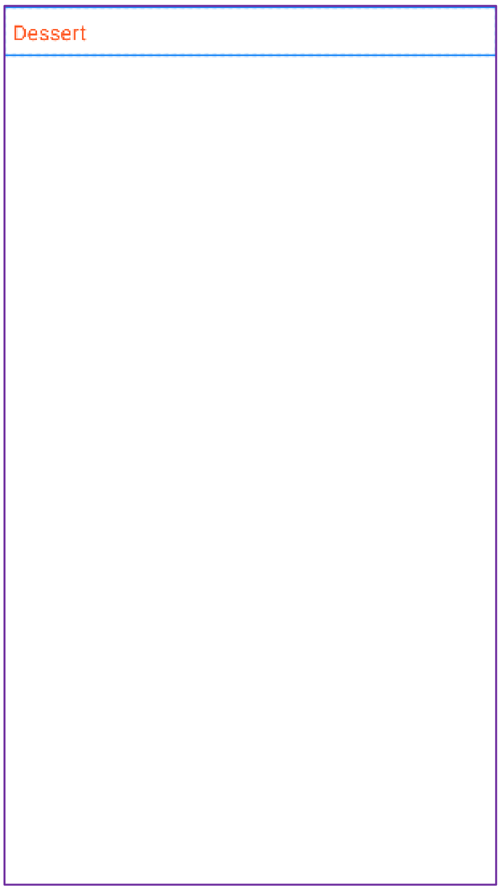


Fig. 3.4 spinner_inner_text.xml design

4.activity_recipe_details.xml

This activity provides the structure for the recipe details.

It has five placeholders for:

a) The recipe name and the source.
b) The dish image.
c) A description of the dish (summary).
d) Step-by-step instructions
e) List of ingredients.

The activity is made scrollable to ensure a smooth usage of the application.

The textviews and imageviews have been nested within a linear layout in order to ensure that the related recipe details are grouped together accordingly.

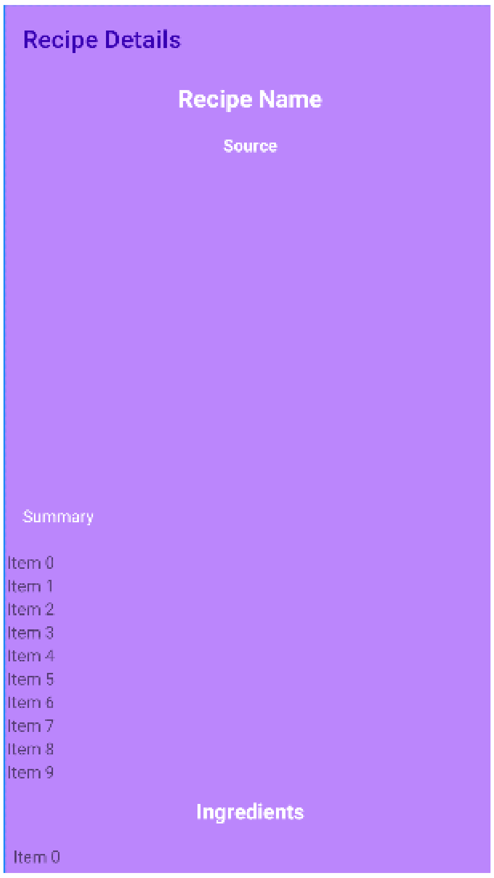


Fig. 3.5 activity_recipe_details.xml design

5.list_instructions.xml

This activity provides the layout for the step-by-step instructions. This activity will have placeholders for the instructions, and the equipments and ingredients that will be required at each step.

Item 0-Item 9 will be used to display an individual step along with the associated equipments and ingredients.

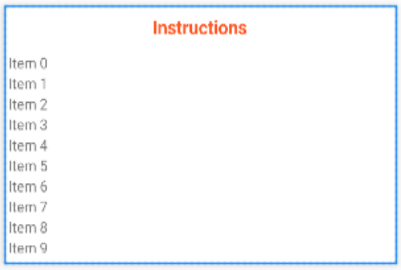


Fig. 3.6 list_instructions.xml design

6.list_instruction_steps.xml

This activity helps design the layout to contain the equipments and ingredients that are associated with a particular step. Has placeholders for the equipments and ingredients which are implemented using recycler views.

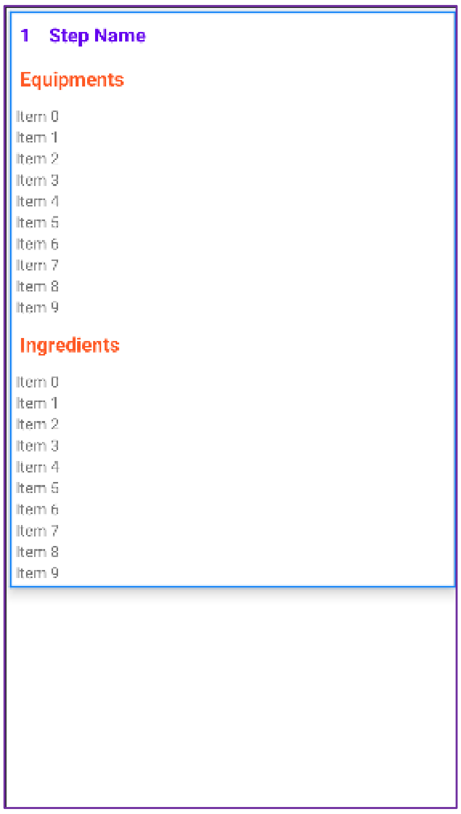


Fig. 3.7 list_instruction_steps.xml design

7.list_instruction_step_items.xml:

The application provides the equipments and ingredients required at each step. This activity provides the layout for containing the equipments and ingredients associated with that particular step. And it will feed these resources to the list_instructions_steps.xml activity.

The placeholder is essentially a card view which has a textview that contains the name of the equipment/ ingredient and an imageview which will render the respective image of these resources.

The textview has its marquee property enabled to ensure that large names can be viewed in the form of a moving text for better understanding.

This activity will feed the list_instruction_steps.xml activity.

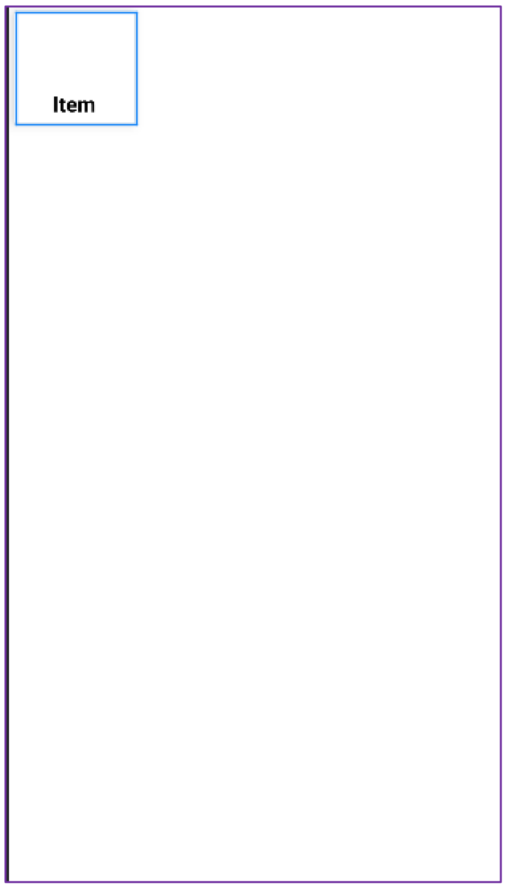


Fig. 3.8 list_instruction_step_items.xml design

8.list_meal_ingredients.xml

The details of the recipe also comprise a complete list of ingredients that are required to prepare the dish. The activity recipe _details has a placeholder for the ingredients. This activity provides placeholders for each individual ingredient.

The placeholders have been implemented using a cardview which comprises of two textviews to display the ingredient name and the quantity and an imageview to display the ingredient image.

The textviews have their marquee property enabled thereby allowing large names to be displayed like a moving text for better understanding.

With respect to the placeholder design, a sample ingredient name "chicken" with a quantity has been provided for ensuring a compact design and efficient interface that the user can access easily.

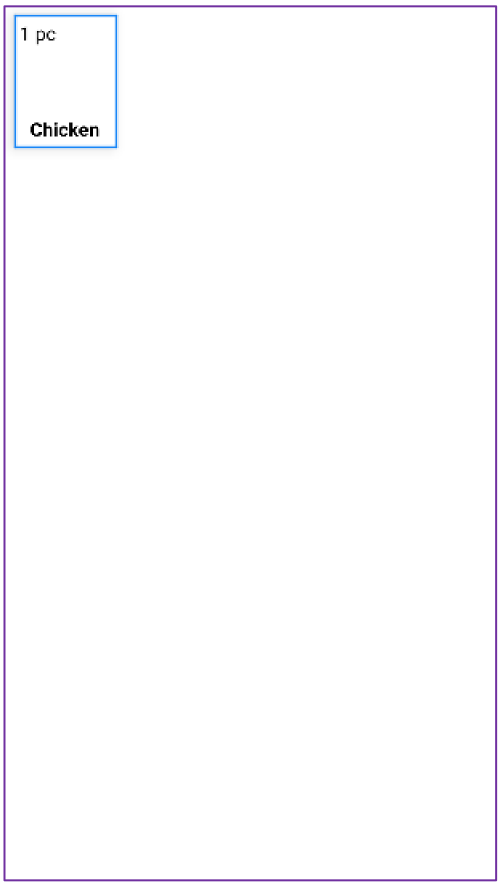


Fig. 3.9 list_meal_ingredients.xml design

CHAPTER 4

IMPLEMENTATION

In the design of this recipe application, we have created placeholders for the image, the number of servings, preparation time, instructions, ingredients that are associated with each recipe. The respective data for each of these parameters are fetched from an Application Programming Interface: Spoonacular API.

By using our own API key, we obtain the data from this API. The corresponding data is fetched and placed into their respective placeholders and will be displayed to the user.

In order to access this API at the time of a request, we need to ensure that the application has permission to access the Internet.
For this we add a permission regarding the same in the Android Manifest File.

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

By adding this permission, the application can access the Spoonacular API using the Internet.

SPOONACULAR API:

It is an API that provides data related to food, menu items, nutrition values and ingredients. It also helps understand the relation between these components.

For our application we have made use of the random recipe listener. This enables our application to render details of a plethora of recipes that increases the scope of our application. The interaction with the API takes place through GET requests.

```
GET https://api.spoonacular.com/recipes/random
```

We can access random recipes using the above GET request. The other key component required to facilitate these requests is an API KEY. We have generated our own API key for this purpose.

The key is used in every request that we make, thereby making it a key component that helps in servicing requests.

?apiKey= b3115b6f519c4670a72b247318da4e3b

Where b3115b6f519c4670a72b247318da4e3b is our own API Key.

We also specify the API key in the strings.xml file.

```
<resources>

    <string name="api_key">b3115b6f519c4670a72b247318da4e3b</string>

</resources>
```

With the API and the respective API key, we also need to include certain dependencies in Gradle,

1) Retrofit manages the process of receiving, sending, and creating HTTP requests and responses. It caches responses to avoid sending duplicate requests.
2) With retrofit creating the HTTP responses, the built-in GSON converter will automatically parse HTTP response into an object or any other types in Java.
3) Picasso helps render (display) the different images in our application.

Dependencies included in Gradle:

‘com.squareup.retrofit2:retrofit:2.9.0’

'com.squareup.retrofit2:converter-gson:latest.version'

'com.squareup.picasso:picasso:2.8'

The Spoonacular API will service data in the form of JSON data. This is in contrast to our Android application which has been developed using the Java programming language. In order to ensure the application connects to the API smoothly, the JSON data that is serviced by the API is converted to corresponding Java classes to facilitate requests and responses.

JAVA Classes:

```
package com.example.recicheese.Models;

import java.util.ArrayList;

public class AnalyzedInstruction {
    public String name;
    public ArrayList<Step> steps;
}


package com.example.recicheese.Models;

public class Equipment {
    public int id;
    public String name;
    public String localizedName;
    public String image;
}


package com.example.recicheese.Models;

import java.util.ArrayList;

public class ExtendedIngredient {
    public int id;
    public String aisle;
    public String image;
    public String consistency;
    public String name;
    public String nameClean;
    public String original;
    public String originalName;
    public double amount;
    public String unit;
    public ArrayList<String> meta;
    public Measures measures;
}


package com.example.recicheese.Models;

public class Ingredient {
    public int id;
    public String name;
    public String localizedName;
    public String image;
}


package com.example.recicheese.Models;

import java.util.ArrayList;

public class InstructionsResponse {
    public String name;
    public ArrayList<Step> steps;
}


package com.example.recicheese.Models;

public class Length {
    public int number;
    public String unit;
}
```

```
package com.example.recicheese.Models;

public class Measures {
    public Us us;
    public Metric metric;
}
package com.example.recicheese.Models;

public class Metric {
    public double amount;
    public String unitShort;
    public String unitLong;
}


package com.example.recicheese.Models;

public class ProductMatch {
    public int id;
    public String title;
    public String description;
    public String price;
    public String imageUrl;
    public double averageRating;
    public double ratingCount;
    public double score;
    public String link;
}


package com.example.recicheese.Models;

import java.util.ArrayList;
```

```
public class RandomRecipeApiResponse {
    public ArrayList<Recipe> recipes;
}


package com.example.recicheese.Models;

import java.util.ArrayList;

public class Recipe {
    public boolean vegetarian;
    public boolean vegan;
    public boolean glutenFree;
    public boolean dairyFree;
    public boolean veryHealthy;
    public boolean cheap;
    public boolean veryPopular;
    public boolean sustainable;
    public boolean lowFodmap;
    public int weightWatcherSmartPoints;
    public String gaps;
    public int preparationMinutes;
    public int cookingMinutes;
    public int aggregateLikes;
    public int healthScore;
    public String creditsText;
    public String license;
    public String sourceName;
    public double pricePerServing;
    public ArrayList<ExtendedIngredient> extendedIngredients;
    public int id;
    public String title;
    public int readyInMinutes;
    public int servings;
    public String sourceUrl;
    public int openLicense;
    public String image;
    public String imageType;
    public String summary;
    public ArrayList<Object> cuisines;
    public ArrayList<String> dishTypes;
    public ArrayList<String> diets;
    public ArrayList<Object> occasions;
    public String instructions;
    public ArrayList<AnalyzedInstruction> analyzedInstructions;
    public Object originalId;
    public String spoonacularSourceUrl;
}


package com.example.recicheese.Models;

import java.util.ArrayList;

public class RecipeDetailsResponse {
    public int id;
    public String title;
    public String image;
    public String imageType;
    public int servings;
    public int readyInMinutes;
    public String license;
    public String sourceName;
    public String sourceUrl;
    public String spoonacularSourceUrl;
    public int aggregateLikes;
    public double healthScore;
    public double spoonacularScore;
    public double pricePerServing;
    public ArrayList<Object> analyzedInstructions;
    public boolean cheap;
    public String creditsText;
    public ArrayList<Object> cuisines;
    public boolean dairyFree;
    public ArrayList<Object> diets;
    public String gaps;
    public boolean glutenFree;
    public String instructions;
    public boolean ketogenic;
    public boolean lowFodmap;
    public ArrayList<Object> occasions;
    public boolean sustainable;
    public boolean vegan;
    public boolean vegetarian;
    public boolean veryHealthy;
    public boolean veryPopular;
    public boolean whole30;
    public int weightWatcherSmartPoints;
    public ArrayList<String> dishTypes;
    public ArrayList<ExtendedIngredient> extendedIngredients;
    public String summary;
    public WinePairing winePairing;
}


package com.example.recicheese.Models;

import java.util.ArrayList;

public class Step {
    public int number;
    public String step;
    public ArrayList<Ingredient> ingredients;
    public ArrayList<Equipment> equipment;
    public Length length;
}


package com.example.recicheese.Models;

public class Us {
    public double amount;
    public String unitShort;
    public String unitLong;
}


package com.example.recicheese.Models;

import java.util.ArrayList;

public class WinePairing {
    public ArrayList<String> pairedWines;
    public String pairingText;
    public ArrayList<ProductMatch> productMatches;
}
```

With the design of the application, the placeholders have been created to display the details of each recipe. Also, the Java classes have been created which map to the JSON data format of the API.

The next step in the implementation is to add the methods and listeners that will help process and service requests and also supervise the interaction between the application and the API.

The application has four listener interfaces. One, for when a recipe is clicked. Two, to generate random recipes on opening of the application. Three, to display details of the respective recipe and the last one dedicated for the instructions.

The two predominant methods associated with these listeners are:

1. void didFetch(RecipeDetailsResponse response, String message): in case of successful request, this method will fetch the corresponding data and service it using the response object 'response'.
2. void didError(String message): In cade the data cannot be fetched, a message will be displayed stating that recipes or recipe details cannot be fetched.

InstructionsListener.java:

```
package com.example.recicheese.Listeners;

import com.example.recicheese.Models.InstructionsResponse;

import java.util.List;

public interface InstructionsListener {
    void didFetch(List<InstructionsResponse> response, String messgae);
    void didError(String message);
}
```

RandomRecipeResponseListener.java:

```
package com.example.recicheese.Listeners;

import com.example.recicheese.Models.RandomRecipeApiResponse;

public interface RandomRecipeResponseListener {
    void didFetch(RandomRecipeApiResponse response,String message);
    void didError(String message);
}
```

RecipeClickListener.java

```
package com.example.recicheese.Listeners;

public interface RecipeClickListener {
    void onRecipeClicked(String id);
}
```

RecipeDetailsListener,java

```
package com.example.recicheese.Listeners;

import com.example.recicheese.Models.RecipeDetailsResponse;

public interface RecipeDetailsListener {
    void didFetch(RecipeDetailsResponse response, String message);
    void didError(String message);
}
```

The drop-down chooser which enables the user to browse recipes belonging to a specific category has its values stored in the string.xml file.

```
<string-array name="tags">
    <item>main course</item>
    <item>side dish</item>
    <item>dessert</item>
    <item>appetizer</item>
    <item>salad</item>
    <item>bread</item>
    <item>breakfast</item>
    <item>soup</item>
    <item>beverage</item>
    <item>sauce</item>
    <item>marinade</item>
    <item>fingerfood</item>
    <item>snack</item>
    <item>drink</item>
</string-array>
```

With the listeners been defined, the data fetched from the server API has to be incorporated into the respective design components. For example, the image view holder has to be provided the image of the respective dish. For this purpose, we have used adapters.
At first, the adapter will recognize all design elements using their IDs and will then place the respective data. It has code for the onclick event as well.

<u>RandomRecipeAdapter.java:</u>
Handles random recipes generation.

```
package com.example.recicheese.Adapters;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.cardview.widget.CardView;
import androidx.recyclerview.widget.RecyclerView;
import com.example.recicheese.Listeners.RecipeClickListener;
import com.example.recicheese.Models.Recipe;
import com.example.recicheese.R;
import com.squareup.picasso.Picasso;
import java.util.List;

public class RandomRecipeAdapter extends
RecyclerView.Adapter<RandomRecipeViewHolder> {
    Context context;
    List<Recipe> list;
    RecipeClickListener listener;

public RandomRecipeAdapter(Context context, List<Recipe>
list,RecipeClickListener listener) {
        this.context = context;
        this.list = list;
        this.listener=listener;
    }

    @NonNull
    @Override
    public RandomRecipeViewHolder onCreateViewHolder(@NonNull
ViewGroup parent, int viewType) {
        return new
RandomRecipeViewHolder(LayoutInflater.from(context).inflate(R.layout.list_ra
ndom_recipe,parent,false));
    }

    @Override
    public void onBindViewHolder(@NonNull RandomRecipeViewHolder holder,
int position) {
        holder.textView_title.setText(list.get(position).title);
        holder.textView_title.setSelected(true);
        holder.textView_likes.setText(list.get(position).aggregateLikes+" Likes");
        holder.textView_servings.setText(list.get(position).servings+" Servings");
        holder.textView_time.setText(list.get(position).readyInMinutes+"
Minutes");
        Picasso.get().load(list.get(position).image).into(holder.imageView_food);

   holder.random_list_container.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View v) {
```

```
listener.onRecipeClicked(String.valueOf(list.get(holder.getAdapterPosition()).id)
);
        }
        });

    }

    @Override
    public int getItemCount() {
        return list.size();
    }
}
class RandomRecipeViewHolder extends RecyclerView.ViewHolder{
    CardView random_list_container;
    TextView textView_title,textView_servings,textView_likes,textView_time;
    ImageView imageView_food;

    public RandomRecipeViewHolder(@NonNull View itemView) {
        super(itemView);

random_list_container=itemView.findViewById(R.id.random_list_container);
        textView_title=itemView.findViewById(R.id.textView_title);
        textView_servings=itemView.findViewById(R.id.textView_servings);
        textView_likes=itemView.findViewById(R.id.textView_likes);
        textView_time=itemView.findViewById(R.id.textView_time);
        imageView_food=itemView.findViewById(R.id.imageView_food);

}
}
```

<u>InstructionsAdapter.java:</u>

Handles instruction generation of the respective recipe.

```
package com.example.recicheese.Adapters;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.example.recicheese.Models.InstructionsResponse;
import com.example.recicheese.R;
import java.util.List;


public class InstructionsAdapter extends
RecyclerView.Adapter<InstructionsViewHolder> {

    Context context;
    List<InstructionsResponse> list;

    public InstructionsAdapter(Context context, List<InstructionsResponse> list)
{
        this.context = context;
        this.list = list;
    }

    @NonNull
    @Override
    public InstructionsViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
        return new
InstructionsViewHolder(LayoutInflater.from(context).inflate(R.layout.list_instru
ctions,parent,false));
    }

    @Override
    public void onBindViewHolder(@NonNull InstructionsViewHolder holder, int
position) {

        holder.textView_instruction_name.setText(list.get(position).name);
        holder.recycler_instruction_steps.setHasFixedSize(true);
        holder.recycler_instruction_steps.setLayoutManager(new
LinearLayoutManager(context,LinearLayoutManager.HORIZONTAL,false));
        InstructionStepAdapter stepAdapter=new
InstructionStepAdapter(context,list.get(position).steps);
holder.recycler_instruction_steps.setAdapter(stepAdapter);

}

    @Override
    public int getItemCount() {
        return list.size();
    }
}
class InstructionsViewHolder extends RecyclerView.ViewHolder{
    TextView textView_instruction_name;
    RecyclerView recycler_instruction_steps;
    public InstructionsViewHolder(@NonNull View itemView) {super(itemView);

textView_instruction_name=itemView.findViewById(R.id.textView_instruction_
name);

recycler_instruction_steps=itemView.findViewById(R.id.recycler_instruction_st
eps);
    }
}
```

The last and most important procedure in the implementation is to coordinate the listeners, adapters and the server API Handling. Also, here we coordinate the working of Retrofit and Picasso libraries.

RequestManager.java:

```
package com.example.recicheese;
import android.content.Context;
import com.example.recicheese.Listeners.InstructionsListener;
import com.example.recicheese.Listeners.RandomRecipeResponseListener;
import com.example.recicheese.Listeners.RecipeDetailsListener;
import com.example.recicheese.Models.InstructionsResponse;
import com.example.recicheese.Models.RandomRecipeApiResponse;
import com.example.recicheese.Models.RecipeDetailsResponse;
import java.util.List;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
import retrofit2.http.GET;
import retrofit2.http.Path;
import retrofit2.http.Query;

public class RequestManager {
    Context context;
    Retrofit retrofit=new Retrofit.Builder()
            .baseUrl("https://api.spoonacular.com/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();

    public RequestManager(Context context) {
        this.context = context;
    }
    public void getRandomRecipes(RandomRecipeResponseListener
listener,List<String> tags){
        CallRandomRecipes
callRandomRecipes=retrofit.create(CallRandomRecipes.class);
        Call<RandomRecipeApiResponse> call =
callRandomRecipes.callRandomRecipe(context.getString(R.string.api_key),"10",
tags);
        call.enqueue(new Callback<RandomRecipeApiResponse>() {
            @Override
            public void onResponse(Call<RandomRecipeApiResponse> call,
Response<RandomRecipeApiResponse> response) {
                if(!response.isSuccessful()){
                    listener.didError(response.message());
                    return;
                }
                listener.didFetch(response.body(), response.message());
            }

            @Override
            public void onFailure(Call<RandomRecipeApiResponse> call,
Throwable t) {
                listener.didError(t.getMessage());

            }
        });
    }

    public void getRecipeDetails(RecipeDetailsListener listener,int id){
        CallRecipeDetails
callRecipeDetails=retrofit.create(CallRecipeDetails.class);
        Call<RecipeDetailsResponse> call =
callRecipeDetails.callRecipeDetails(id,context.getString(R.string.api_key));
        call.enqueue((new Callback<RecipeDetailsResponse>() {
            @Override
            public void onResponse(Call<RecipeDetailsResponse> call,
Response<RecipeDetailsResponse> response) {
                if(!response.isSuccessful()){
                    listener.didError(response.message());
                    return;
                }
                listener.didFetch(response.body(), response.message());
            }

            @Override
            public void onFailure(Call<RecipeDetailsResponse> call, Throwable t) {
                listener.didError(t.getMessage());

            }
        }));
    }
    //
    public void getInstructions(InstructionsListener listener,int id){
        CallInstructions callInstructions=retrofit.create(CallInstructions.class);
        Call<List<InstructionsResponse>> call=
callInstructions.callInstructions(id,context.getString(R.string.api_key));
        call.enqueue(new Callback<List<InstructionsResponse>>() {
            @Override
            public void onResponse(Call<List<InstructionsResponse>> call,
Response<List<InstructionsResponse>> response) {
                if(!response.isSuccessful()){
                    listener.didError(response.message());
                    return;
                }
                listener.didFetch(response.body(), response.message());
            }

            @Override
            public void onFailure(Call<List<InstructionsResponse>> call,
Throwable t) {
                listener.didError(t.getMessage());

            }
        });
    }

    private interface CallRandomRecipes{
        @GET("recipes/random")
        Call<RandomRecipeApiResponse> callRandomRecipe(
                @Query("apiKey") String apiKey,
                @Query("number") String number,
                @Query("tags") List<String> tags
        );



    }

    private interface CallRecipeDetails{
        @GET("recipes/{id}/information")
        Call<RecipeDetailsResponse> callRecipeDetails(
                @Path("id") int id,
                @Query("apiKey") String apiKey
        );
    }
    //
    private interface CallInstructions{
        @GET("recipes/{id}/analyzedInstructions")
        Call<List<InstructionsResponse>> callInstructions(
                @Path("id") int id,
                @Query("apiKey") String apiKey
        );
    }}
```

RecipeDetailsActivity.java

```
package com.example.recicheese;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.text.Html;
import android.app.ProgressDialog;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;
import com.example.recicheese.Adapters.IngredientsAdapter;
import com.example.recicheese.Adapters.InstructionsAdapter;
import com.example.recicheese.Listeners.InstructionsListener;
import com.example.recicheese.Listeners.RecipeDetailsListener;
import com.example.recicheese.Models.InstructionsResponse;
import com.example.recicheese.Models.RecipeDetailsResponse;
import com.squareup.picasso.Picasso;
import java.util.List;

public class RecipeDetailsActivity extends AppCompatActivity {
    int id;
    TextView
textView_meal_name,textView_meal_source,textView_meal_summary;
    ImageView imageView_meal_image;
    RecyclerView recycler_meal_ingredients,recycler_meal_instructions;
    RequestManager manager;
    ProgressDialog dialog;
    IngredientsAdapter ingredientsAdapter;
    InstructionsAdapter instructionsAdapter;

    @Override
```

```
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_recipe_details);

      findViews();
      id=Integer.parseInt(getIntent().getStringExtra("id"));
      manager=new RequestManager(this);
      manager.getRecipeDetails(recipeDetailsListener,id);
      manager.getInstructions(instructionsListener,id);
      dialog=new ProgressDialog(this);
      dialog.setTitle("Loading Details");
      dialog.show();
 }
 private void findViews() {
      textView_meal_name=findViewById(R.id.textView_meal_name);
      textView_meal_source=findViewById(R.id.textView_meal_source);
      textView_meal_summary=findViewById(R.id.textView_meal_summary);
      imageView_meal_image=findViewById(R.id.imageView_meal_image);
      recycler_meal_ingredients=findViewById(R.id.recycler_meal_ingredients);
    recycler_meal_instructions=findViewById(R.id.recycler_meal_instructions);
}
private final RecipeDetailsListener recipeDetailsListener = new
RecipeDetailsListener() {
      @Override
      public void didFetch(RecipeDetailsResponse response, String message) {
          dialog.dismiss();
          textView_meal_name.setText(response.title);
          textView_meal_source.setText(response.sourceName);
          textView_meal_summary.setText(response.summary);
          Picasso.get().load(response.image).into(imageView_meal_image);
          recycler_meal_ingredients.setHasFixedSize(true);
          recycler_meal_ingredients.setLayoutManager(new
LinearLayoutManager(RecipeDetailsActivity.this,LinearLayoutManager.
HORIZONTAL,false));
ingredientsAdapter=new
IngredientsAdapter(RecipeDetailsActivity.this,response.extendedIngredients);
recycler_meal_ingredients.setAdapter(ingredientsAdapter);
 }
       @Override
        public void didError(String message) {

Toast.makeText(RecipeDetailsActivity.this,message,Toast.LENGTH_SHORT).sho
w();


        }
    };

    private final InstructionsListener instructionsListener = new
InstructionsListener() {
        @Override
        public void didFetch(List<InstructionsResponse> response, String message)
{
            recycler_meal_instructions.setHasFixedSize(true);
            recycler_meal_instructions.setLayoutManager(new
LinearLayoutManager(RecipeDetailsActivity.this,LinearLayoutManager.VERTIC
AL,false));
            instructionsAdapter = new
InstructionsAdapter(RecipeDetailsActivity.this,response);
            recycler_meal_instructions.setAdapter(instructionsAdapter);


        }


        @Override
        public void didError(String message) {
}
    };
}
```

MainActivity.java:

```
package com.example.recicheese;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.SearchView;
import androidx.recyclerview.widget.GridLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;
import com.example.recicheese.Adapters.RandomRecipeAdapter;
import com.example.recicheese.Listeners.RandomRecipeResponseListener;
import com.example.recicheese.Listeners.RecipeClickListener;
import com.example.recicheese.Models.RandomRecipeApiResponse;
import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
    ProgressDialog dialog;
    RequestManager manager;
    RandomRecipeAdapter randomRecipeAdapter;
    RecyclerView recyclerView;
    Spinner spinner;
    List<String> tags = new ArrayList<>();
    SearchView searchView;
 @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dialog = new ProgressDialog(this);
        dialog.setTitle("Loading...");

        searchView=findViewById(R.id.searchView);
        searchView.setOnQueryTextListener(new
SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextSubmit(String query) {
                tags.clear();
                tags.add(query);
                manager.getRandomRecipes(randomRecipeResponseListener,tags);
                dialog.show();
                return true;
            }

            @Override
            public boolean onQueryTextChange(String newText) {
                return false;
            }
        });

        spinner=findViewById(R.id.spinner_tags);
        ArrayAdapter arrayAdapter=ArrayAdapter.createFromResource(
                this,
                R.array.tags,
                R.layout.spinner_text


        );
        arrayAdapter.setDropDownViewResource(R.layout.spinner_inner_text);
        spinner.setAdapter(arrayAdapter);
        spinner.setOnItemSelectedListener(spinnerSelectedListener);
        manager=new RequestManager(this);
        //manager.getRandomRecipes(randomRecipeResponseListener);
        //dialog.show();
```

```
    }
 private final RandomRecipeResponseListener randomRecipeResponseListener =
new RandomRecipeResponseListener() {
        @Override
        public void didFetch(RandomRecipeApiResponse response, String
message) {
            recyclerView=findViewById(R.id.recycler_random);
            recyclerView.setHasFixedSize(true);
            recyclerView.setLayoutManager(new
GridLayoutManager(MainActivity.this,1));
            randomRecipeAdapter = new
RandomRecipeAdapter(MainActivity.this,response.recipes,recipeClickListener);
            recyclerView.setAdapter(randomRecipeAdapter);

        }

        @Override
        public void didError(String message) {

Toast.makeText(MainActivity.this,message,Toast.LENGTH_SHORT).show();

        }
    };
    private final AdapterView.OnItemSelectedListener
spinnerSelectedListener=new AdapterView.OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> adapterView, View view, int
position, long id) {
            tags.clear();
            tags.add(adapterView.getSelectedItem().toString());
            manager.getRandomRecipes(randomRecipeResponseListener,tags);
            dialog.show();

        }
        @Override
        public void onNothingSelected(AdapterView<?> parent) {

        }
    };

    private final RecipeClickListener recipeClickListener = new
RecipeClickListener() {
        @Override
        public void onRecipeClicked(String id) {
            startActivity(new Intent(MainActivity.this,RecipeDetailsActivity.class)
                    .putExtra("id",id));

        }
    };
}
```

CHAPTER 5

RESULTS

On installing and running the application on an Android Device, the following results were obtained.

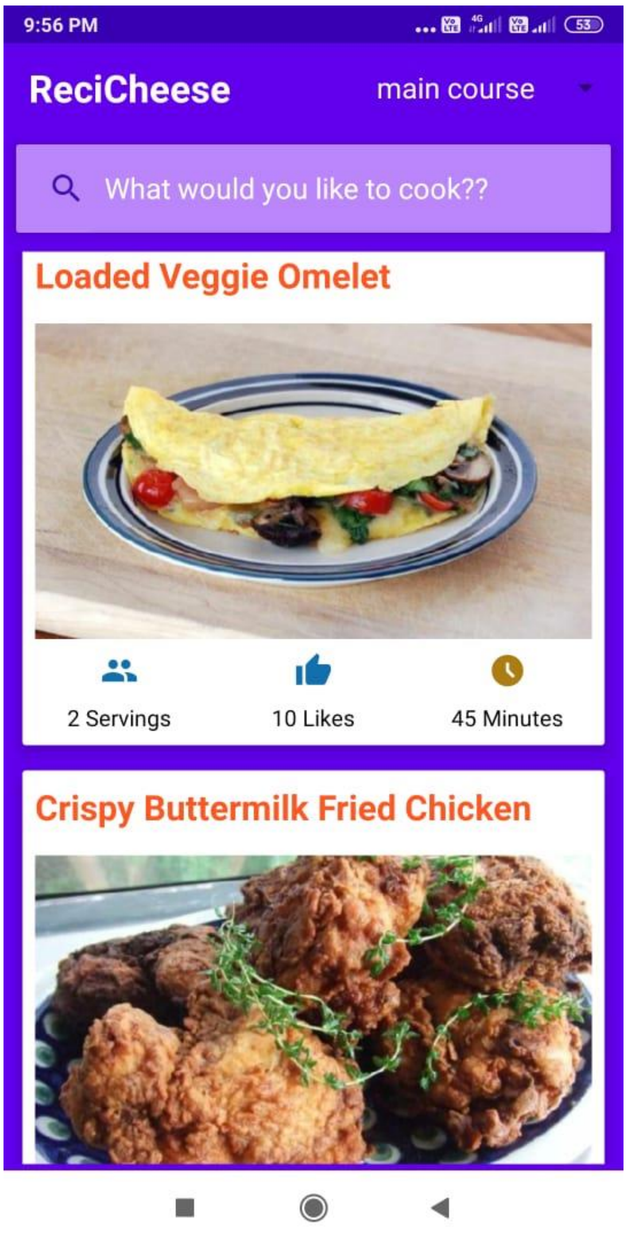


Fig 5.1 Main Recipe Page

Figure 5.1 shows the main page, I.e., the page visible to the user when he/she opens the application.

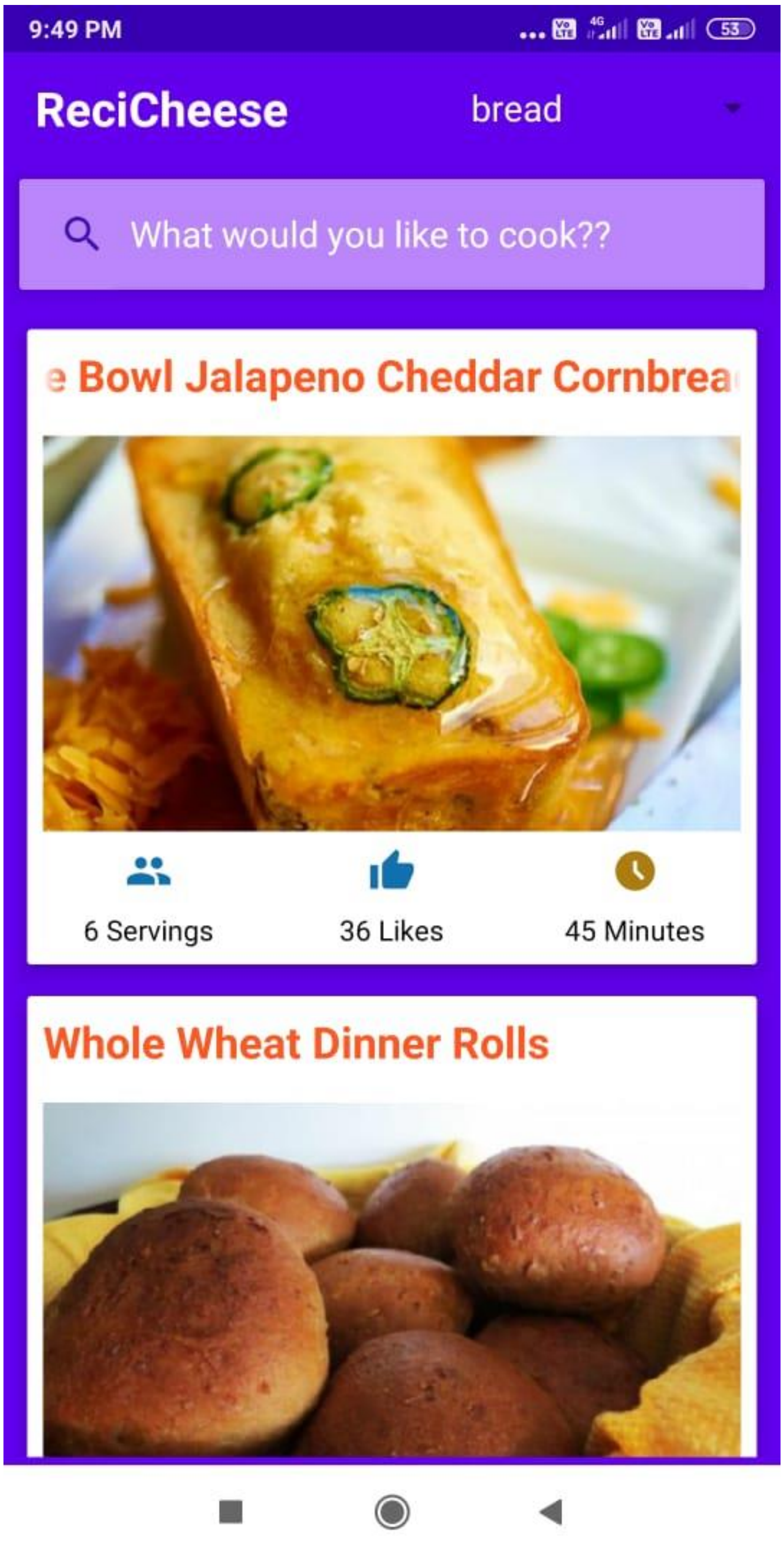


Fig 5.2 Recipe display based on category

Figure 5.2 shows the display of recipes based on the category selected by the user from the drop-down chooser. Here the user has selected the category 'bread' and as a result, dishes such as whole wheat dinner rolls etc., are generated.

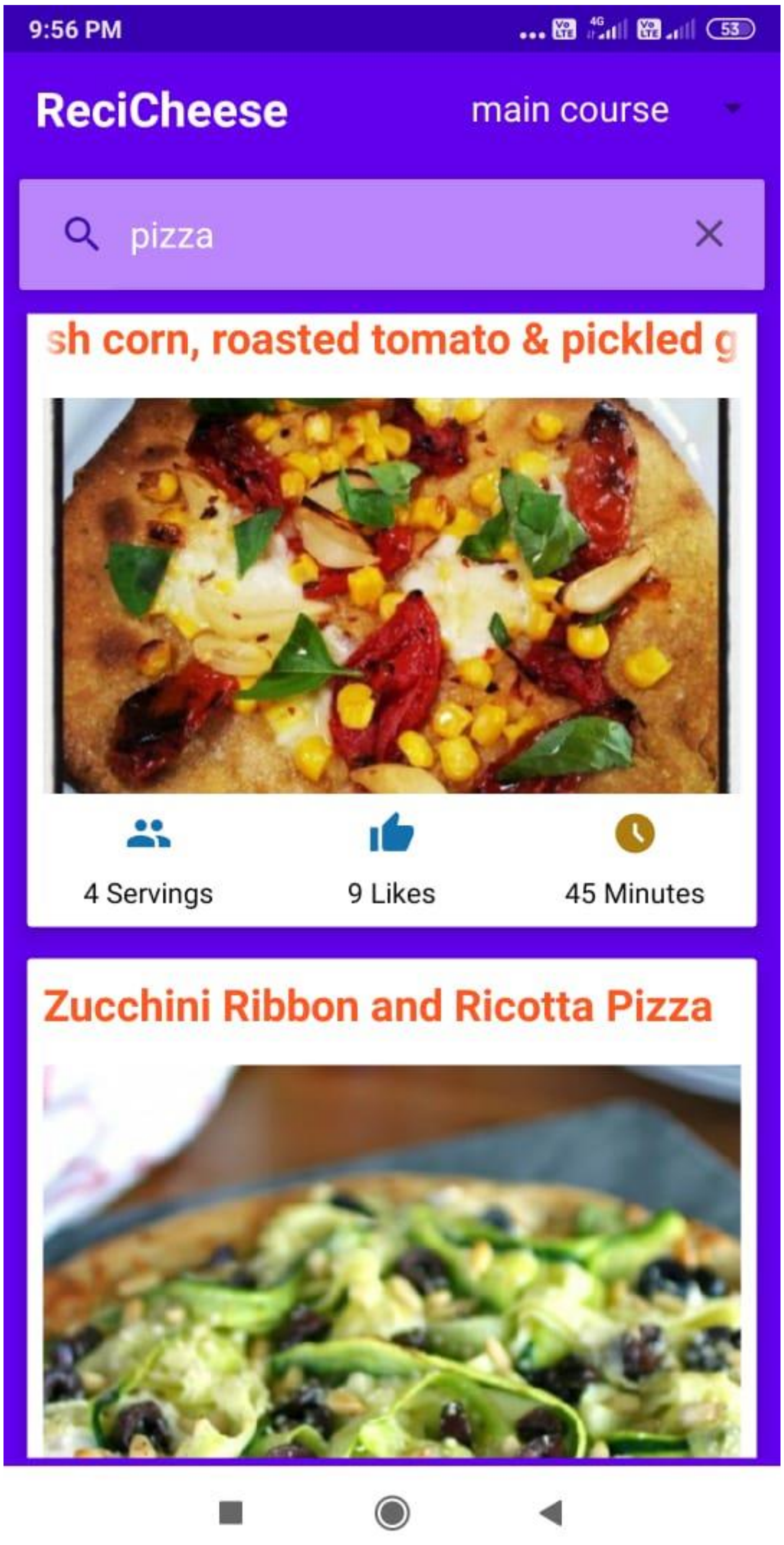
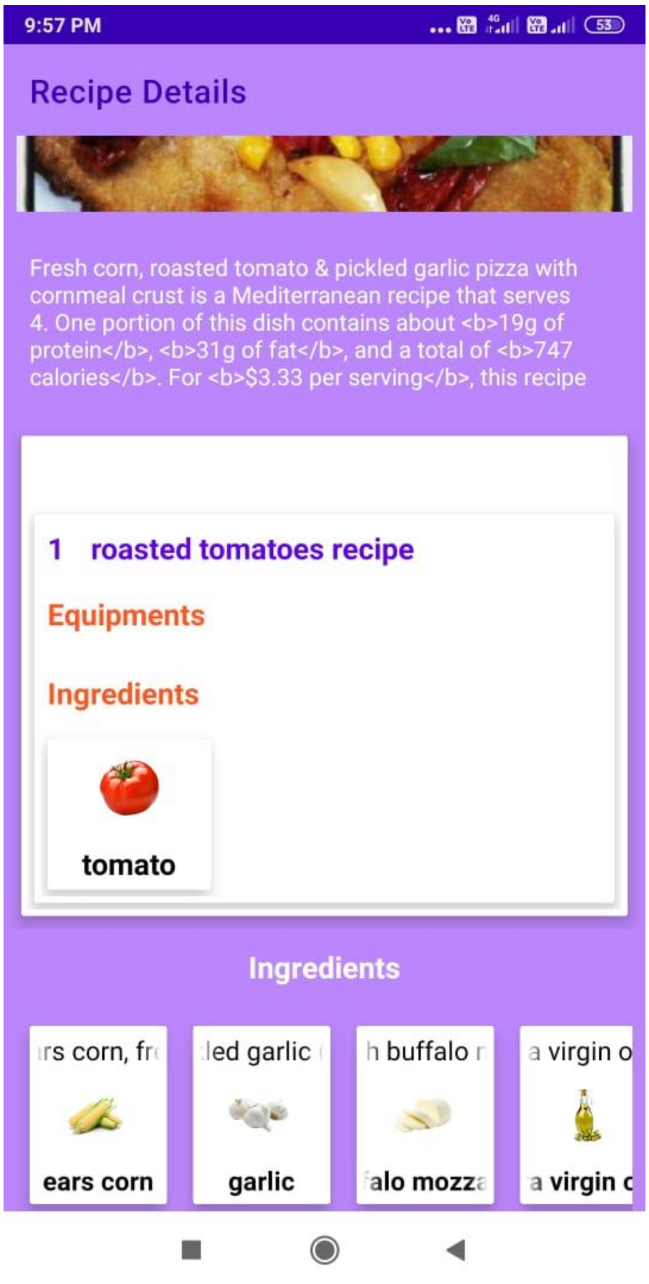

9:56 PM
ReciCheese
main course
pizza
sh corn, roasted tomato & pickled g
4 Servings
9 Likes
45 Minutes
Zucchini Ribbon and Ricotta Pizza

Fig 5.3 Recipe Display based on search-entry

Figure 5.3 shows the display of recipes based on the search-entry by the user. Here the user has searched for pizza. As a result, a list of pizza recipes will be displayed. The user can then proceed to view the recipe details of each pizza.



Fresh corn, roasted tomato & pickled garlic pizza with cornmeal crust is a Mediterranean recipe that serves 4. One portion of this dish contains about <b>19g of protein</b>, <b>31g of fat</b>, and a total of <b>747 calories</b>. For <b>$3.33 per serving</b>, this recipe

Fig 5.4 Recipe Details

Figure 5.4 displays the recipe details. First the name of the dish followed by its source. Then details associated with the recipe such as a general description, the instructions, the ingredients, the equipments are all displayed in this page.

Conclusion and Future Enhancement

Conclusion

The recipe application ‘ReciCheese’ has been developed using Android Studio IDE. Various elements such as layouts, activities, assets have been used for the development of this application. In order to expand the scope of our application and not merely restrict it to a limited target group, an external API ‘Spoonacular’ was used to service any recipe request at any time. Listeners, adapters, interfaces, classes and methods were incorporated in implementing the functionality for the application.

By developing this Android application, we understood what goes behind the scenes in developing an Android app. Throughout the process of developing this application, we were able to learn the various features of Android Studio and how to maximize the potential of the resources, plugins that it offers, in order to build powerful and efficient applications.

Future Enhancement

The food industry is an industry whose market value will not depreciate in the near future. The Coronavirus Pandemic has also highlighted the importance of home food, and in the present world, there is an understanding among the larger population that ‘cooking-at-home’ is a key component in maintaining a healthy lifestyle. Thus, by accessing recipes, people find it easier to cook various dishes at home using simple methods.

The recipe app has a large target market as a result of the above factors. With respect to future enhancement, in order to support healthy lifestyles, the app can be enhanced by adding search features for filtering recipes based on calories, nutrition value, nutrient constitution per ingredient etc. Also, in order to assist people with visual disability, a text-to-speech facility that helps dictate the instructions can also be implemented.

BIBLIOGRAPHY

[1]. **https://developer.android.com/studio**

[2]. **https://source.android.com**

[3]. **https://www.wikipedia.org/**

[4]. **https://spoonacular.com/food-api**

[5]. **https://images.google.com/**