# ABSTRACT

The modern era of technology has a tremendous impact on the society. With the creation of the ultimate virtual assistants, chatbots have become a popular entity in the conversational services. Chatbots are software programs that use natural language understanding and processing. Chatbots are not just restricted to help the user to complete his tasks such as booking a movie ticket or finding the nearest restaurant, but they also provide a source of entertainment, play a major role in home automation projects, give business strategy tips and help in other ways.

The project ‘ A Custom Chatbot Application Using OpenAI’ is typically based on OpenAI's language model technology, which uses advanced artificial intelligence techniques to analyze and generate natural language text. Specifically, it may be based on OpenAI's GPT (Generative Pre-trained Transformer) models, which are large neural networks that can generate coherent and contextually relevant responses to user inputTo create a custom chatbot, developers can fine-tune one of these pre-trained models on a specific dataset that is relevant to the chatbot's purpose or domain. The fine-tuning process involves training the model on the dataset to adapt it to the specific task at hand, such as customer service, language translation, or conversational assistance. Once the model is fine-tuned, it can be deployed as a chatbot that users can interact with through text or voice commands.Overall, a custom chatbot based on OpenAI is designed to use natural language processing and machine learning techniques to understand and respond to user inputs in a way that simulates human-like conversation.

# TABLE OF CONTENTS

| CHAPTER NO | TITLE | PAGE NO |
|---|---|---|
| | ABSTRACT | 1 |
| | LIST OF FIGURES | 4 |
| 1 | INTRODUCTION | 5 |
| | 1.1 PROBLEM STATEMENT | 5 |
| | 1.2 OBJECTIVE | 6 |
| | 1.3 SCOPE | 6 |
| 2 | **LITERATURE SURVEY** | **7** |
| 3 | **SYSTEM DESIGN** | 8 |
| | 3.1 GENRAL ARCHITECTURE DESIGN | 8 |
| | 3.2 LEVEL ZERO DATA FLOW DIAGRAM | 9 |
| | 3.3 LEVEL ONE DATAFLOW DIAGRAM | 9 |
| | 3.4 USE CASE DIAGRAM | 10 |
| | 3.5 SEQUENCE DIAGRAM | 11 |
| | 3.6 ACTIVITY DIAGRAM | 12 |
| | 3.7 COMPONENT DIAGRAM | 13 |
| | 3.8 DEPLOYMENT DIAGRAM | 14 |
| 4 | **REQUIREMENT SPECIFICATION** | **15** |
| | 4.1 HARDWARE REQUIREMENTS | 15 |
| | 4.2 SOFTWARE REQUIREMENTS | 15 |

| | | |
|---|---|---|
| 5 | IMPLEMENTATION | 16 |
| | 5.1 STEP BY STEP IMPLEMENTATION | 16 |
| | 5.2 ACTIVITY MAIN.XML CODE | 16 |
| | 5.3 CODE FOR MESSAGEMODAL CLASS | 18 |
| | 5.4 LAYOUT FOR STORING USERMSG | 19 |
| | 5.5 LAYOUT FOR STORING BOTMSG | 20 |
| | 5.6 ADAPTER CLASS | 21 |
| | 5.7 MAIN ACTIVITY.JAVA FILE | 24 |
| | 5.8 ANDROID MANIFEST.XML FILE | 28 |
| | 5.9 RETROFIT API.JAVA | 29 |
| **6** | **INTERFACING** | **30** |
| | 6.1 GENRATING API KEY | 30 |
| | 6.2 TRAINING THE BRAIN | 35 |
| | 6.3 DEFAULT ATTRIBUTES | 36 |
| | 6.4 OUTPUT SCREENSHOTS | 37 |
| **7** | **CONCLUSION** | **40** |
| | 7.1 CONCLUSION | 40 |
| | 7.2 LIMITATIONS OF THE SYSTEM | 41 |
| 8 | REFERENCE | |

# LIST OF FIGURES

| FIGURE NO | FIGURE NAME | PAGE NO. |
|---|---|---|
| 1 | GENERAL CHATBOT ARCHITECTURE | 8 |
| 2 | LEVEL 0 DFD DIAGRAM | 9 |
| 3 | LEVEL 1 DFD DIAGRAM | 9 |
| 4 | USECASE DIAGRAM | 10 |
| 5 | SEQUENCE DIAGRAM | 11 |
| 6 | ACTIVITY DIAGRAM | 12 |
| 7 | COMPONENT DIAGRAM | 13 |
| 8 | DEPLOYMENT DIAGRAM | 14 |
| 9 | BRAINSHOPAI WEBPAGE | 30 |
| 10 | CREATING BRAIN | 31 |
| 11 | ADDING BRAIN | 32 |
| 12 | TRAINING THE MODEL | 33 |
| 13 | GENRATING API KEY | 34 |
| 14 | TRAING THE BRAIN | 35 |
| 15 | DEFAULT ATTRIBUTES | 36 |
| 16 | OUTPUT SCREENSHOTS | 37 |

# 1 INTRODUCTION

Technology plays a massive role in the industry and daily chores. It serves a variety of purposes and is applied in a different way in different parts of the world. Recently, the public has been fantasized by Artificial Intelligence. Artificial Intelligence simulates the cognitive abilities of a human. To be more precise and closely related to humans, the AI Chatbots are now replacing human responses with this software. A Chatbot is a computerized program that acts like a colloquist between the human and the bot, a virtual assistant that has become exceptionally popular in recent years mainly due to dramatic improvements in the areas like artificial intelligence , machine learning and other underlying technologies such as neural networks and natural language processing. These chatbots effectively communicate with any human being using interactive queries. Recently, there’s been a massive increase in many cloud-based chatting bot services which have been made available for the development and improvement of the chatbot sector such as IBM Watson, Cleverbot, ELIZA chatbot and many others. These conversational agents have become more responsive and the art of conversation between humans and robots over the past few years have improved drastically. In this paper, we have generalized the AI chatbot using brain shops openAI.

## 1.1 Problem Statement

Providing effective and efficient customer service is one of the main issues facing organisations today. Traditional customer service channels, including phone and email, can be time-consuming and expensive, and they might not give clients the fast answer they want. As a result, the demand for AI chatbots to deliver prompt and effective customer care is rising. AI chatbots can help businesses automate customer support and provide 24/7 service. They are capable of doing a wide range of responsibilities, including responding to frequently asked queries, making product recommendations, setting up appointments, and handling straightforward client difficulties. This not only saves time and money for businesses, but it also improves the customer experience by providing instant responses and reducing wait times.

## 1.2 Objective

Through the use of natural language processing, AI chatbots can offer quick and effective customer service and support Numerous duties, like responding to frequently asked inquiries, making tailored recommendations, addressing straightforward client complaints, and organising appointments, should be handled by the chatbot.

We expect the AI chatbot to improve customer service and efficiency by reducing response times and providing accurate and relevant responses to user input. The chatbot should also reduce costs by automating customer support and improving the scalability of customer service operations. Additionally, we expect the chatbot to improve the customer experience by providing personalized responses and integrating with other systems and applications. Finally, we expect the chatbot to provide valuable insights through analytics on user interactions and feedback, which can be used to improve the overall customer experience.

To achieve the objective, we will develop an AI chatbot using a combination of natural language processing and machine learning algorithms. The chatbot will be trained on relevant data using brainshopAI's brain to provide accurate and relevant responses to user input. The chatbot will also be designed to personalize responses based on user preferences and interest or search history, and integrate with other systems and applications to provide a seamless user experience. We will evaluate the performance of the chatbot by measuring its accuracy, response time, and user satisfaction.

## 1.3 Scope

The primary purpose of an AI chatbot is to provide fast and efficient customer service and support using natural language processing and machine learning algorithms. The scope of an AI chatbot project can vary depending on the needs and goals of the business. Some common use cases for AI chatbots include customer support, sales and marketing, appointment scheduling, human resources, and entertainment and gaming. Each of these use cases may require different functionalities and capabilities for the chatbot to be effective and AI chatbot is generally broader and more advanced than a traditional chatbot due to the use of natural language processing and machine learning algorithms. While a traditional chatbot may rely on a set of predefined rules or responses, an AI chatbot has the ability to understand and respond to natural language input in a more sophisticated manner.

# 2.Literature Survey

Though the desire to be able to create something that understands and can converse with its creator has a long history, Alan Turing is credited as being the first to have conceptualised the idea of a chatbot in 1950 when he posed the question, "Can machines think?" Turing's depiction of an intelligent machine's activity inspires the idea of a chatbot as it is generally known.

With the creation of ELIZA in 1966, the first chatbot was put into use. This system largely depended on linguistic rules and pattern matching algorithms. Through the use of a keyword matching programme, it could converse with the user. It looks for a suitable transformation rule to restructure the input and produce an output, or the user's response. Eliza was a seminal system that inspired additional study in the area. However, ELIZA's knowledge base was constrained since it relied on a minimal amount of context identification and because pattern matching algorithms are typically rigid and difficult to apply to new domains.

A marked evolution in chatbot in the 1980s is the use of Artificial Intelligent. A.L.I.C.E. (Artificial Intelligent Internet Computer Entity) is based on the Artificial Intelligence Mark-up Language (AIML), which is an extension of XML. It was developed especially so that dialogue pattern knowledge could be added to A.L.I.C.E.'s software to expand its knowledge base. Data objects in AIML are composed of topics and categories. Categories are the basic unit of knowledge, which are comprised of a rule to match user inputs to chatbot's outputs. The user input is represented by rule patterns, while the chatbot's output is defined by rule template, A.L.I.C.E. knowledge base. The addition of new data objects in AIML represented a significant improvement on previous pattern matching systems since the knowledgebase was easily expandable.

the most interesting application, is the development of smart personal assistants (such as Amazon's Alexa, Apple's Siri, Google's Google Assistant, Microsoft's Cortana, and IBM's Watson). Personal assistants chatbots or conversational agents that can usually communicate with the user through voice are usually integrated in smartphones, smartwatches, dedicated home speakers and monitors, and even cars. For example, when the user utters a wake word or phrase the device activates, and the smart personal assistant starts to listen. Through Natural Language Understanding the assistant can then understand commands and answer the user’s requests, usually by providing pieces of information. Microsoft's XiaoIce is another intriguing social chatbot example. In order to achieve high user engagement, XiaoIce has been created to have a personality, an Intelligent Quotient (IQ), and an Emotional Quotient (EQ). XiaoIce is intended to be a long-term friend for the user. IQ skills include things like the ability to model knowledge and memory, comprehend images and natural language, reason, generate ideas, and anticipate the future. These are essential elements in the growth of dialogue skills. They are necessary for social chatbots to assist users and cater to their unique needs. Core Chat, which can hold extended chats with users in an open-domain setting, is the most important and advanced capability. The two essential elements of EQ are empathy and social intelligence. The XiaoIce conversational engine makes use of a dialogue manager to keep track of the state of the conversation and selects either the Core Chat (the open domain Generative component) or the dialogue skill in order to generate a response.

# 3.Proposed System Architecture Design

## 3.1 General chatbot architecture

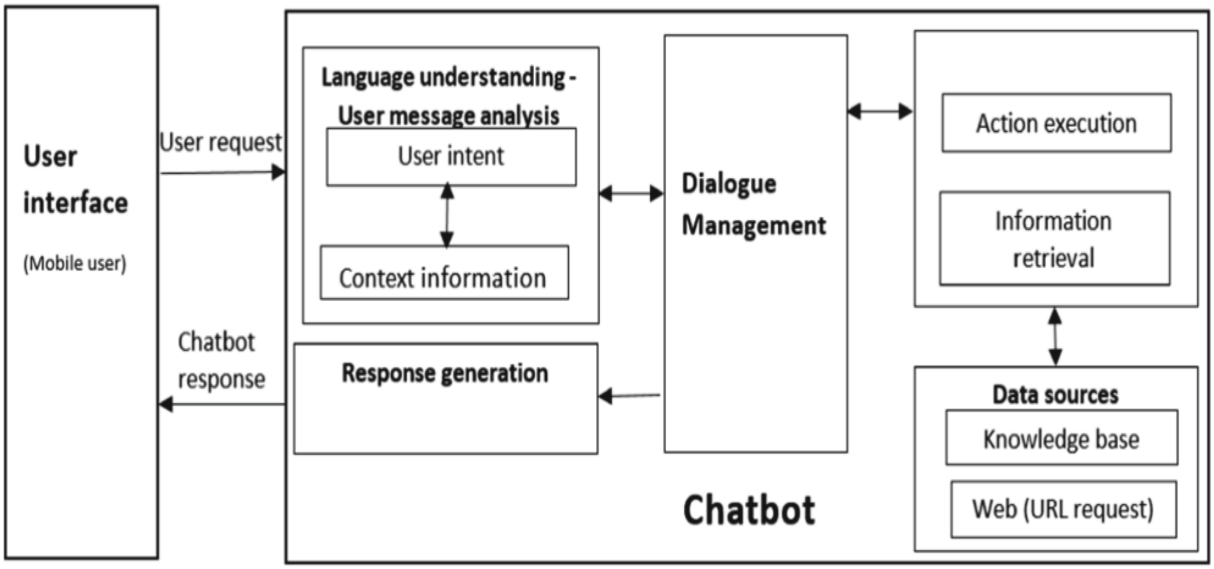


Fig 1: genral architecture

## 3.2 Dataflow diagram for proposed architecture

## Level 0:

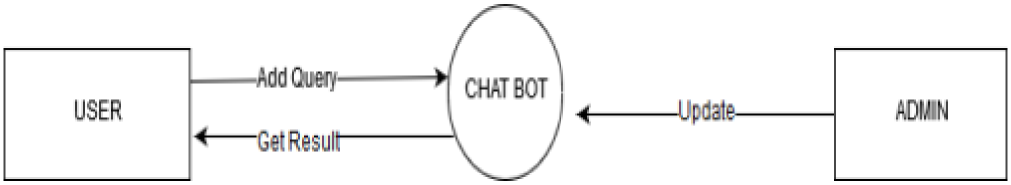


**Fig 2: Zero level DFD of Chat Bot System**

## 3.3
## Level 1:

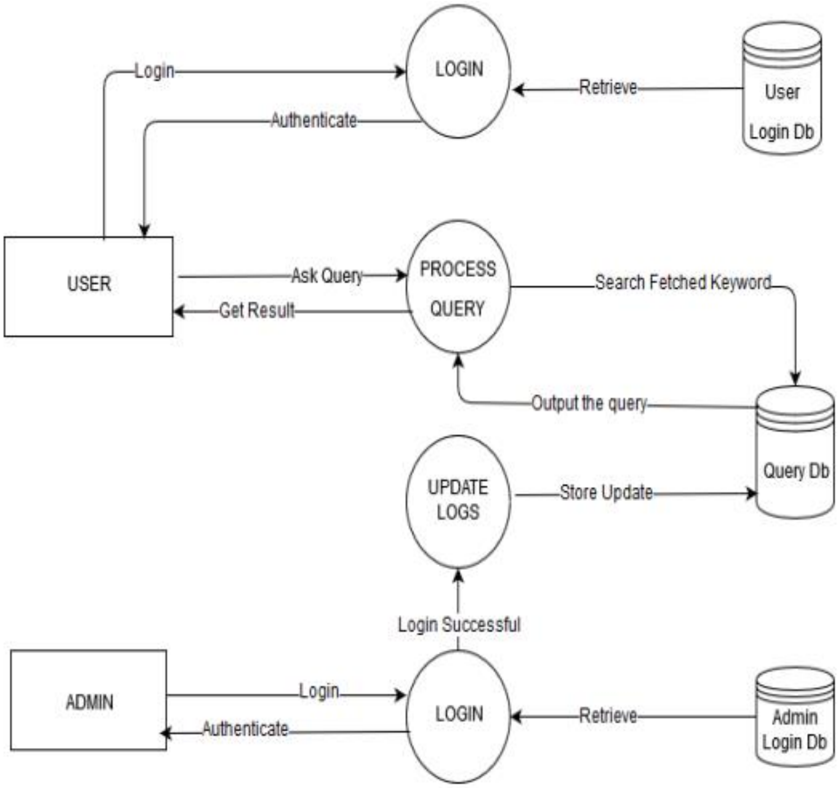


**Fig 3: First level DFD of Chat Bot System**

## 3.4 Use Case Diagram

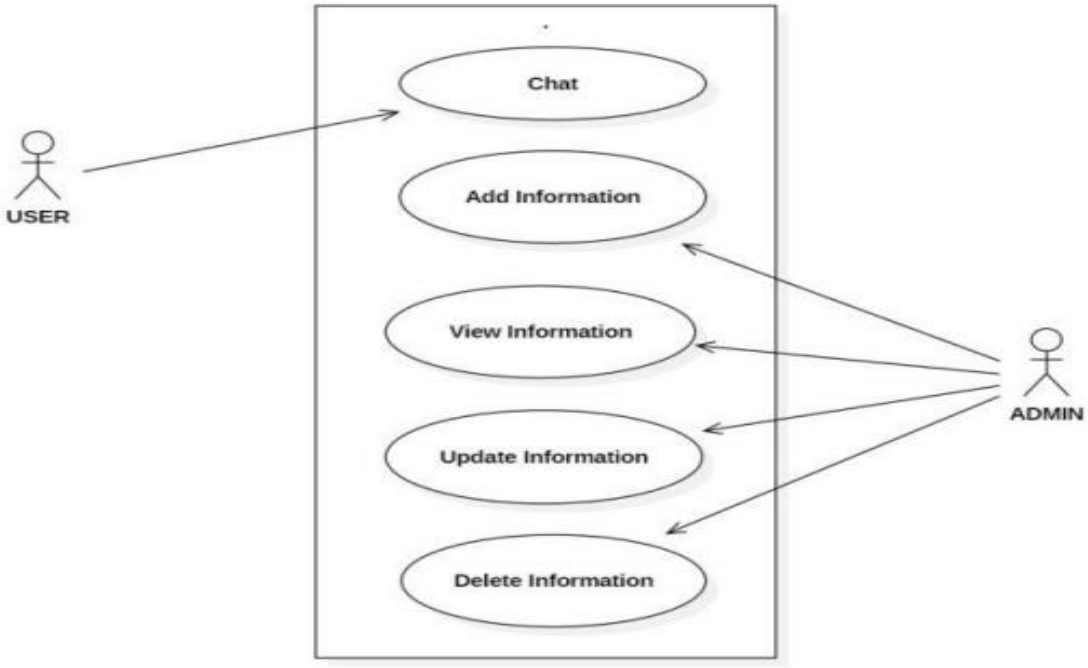


Fig 4: Use case diagram for user and admin roles

## 3.5 Sequence Diagram

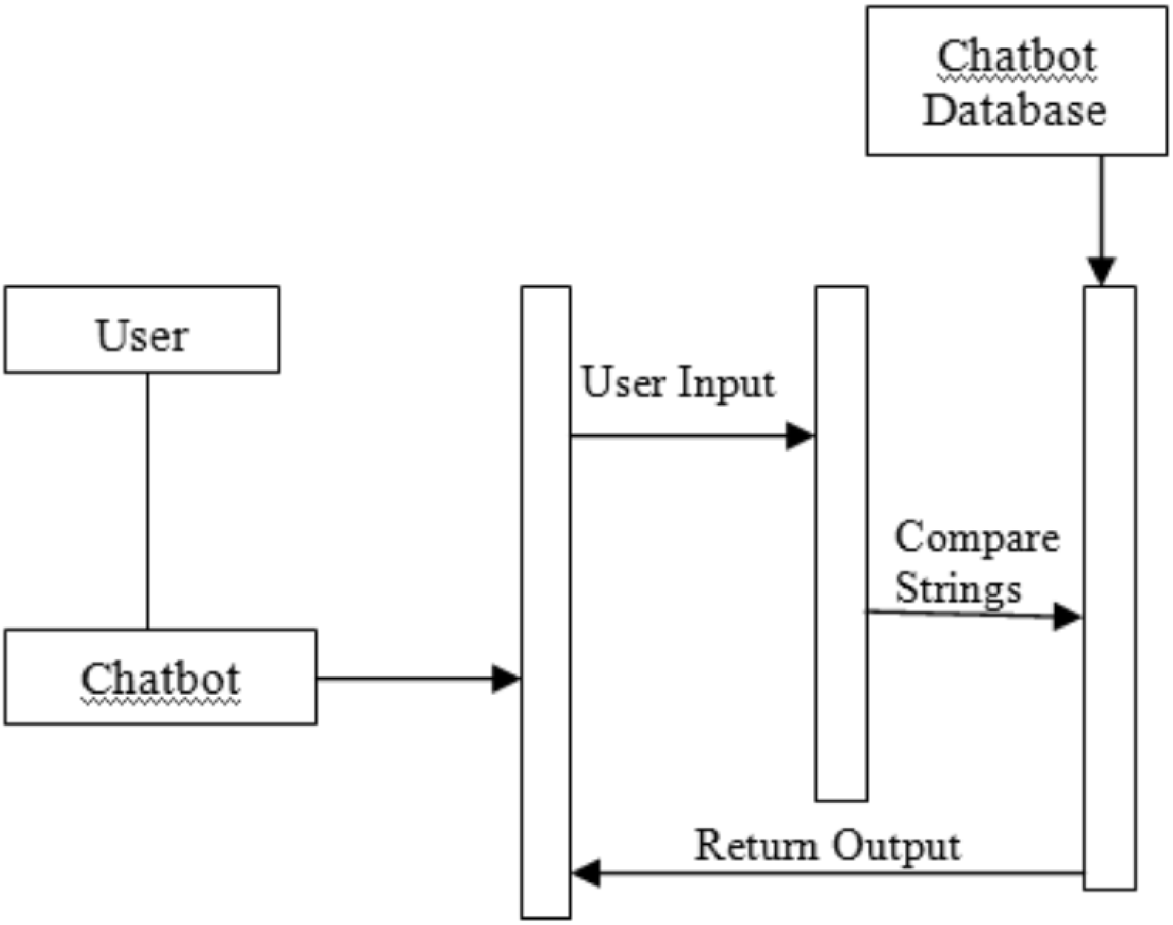
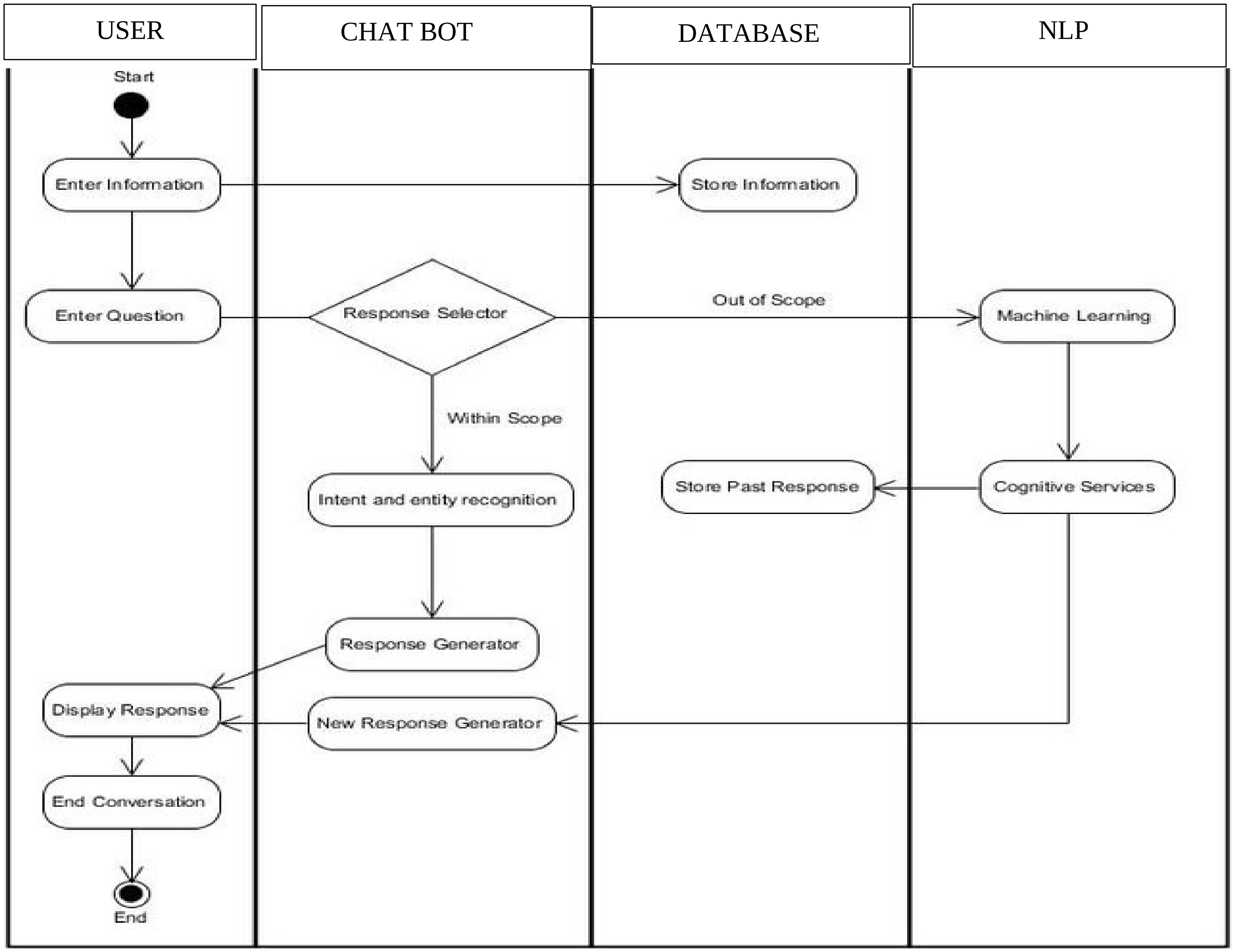


**Fig 5: Sequence Diagram For ChatBot**

## 3.6 Activity Diagram



## 3.7 Component Diagram

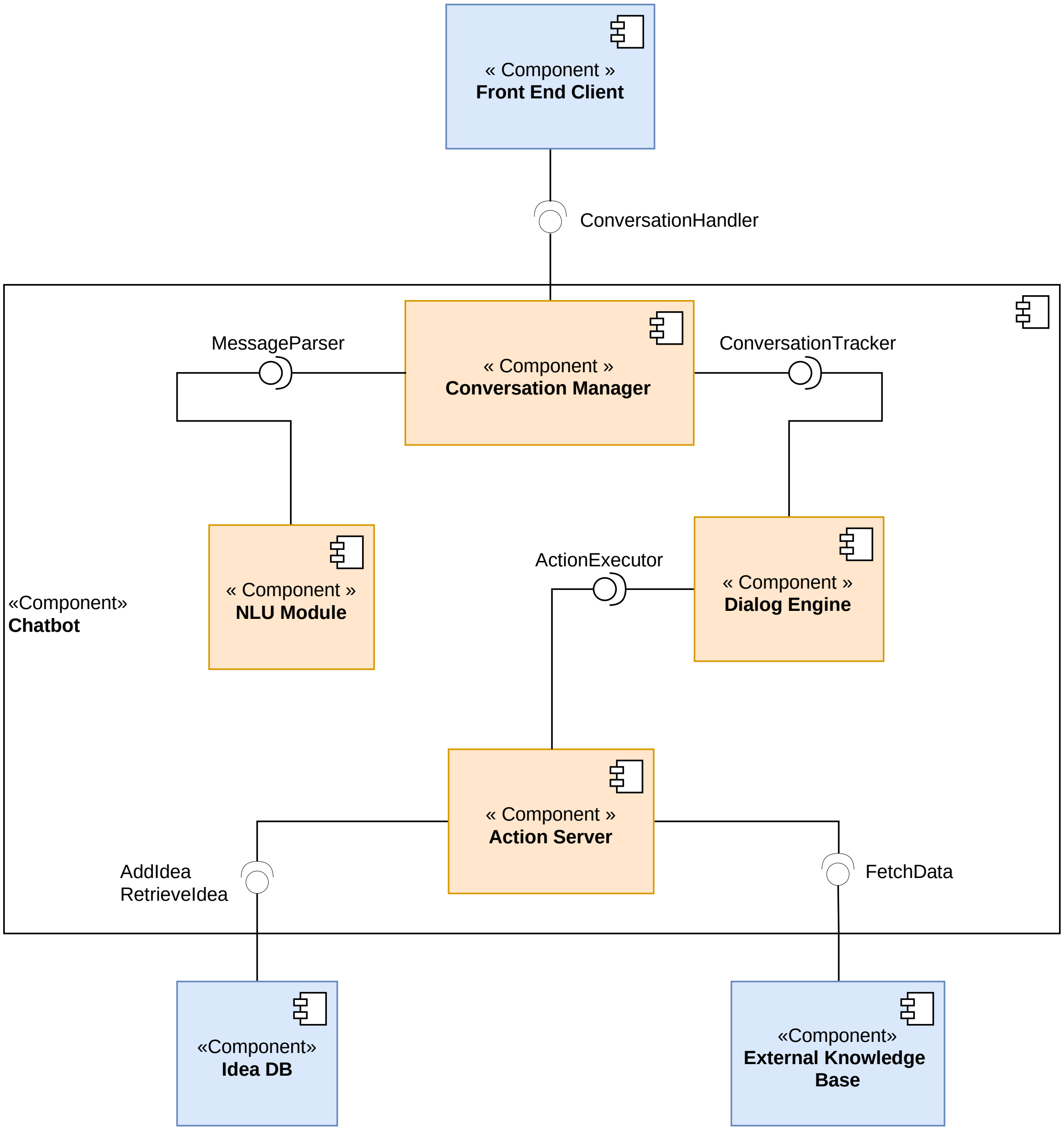


Fig 7: Component Diagram

## 3.8 Deployment Diagram

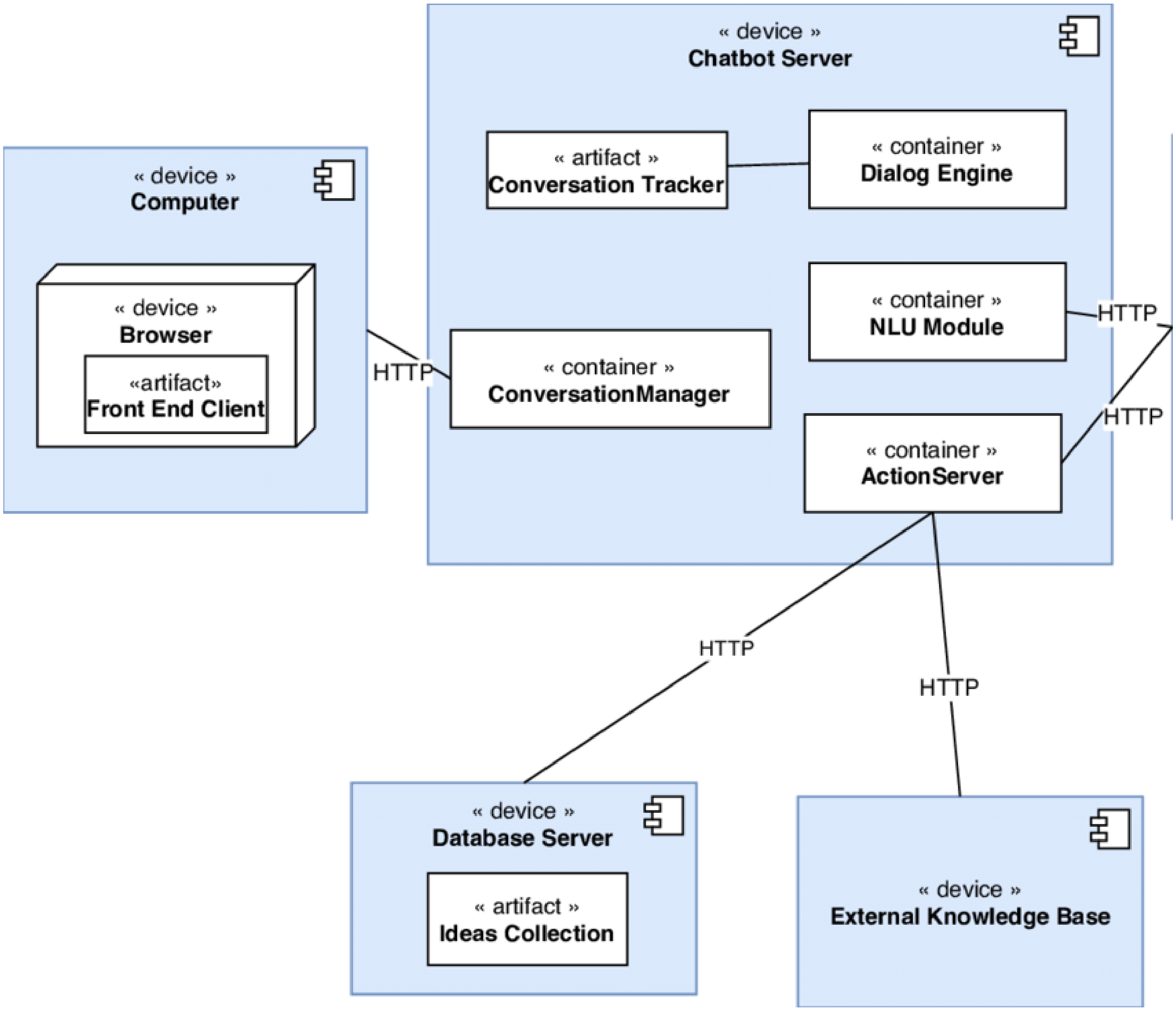


**Fig 8: Deployment Diagram**

# 4. Requirement Specification

## 4.1 Hardware Specification

To run android studios the required hardware specifications are

1.supported cpu models

i) 2nd generation Intel CPU (Sandy Bridge) or newer, AMD CPU with support for a Windows Hypervisor
ii) ARM-based chips, or 2nd generation Intel Core or newer with support for Hypervisor.Framework
iii) Intel Core i5-8400 or better

2.Memory
i)Minimum 8gb RAM is required 16 gb RAM is great to use
ii)Free storage-30 GB (SSD is strongly recommended)

3.Screen Resolution

i) 1920 x 1080

## 4.2 Software Specification

1.supported operating system versions

i) Windows 8/8.1/10/11 (64-bit)
ii) macOS 10.15 (Catalina)
**iii)** Any 64-bit Linux distribution that supports Gnome, KDE, or Unity DE

2.Latest version of android studio(flamingo or eel)
3. brainshop AI key

# 5. Implementation

## 5.1 Step by Step Implementation

Step 1: Creating  a new project

<<img>>

Step 2: Adding  the dependencies in build.gradle file

```
implementation ‘com.android.volley:volley:1.1.1’
```

Volley is an HTTP library that makes networking very easy and fast, for Android apps

Step 3: Adding permission to access internet in androidxml.manifest file

Navigate to the **app > AndroidManifest.xml** and add the below code to it.

```
<!--permissions for internet-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

## 5.2 Activity main.xml code

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

        <!--recycler view to display our chats-->
        <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/idRVChats"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_above="@id/idLLMessage" />

        <LinearLayout
                android:id="@+id/idLLMessage"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_alignParentBottom="true"
                android:orientation="horizontal"
                android:weightSum="5">

                <!--edit text to enter message-->
                <EditText
                    android:id="@+id/idEdtMessage"
                    android:layout_width="0dp"
                    android:layout_height="wrap_content"
                    android:layout_weight="4"
                    android:hint="Enter Message" />

                <!--button to send message-->
                <ImageButton
                    android:id="@+id/idIBSend"
                    android:layout_width="0dp"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center_vertical"
                    android:layout_weight="1"
                    android:background="@color/purple_200"
                    android:src="@android:drawable/ic_menu_send"
                    android:tint="@color/white" />

        </LinearLayout>

</RelativeLayout>
```

### 5.3 Code for MessageModal Class

Message Modal class is created  for storing our messages and to create it
Navigate to the **app > java > your app’s package name > Right-click on it > New > Java class** and name it as MessageModal and we add the below code to it.

```
public class MessageModal {

// string to store our message and sender

        private String message;

        private String sender;

// constructor.

        public MessageModal(String message, String sender) {

                this.message = message;

                this.sender = sender;

        }

// getter and setter methods.

        public String getMessage() {

                return message;

        }

public void setMessage(String message) {

                this.message = message;

        }

public String getSender() {
        return sender;

    }

public void setSender(String sender) {

        this.sender = sender;

    }

}
```

### 5.4 Layout file for storing user messages

By Navigating to the **app > res > layout > Right-click on it > New > layout resource file** and name the file as **user_msg** and we add the below code to it.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end"
    android:layout_margin="5dp"
    android:elevation="8dp"
    app:cardCornerRadius="8dp">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <!--text view for displaying user message-->
        <TextView
            android:id="@+id/idTVUser"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:layout_margin="5dp"
            android:padding="3dp"
            android:text="User message"
            android:textColor="@color/black" />

        <!--we are displaying user icon-->
        <ImageView
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_margin="10dp"
            android:src="@drawable/ic_user" />
        </LinearLayout>
</androidx.cardview.widget.CardView>
```

## 5.5 Layout file for storing Bot Messages

By Navigating to the **app > res > layout > Right-click on it > New > layout resource file** and we name the file as **bot_msg** and add the below code to it.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="start"
    android:layout_margin="5dp"
    android:elevation="8dp"
    app:cardCornerRadius="8dp">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <!--below widget is for image of bot-->
        <ImageView
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_margin="10dp"
            android:src="@drawable/ic_bot" />
        <!--below widget is for
            displaying message of bot-->
        <TextView
            android:id="@+id/idTVBot"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:layout_margin="5dp"
            android:padding="3dp"
            android:text="Bot message"
            android:textColor="@color/black" />

    </LinearLayout>

</androidx.cardview.widget.CardView>
```

### 5.6 Adapter class

For setting data to our items of Chat RecyclerView we have to create an Adapter class. Navigate to the **app > java > our app's package name > Right-click on it > New > Java class** and we name our class as MessageRVAdapter and add the below code to it.

```
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.RecyclerView;

import java.util.ArrayList;

public class MessageRVAdapter extends RecyclerView.Adapter {

    // variable for our array list and context.
    private ArrayList<MessageModal> messageModalArrayList;
```

```
	private Context context;

	// constructor class.
	public MessageRVAdapter(ArrayList<MessageModal>
messageModalArrayList, Context context) {
		this.messageModalArrayList = messageModalArrayList;
		this.context = context;
	}

	@NonNull
	@Override
	public RecyclerView.ViewHolder onCreateViewHolder(@NonNull
ViewGroup parent, int viewType) {
		View view;
		// below code is to switch our
		// layout type along with view holder.
		switch (viewType) {
			case 0:
				// below line we are inflating user message layout.
				view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.user_msg, parent,
false);
				return new UserViewHolder(view);
			case 1:
				// below line we are inflating bot message layout.
				view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.bot_msg, parent,
false);
				return new BotViewHolder(view);
		}
		return null;
	}

	@Override
	public void onBindViewHolder(@NonNull RecyclerView.ViewHolder
holder, int position) {
		// this method is use to set data to our layout file.
		MessageModal modal = messageModalArrayList.get(position);
		switch (modal.getSender()) {
			case "user":
```

```
                // below line is to set the text to our text view of user
layout
                ((UserViewHolder)
holder).userTV.setText(modal.getMessage());
                break;
            case "bot":
                // below line is to set the text to our text view of bot
layout
                ((BotViewHolder)
holder).botTV.setText(modal.getMessage());
                break;
        }
    }

    @Override
    public int getItemCount() {
        // return the size of array list
        return messageModalArrayList.size();
    }

    @Override
    public int getItemViewType(int position) {
        // below line of code is to set position.
        switch (messageModalArrayList.get(position).getSender()) {
            case "user":
                return 0;
            case "bot":
                return 1;
            default:
                return -1;
        }
    }

    public static class UserViewHolder extends RecyclerView.ViewHolder {

        // creating a variable
        // for our text view.
        TextView userTV;

        public UserViewHolder(@NonNull View itemView) {
            super(itemView);
            // initializing with id.
            userTV = itemView.findViewById(R.id.idTVUser);
        }
    }

    public static class BotViewHolder extends RecyclerView.ViewHolder {

        // creating a variable
        // for our text view.
        TextView botTV;

        public BotViewHolder(@NonNull View itemView) {
            super(itemView);
            // initializing with id.
            botTV = itemView.findViewById(R.id.idTVBot);
        }
    }
}
```

### 5.7 **Working with the MainActivity.java file**

On **MainActivity.java** file we have to add our api key and api url to the code. Below is the code for the **MainActivity.java** file

```
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.android.volley.toolbox.Volley;

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

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

        // creating variables for our
        // widgets in xml file.
        private RecyclerView chatsRV;
        private ImageButton sendMsgIB;
        private EditText userMsgEdt;
        private final String USER_KEY = "user";
        private final String BOT_KEY = "bot";

        // creating a variable for
        // our volley request queue.
        private RequestQueue mRequestQueue;

        // creating a variable for array list and adapter class.
        private ArrayList<MessageModal> messageModalArrayList;
        private MessageRVAdapter messageRVAdapter;

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

                // on below line we are initializing all our views.
                chatsRV = findViewById(R.id.idRVChats);
                sendMsgIB = findViewById(R.id.idIBSend);
                userMsgEdt = findViewById(R.id.idEdtMessage);

                // below line is to initialize our request queue.
                mRequestQueue = Volley.newRequestQueue(MainActivity.this);
                mRequestQueue.getCache().clear();
        // creating a new array list
        messageModalArrayList = new ArrayList<>();

        // adding on click listener for send message button.
        sendMsgIB.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // checking if the message entered
                // by user is empty or not.
                if (userMsgEdt.getText().toString().isEmpty()) {
                    // if the edit text is empty display a toast message.
                    Toast.makeText(MainActivity.this, "Please enter your
message..", Toast.LENGTH_SHORT).show();
                    return;
                }

                // calling a method to send message
                // to our bot to get response.
                sendMessage(userMsgEdt.getText().toString());

                // below line we are setting text in our edit text as empty
                userMsgEdt.setText("");
            }
        });

        // on below line we are initializing our adapter class and passing our array
list to it.
        messageRVAdapter = new MessageRVAdapter(messageModalArrayList,
this);

        // below line we are creating a variable for our linear layout manager.
        LinearLayoutManager linearLayoutManager = new
LinearLayoutManager(MainActivity.this, RecyclerView.VERTICAL, false);

        // below line is to set layout
        // manager to our recycler view.
        chatsRV.setLayoutManager(linearLayoutManager);

        // below line we are setting
        // adapter to our recycler view.
        chatsRV.setAdapter(messageRVAdapter);
    }

    private void sendMessage(String userMsg) {
        // below line is to pass message to our
        // array list which is entered by the user.
        messageModalArrayList.add(new MessageModal(userMsg,
USER_KEY));
        messageRVAdapter.notifyDataSetChanged();

        // url for our brain
        // make sure to add mshape for uid.
        // make sure to add your url.
        String url
=http://api.brainshop.ai/get?bid=174808&key=B9R4pgFtegdj9CLX&uid=[uid]&msg=[msg] + userMsg;

        // creating a variable for our request queue.
        RequestQueue queue = Volley.newRequestQueue(MainActivity.this);

        // on below line we are making a json object request for a get request and
passing our url .
        JsonObjectRequest jsonObjectRequest = new
JsonObjectRequest(Request.Method.GET, url, null, new
Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
                try {
                    // in on response method we are extracting data
                    // from json response and adding this response to our
array list.
                    String botResponse = response.getString("cnt");
                    messageModalArrayList.add(new
MessageModal(botResponse, BOT_KEY));

                    // notifying our adapter as data changed.
                    messageRVAdapter.notifyDataSetChanged();
                } catch (JSONException e) {
                    e.printStackTrace();

                    // handling error response from bot.
                    messageModalArrayList.add(new
MessageModal("No response", BOT_KEY));
                    messageRVAdapter.notifyDataSetChanged();
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                // error handling.
                messageModalArrayList.add(new MessageModal("Sorry no
response found", BOT_KEY));
                Toast.makeText(MainActivity.this, "No response from the
bot..", Toast.LENGTH_SHORT).show();
            }
        });

        // at last adding json object
        // request to our queue.
        queue.add(jsonObjectRequest);
    }}
```

## 5.8 Android manifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.chatbot">

    <application
        android:allowBackup="true"
        android:usesCleartextTraffic="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Chatbot"
        tools:targetApi="31">
        <activity
```

```
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

## 5.9 RETROFIT API.JAVA

```
package com;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Url;

public interface RetrofitAPI

{
    @GET
    Call<MsgModal> getMessage(@Url String url);

    Call<MsgModal> getmessage(String url);
}
```

# 6.INTERFACING

## 6.1 Genrating  API key for interfacing the chatbot service

Using  Brainshop.ai  website we  generate our user account with our username and password.After  Simply creating our account on the website. will get to see the below screen, After creating our account we have to request a new password from the request password option and enter our email address. After adding our email address we have to add the password to our account. Now we will be to generate our API key.

After logging in we have to create a brain by clicking on the add brain option we get an popup window  to create the brain and  generate API key

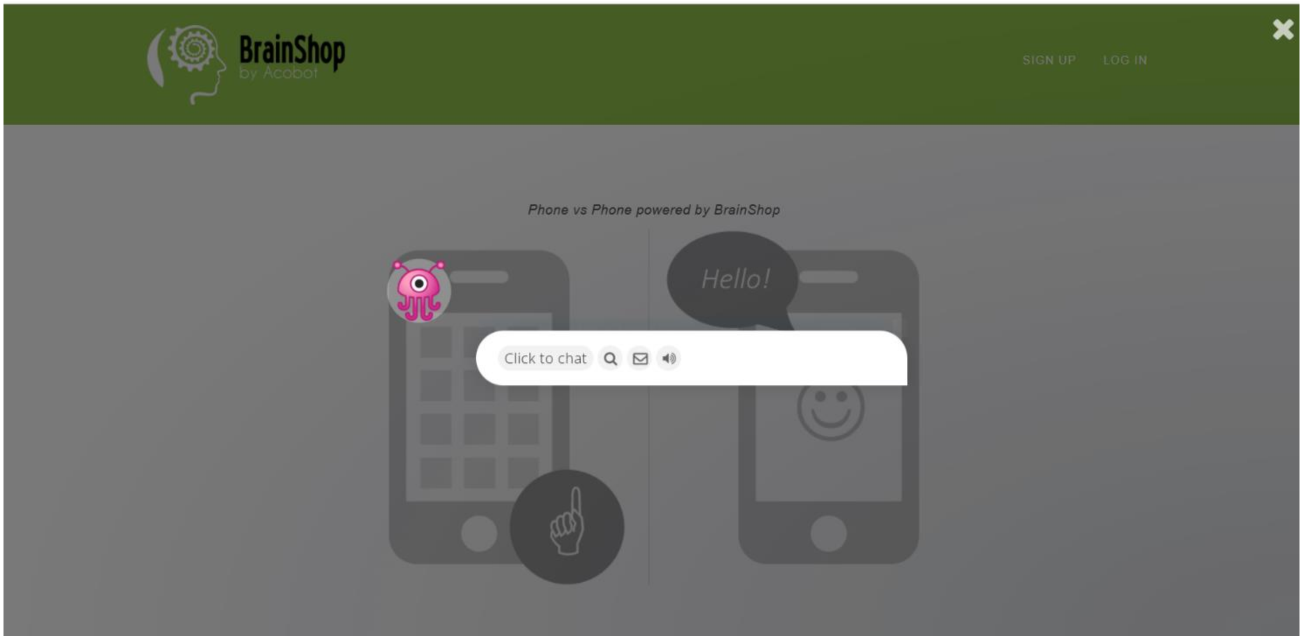
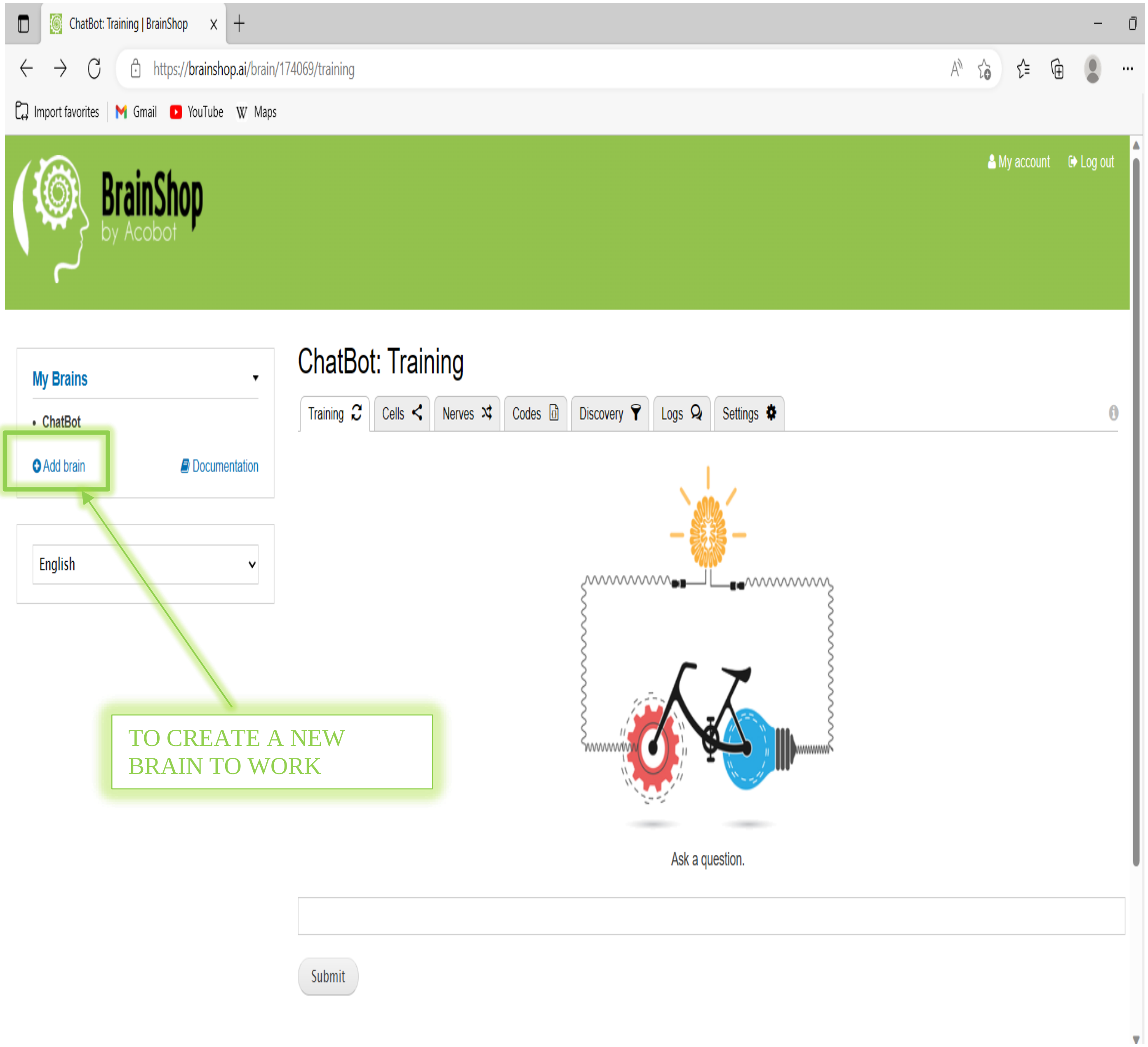


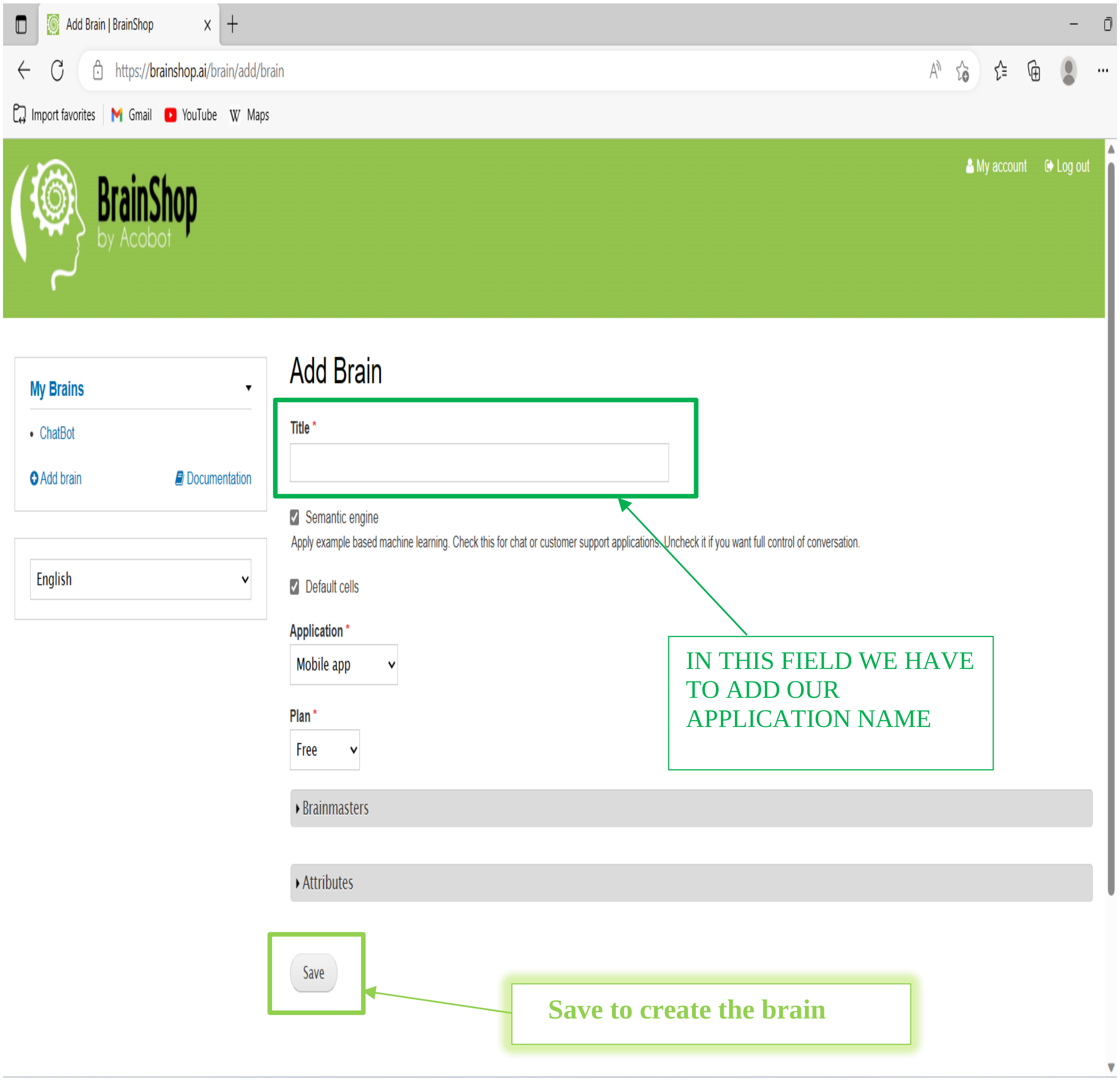
Fig 9: Brainshopai web page



TO CREATE A NEW BRAIN TO WORK

Fig 10: Creating brain

Click +add brain > create a root brain -→ we get this tab



Add Brain

Title *

☑ Semantic engine

Apply example based machine learning. Check this for chat or customer support applications. Uncheck it if you want full control of conversation.

☑ Default cells

Application * Mobile app

Plan * Free

Brainmasters

Attributes

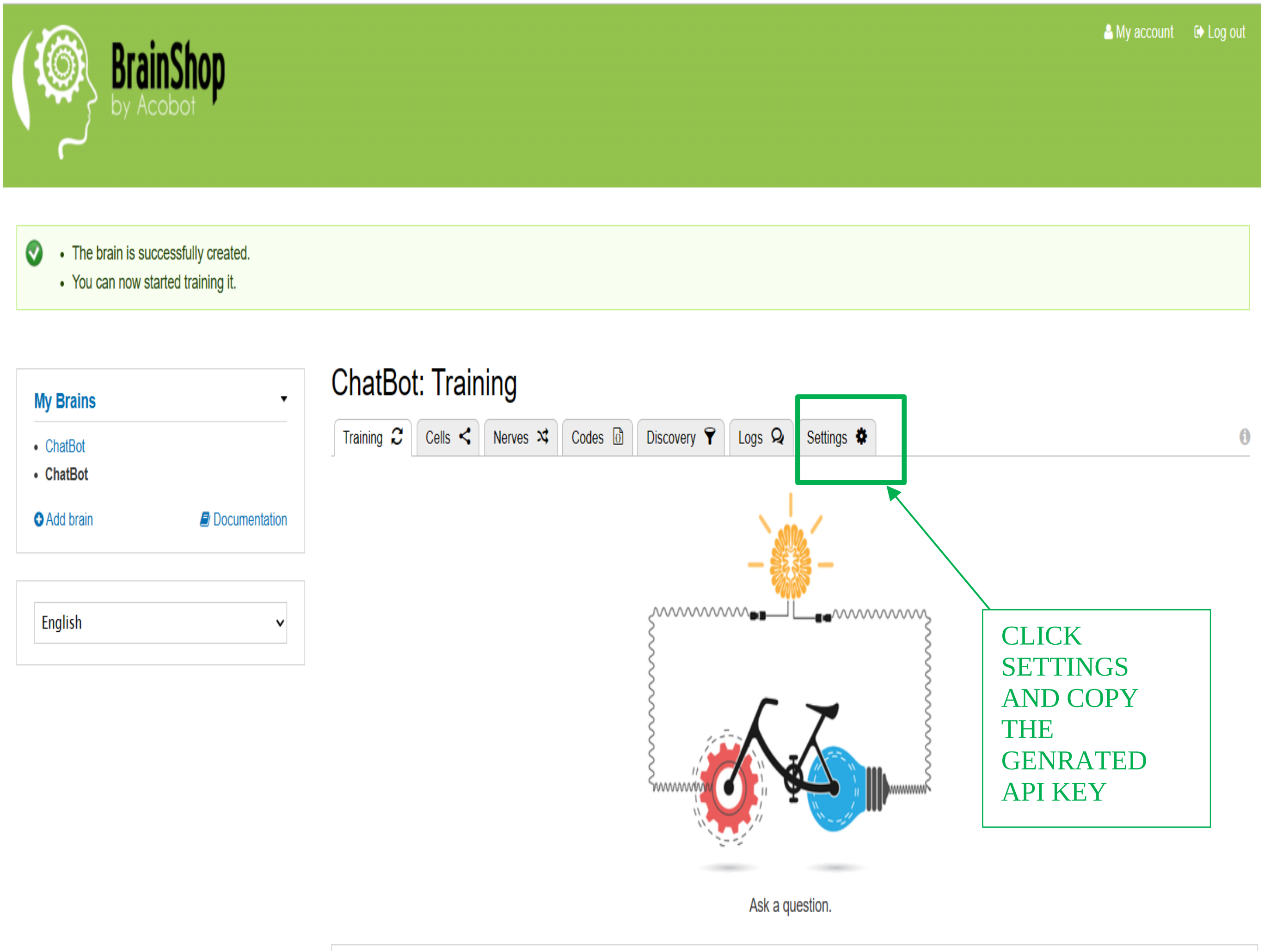
Save

IN THIS FIELD WE HAVE TO ADD OUR APPLICATION NAME

Save to create the brain

Fig 11: Adding  brain

After creating the brain click > settings >copy the API url and API key



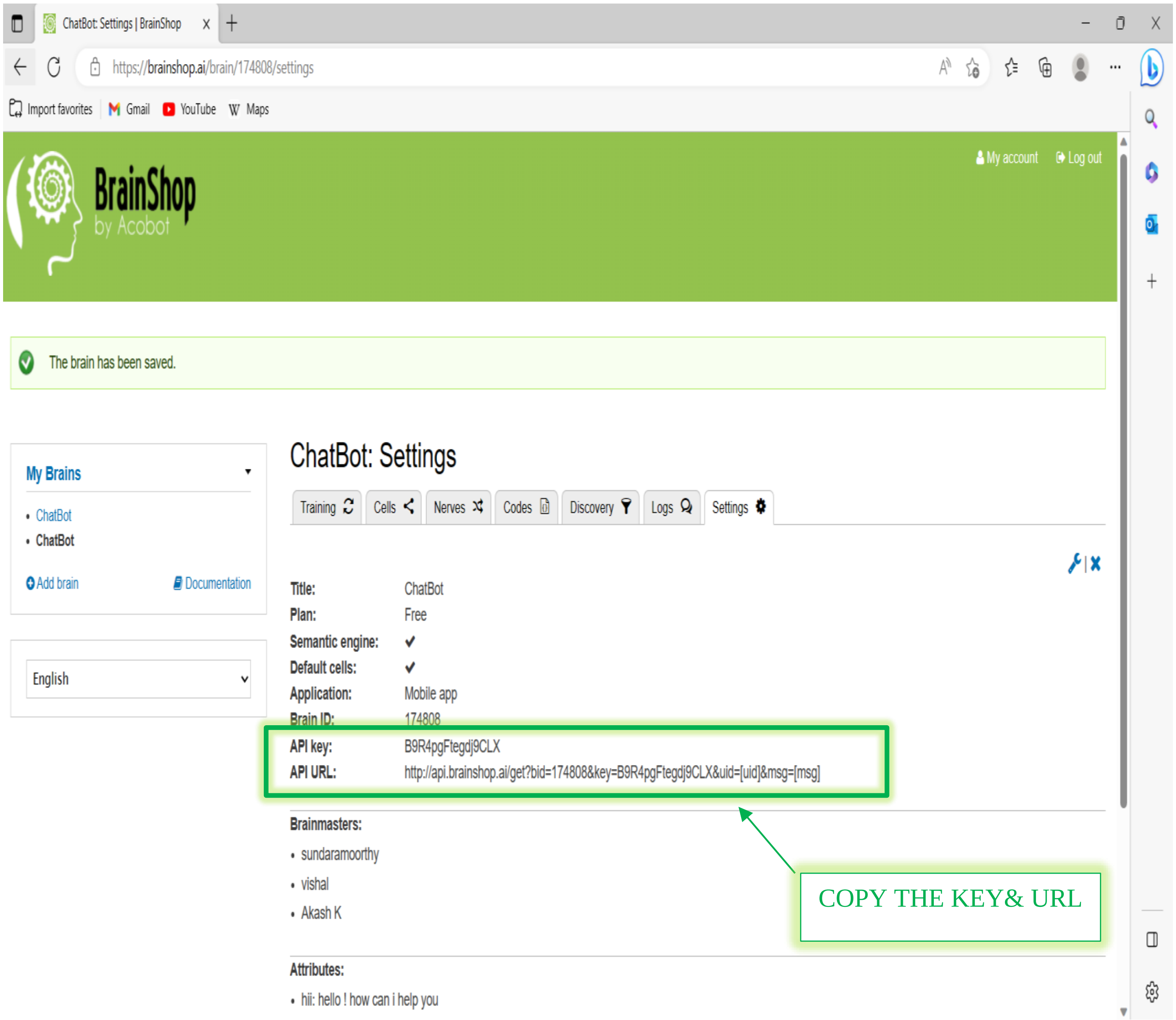
- The brain is successfully created.
- You can now started training it.

ChatBot: Training

Ask a question.

CLICK SETTINGS AND COPY THE GENRATED API KEY

Fig 12: Traing The Model



The brain has been saved.

- ChatBot
- ChatBot

English

ChatBot: Settings

| | |
|---|---|
| Title: | ChatBot |
| Plan: | Free |
| Semantic engine: | ✓ |
| Default cells: | ✓ |
| Application: | Mobile app |
| Brain ID: | 174808 |
| API key: | B9R4pgFtegdj9CLX |
| API URL: | http://api.brainshop.ai/get?bid=174808&key=B9R4pgFtegdj9CLX&uid=[uid]&msg=[msg] |

Brainmasters:

- sundaramoorthy
- vishal
- Akash K

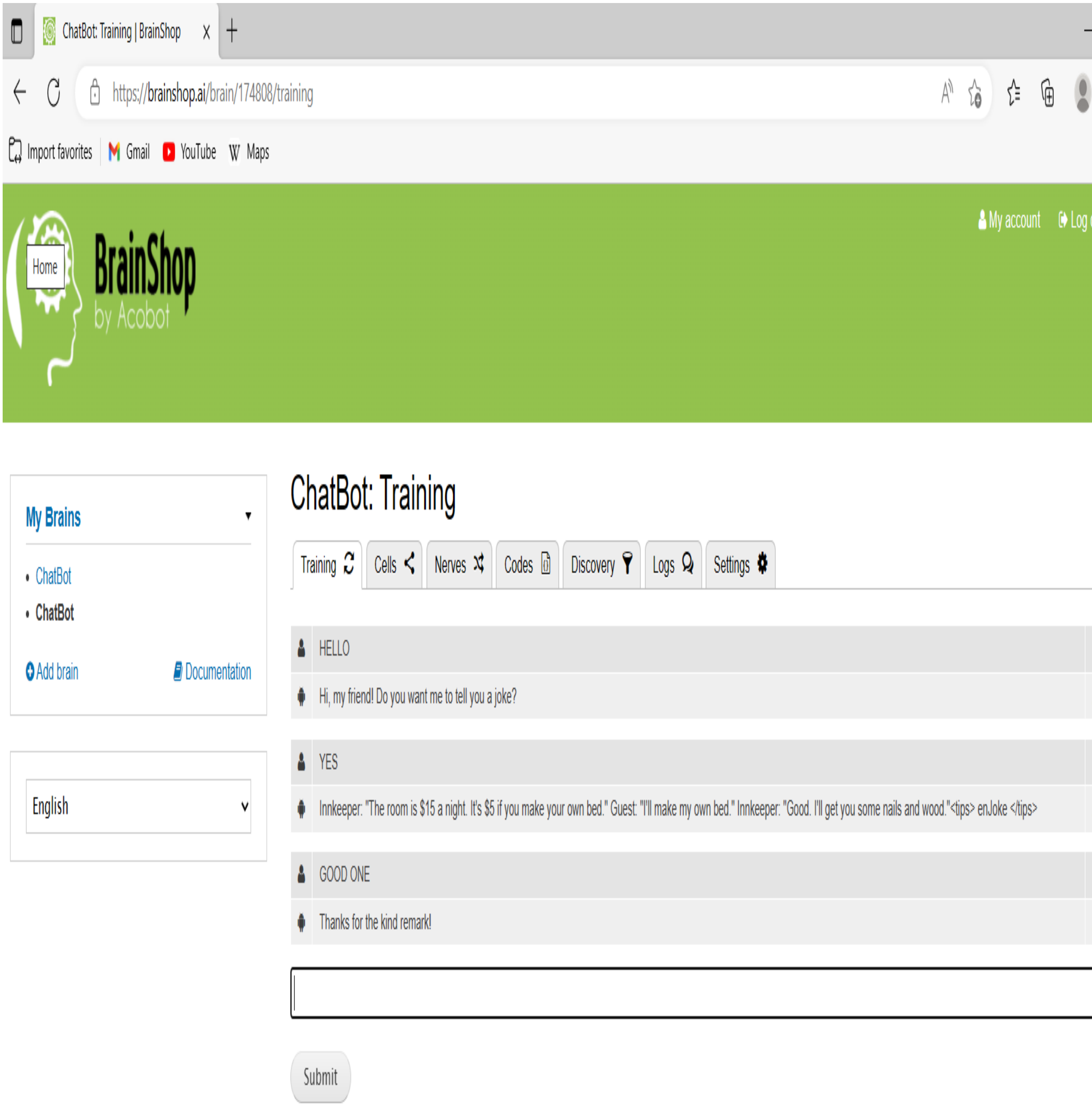
Attributes:

- hii: hello ! how can i help you

COPY THE KEY& URL

Fig 13: Genrating API Key

## 6.2 TRAINING THE BRAIN



ChatBot: Training

HELLO

Hi, my friend! Do you want me to tell you a joke?

YES

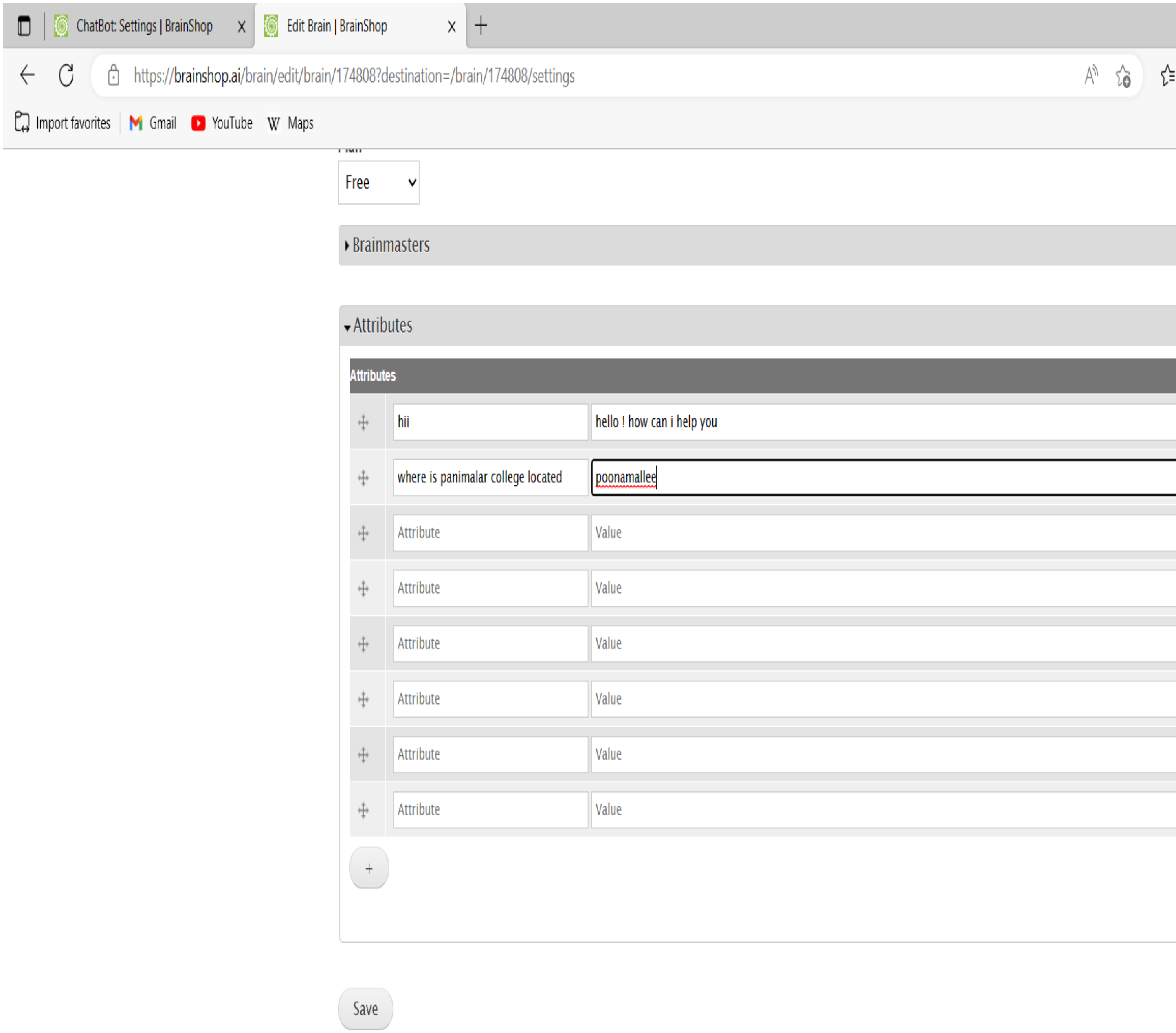
Innkeeper: "The room is $15 a night. It's $5 if you make your own bed." Guest: "I'll make my own bed." Innkeeper: "Good. I'll get you some nails and wood."<tips> enJoke </tips>

GOOD ONE

Thanks for the kind remark!

## 6.3 DEFAULT ATTRIBUTES

Default attributes can also be feeded  to the brain which provides a predefined text for the user queries Brain shop provides n amount of default attributes to add to the brain.

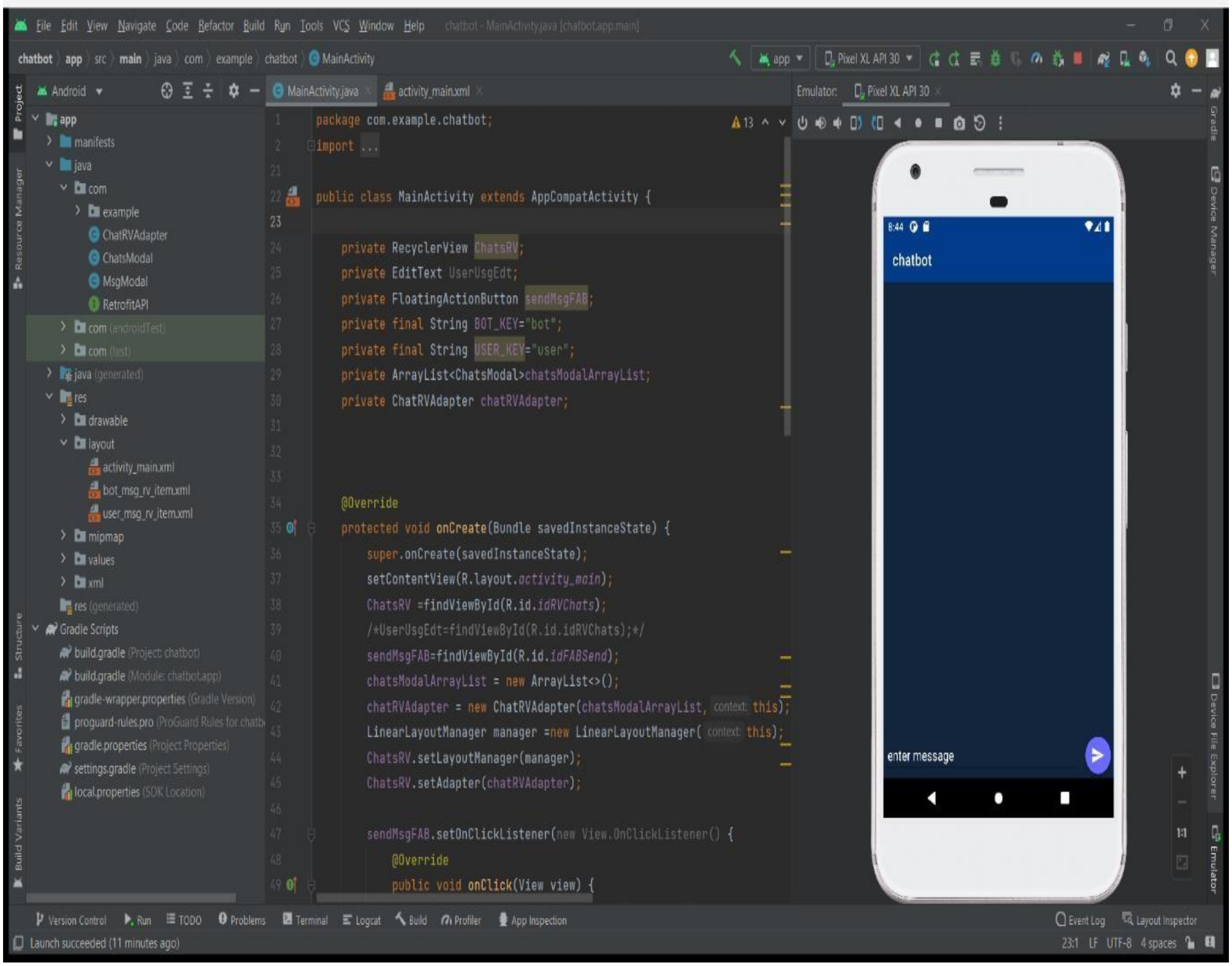


ChatBot: Settings | BrainShop
Edit Brain | BrainShop
https://brainshop.ai/brain/edit/brain/174808?destination=/brain/174808/settings
Import favorites
Gmail
YouTube
Maps
Free
Brainmasters
Attributes
+
Save

| Attributes | |
|---|---|
| hii | hello ! how can i help you |
| where is panimalar college located | poonamallee |
| Attribute | Value |
| Attribute | Value |
| Attribute | Value |
| Attribute | Value |
| Attribute | Value |
| Attribute | Value |

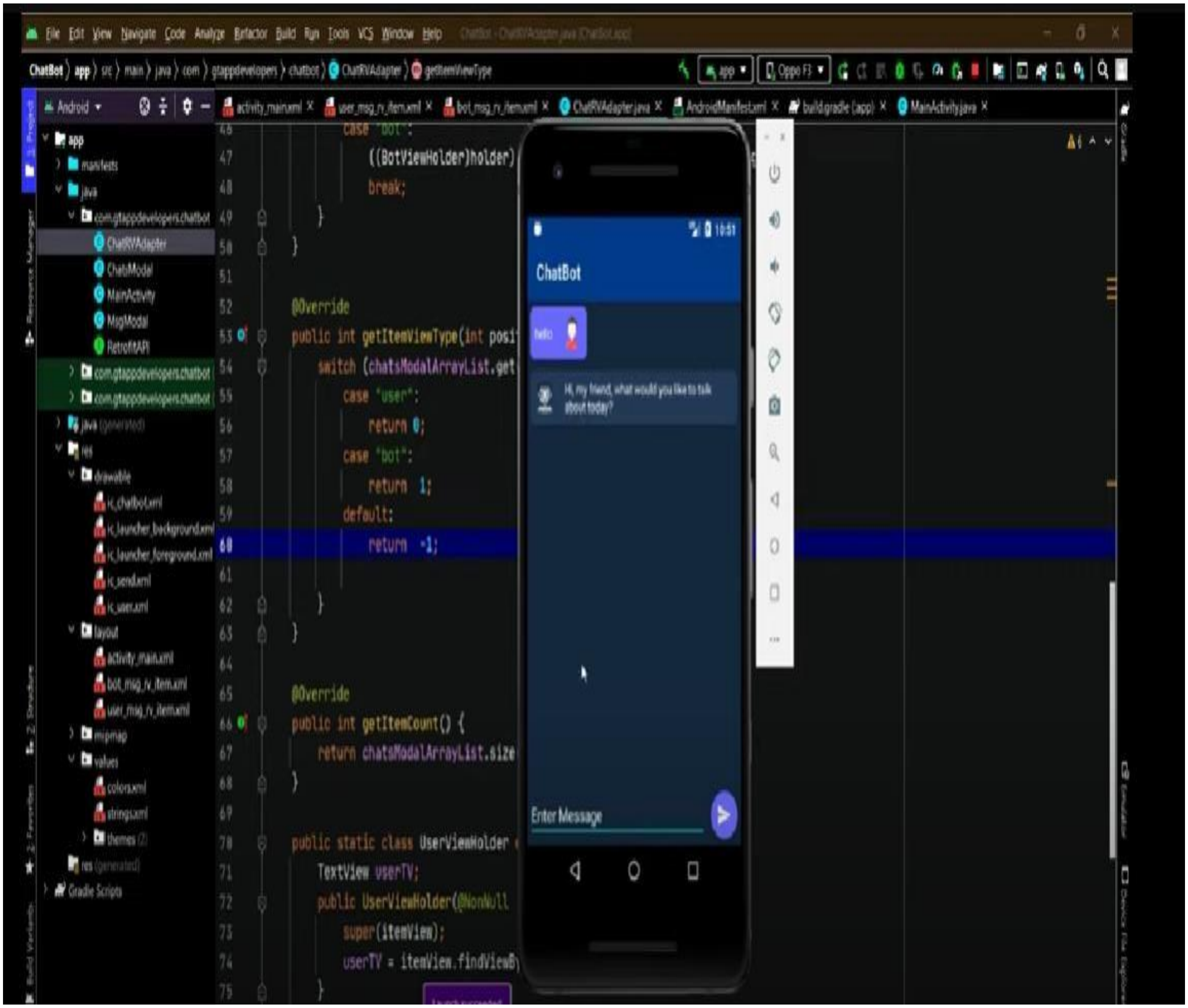
Fig 15: Default Attributes

## 6.4 OUTPUT SCREENSHOTS



```java
package com.example.chatbot;
import ...

public class MainActivity extends AppCompatActivity {

    private RecyclerView ChatsRV;
    private EditText UserUsgEdt;
    private FloatingActionButton sendMsgFAB;
    private final String BOT_KEY="bot";
    private final String USER_KEY="user";
    private ArrayList<ChatsModal>chatsModalArrayList;
    private ChatRVAdapter chatRVAdapter;



    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ChatsRV =findViewById(R.id.idRVChats);
        /*UserUsgEdt=findViewById(R.id.idRVChats);*/
        sendMsgFAB=findViewById(R.id.idFABSend);
        chatsModalArrayList = new ArrayList<>();
        chatRVAdapter = new ChatRVAdapter(chatsModalArrayList, this);
        LinearLayoutManager manager =new LinearLayoutManager( this);
        ChatsRV.setLayoutManager(manager);
        ChatsRV.setAdapter(chatRVAdapter);

        sendMsgFAB.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
chatbot - MainActivity.java [chatbot.app.main]
chatbot › app › src › main › java › com › example › chatbot › MainActivity
app
Pixel XL API 30
Android
MainActivity.java
activity_main.xml
Emulator: Pixel XL API 30
app
manifests
java
com
example
ChatRVAdapter
ChatsModal
MsgModal
RetrofitAPI
com (androidTest)
com (test)
java (generated)
res
drawable
layout
activity_main.xml
bot_msg_rv_item.xml
user_msg_rv_item.xml
mipmap
values
xml
res (generated)
Gradle Scripts
build.gradle (Project: chatbot)
build.gradle (Module: chatbot.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for chatb
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
8:44
chatbot
enter message
Version Control
Run
TODO
Problems
Terminal
Logcat
Build
Profiler
App Inspection
Event Log
Layout Inspector
Launch succeeded (11 minutes ago)
23:1 LF UTF-8 4 spaces



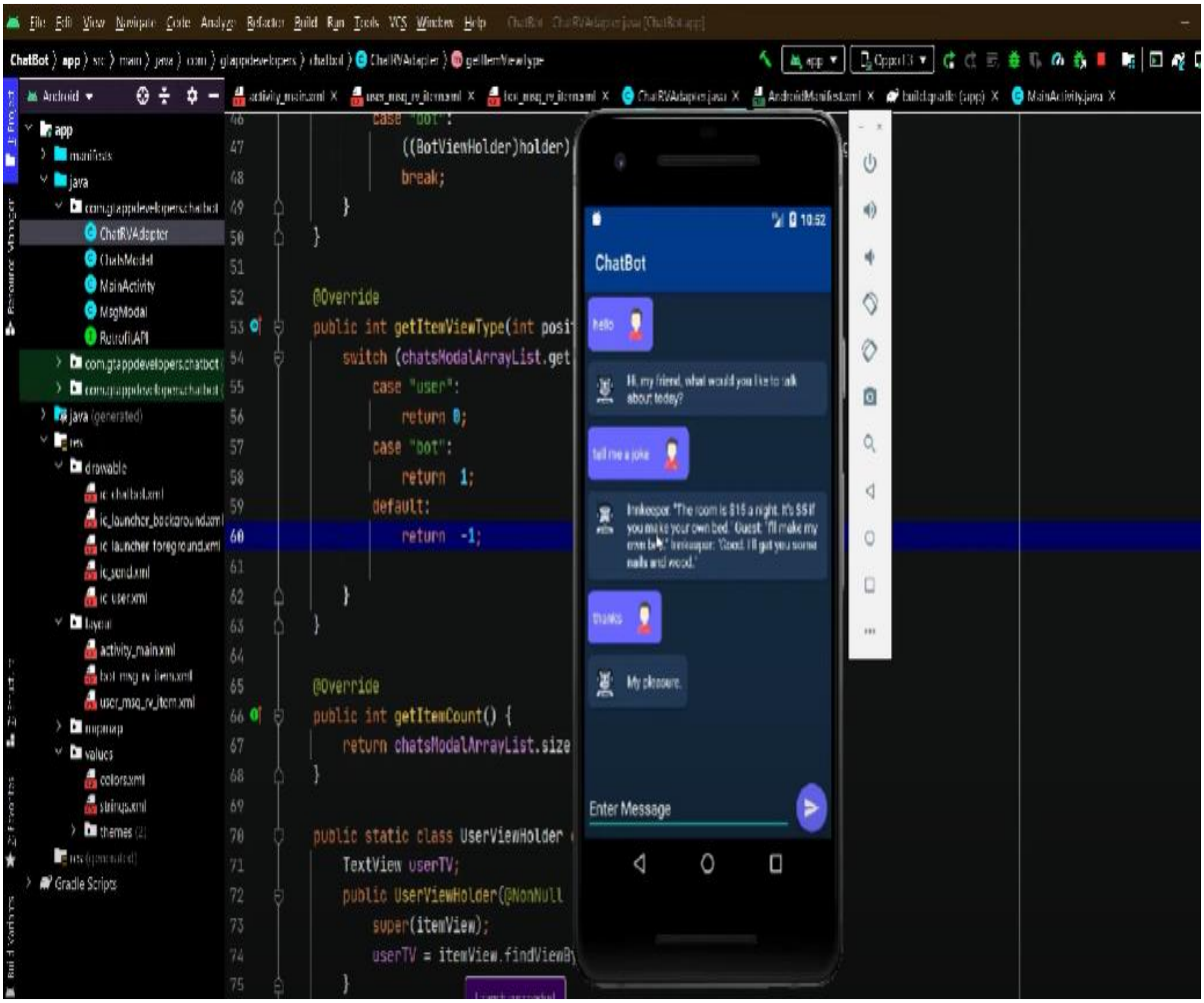
```
                case "bot":
                    ((BotViewHolder)holder)
                    break;
            }
        }

    @Override
    public int getItemViewType(int posi
        switch (chatsModalArrayList.get
            case "user":
                return 0;
            case "bot":
                return  1;
            default:
                return  -1;

        }
    }

    @Override
    public int getItemCount() {
        return chatsModalArrayList.size
    }

    public static class UserViewHolder e
        TextView userTV;
        public UserViewHolder(@NonNull
            super(itemView);
            userTV = itemView.findViewBy
        }
```

ChatBot

hello

Hi, my friend, what would you like to talk about today?

Enter Message



```
                case "bot":
                    ((BotViewHolder)holder)
                    break;
            }
        }

        @Override
        public int getItemViewType(int posi
            switch (chatsModalArrayList.get
                case "user":
                    return 0;
                case "bot":
                    return  1;
                default:
                    return  -1;

            }
        }

        @Override
        public int getItemCount() {
            return chatsModalArrayList.size
        }

        public static class UserViewHolder
            TextView userTV;
            public UserViewHolder(@NonNull
                super(itemView);
                userTV = itemView.findViewBy
            }
```

ChatBot

hello

Hi, my friend, what would you like to talk about today?

tell me a joke

Innkeeper: "The room is $15 a night. It's $5 if you make your own bed." Guest: "I'll make my own bed." Innkeeper: "Good. I'll get you some nails and wood."

thanks

My pleasure.

Enter Message

# 7.CONCLUSION

## 7.1 Conlusions

In conclusion, a customized chatbot can have significant benefits for businesses, particularly in the area of customer engagement and support. Through the use of natural language processing and artificial intelligence, chatbots can provide personalized assistance and support to customers, addressing their needs quickly and efficiently.we created an effective customized chatbot which can be used for businesses and can be customized according to user needs of their customers, many types of interactions the chatbot has, and it can be used in various platforms on which we can deploy. The chatbot has designed to be user-friendly and intuitive, allowing customers to interact with it easily and without frustration. Overall our customized chatbot is a valuable tool for businesses looking to improve customer engagement and support. By providing personalized assistance and support, our chatbot can enhance the customer experience, increase customer satisfaction, and ultimately drive business growth.

a) This project provides the user to customize the chatbot application by allowing user to change the application According to his needs.

b) Unlike other bots ,The user can change font, color ,theme ,icons and widgets manually .

c)The bot can me made to give responses according to the users business and users need by training the brain in brainshopai on any domain.

d) This projects can be deployed in many applications such as Google Play Store, Amazon Appstore, Samsung Galaxy Store, Huawei AppGallery and more and can be integarated and used in mobile, laptops ,etc,.

## 7.2 limitations of the system

a) Chatbot may not fully comprehend a customer's input and may provide incoherent answers.

b) Chatbot lack decision-making.

c) Chatbot may collect sensitive data from users, and if not secured properly, it can lead to data breaches.

d) Our Chatbot is fully reliant on brainshop open ai and may experience downtime or errors that can disrupt customer service and support.

# 8. REFERENCE

1. Lo, A., & Ren, Y. (2020). A study of the effectiveness of customized chatbot for e-commerce. Journal of Electronic Commerce Research, 21(3), 267-280.
2. Li, X., Li, Y., Wang, W., & Li, L. (2020). Development of a customized chatbot for customer service in the retail industry. International Journal of Industrial Ergonomics, 79, 103068.
3. Radziwon, A., Jabloński, B., & Lupa-Zatwarnicka, K. (2020). Customizing chatbots in the healthcare sector: A review of the literature. Healthcare, 77.
4. Kesharwani, A., & Mishra, A. (2021). Customized chatbots: An analysis of the key determinants for adoption in e-commerce. Journal of Retailing and Consumer Services, 61, 102556.
5. Emanuela Haller, Traian Rebedea, “Designing a Chat-bot that Simulates an Historical Figure”, IEEE Conference Publications, July 2013.
6. Pratik Slave, Vishruta Patil, Vyankatesh Gaikwad, Girish Wadhwa, “College Enquiry Chat Bot”, International Journal on Recent and Innovation Trends in Computing and Communication, Volume 5, Issue 3, March 2015.
7. “AIML Based Voice Enabled Artificial Intelligent Chatterbot”, International Journal of u- and e- Service, Science and Technology Volume 8 - No. 2, 2015.
8. Amey Tiwari, Rahul Talekar, Prof. S. M. Patil, “College Information Chatbot System”, International Journal of Engineering Research and General Science, Volume 2, Issue 2, April 2017.
9. Nimavat, K., Champaneria, T.: Chatbots: an overview types, architecture, tools and future possibilities. Int. J. Sci. Res. Dev. 5, 1019–1024 (2017) 35.
10. Kucherbaev, P., Bozzon, A., Houben, G.-J.: Human-aided bots. IEEE Internet Comput. 22, 36–43 (2018). https://doi.org/10.1109/MIC.2018.252095348