NATIONAL INSTITUTE OF TECHNOLOGY TIRUCHIRAPPALLI

Department of Computer Applications

COVID-19 CHATBOT

AI PROJECT WORK

Submitted By:

VAIBHAV VIKAS (205118081)

Under the guidance of

Dr. (Mrs.) R. Eswari.

NATIONAL INSTITUTE OF TECHNOLOGY TIRUCHIRAPPALLI

<u>CERTIFICATE</u>

*This is to certify that **Vaibhav Vikas,** bearing Roll No.: **205118081**, student of 5th semester MCA (batch 2018-2021) of National Institute of Technology, Tiruchirappalli has successfully completed the project **Covid19 Chatbot** under the guidance of **Dr. (Mrs.) R. Eswari.***

Dr. (Mrs.) R. Eswari	**Dr. P.J.A. Alphonse**
Project Guide	**Head of the Department**

BONAFIDE CERTIFICATE

This is to certify that the project “**Covid19 Chatbot"** is a project work successfully done by **Vaibhav Vikas** (**205118081**) for the subject of **Artificial Intelligence** in partial fulfilment of the requirements for the awards of the degree of **Master of Computer Applications** from **National Institute of Technology**, **Tiruchirappalli**, during the academic year 2019–2020 (5th Semester).

Dr. (Mrs.) R. Eswari.	**Dr. P.J.A. Alphonse**
Project Guide	**Head of the Department**

ACKNOWLEDGEMENTS

At the outset, I thank Lord Almighty for gracing with all strength and health for carrying out my project work.

I express my sincere and heartfelt gratitude to my project supervisor and well-wisher **Dr. (Mrs.) R. Eswari**, Professor, Department of Computer Applications, National Institute of Technology, Tiruchirappalli. I sincerely thank for her constant support, care, guidance, and regular interaction throughout my project.

I am also grateful to **Dr. Mini Shaji Thomas**, Director, National Institute of Technology, Tiruchirappalli for providing the infrastructure and facilities to carry out the project. I am very much grateful to **Dr. P.J.A. Alphonse**, Professor and Head, Department of Computer Applications, National Institute of Technology, Tiruchirappalli and faculty members of Computer Applications department for their support throughout my project work.

Last but not least, I thank my family members for their understanding, affection and moral support through the ups and downs of my project.

ABSTRACT

Conversational systems are becoming pervasive as a basis for human computer interaction as we seek more natural ways to integrate automation via voice assistants and chatbots. Well-known examples of conversational AI include Google Assistant, Apple's Siri, Amazon's Alexa and Microsoft's Cortana but conversational systems are becoming widespread with platforms like Facebook Messenger, Discord, reddit opening to chatbot developers and companies. Common tasks for conversational systems include scheduling meetings, booking flights, customer support tasks, and solving general frequently asked questions.

The purpose is to make a full-fledged conversational system that can interact to users in a much natural way so that user feels like they are talking to a real person and not an AI. Also, the system must be able to handle various tasks such as opening applications, making API calls, and handling Fallbacks much efficiently. Furthermore, we are also developing a web based react User Interface which can be used as a frontend to handle the various inputs and demonstrated the output to the users in a much user-friendly way with the support for quick replies, images, gifs, videos, charts, and buttons etc.

TABLE OF CONTENTS

1. Introduction:

Chatbots are software agents used to interact between a computer and a human in natural language. Just as people use language for human communication, chatbots use natural language to communicate with human users. It can do real conversations with textual and/or auditory methods. Using Artificial Intelligence (AI), chatbots can simulate human conversations. There are two categories of chatbots. One category is command based chatbots where chatbots rely on a databank of replies and heuristics. The user must be very specific while asking the questions so that the bot can answer. Hence, these bots can answer limited set of questions and cannot perform function outside of the code. The other category is chatbots based on AI or machine learning algorithms, these bots can answer ambiguous questions which means the user do not have to be specific while asking questions. Thus, these bots create replies for the user's queries using Natural Language Processing (NLP).

AI-powered chatbots are motivated by the need of traditional websites to provide a chat facility where a bot is required to be able to chat with user and solve queries. When live agent can handle only two to three operations at a time, chatbots can operate without an upper limit which really scales up the operations. Also, if any school or business is receiving lots of queries, having a chatbot on a website takes off the load from support team. Having a chatbot clearly improves the response rate compared to human support team. In addition, since millennials prefer live chats over a phone call, they find a chatbot, which provide a highly interactive marketing platform, very attractive. Furthermore, a chatbot can automate the repetitive tasks. There can be some scenarios where a business or school receives same queries in a day for many times and support team must respond to each query repetitively. Lastly, the most important advantage of having a chatbot is that it is available 24/7. No matter what time it is, a user can get a query solved. All these advantages of a chatbot constitute the motivation to implement a Covid Enquiry Chatbot.

Our project takes inspiration from a number of sources such as from scikit-learn (focus on consistent APIs over strict inheritance) and Keras (consistent APIs with different backend implementations), and indeed both of these libraries are (optional) components of a Rasa application. Text classification is loosely based on the fastText approach. Sentences are represented by pooling word vectors for each constituent token. Using pre-trained word embeddings such as GloVe, the trained intent classifiers are remarkably robust to variations in phrasing when trained with just a few examples for each intent. Braun et al [3] show that The NLU's performance compares favorably to various closed-source solutions. Custom entities are recognized using a conditional random field. The approach to dialogue management is most similar to Hybrid code networks but takes a different direction from many recent research systems. There is currently no support for end-to-end learning, as in A network-based end-to-end trainable task-oriented dialogue system, where natural language understanding, state tracking, dialogue management, and response generation are all jointly learned from dialogue transcripts.

2. Problem Statement:

The use of chat services on phone as well as on web have increased to a significant level. A lot of Companies rely on Chat system for solving their user enquiries as well as general FAQs. Most of the times the questions asked by the users are very trivial i.e. can simply be answered easily rather than requiring a human interaction. Using Chatbot we can simply reply to user FAQs queries and in the case, user asks something out of the bot knowledge base we can redirect them to the Customer Service.

The challenges with pre-existing bot frameworks includes issues such as bot response accuracy, time taken to train the bot, no pre-available methods of logging chats, there is no seamless communication between the backend and the frontend, no support for emojis, images, videos, coursels etc. and also lack many ways to open actions and do custom task from the chatbot.

A domain specific chatbot will be implemented to assist users with their queries. In order to overcome the user satisfaction issues associated with covid doubts and questions. The chatbot will provide personal and efficient communication between the user and application in order to resolve their doubts and get assistance when needed, such as; answering any queries and checking symptoms. The chatbot will allow users to feel confident and comfortable when using this service regardless of the user's computer literacy due to the natural language used in messages. It also provides a very accessible and efficient service as all interactions will take place within the one chat conversation negating the need for the user to navigate through a site.

The proposed solution is to create a chatbot to simulate a human conversation to assist users with their banking needs and to provide a more personal experience. Advancements in artificial Intelligence, machine learning techniques, improved aptitude for decision making, larger availability of domains and corpus, have increased the practicality of integrating a chat bot into applications (Dole et al., 2015). Users will be able to ask any Covid19 related queries in natural language that they are comfortable using such as; view covid information, stats in states and check their doubts. The chatbot will identify and understand what the user is asking and generate an appropriate response based on the conversational context. Immediate responses will be provided by the chatbot to redeem the need for the user to have to call or visit Hospitals and wait in queue in order to get through to an advisor for assistance.

3. **Platform (Hardware and Software Requirements**):

Hardware:

- Processor – Intel® Core™ i3
- RAM – 4GB
- Operating system – Windows 7 to 10 or Ubuntu 18.04 or higher

Software:

- WSL if using Windows 10
- Python 3.7.9
- Node 10.19.0 or higher
- Visual Studio Code
- Google Chrome or an equivalent Web Broweser
- JavaScript
- Postman

Library:

- TensorFlow
- RASA Framework
- Scikit-learn
- DialogFlow
- Keras
- Numpy
- Pandas
- Json
- Nginx

4. Requirements Gathering/Analysis:

To determine the requirements of an application, information gathering must be carried out in order to know what the end user wants the system to be able to do and what the user hopes to achieve through using the application. The most efficient way to gather information for this project would be the use of a Questionnaire, distributed online to the general public. This will reach a large user group, producing a diverse and varied result set for analysis providing detailed scope on what users from different age groups and backgrounds are expecting from an application such as this. It will give valuable insight into the age group, how often the user utilizes online banking, if they would prefer assistance through an advisor on phone or using a friendly chatbot service, level of computer literacy and their expectations from the application. These questionnaires will be completed using Survey Monkey, which provides free cloud-based expert custom templates for surveys and robust analytical features with a UI to display data analysis in a number of formats that make it straight forward to read and analyses the data to identify requirements. The structured questionnaire will be made up of open and close ended questions as a convenient means of extracting information from users. There will be clear descriptions throughout the online questionnaire making it easy to navigate through and use, increasing the likelihood of participation from users. Clear and concise information will also be given on how the user should respond to each question accordingly (M. McGuirk , P.M and P. O'Neill, P. 2016). The questionnaire will be well presented with a consistent color theme throughout provided by Survey Monkey. A range of different questioning techniques will be used such as point scales, ranges, open and closed question. It’s clear from analysing the responses of this question that a small number of the respondents have interacted with a chatbot before as only 11.54 % had previously interacted with a chatbot. The majority of users that have interacted with a chatbot, even if the interaction did not go seamlessly had a very positive opinion of chatbots in general and actually felt as though they were communicating with a human as the conversation felt very natural. , this is due to the chatbot having the ability to understand the natural language used in messages and generate an appropriate response through various artificial intelligence methods such as natural language processing. Users found it to be very informative and efficient when replying to their queries . Although some respondents found that it could take a long time to reply or provide very simplistic responses to more complex tasks and become unresponsive. This reaffirms the need for the chatbot to have a method that will give the user guidance if it does not understand the context of the conversation or if it cannot assist the user any further.

4.1 Functional and Non-Functional Requirements:

Functional and Non-functional requirements are identified through the analysis of the data collected from the survey. Functional requirements are the features and functionality that the system must have or be able to perform whereas non-functional requirements define the manner or characteristic the system must have such as: performance, usability, modifiability, maintainability, security, scalability, reliability, availability, configurability and design constraints (Tsui et al, 2016)

4.1.1 Functional Requirements:

- The chat bot must log the conversations between the bot and the user so the admin can see it later to fix and improve various queries.
- The Chat bot will integrate with the covid19india.org API to get status about the counts and numbers of the patients in different states.
- The Chat bot will assist users with their queries and carry out appropriate actions such as opening apps, showing them video tutorials.
- Users will be able to converse with the Chat bot through voice or text commands and it will understand what the user is saying through natural language processing provided through the integration of Dialogflow API.
- The chat bot should be able to maintain the conversational state when the context may be unclear through previous messages and conversations.
- Provide text and audio responses.

4.1.2 Non-Functional Requirements

- The chatbot must be efficient with very little lag in response time for instance no longer than 5 seconds to reply to a user message.
- The chatbot must be reliable with next to no faults or bugs
- The database must be scalable to adopt to a growing number of users
- The chatbot must be secure as sensitive data is being used
- Comply with data protection laws such as the Data Protection Act 1198
- The use of natural language used to interact with the chatbot promotes human computer interaction.
- Provide accurate responses to input.

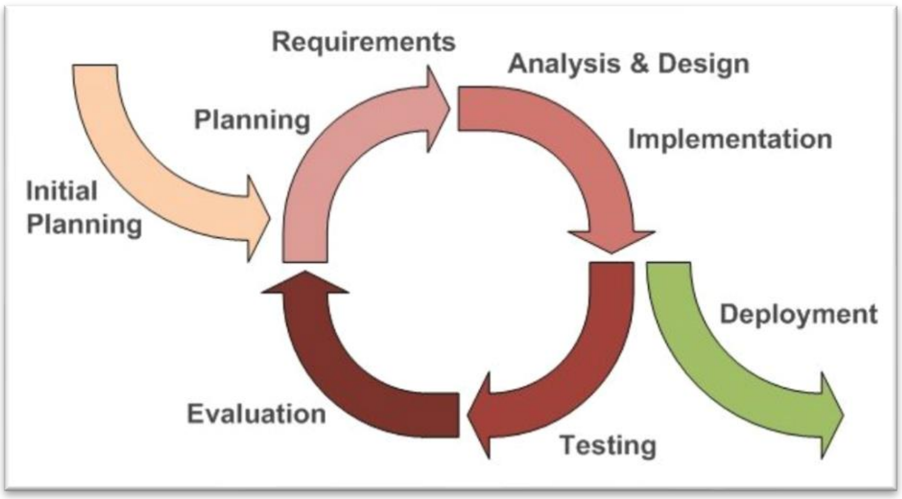
Appropriate handling of unexpected input , and correctly inform the user if it cannot aid and further do a google search of such queries.

5. Methodologies:

Deciding upon an appropriate methodology is vital for the overall development of any software application to ensure a realistic timeframe is established for each stage of the project and requirements are clearly outlined.

5.1 Incremental Model

This software methodology evolved from the waterfall model. The application is designed, developed and tested using iterative incremental build stages. At the end of each build a subsystem or feature will be created. The project will progress in complexity as new requirements are likely to be discovered and implemented in each incremental build, developing on top of the functionality from the last build leading to the overall development of the application. It is very common for software to be released in stages; it is critical that component versions utilized within the software are managed throughout the entire lifecycle using version control tools such as GitHub. Each build will only last a few weeks to produce a baseline version of the application. Feedback can be given on any requirement errors or faults found in the application. Distributing the development of the project over various build cycles can lower the risks associated with development to a more manageable level as requirements are broken down into smaller functionality to be implemented at the end of each build.

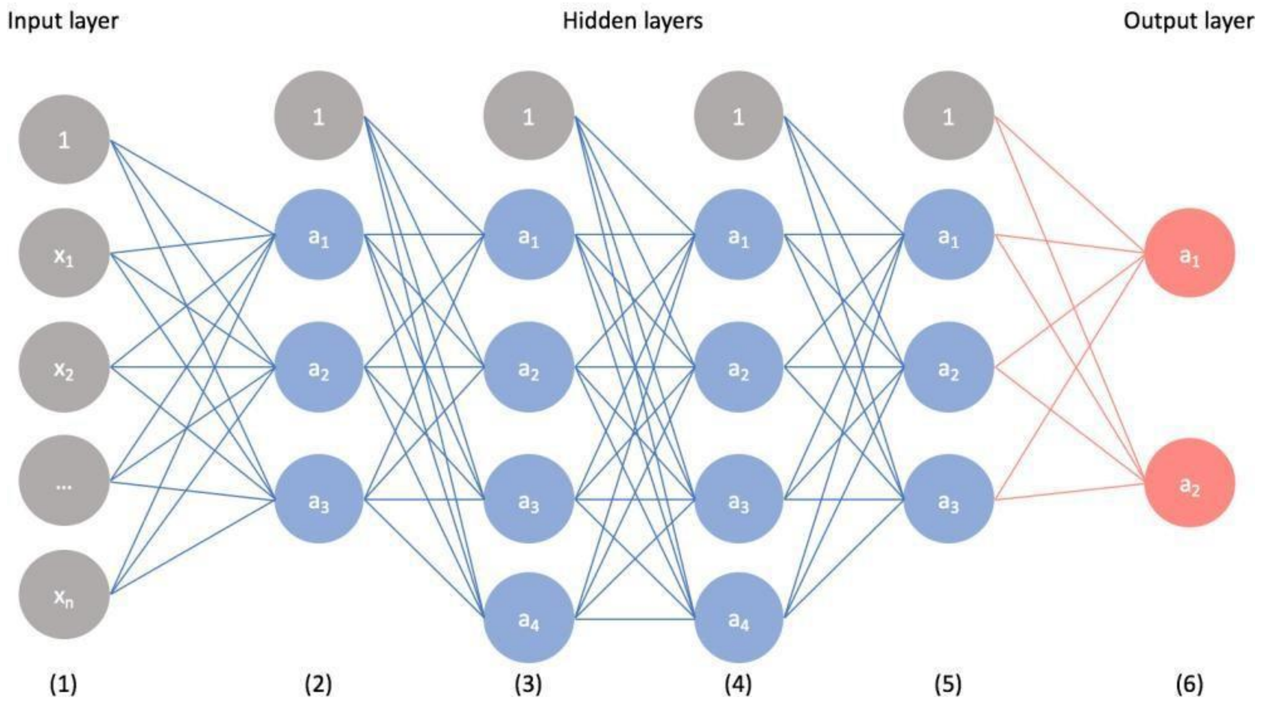


Testing and debugging are made easier as faults are identified early on during small manageable cycles. This methodology is flexible during implementation as new requirements identified are easily integrated at each build and a, updated version is released.

The incremental model is the most suitable development methodology to implement for this project. The flexibility of the incremental model makes it ideal for this project as it is likely new requirements will be identified during the later stages of development and each iterative build makes it easy to implement new requirements throughout the development process.

5.2 Basics of Deep Learning:

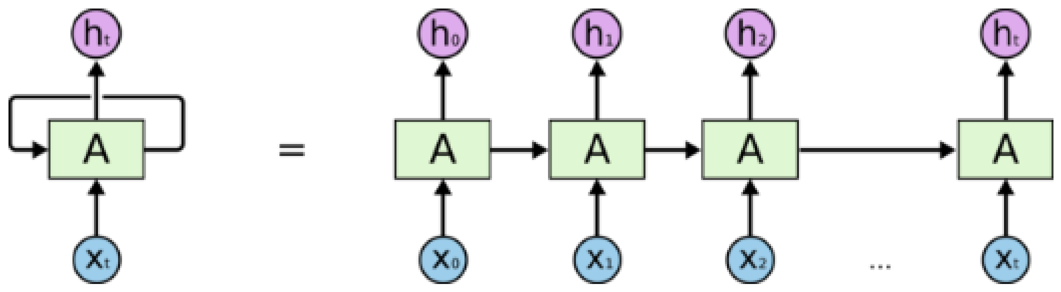
Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. To the greatest extent, it is based on a concept of a human brain and the interaction of neurons. Today, it has a variety of applications and significant place among them is occupied by face detection. First of all, deep learning gives the power to build biometric software that is capable of identifying a face of a human. As deep learning models are able to leverage very large datasets of faces and learn rich and compact representations of faces, they outperform the face detection capabilities of humans.



5.3 RNN (Recurrent Neural Network):

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.

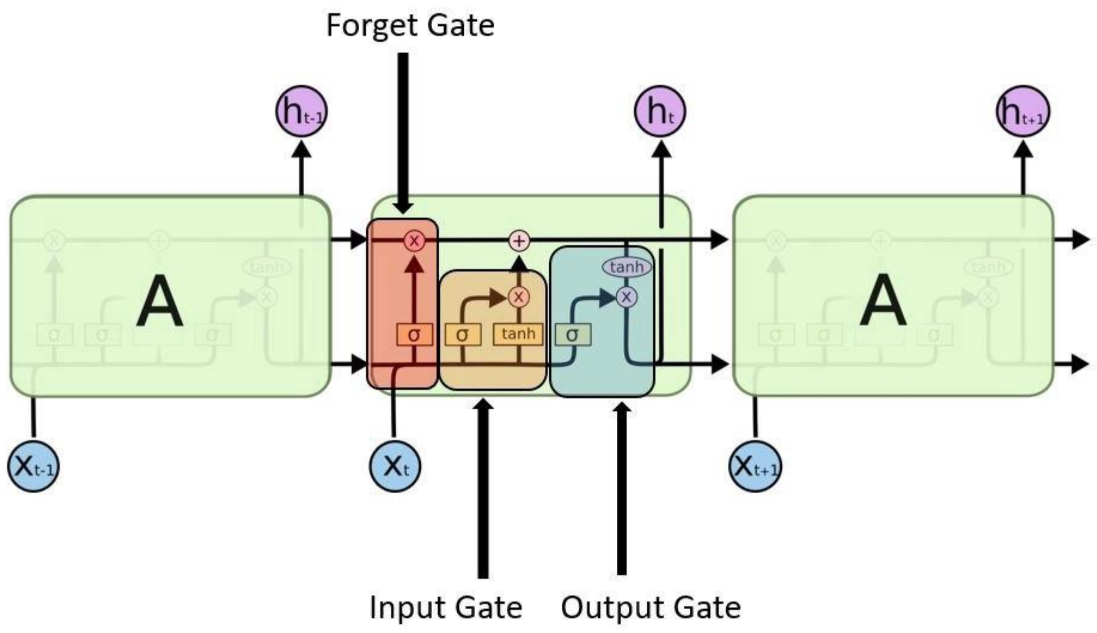
Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.



An unrolled recurrent neural network.

5.4 LSTM (Long Short-Term Memory):

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present:



5.4.1. Input gate — discover which value from input should be used to modify the memory. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1.

5.4.2. Forget gate — discover what details to be discarded from the block. It is decided by the sigmoid function. it looks at the previous state(ht-1) and the content input(Xt) and outputs a number between 0(omit this)and 1(keep this)for each number in the cell state Ct−1.

5.4.3. Output gate — the input and the memory of the block is used to decide the output. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1 and multiplied with output of Sigmoid.

5.5 Natural Language Understanding:

Rasa NLU is a kind of natural language understanding module. It comprises loosely coupled modules combining a number of natural language processing and machine learning libraries in a consistent API. There are some predefined pipelines like spacy_sklearn, tensorflow_embedding, mitie, mitie_sklearn with sensible defaults which work well for most use cases. For example, the recommended pipeline, spacy_sklearn, processes text with the following components. First, the text is tokenised and parts of speech (POS) annotated using the spaCy NLP library. Then the spaCy featuriser looks up a GloVe vector for each token and pools these to create a representation of the whole sentence. Then the scikit-learn classifier trains an estimator for the dataset, by default a mutliclass support vector classifier trained with five-fold cross-validation. The ner_crf component then trains a conditional random field to recognize the entities in the training data, using the tokens and POS tags as base features. Since each of these components implements the same API, it is easy to swap the GloVe vectors for custom, domain-specific word embeddings, or to use a different machine learning library to train the classifier. There are further components for handling out-of-vocabulary words and many customization options for more advanced users.

5.6 Natural Language Generation (NLG)

Natural Language Generation (NLG) is a subdivision of Artificial Intelligence (AI) that aims to reduce communicative gaps between machines and humans. The technology, typically accepts input in non-linguistic format and turn it into human understandable formats like reports, documents, text messages etc.

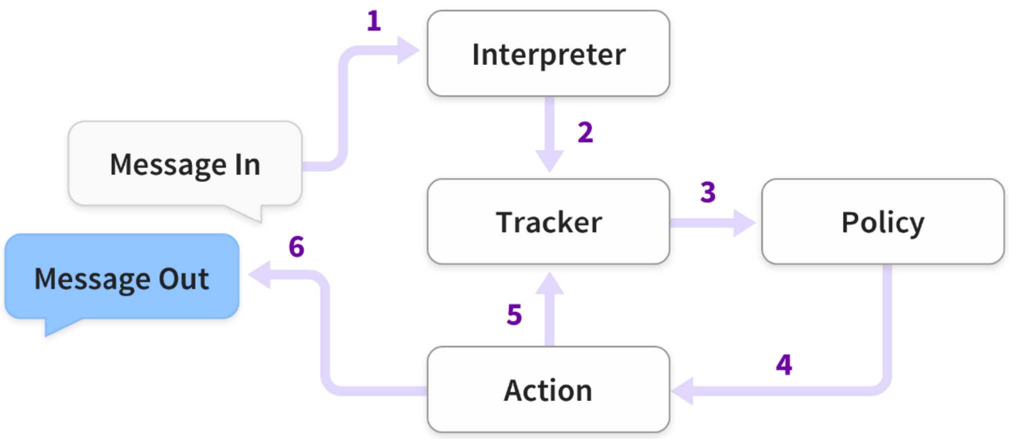
6. System Design:

Rasa's architecture is modular by design. This allows easy integration with other systems. For example, Rasa Core can be used as a dialogue manager in conjunction with NLU services other than Rasa NLU. While the code is implemented in Python, both services can expose HTTP APIs so they can be used easily by projects using other programming languages.

6.1 Architecture:

6.1.1 Rasa Core Architecture

Dialogue state is saved in a tracker object. There is one tracker object per conversation session, and this is the only stateful component in the system. A tracker stores slots, as well as a log of all the events that led to that state and have occurred within a conversation. The state of a conversation can be reconstructed by replaying all the events. When a user message is received Rasa takes a set of steps as described in figure 1. Step 1 is performed by Rasa NLU, all subsequent steps are handled by Rasa Core.



1. A message is received and passed to an Interpreter (e.g. Rasa NLU) to extract the intent, entities, and any other structured information.

2. The Tracker maintains conversation state. It receives a notification that a new message has been received.

3. The policy receives the current state of the tracker.

4. The policy chooses which action to take next.

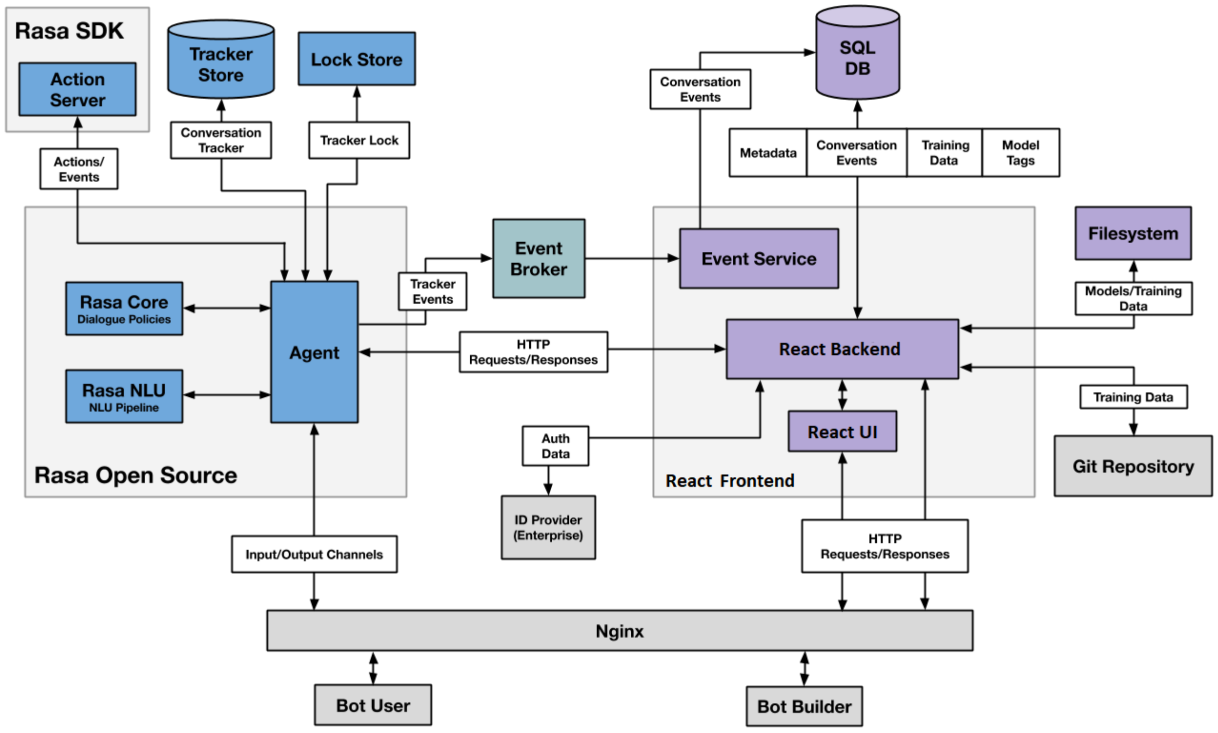
5. The chosen action is logged by the tracker.

6. The action is executed (this may include sending a message to the user).

7. If the predicted action is not 'listen', go back to step 3.

6.1.2 Complete Project Architecture

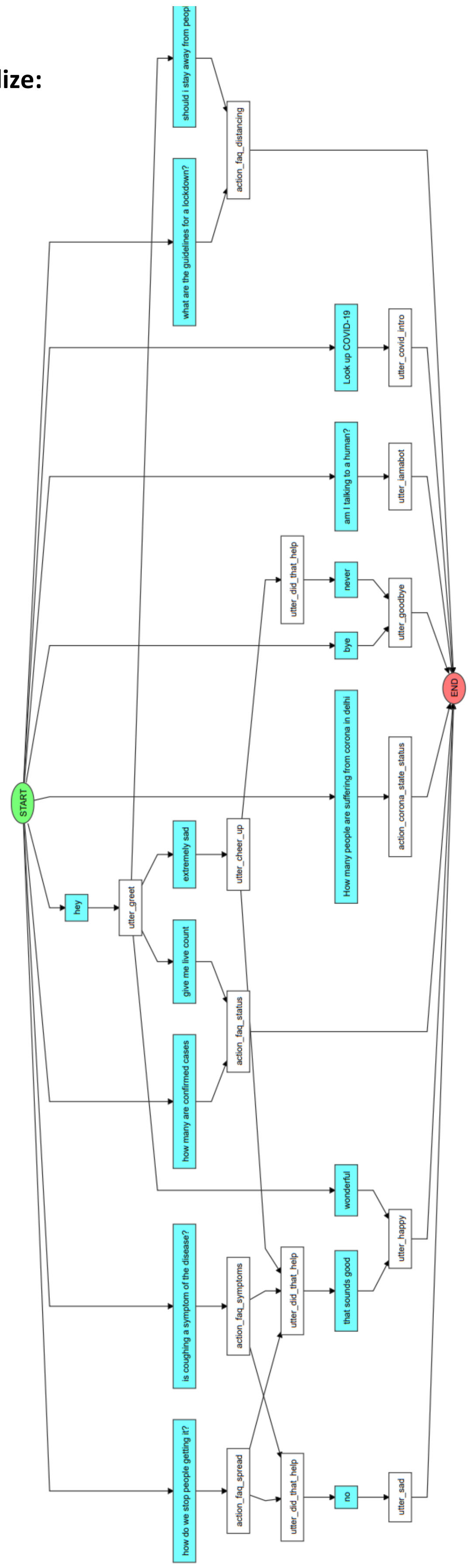
Covid19 Chatbot is built on a microservice architecture. The diagram below provides a high-level overview of the services started when the project is launched and their interaction with each other.



The diagram shows two main categories of services; the blue components relate to Rasa Open Source, and the purple ones relate to React. Rasa Open Source and React have independent databases. Conversation events data flows from Rasa Open Source to React via the event broker and React in turn makes API calls to Rasa Open Source to train, run models and trigger conversation events.

The Rasa Open Source source service trains and runs NLU and dialogue models. When a conversation is being handled by a model, it adds events to a conversation tracker. It also makes API calls to the action server to run custom actions. Conversation trackers are stored in the tracker store, which can be any kind of database or an in-memory store. If an event broker is configured, events that are stored in the tracker store are published to it as well. The lock store ensures that, given multiple Rasa Open Source nodes, only one node can work on a single conversation tracker at a time.

6.1.3 Project Stories Visualize:



7. System Development and Implementation:

7.1 Rasa NLU:

Rasa NLU is an open-source natural language processing tool for intent classification and entity extraction in Conversational AI chatbots. Rasa NLU contains all the information about the chat data regarding how the particular sentences will be taken in and different ways in which that particular query can be asked.

Rasa nlu.md: The nlu file contains all the different ways in which a particular message can be typed and giving it a particular intent such that it can be easily categorized.

```
## intent:greet
- hey
- hello
- hi
- good morning
- good evening
- hey there

## intent:goodbye
- bye
- goodbye
- see you around
- see you later

## intent:affirm
- yes
- indeed
- of course
- that sounds good
- correct

## intent:deny
- no
- never
- I don't think so
- don't like that

## intent:covid_intro
- Are corona virus and COVID-19 different?
- are coronavirus and covid 19 the same
- are coronaviruses common
- are coronaviruses seasonal
```

Rasa stories.md: Stories are type of training data used to train the assistant's dialogue management model.

```
## happy path
* greet
  - utter_greet
* mood_great
  - utter_happy

## sad path 1
* greet
  - utter_greet
* mood_unhappy
  - utter_cheer_up
  - utter_did_that_help
* affirm
  - utter_happy

## sad path 2
* greet
  - utter_greet
* mood_unhappy
  - utter_cheer_up
  - utter_did_that_help
* deny
  - utter_goodbye

## say goodbye
* goodbye
  - utter_goodbye
```

Domain.yml: The domain defines the universe in which the assistant operates. It specifies the intents, responses, forms, and actions the bot should know about. It also defines a configuration for conversation sessions.

```
intents:
  - greet
  - goodbye
  - affirm
  - deny
  - mood_great
  - mood_unhappy
  - bot_challenge
  - faq_vaccine
  - faq_vulnerable
  - company_open
  - company_wfh
  - faq_status
  - corona_state_status

responses:
  utter_greet:
  - text: "Hey! I'm Covid Bot. How are you feeling?"

  utter_covid_intro:
  - text: "Coronavirus disease (COVID-
19) is an infectious disease caused by a newly discovered coronavirus."
    image: "https://i.pinimg.com/originals/3f/0a/7d/3f0a7d813b7e1402f9c66cab2f82d
d38.gif"

  utter_did_that_help:
  - buttons:
    - payload: /affirm
      title: 👍
    - payload: /deny
      title: 👎
    text: "Did that help you?"

  utter_sad:
    - text: "Thanks, I appreciate that, and I'm sorry 😔"

  utter_iamabot:
  - text: "I am a bot, powered by Rasa."

  utter_default:
    - text: "Sorry! I am unable to understand what you said. Try again..."

actions:
- action_faq_distancing
- action_faq_spread
- action_faq_symptoms
- action_default_fallback
- action_corona_state_status

session_config:
  session_expiration_time: 60
  carry_over_slots_to_new_session: true
```

config.yml:

```
# Configuration for Rasa NLU.
language: en
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100

# Configuration for Rasa Core.
policies:
  - name: MemoizationPolicy
  - name: TEDPolicy
    max_history: 5
    epochs: 100
  - name: MappingPolicy
  - name: FallbackPolicy
    nlu_threshold: 0.7
    core_threshold: 0.7
    fallback_action_name: "action_default_fallback"
```

socketchannel.py: For communication between backend and the react part.

```python
import logging
import warnings
import uuid
import json
from sanic import Blueprint, response
from sanic.request import Request
from sanic.response import HTTPResponse
from socketio import AsyncServer
from typing import Optional, Text, Any, List, Dict, Iterable, Callable, Awaitable

from rasa.core.channels.channel import InputChannel
from rasa.core.channels.channel import UserMessage, OutputChannel
```

```python
logger = logging.getLogger(__name__)

def update_logs():
    try:
        with open("logs.json", "r") as fp:
            return json.load(fp)
    except:
        logs = []
        return logs

def update_logs_dump(logs):
    with open("logs.json", "w") as fp:
        return json.dump(logs, fp)

class SocketBlueprint(Blueprint):
    def __init__(self, sio: AsyncServer, socketio_path, *args, **kwargs):
        self.sio = sio
        self.socketio_path = socketio_path
        super().__init__(*args, **kwargs)

    def register(self, app, options) -> None:
        self.sio.attach(app, self.socketio_path)
        super().register(app, options)


class SocketIOOutput(OutputChannel):
    @classmethod
    def name(cls) -> Text:
        return "socketio"

    def __init__(self, sio, sid, bot_message_evt) -> None:
        self.sio = sio
        self.sid = sid
        self.bot_message_evt = bot_message_evt

    async def _send_message(self, socket_id: Text, response: Any) -> None:
        """Sends a message to the recipient using the bot event."""
        logs = update_logs()
        logs.append({"bot":response})
        update_logs_dump(logs)

        await self.sio.emit(self.bot_message_evt, response, room=socket_id)

    async def send_text_message(
        self, recipient_id: Text, text: Text, **kwargs: Any
    ) -> None:
        """Send a message through this channel."""

        await self._send_message(self.sid, {"text": text})

    async def send_image_url(
        self, recipient_id: Text, image: Text, **kwargs: Any
    ) -> None:
        """Sends an image to the output"""

        message = {"attachment": {"type": "image", "payload": {"src": image}}}
        await self._send_message(self.sid, message)

    async def send_text_with_buttons(
        self,
        recipient_id: Text,
        text: Text,
        buttons: List[Dict[Text, Any]],
        **kwargs: Any,
    ) -> None:
        """Sends buttons to the output."""

        message = {"text": text, "quick_replies": []}

        for button in buttons:
            message["quick_replies"].append(
                {
                    "content_type": "text",
                    "title": button["title"],
                    "payload": button["payload"],
                }
            )

        await self._send_message(self.sid, message)

    async def send_elements(
        self, recipient_id: Text, elements: Iterable[Dict[Text, Any]], **kwargs:
Any
    ) -> None:
        """Sends elements to the output."""

        for element in elements:
            message = {
                "attachment": {
                    "type": "template",
                    "payload": {"template_type": "generic", "elements": element},
                }
            }

            await self._send_message(self.sid, message)

    async def send_custom_json(
        self, recipient_id: Text, json_message: Dict[Text, Any], **kwargs: Any
    ) -> None:
        """Sends custom json to the output"""

        json_message.setdefault("room", self.sid)

        await self.sio.emit(self.bot_message_evt, **json_message)

    async def send_attachment(
        self, recipient_id: Text, attachment: Dict[Text, Any], **kwargs: Any
    ) -> None:
        """Sends an attachment to the user."""
        await self._send_message(self.sid, {"attachment": attachment})


class SocketIOInput(InputChannel):
    """A socket.io input channel."""

    @classmethod
    def name(cls) -> Text:
        return "socketio"

    @classmethod
    def from_credentials(cls, credentials: Optional[Dict[Text, Any]]) -
> InputChannel:
        credentials = credentials or {}
        return cls(
            credentials.get("user_message_evt", "user_uttered"),
            credentials.get("bot_message_evt", "bot_uttered"),
            credentials.get("namespace"),
            credentials.get("session_persistence", False),
            credentials.get("socketio_path", "/socket.io"),
        )

    def __init__(
        self,
        user_message_evt: Text = "user_uttered",
        bot_message_evt: Text = "bot_uttered",
        namespace: Optional[Text] = None,
        session_persistence: bool = False,
        socketio_path: Optional[Text] = "/socket.io",
    ):
        self.bot_message_evt = bot_message_evt
        self.session_persistence = session_persistence
        self.user_message_evt = user_message_evt
        self.namespace = namespace
        self.socketio_path = socketio_path

    def blueprint(
        self, on_new_message: Callable[[UserMessage], Awaitable[Any]]
    ) -> Blueprint:

        sio = AsyncServer(async_mode="sanic", cors_allowed_origins=[])
        socketio_webhook = SocketBlueprint(
            sio, self.socketio_path, "socketio_webhook", __name__
        )

        @socketio_webhook.route("/", methods=["GET"])
        async def health(_: Request) -> HTTPResponse:
            return response.json({"status": "ok"})

        @sio.on("connect", namespace=self.namespace)
        async def connect(sid: Text, _) -> None:
            logger.debug(f"User {sid} connected to socketIO endpoint.")

        @sio.on("disconnect", namespace=self.namespace)
        async def disconnect(sid: Text) -> None:
            logger.debug(f"User {sid} disconnected from socketIO endpoint.")

        @sio.on("session_request", namespace=self.namespace)
        async def session_request(sid: Text, data: Optional[Dict]):
            if data is None:
                data = {}
            if "session_id" not in data or data["session_id"] is None:
                data["session_id"] = uuid.uuid4().hex
            await sio.emit("session_confirm", data["session_id"], room=sid)
            logger.debug(f"User {sid} connected to socketIO endpoint.")

        @sio.on(self.user_message_evt, namespace=self.namespace)
        async def handle_message(sid: Text, data: Dict) -> Any:
            output_channel = SocketIOOutput(sio, sid, self.bot_message_evt)
            if self.session_persistence:
                if not data.get("session_id"):
                    warnings.warn(
                        "A message without a valid sender_id "
                        "was received. This message will be "
                        "ignored. Make sure to set a proper "
                        "session id using the "
                        "`session_request` socketIO event."
                    )
                    return

                sender_id = data["session_id"]
            else:

                sender_id = sid

            logs = update_logs()
            logs.append({"user": data["message"]})
            update_logs_dump(logs)

            message = UserMessage(
                data["message"], output_channel, sender_id, input_channel=self.na
me(),metadata=data["customData"]
            )
            await on_new_message(message)

        return socketio_webhook
```

actions.py:

This file contains custom tasks for handling different actions such as opening applications, displaying videos, making API calls etc.

Code for Fallback Policy to make a google search in case the user ask something out of the bot knowledge base.

```
class ActionDefaultFallback(Action):
    def name(self) -> Text:
        return "action_default_fallback"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
```

```
        text = list(map(str, tracker.latest_message['text'].split()))
        appendedtext = "+".join(text)
        searchtext = "Sorry! I cant understand. Here is a google search [Link]("
+ "https://www.google.com/search?q=" + appendedtext + ")"
        dispatcher.utter_message(text=searchtext)

        return []
```

Code for making API call to get statewise Corona stats from the covid19india.org.

```
class ActionCovidStateStatus(Action):
    def name(self) -> Text:
        return "action_corona_state_status"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        response = requests.get("https://api.covid19india.org/data.json").json()

        entities = tracker.latest_message["entities"]

        state = None
        for e in entities:
            if e['entity'] == 'state':
                state = (e['value']).lower()

        if state == "india": state = "Total"

        message = "Please enter correct state name."

        if state:
            for data in response["statewise"]:
                if data["state"] == state.title():
                    message = "Status of " + state.title() + "\n Active: " + data
["active"] + " Confirmed: " + data["confirmed"] + " Recovered: " + data["recovere
d"]

        dispatcher.utter_message(text=message)

        return []
```

Code for displaying an youtube video.

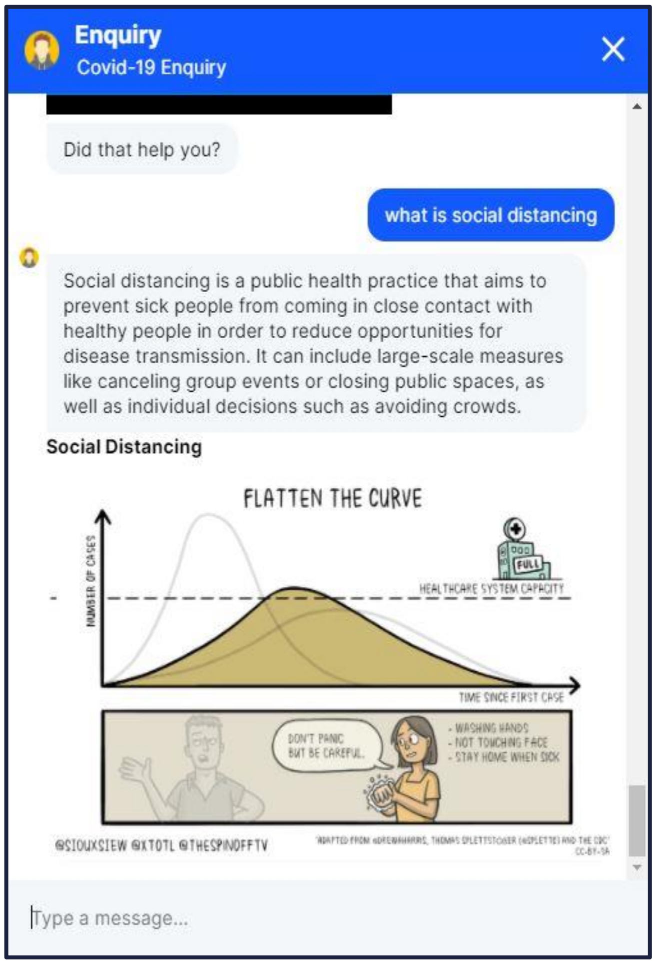
```
class ActionFaqSymptoms(Action):
    def name(self) -> Text:
        return "action_faq_symptoms"

    def run(self, dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        intent = tracker.latest_message["intent"].get("name")

        logger.debug("Detected FAQ intent: {}".format(intent))

        if intent in ["faq_symptoms"]:
            message = {
                "type": "video",
                "payload": {
                    "title": "COVID-
19 Animation: What Happens If You Get Coronavirus?",
                    "src": "https://www.youtube.com/embed/5DGwOJXSxqg",
                },
            }

            dispatcher.utter_message( text="People with COVID-
19 generally develop signs and symptoms, including mild respiratory symptoms and
fever, on an average of 5-6 days after infection (mean incubation period 5-
6 days, range 1-14 days). Most people infected with COVID-
19 virus have mild disease and recover.", attachment=message)

        return []
```

App.js code for the react Component:

```
import React from 'react';
import './App.css';
import { Widget } from 'rasa-webchat';
import { makeStyles } from '@material-ui/core/styles';
import AppBar from '@material-ui/core/AppBar';
import Toolbar from '@material-ui/core/Toolbar';
import Typography from '@material-ui/core/Typography';
import Button from '@material-ui/core/Button';

const useStyles = makeStyles((theme) => ({
  root: {
    flexGrow: 1,
  },
  menuButton: {
    marginRight: theme.spacing(2),
  },
  title: {
    flexGrow: 1,
  },
}));

function App() {

  const classes = useStyles();

  return (
    <div className="App">

      <AppBar position="static">
        <Toolbar>
          <Typography variant="h6" className={classes.title}>
          Covid-19 Enquiry
          </Typography>
          <Button color="inherit">Login</Button>
        </Toolbar>
      </AppBar>

      <Widget
      initPayload={"/greet"}
      subtitle={"Covid-19 Enquiry"}
      socketUrl={"http://localhost:5005"}
      socketPath={"/socket.io/"}
      profileAvatar={"https://icon-library.com/images/avatar-icon-images/avatar-
icon-images-4.jpg"}
      customData={{"language": "en"}}
      title={"Enquiry"}
      />
    </div>

  );
}
export default App;
```

8. Conclusion and Future Work:

The rise and popularity of chatbots is clearly outlined here. With this in mind the data gathered from testing the chatbot justifies the recent growth and demand for companies wanting to integrate a chatbot. It was determined that chatbots perform at a very high standard and provide reliable and rapid responses to users compared to that of traditional methods. The average time spent interacting with the chatbot is very low as it provides an efficient way for users to manage their banking. The low interaction time reflects the high understanding and speech recognition rates, offered through the adoption of conversational user interfaces thus allowing users to freely interact with the chatbot to meet the demands of modern life. The chatbot has proven to fulfil the demand of users wanting instant access and availability information and services.



[redacted]

Did that help you?

what is social distancing

Social distancing is a public health practice that aims to prevent sick people from coming in close contact with healthy people in order to reduce opportunities for disease transmission. It can include large-scale measures like canceling group events or closing public spaces, as well as individual decisions such as avoiding crowds.

Social Distancing

Future Work:

From observing the users interact with the chatbot during user testing, there were numerous suggestions to integrate the chatbot across other popular platforms such as Amazon Echo or Dot, this would increase the availability of the chatbot and the integration of the DialogFlow API allow the chatbot to be easily exported Amazon Alexa. Add voice processing to the chatbot so that instead of typing user can simply speak and the bot will respond too via voice/text. Support for secure payment services for the purpose of crowd funding and donations. Check if a person has symptoms of Corona by asking various questions regarding their health and then evaluating it.

References:

[1] D. Bohus and A. I. Rudnicky. The ravenclaw dialog management framework: Architecture and systems. Comput. Speech Lang., 23(3):332–361, July 2009.

[2] A. Bordes and J. Weston. Learning end-to-end goal-oriented dialog. CoRR, abs/1605.07683, 2016

[3] F. Chollet et al. Keras, 2015

[4] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759, 2016.

[5] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001

[6] P. Lison and C. Kennington. Opendial: A toolkit for developing spoken dialogue systems with probabilistic rules. ACL 2016, page 67, 2016.