Lolly

Project Engineering

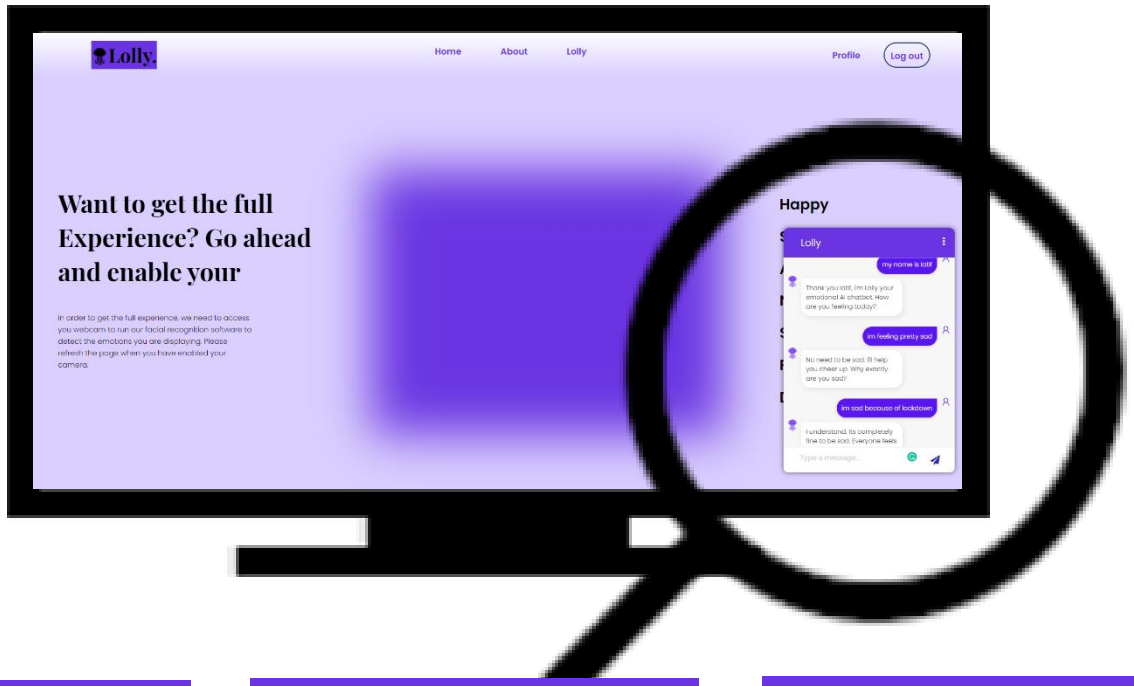
Year 4

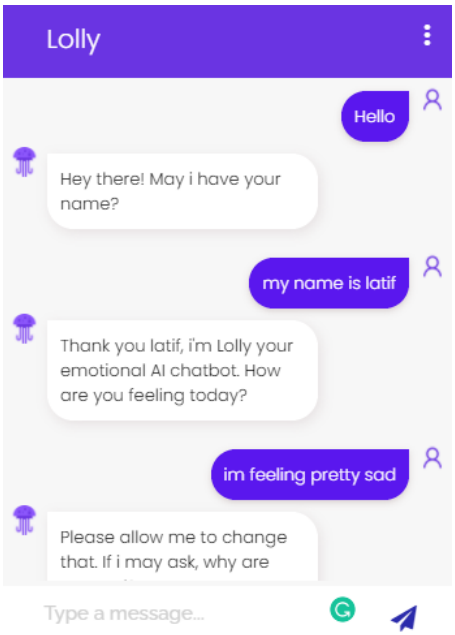
Latif Yahia

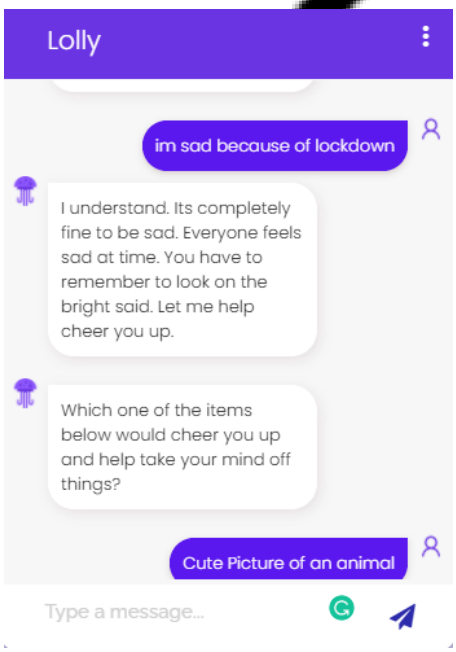
Bachelor of Engineering (Honours) in Software and Electronic Engineering

Galway-Mayo Institute of Technology

2020/2021


Lolly.
Home
About
Lolly
Profile
Log out
Want to get the full Experience? Go ahead and enable your
Happy


Lolly
Hello
Hey there! May i have your name?
my name is latif
Thank you latif, i'm Lolly your emotional AI chatbot. How are you feeling today?
im feeling pretty sad
Please allow me to change that. If i may ask, why are
Type a message...


Lolly
im sad because of lockdown
I understand. Its completely fine to be sad. Everyone feels sad at time. You have to remember to look on the bright said. Let me help cheer you up.
Which one of the items below would cheer you up and help take your mind off things?
Cute Picture of an animal
Type a message...


Lolly
Here is a cute gif of an animal. Hope this cheers you up!
Did that help you?
Type a message...

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Name: Latif Yahia Date: 12/05/2021

Acknowledgements

I want to acknowledge Paul Lennon and Brian O'Shea for their help and support throughout this project.

Table of Contents

1 Summary

During my final year of study in BEng Software and Electronic Engineering, I had to complete a project as part of my project engineering module. I decided to create an emotion-based machine-learning chatbot that uses facial recognition for emotion detection called “Lolly”.

Lolly’s goal is to help people overcome their emotions throughout the corona-virus pandemic. I know people are on lockdown and find it hard to talk to other people about their feelings. I was inspired to create this project as it solves a significant problem for people living in lockdown.

Lolly aims to cheer people up in real-time using facial recognition and machine learning. Lolly does this by providing a platform that allows people to talk about their feelings to a bot. Lolly also provides users with a personal profile that will enable them to view individual statistics based on different emotions detected throughout various conversations with Lolly.

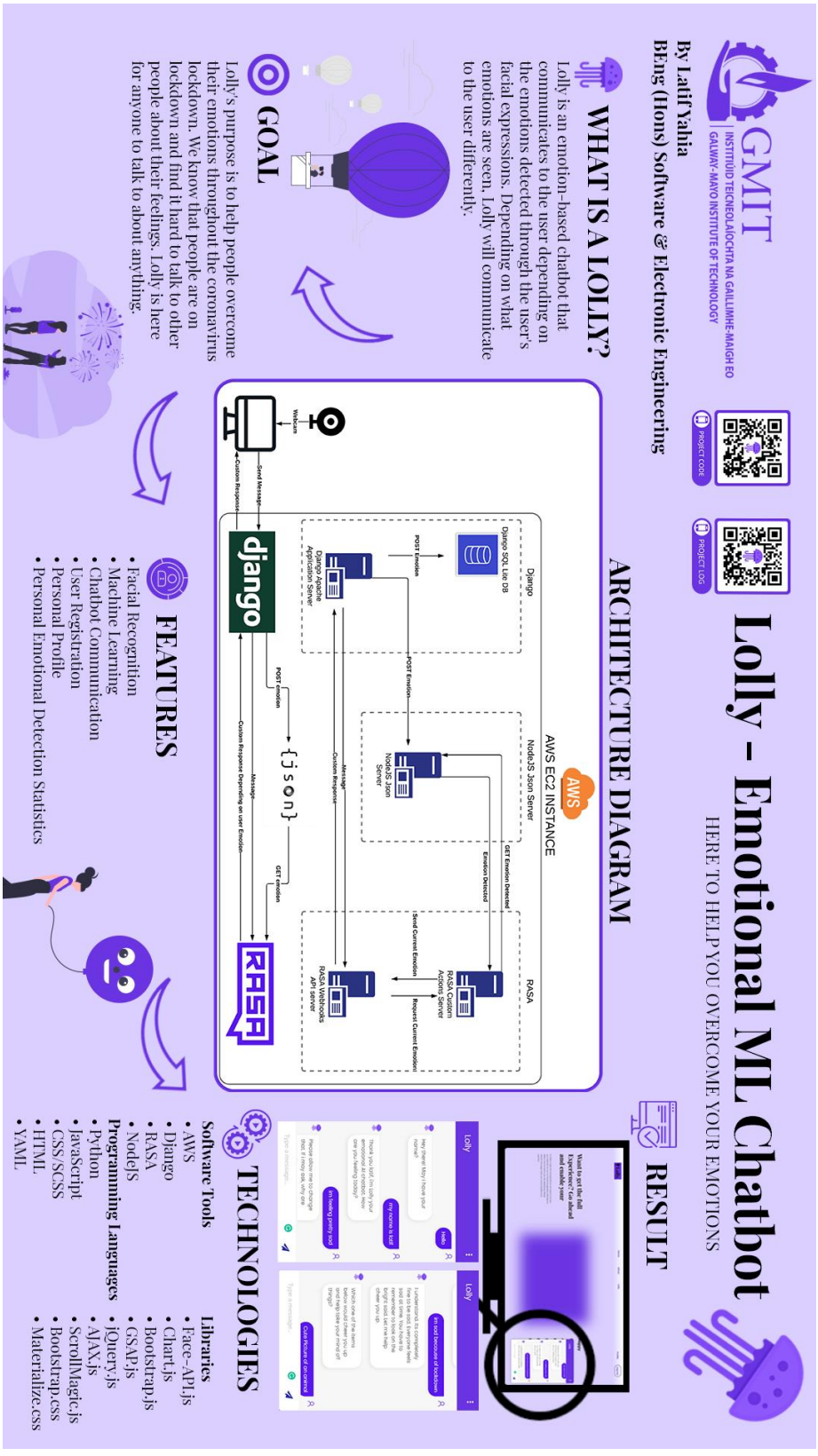
The approach to this project was challenging to say the least. Although I had experience working with machine learning, I didn’t have experience using facial recognition. I had to conduct a lot of research on the topic, which benefited me as I learned about the subject, and how it all worked. I endured a lot of trial and error through the course of this project as I experimented with different technologies

I planned to create a web application using Django, which is a web development framework. Using Django, I was able to create a visually stunning and fully functional web application. I also used different technologies such as RASA, a contextual open-source machine learning framework for building my chatbot. Within my web application, I also used a JavaScript library called face-api.js, a library used for facial recognition built using JavaScript and pre-trained machine learning models. I also set up a JSON server using NodeJS to work as a fake restful API to store emotions detected through Django and pass them across to RASA when required. I also decided to use AWS as the hosting service for my web application.

Just as planned, I completed the project, and the result of the project surpassed my expectations. Lolly is fully functional, using facial recognition to detect the user’s current emotion through their webcam. Lolly will communicate to the user differently depending on the users’ present emotion.

In conclusion, I am pleased with the result of the project. If I were to continue working on this project, I would improve the NLU (natural-language understanding) model for the chatbot. Improving the NLU (natural-language understanding) model would give more variety and complexity to the chatbot and how it handles incoming messages by the user.

2 Poster


GMIT
INSTITIUID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO
GALWAY-MAYO INSTITUTE OF TECHNOLOGY
PROJECT CODE
PROJECT LOG
Lolly - Emotional ML Chatbot
HERE TO HELP YOU OVERCOME YOUR EMOTIONS
By Latif Yahia
BEng (Hons) Software & Electronic Engineering
WHAT IS A LOLLY?
Lolly is an emotion-based chatbot that communicates to the user depending on the emotions detected through the user's facial expressions. Depending on what emotions are seen, Lolly will communicate to the user differently.
GOAL
Lolly's purpose is to help people overcome their emotions throughout the coronavirus lockdown. We know that people are on lockdown and find it hard to talk to other people about their feelings. Lolly is here for anyone to talk to about anything.
ARCHITECTURE DIAGRAM
AWS EC2 INSTANCE
Django
Django SQL Lite DB
Django Rasa Application Server
NodeJS Json Server
NodeJS Json Server
RASA
RASA Custom Actions Server
RASA Webhooks API server
Webcam
Send Message
Custom Response
POST Emotion
GET Emotion
Emotion Detected
Send Current Emotion
Request Current Emotion
Message
Custom Response Depending on user Emotion
{json}
RESULT
FEATURES
• Facial Recognition
• Machine Learning
• Chatbot Communication
• User Registration
• Personal Profile
• Personal Emotional Detection Statistics
TECHNOLOGIES
Software Tools
• AWS
• Django
• RASA
• NodeJS
Programming Languages
• Python
• JavaScript
• CSS/SCSS
• HTML
• YAML
Libraries
• Face-API.js
• Chart.js
• BootStrap.js
• GSAP.js
• jQuery.js
• AJAX.js
• ScrollMagic.js
• Bootstrap.css
• Materialize.css

3 Introduction

This report concludes my final year project. The information in this report is about my project named Lolly. Lolly is an emotion-based machine-learning chatbot that communicates to the user differently depending on the emotions detected through the user webcam using facial recognition. Lolly's goal is to help the user overcome their emotions and provides a platform for people to talk about their feelings to a bot. Lolly will try to cheer up the user if it detects that the user isn't happy. I know 2020/2021 has been a tough year for everybody, especially for the elderly in our community who cannot visit a family member and who might feel lonely at times due to lockdown restrictions. I was inspired to create a solution to this problem.

Lolly was part of my project engineering module, which took place in my final year in BEng Software and Electronic Engineering. The module spanned two semesters and is worth 30 credits. I was tasked with creating a fully functional web application, a poster, a video demonstration and a report by the 17th of May 2021.

4 Corona-Virus Pandemic and its Effect on Mental Health

The corona-virus pandemic affects nearly everybody in the world. The effects of the global pandemic are still ongoing due to the virus still being around. A lot of people have suffered both financially as well as mentally.

People have experienced several pandemic-related consequences, such as the closures of universities, loss of employment, and loss of income, which are all contributing to poor mental health. A larger than average share of young adults (ages 18-24) reported anxiety and/or depressive disorder during the pandemic. Research conducted during the pandemic indicates poor mental health and well-being for children and their parents. Parents face challenges with school closures and lack of childcare. Studies show that women with children are more likely to report anxiety and/or depressive disorder symptoms than men with children[1].

By creating Lolly, I wanted to stop the increase in mental health-related problems. Many of the mental health issues are related to people being stuck in lockdown. Being on lockdown, not being able to socialise and move freely can take a toll on peoples mental health.

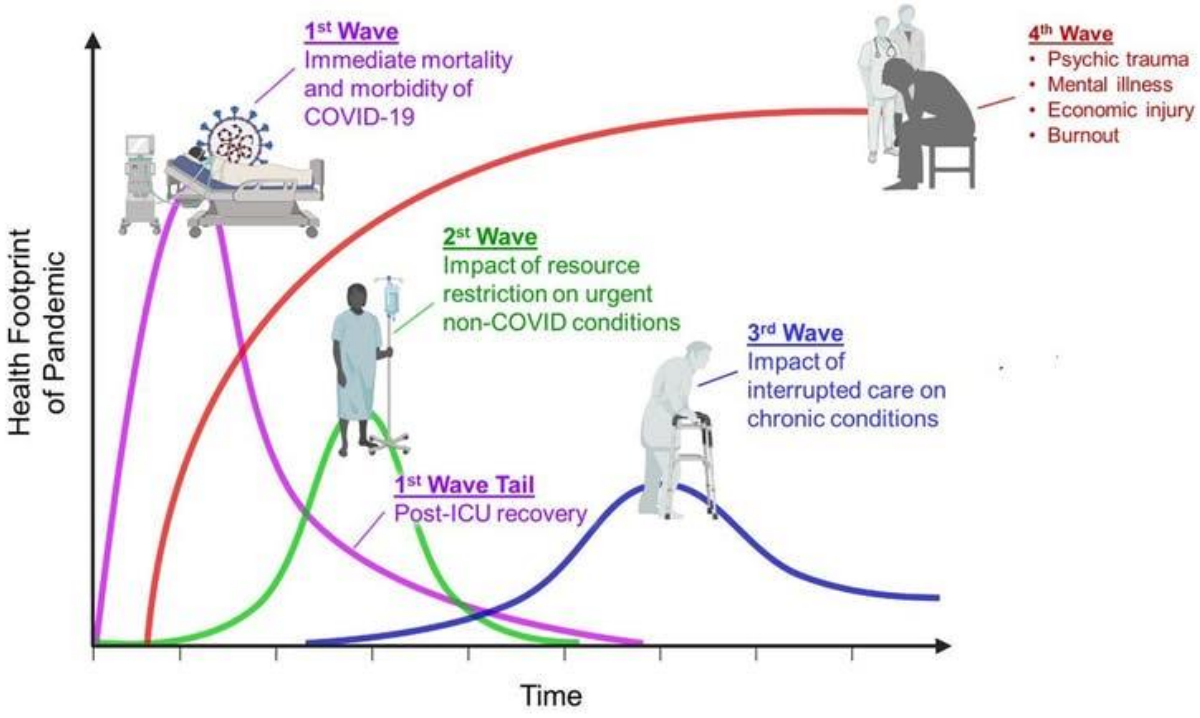


Figure 4-1 Effect on Mental Health Over Time During the Pandemic [2]

5 Project Architecture

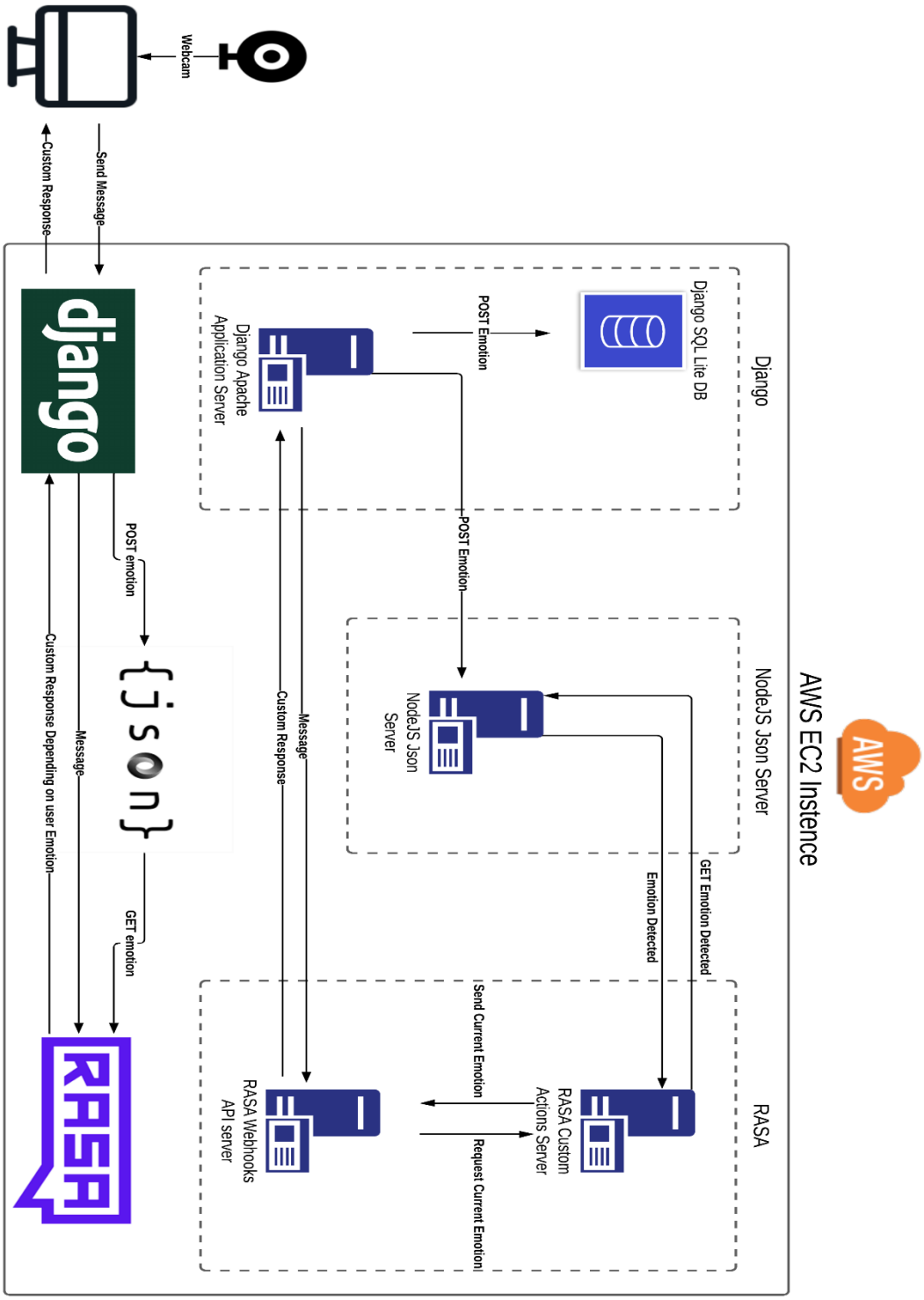


Figure 5-1 Architecture Diagram

6 Project Plan

Task Mode	Task Name	Duration	Start	Finish	Predecessors
	Final Year Project - Emotional AI Assistant	**159 days?**	**Mon 05/10/20**	**Thu 13/05/21**	
	Scope	**20 days**	**Mon 05/10/20**	**Fri 30/10/20**	
	Deteremine Project	2 wks	Mon 05/10/20	Fri 16/10/20	
	Research	**10 days**	**Mon 19/10/20**	**Fri 30/10/20**	**2**
	ESP32	0.5 days	Mon 19/10/20	Mon 19/10/20	
	Camera Module for	0.5 days	Mon 19/10/20	Mon 19/10/20	4
	Azure Facial Recognition	1 day	Tue 20/10/20	Tue 20/10/20	5
	RASA	3 days	Wed 21/10/20	Fri 23/10/20	6
	NLU Models	3 days	Mon 26/10/20	Wed 28/10/20	7
	Project Proposal	2 days	Thu 29/10/20	Fri 30/10/20	8
	Setup	**18 days?**	**Mon 02/11/20**	**Wed 25/11/20**	
	ESP32 with camera module	1 day	Mon 02/11/20	Mon 02/11/20	
	AWS server	3 days	Fri 06/11/20	Tue 10/11/20	
	Azure Facial Recongnition	1 day	Wed 11/11/20	Wed 11/11/20	12,11
	RASA	1 day	Wed 18/11/20	Wed 18/11/20	12
	Installation	**10 days**	**Thu 12/11/20**	**Wed 25/11/20**	
	Installation Research	**1.4 wks**	**Thu 12/11/20**	**Fri 20/11/20**	
	Installing Azure	2 days	Thu 12/11/20	Fri 13/11/20	13
	Installing RASA	2 days	Thu 19/11/20	Fri 20/11/20	14
	Install Azure Facial Recongnition on AWS	2 days	Mon 16/11/20	Tue 17/11/20	17
	Install RASA on AWS	3 days	Mon 23/11/20	Wed 25/11/20	18
	Development	**98 days**	**Fri 18/12/20**	**Tue 04/05/21**	
	Get rasa bot communicati	5 days	Fri 18/12/20	Thu 24/12/20	
	Azure face detecing faces	4 days	Mon 28/12/20	Thu 31/12/20	22
	Get django running with rasa and azure	7 days	Sat 02/01/21	Sun 10/01/21	23
	Research Azure Face Detection via live feed	7 days	Sat 23/01/21	Sun 31/01/21	
	Develop Rasa Chat Paths	11 days	Mon 01/02/21	Sun 14/02/21	
	Develop Nicer looking website	6 days	Mon 15/02/21	Sun 21/02/21	
	Starting Using Face-API library	7 days	Mon 01/03/21	Tue 09/03/21	
	Exams this week, also doing other mini project work	7 days	Thu 04/03/21	Fri 12/03/21	
	Get Data sending into django from face-api - Also send that data into rasa	13 days	Mon 15/03/21	Wed 31/03/21	28
	Exact User Emotion from Users face using JavaScript	3 days	Thu 01/04/21	Sun 04/04/21	
	Set up nodeJS json server to send the exacted emotions to	4 days	Sun 04/04/21	Wed 07/04/21	31
	Get the emotion from json sever in RASA and get the chatbot to talk different depending on about emotion is pulled	3 days	Thu 08/04/21	Mon 12/04/21	32
	Create more Path for RASA chatbot	10 days	Tue 13/04/21	Mon 26/04/21	33
	Add more pages / Design to the website	4 days	Tue 27/04/21	Fri 30/04/21	
	Add the project to AWS	3 days	Sat 01/05/21	Tue 04/05/21	
	Project Submissions	**7 days**	**Wed 05/05/21**	**Thu 13/05/21**	
	Porject Poster, Project Video, Project Report	7 days	Wed 05/05/21	Thu 13/05/21	

7 Facial Recognition

I am using facial recognition to detect emotions through the user's facial expressions. Facial recognition is a technology that matches a human face from a digital image with a database of images. In my case, I am using facial recognition for emotion detection using a machine learning FaceNet model. A FaceNet model is a machine learning model that takes an image of a face as an input and outputs a vector of 128 numbers representing points on a face. The FaceNet model can detect different emotions by reading the position of the points on the user's face[3]. I first wanted to implement facial recognition in my project using Azure Face, a facial recognition system developed by Microsoft. Unfortunately, I wasn't able to do live detection using Azure Face, so I decided to implement facial recognition in my project using a JavaScript library called face-api.js, which allowed for real-time detection of emotions.

7.1 Face-api.js

Face-api.js is a JavaScript API library for facial recognition implemented using tenserflow.js. The Face-API.js library comes with pre-existing models for facial recognition, such as a facial expressions model, facial detection model and a facial landmarks model. I am only using the facial expressions model as it's the model that can detect different emotions. The facial expressions model is a model that has been trained on a variety of images from both publicly available datasets and images scraped from the web. Training the facial expressions model allows the model to increase its accuracy as the model has more references between different emotions[4].

7.2 Emotion Detection

I want to acknowledge that the code used to set up the facial recognition was taken and changed to suit my own needs from a fellow developer who goes by the name of WebDevSimplifed through their GitHub repository called Face-Detection-JavaScript [5].

To detect emotions in real-time, I had to create a video element within my HTML code to show the user's webcam feed. I also had to implement an asynchronous function using JavaScript to detect emotions every five seconds. Suppose we have a look at Figure 7-2-1. In that case, we can see an asynchronous function that declares a constant called 'detections', which is used to create an object that uses the facial expressions model with the video element as the model's input. The data stored in the object can be seen in Figure 7-2-2.

```
setInterval(async() => {

    const detections = await faceapi.detectAllFaces(video, new faceapi.TinyFaceDetectorOptions()).withFaceExpressions()
    const resizedDetections =faceapi.resizeResults(detections, displaySize)
    canvas.getContext('2d').clearRect(0,0,canvas.width, canvas.height)
    faceapi.draw.drawFaceExpressions(canvas, resizedDetections)


    try {
        //converting dections into json format
        var json_data = JSON.stringify(detections)

        //changing json data to an jobject so i can parse the data
        var json_obj = JSON.parse(json_data)[0]

        //creating a array with emotions indexed so later i can retrive the type of emotion being displayed
        var Emotions = {Happy:json_obj.expressions.happy, Neutral:json_obj.expressions.neutral,
            Sad:json_obj.expressions.sad, Angry:json_obj.expressions.angry,
            Fearful:json_obj.expressions.fearful, Surprised:json_obj.expressions.surprised,
            Disgusted:json_obj.expressions.disgusted }
```

Figure 7-2-1 Start of the asynchronous function

Figure 7-2-2 Data inside the 'detections' object

The detections object stores information that the facial expressions model generates. Later in the function seen in Figure 7-2-1, I create a 'json_data' object that uses a function within JavaScript known as 'JSON.stringify()' to create a JSON object out of the detection object. Creating the 'json_data' object enabled me to parse the data within the 'detections' object. I proceed to create another JSON object by the name of 'json_obj' that uses another function within JavaScript called 'JSON.parse()' that allowed me to set the contents of the 'json_obj' to the first element of the 'json_data' object. I did this to make the parsing of the 'json_data' simpler. I create another JavaScript object by the name of 'Emotions', which hold the value of all possible emotions detected, such as happy, sad, angry, neutral, fearful, surprised and

disgusted. I set the value of the elements in the 'Emotions' object by parsing the 'json_obj' further.

The confidence level (how sure the model is) of each emotion ranges between zero and one, with zero meaning not confident, while one meaning very confident. This means that the emotion that is currently detected would hold the highest value between 0 and 1.

Later in the asynchronous function, as seen In Figure 7-2-3, I wrote code to check each element's value inside the 'Emotions' object. Depending on what detected emotion has a value of > 0.5, I set the value of another variable called 'emotion' as the name of that element. The value of 'emotion' is used to put the currently detected emotion and replace the previously detected emotion on my JSON server, which Rasa later uses. I also post the value of 'emotion' to a function inside my views.py file to save the currently detected emotion to the Django database, as seen in Figure 7-5 using jQuery and Ajax[6].

```
//sending the emotion to my json server so i can use in rasa
$.ajax({
    type: "PUT",
    url: "http://localhost:3000/emotion/1",
    data:{
        currentEmotion: emotion,
    },
    datatype:'json',
    success: function(data) {
        if (data['success'])
            console.log("Emotion successfully to json server")
    }
});
//sending the emotion to my views.py so i can save the emotions with
$.ajax({
    type: "POST",
    url: "/updateEmotions/",
    data:{
        emotion: emotion,
        'csrfmiddlewaretoken': '{{ csrf_token }}'
    },
    datatype:'json',
    success: function(data) {
        if (data['success'])
            console.log("successfully added to user emotions")
    }
});
```

Figure 7-2-3 PUT & POST using jQuery and Ajax

7.3 Live Emotion Detection Using Azure Face Issue

As mentioned previously, I first planned to use Azure Face for facial recognition. This turned out to be more challenging than first anticipated. I spent months trying to figure out how to detect emotions in real-time using Azure Face but came up empty-handed. I was using python to

develop my project, which there wasn't much support regarding Azure Face. Azure Face mainly provided support for languages such as C++; although I had Azure facial recognition working with python, I was only detecting emotions using static images. At first, I thought I would take a snapshot of the user's webcam every five seconds, save them to Django's database, and run the Azure facial recognition on the images to get live emotion detection. Still, it turned out to be more hassle than expected. After researching, I found out that saving an image would require PHP, a programming language I was unfamiliar with. After consulting one of my supervisors, I was advised to stay away from PHP as it's an outdated language. I took his advice and proceeded to do more research on different facial recognition technologies. That's when I came across face-api.js. After researching the face-api.js library in detail, I was happy to use it. The documentation for the library was tremendous, and it was easy to implement into my Django project.

8 Machine Learning

I developed Lolly using machine learning. “Machine learning is a method of data analysis that automates analytical model building”. Machine learning is related to artificial intelligence because machine learning systems can learn from data, identify different patterns, and make decisions independently with minimal human intervention[7]. I chose to develop my machine learning chatbot using Rasa.

8.1 Rasa

Rasa is an open-source machine learning framework used for building chatbots and AI assistants without much knowledge of programming[8]. However, people who plan to use the Rasa framework should familiarise themselves with the syntax of YAML as it’s the main syntax used within Rasa. Rasa provides the developer with an NLU (natural-language understanding) model to train the chatbot. The NLU model is a machine learning model that uses speech recognition methods that add numeric structure to large datasets. The NLU model improves as its trained to recognise different syntax, context, language patterns, unique definitions, sentiment, and intent [9]. Rasa also provides different configuration for model training known as ‘policies’ which can be applied to the model to create the desired NLU model. Rasa implements the NLU data – domain – stories – actions architecture.

8.2 NLU Data, Domain, Stories, Custom Actions

NLU data is data that the user can send to the chatbot, also known as an ‘intent’. The domain is data that the chatbot can use to reply to an intent, also known as an ‘action’. The domain data also contains different data variables known as ‘entities’, ‘slots’ and ‘forms’. Entities are data extracted from the user during a conversation, for example, a name, place or date. Slots are mainly used to set data variables from custom actions. A custom action is an action that can run any code the developer wants. This can be used to make an API call or query a database [10]. Depending on the slot value, the conversation can follow different stories. Stories can be expressed as a type of training data used to train a chatbots dialogue management model. A story represents a conversation between a user and the chatbot, converted into a YAML format where user messages are considered as intents while the chatbot’s responses and actions are expressed as action names[11].

In 'Figure 8-2-1', I wrote a python class to create a custom action called 'action_submit_emotion'. When the custom action is triggered in a story, it runs python code to send a GET request to my JSON server, which returns the value of the latest emotion detected by the user. A slot called ‘emotion’ is then set with the value of the emotion got from the JSON server.

```
class ActionEmotionSubmit(Action):


    def name(self) -> Text:
        return "action_submit_emotion"

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

            r = requests.get('http://localhost:3000/emotion')
            jsonData = r.json()
            emotion = jsonData[0]['currentEmotion']
            return [SlotSet("emotion", emotion)]
```

Figure 8-2-1 A custom action filling a ‘slot’ called emotion with the latest detected emotion

In ‘Figure 8-2-2’ and ‘Figure 8-2-3’, You can see two stories that the bot will follow depending on the slot value of the emotion detected. If we look at the story itself, we can see both stories are very similar in how they proceed with the conversation. If we look at the stories step by step, we can see when a user sends a message to the chatbot with the intent ‘greet’; we follow both stories. We then run the custom action called ‘action_submit_emotion’, which will then fill a slot named ‘emotion’ with the user-detected emotion. I then use the function ‘slot_was_set’ to tell the chatbot if the emotion detected in the slot is ‘happy’, then to proceed with the story called ‘greet_name_happy’ or else if the emotion detected in the slot is ‘neutral’, then proceed with the story called ‘greet_name_neutral’. I have created different stories for all possible emotions that can be detected, such as happy, sad, neutral, angry, surprised, disgusted and fearful. If we continue following the stories, we can see I’m also extracting the entity’ name’ through the conversation and setting the name-value into a slot called ‘name’. Extracting the name as an entity and filling it into a slot later allows the chatbot to use that entity when speaking to the user.

```
- story: greet_name_happy
  steps:
  - intent: greet
  - action: action_submit_emotion
  - slot_was_set:
    - emotion: Happy
  - action: utter_greet_happy
  - intent: my_name
    entities:
    - name: latif
  - slot_was_set:
    - name: latif
  - action: utter_reply_name_happy
```

Figure 8-2-2 Story for the chatbot to follow if the emotion detected is 'happy'

```
- story: greet_name_neutral
  steps:
  - intent: greet
  - action: action_submit_emotion
  - slot_was_set:
    - emotion: Neutral
  - action: utter_greet_neutral
  - intent: my_name
    entities:
    - name: latif
  - slot_was_set:
    - name: latif
  - action: utter_reply_name_neutral
```

Figure 8-2-3 Story for the chatbot to follow if the emotion detected is 'neutral'

8.3 Slot Value Issue

Throughout my development, I faced many problems. The biggest problem I faced would probably be the issue of the chatbot not following the assigned stories for different emotion detection. The case was that the slot called emotion was being filled, but the slot's value did not match the detected emotion. It turns out the slot was just being acknowledged as being set. I research the issue through StackOverflow but came up empty-handed. I turned to Rasa's community forums, where developers working with Rasa can post a question concerning Rasa. I was lucky enough to get a reply from a verified Rasa developer who told me my issue[12].

While I was creating my slot type, I assigned the slot type to be of ‘text’ instead of ‘categorical, meaning the slot’s value did not matter within the story. After changing the slot type to ‘categorical and adding all possible slot values, the chatbot started following the stories correctly. Although the issue wasn’t a major one, it still took me a lot of time to figure out. I have previous experience with Rasa and am familiar with slot types; the slot type ‘categorical wasn’t in the earlier versions of Rasa that I was using hence why I didn’t figure out the issue for a while.

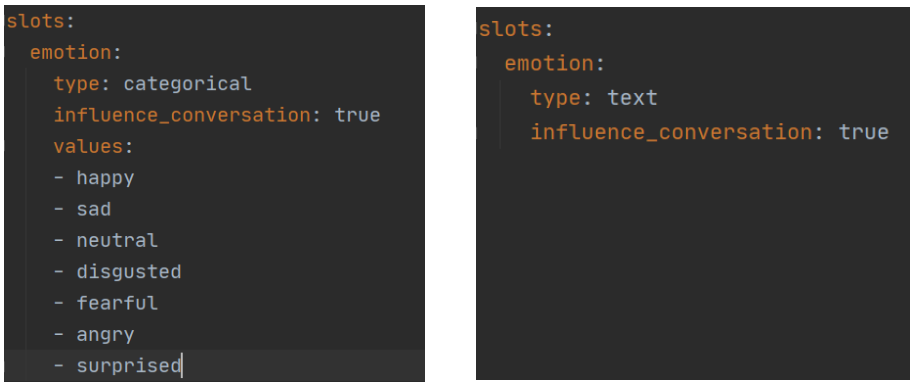


Figure 8-3-1 Categorical slot type VS Text slot type

9 Django

To develop my web application, I chose to use Django. Django is a python-based open-source web development framework. Django follows models-templates-views architecture [13]. I chose to use the Django framework as it allows for great scalability. It allows for multiple smaller applications to run as one big application. Django provides the developer with a SQL lite database that the developer can use to store information, passwords, profiles etc. I also decided to use Django as it takes care of all the hassle of web development. This, in turn, allowed me to focus more on writing my application. Django also takes care of security and authentication using middleware and Csrf (Cross-site request forgery) tokens, meaning I didn't have to write any new code relating to security and authentication when implementing profile creation.

9.1 Registration, Login, Logout

I want to acknowledge that the code used to set up the registration application was taken and changed to suit my own needs from a fellow developer named Tech With Tim through his GitHub repository Django-Website [14].

Within my main Django project, I created a new app (application) called registration. The new app was implemented to take care of user registration, login and logout. This was done by using Django's built-in authentication system, which is a default package supplied with the Django framework. I first started off by creating new URL paths for '/register', '/login', '/logout' as seen in Figure 9-1-1.

```
path('register/', v.register, name='register'), # url for registration and assigning the url to an function inside view.py
path('', include('django.contrib.auth.urls')),# url for /login , /logout
```

Figure 9-1-1 URL paths for '/register', '/login', '/logout'

In the URL path for '/register', I'm assigning a function written in the register applications view.py file that handles user registration. For the '/login' and '/logout' URLs, I only have to call a package within Django called 'django.contrib.auth.urls' that automatically creates those URLs. In the views.py file, I have a function to handle the registration of a user, as seen in figure 9-1-2.

```python
from django.shortcuts import render, redirect
from .forms import RegisterForm


# Create your views here.

def register(response):

    if response.method == "POST":
        form = RegisterForm(response.POST)
        if form.is_valid():
            form.save()
            return redirect("/login")
    else:
        form = RegisterForm()

    if not response.user.is_authenticated:
        return render(response, "register/register.html", {"form":form})

    else:
        return redirect('/profile')
```

Figure 9-1-2 Register function inside views.py

In the view.py file, I am first importing a class called 'RegisterForm' from a file called forms.py within the registration app. I have created a form that the user must fill out during registration. I have set up the form to ask for the user's username, first name, last name, email, password and password confirmation. This can be seen in Figure 9-1-3. If we progress through the register function, you can see I'm checking for a POST request. A POST request is sent when a user has registered a profile through the website. If a POST request has been sent, I check the form to make sure it's valid. If the form is correct, the user's information will be saved to Django's database. The user will then be able to log in using their credentials. My code for the registration page can be seen in Figure 9-1-4, and the rendered version of the page can be seen in Figure 9-1-5.

```python
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth import login, authenticate
from django.contrib.auth.forms import UserCreationForm

class RegisterForm(UserCreationForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ["username",'first_name', 'last_name','email', 'password1', 'password2']
```

Figure 9-1-3 RegisterForm class in forms.py

In the RegisterForm class, I am importing a function within a pre-existing Django class called ‘UserCreationForm’. The ‘UserCreationForm’ creates a user, with no privileges, from the given username and password. Django automatically does this for me when the form has been filled and submitted. Within the class, I am creating a new class called ‘meta’ where the existing ‘UserCreationForm’ can be modified. I am then creating an array called ‘fields’ which will hold the name of the fields I want the user to fill when registering their profile. I am doing this since the ‘UserCreationForm’ only requests the user’s username, password and password confirmation, and I want to override it with my own created fields.

```
<!--some code taken and customized to meet my own needs from https://www.youtube.com/watch?v=Ev5xgwndmfc-->
<div class="container">
    <div class="row">
        <div class="col-4">
        </div>
        <div class="col-4">
            <div style="justify-content: center; margin-top: -70px" class="center">
                <h3>Registration</h3>
                <form style="justify-content: center;" method="POST" class="form-group">
                    {% csrf_token %}
                    {{form | crispy}}

                    <button style="background-color: #6a34e2; border-color: #6a34e2; margin-top: 25px"
                            type="submit" class="btn btn-success">Register
                    </button>
                </form>
            </div>.
        </div>
    </div>
</div>
```

Figure 9-1-4 Registration page HTML code

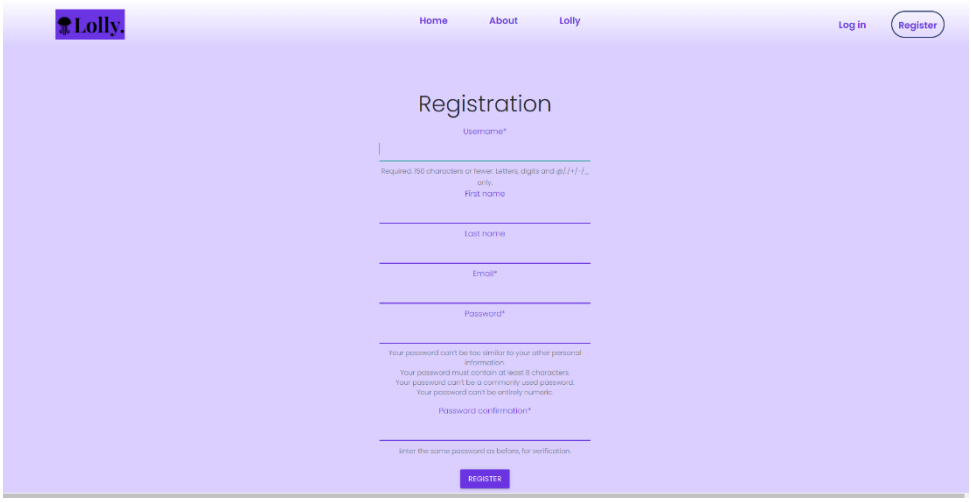


Figure 9-1-5 Rendered registration page

In Figure 9-1-4, I am simply creating a form element and a button element. The form elements data is posted across to my register function in views.py when the register button is pressed. I am also using 'crispy_forms', which styles the form automatically[15].

9.2 Security, Authentication, Cross-site request forgery

Django takes care of security, authentication and cross-site request forgery by using different plugins known as middleware. Middleware is a significant aspect of Django's security and what makes Django so great. Middleware is a plugin that precesses during requests and response execution. Middleware can be used to perform a different function such as security, Csrf (cross-site request forgery) protection and authentication[16].

Through my Django project, I used the Csrf token a lot. I was required to use the Csrf token unique to my application to make POST requests throughout my Django application.

```
$.ajax({
    type: "POST",
    url: "/updateEmotions/",
    data:{
        emotion: emotion,
        'csrfmiddlewaretoken': '{{ csrf_token }}'
    },
    datatype:'json',
    success: function(data) {
        if (data['success'])
            console.log("successfully added to user emotions")
    }
});
```

Figure 9-2-1 POST request using jQuery & Ajax with the use of Csrf token

In figure 9-2-1, I'm using jQuery and Ajax to post the users detected emotion to a function called 'updateEmotions' in my views.py file, which saves the detected emotion to the database. I include my unique Csrf token in the data I'm posting over to my 'updateEmotions' function, which allows Django to process the request as legitimate. Without the Csrf token, the POST request would fail since Django would think it's a foreign POST request not requested by the application.

By default, Django stores a password as an attribute of a user object in a string format using the PBKDF2 (Password-Based Key Derivation Function 2) algorithm with a SHA256 hash, as seen in Figure 9-2-2. Using the combination of the two allows for a secure password that would take a lot of computing power to break. When a user creates a profile, a user object is created in Django's database containing the user's information and password. The user's password is encrypted and not visible to website admins within the users object in the database, as seen in Figure 9-2-3.

```
<algorithm>$<iterations>$<salt>$<hash>
```

Figure 9-2-2 Django's password string format

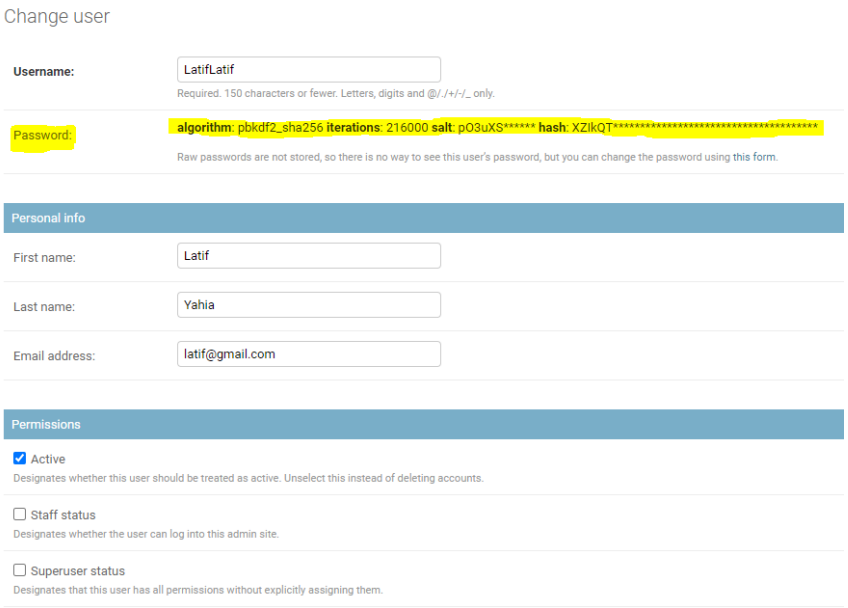


Figure 9-2-3 Admin view of a user's object inside Django's database

I am also restricting certain pages to users who are not logged in; for example, users who logged out cannot access the 'profile' page and will be redirected to the login page if they try to access the URL. I am also displaying different content throughout the website depending on whether the user is logged in or logged out. By doing this, I am taking full advantage of Django's built-in authentication features, which results in a much better project.

9.3 PBKDF2 (Password-Based Key Derivation Function) Algorithm & SHA265 Hash

The PBKDF2 is a secure password algorithm that uses key derivation functions[17]. The PBKDF2 key derivation function has five input parameters: [18]

DK = PBKDF2(PRF, Password, Salt, c, dkLen)

Where:

PRF = Pseudorandom function of two-parameters

Password = The master password which a key is generated from

Salt = A sequence of bits

c = The number of iterations

dkLen = The desired bit-length of the derived key

DK = The generated Key

SHA256 (Secure-Hash Algorithm 256) is a cryptographic hash function with a length of 256 bits that is developed by the united states national security agency[19].

9.4 CORS Policy No 'Access-Control-Allow-Origin' Issue

I want to acknowledge that the code for the middleware CORS (Cross-Origin Resource Sharing) class was taken from a solution on Stackoverflow and used in my project to fix the issue of CORS policy no 'access-control-allow-origin'[20].

I faced many problems throughout my development of Lolly. By far, my biggest issue relating to Django was the issue of CORS (Cross-Origin Resource Sharing) policy. Due to Djangos strict security measure, POST and GET requests from Django to Rasa's webhook server to allow communication to the chatbot through the chat widget were being blocked. The solution to the issue was simple and required installing a middleware plugin called 'django-cors-header'. After installing the middleware plugin and configuring it in my settings.py file, I was still facing the issue. For an unknown reason, the installation of the middleware plugin was not working for me. The issue left me frustrated as all the research I conducted told me to fix the issue; I had to install the middleware plugin. I continued my research further and tried different methods to bypass CORS policy. I experimented with Crsf (Cross-site request forgery) tokens but still came up empty-handed. After hours of research, I came across a similar issue on StackOverflow, where the installation of the middleware plugin was not working. It turns out the issue only persisted in the version of Django I was using. A user on the post suggested writing the middleware class within the Django project to manually allow all POST and GET requests from any origin through the Django application. Although this is not ideal, it was the only way to resolve the issue to have a fully working project. I took the code the user wrote for their middle

was class, as seen in figure 9-4-1, and put it into my own Django project. This fixed the issue for me, and I started communicating with the chatbot through the chat widget.

```python
class CustomCorsMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        # One-time configuration and initialization.

    def __call__(self, request):
        # Code to be executed for each request before
        # the view (and later middleware) are called.

        response = self.get_response(request)
        response["Access-Control-Allow-Origin"] = "*"
        response["Access-Control-Allow-Headers"] = "*"

        # Code to be executed for each request/response after
        # the view is called.

        return response
```

Figure 9-4-1 Middleware CORS (Cross-Origin Resource Sharing) policy class

All the class is doing is configuring Django to allow any origin (host/computer) to make POST and GET requests to the website. Although this would not be ideal for a mainstream product as it opens the website to hacking and data breaches.

10 UI (User Interface) Design

For my applications UI (User Interface), I wanted to create a modern, professional-looking website that attracts the user's attention to the website itself—the website consists of free, open-source illustrations and animations. The use of animations and illustrations bring the website to life. To design a clean looking UI, I made good use of different CSS and JavaScript libraries such as 'bootstrap.css', which I used to create the layout for the charts and buttons[21]. 'materialize.css', which I used primarily for the design of the chat widget[22]. 'gsap.js', which I used to create animations[23]. 'chart.js', which I used to render charts[24].

10.1 Layout

The website layout consists of five pages by the name of 'home', 'about', 'lolly', 'profile' and 'registration'. Each page consists of a top navigation bar that is written once and extended to all pages with the use of a 'tag' within the jinja2 templating language. As seen in Figure 10-1-1, the home page introduces the user's to the website and displays some relevant information about the website itself.

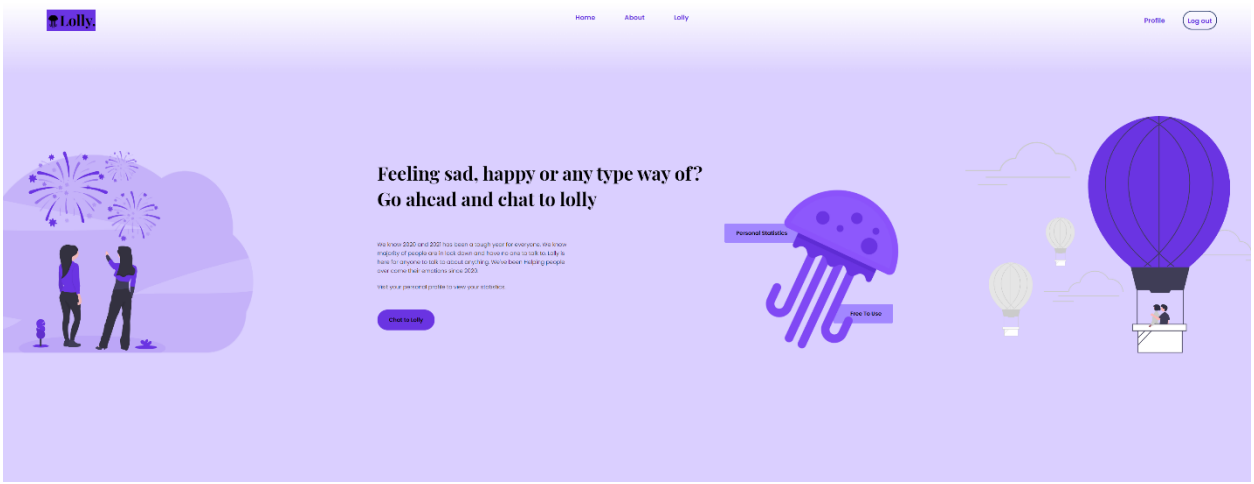


Figure 10-1-1 Home Page

As seen in Figure 10-1-2, the about page informs the user about relevant information regarding the project. The about page informs the user about the project's goal, Lolly, the developer behind the project, the features, facial recognition and machine learning.

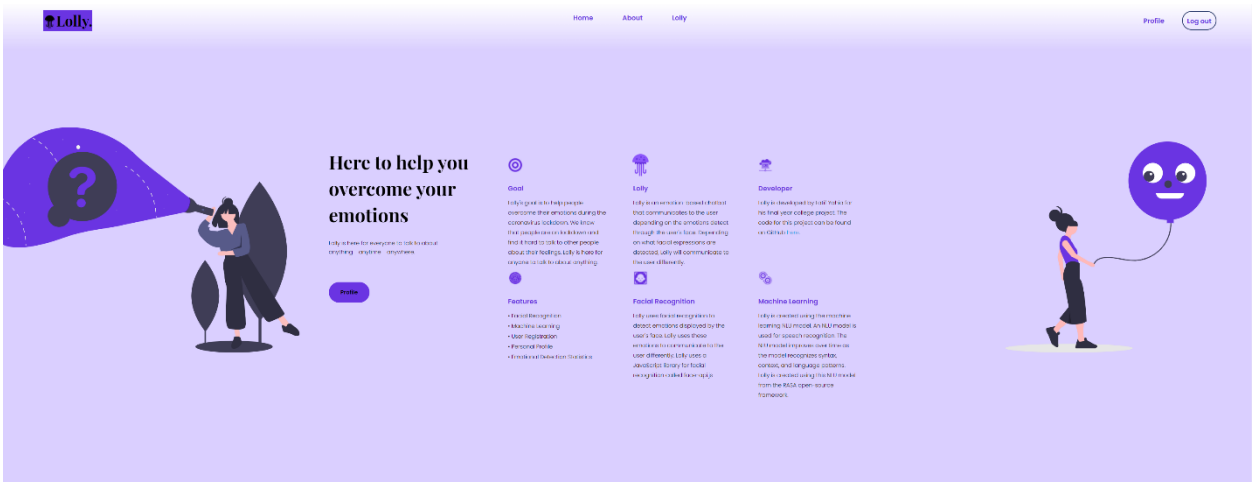


Figure 10-1-2 About Page

As seen in Figure 10-1-3, the Lolly page allows the user to communicate with the chatbot. The Lolly page allows users to enable their webcam, which is used to run facial recognition to detect current emotions. The Lolly page also consists of all the different emotions that can be detected. Depending on the emotion detected, the emotion name would animate to purple. The Lolly page also contains a chat widget, which is used to send and receive messages from Rasa's webhook server.

I want to acknowledge that the code for the chat widget was taken and modified to meet my projects requirements from a developer by the name of Jitesh Gaikwad through their GitHub repository[25]. Although the repository has since been deleted for unknown reasons, I would still like to give credit to Jitesh.

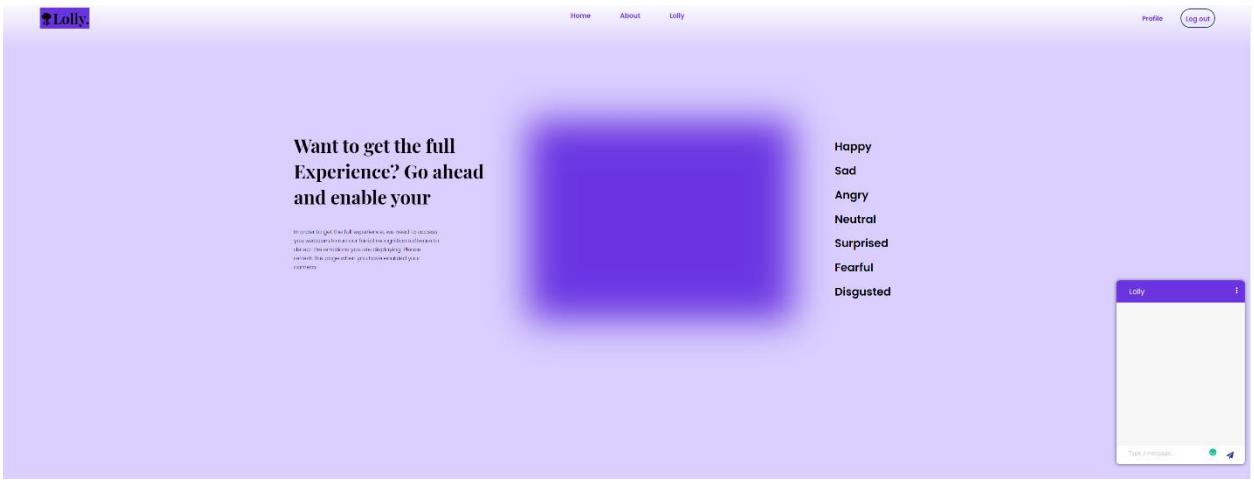


Figure 10-1-3 Lolly Page

As seen in Figure 10-1-4, The profile page allows users to access their personal information as well as personal statistics in relation to emotions detected throughout conversations with Lolly. The user is presented with three different charts created using chart.js. The three different charts are a pie chart, polar chart and bar chart. Although I'm only presenting one type of data, I wanted to add a variety of charts that the user can choose from when viewing statistics.

Figure 10-1-4 Profile Page

10.2 Personal Information & Charting

To chart the different types of emotions detected and to display the user's personal information, I am extracting information from two objects within Django's database called 'userData' and 'User'. I'm doing this in a function called 'profile' in my views.py file, as seen in Figure 10-2-1.

```python
@login_required(login_url='login')
def profile(request):

    userEmotions = { 'happy': 0, 'sad': 0, 'angry': 0, 'neutral': 0,
                     'fearful': 0, 'surprised': 0, 'disgusted': 0,
                     'userName': '', 'firstName': '', 'lastName': '',
                     'emailAddress': '', 'dateJoined': '', 'lastLogin': '',
    }
    current_user = request.user
    userData = UserData.objects.all()
    userEmotions['userName'] = str(current_user.username)
    userEmotions['firstName'] = str(current_user.first_name)
    userEmotions['lastName'] = str(current_user.last_name)
    userEmotions['emailAddress'] = str(current_user.email)
    dateJoined = str(current_user.date_joined)
    lastLogin = str(current_user.last_login)

    dateJoinedParsed = dateJoined[0:10]
    lastLoginParsed = lastLogin[0:10]

    userEmotions['dateJoined'] = dateJoinedParsed
    userEmotions['lastLogin'] = lastLoginParsed
```

Figure 10-2-1 Profile Function

I'm first creating an object called 'userEmotions' that will house the number of each emotion the user has detected and their personal information. I get the current user logged in and use that information to call an object within Django's database that houses all the user's personal information. I then parse the users object to extract the desired information and set elements of the 'userEmotions' object. I am also getting all the objects within a table called 'userData' inside of Django's database. The 'userData' table stores all the detected emotions from all users. If we continue through the 'profile' function as seen in Figure 10-2-2, I write a for-loop to check for each emotion inside the 'userData' object. For each iteration of the 'userData' object, I create an object called 'emotion' and set it to the iteration of the 'userData' object. The object 'emotion' will hold the value of the emotion, for example, happy, sad, angry etc. The object 'emotion' will also hold the author of that emotion (which user the emotion belongs to). I then parse the 'emotion' object into a string and check whether the current user logged in is the author of that emotion. Suppose the author of that emotion is the current logged in user. In that case, the code will check the value of the emotion and increment an element relating to that emotion in the 'userEmotions' object. If the author of the emotion isn't currently logged in, it will pass and go through to the next emotion in the 'userData' object. After the for-loops completion, I render the page to the user and pass across the 'userEmotion' object to my HTML page for further processing.

```
for data in userData:
    emotion = data
    if(str(emotion.user) == str(current_user)):
        if(str(emotion) == 'Happy'):
            userEmotions['happy'] += 1

        elif(str(emotion) == 'Sad'):
            userEmotions['sad'] += 1

        elif(str(emotion) == 'Angry'):
            userEmotions['angry'] += 1

        elif(str(emotion) == 'Neutral'):
            userEmotions['neutral'] += 1

        elif(str(emotion) == 'Fearful'):
            userEmotions['fearful'] += 1

        elif(str(emotion) == 'Surprised'):
            userEmotions['surprised'] += 1

        elif(str(emotion) == 'Disgusted'):
            userEmotions['disgusted'] += 1

return render(request, "data.html", {'userEmotions':userEmotions})
```

Figure 10-2-2 Profile Function Continuation

To display the user's information, I called the element I want to display within the 'userEmotions' object in my HTML page, as seen in Figure 10-2-3.

```
<li>
    <h3>
        User Name
    </h3>
    <h2>
        {{userEmotions.userName}}
    </h2>
</li>
<li>
    <h3>
        First Name
    </h3>
    <h2>
        {{userEmotions.firstName}}
    </h2>
</li>
```

Figure 10-2-3 HTML code displaying personal information

To chart the data, I am setting seven hidden input text fields to the elements that hold the emotion detection data within the ‘userData’ object. I am also creating three canvas elements, as seen in Figure 10-2-4

```
<div class="container-fluid">
    <div class="row h-10">
        <input id="happy" hidden="true" value={{userEmotions.happy}}>
        <input id="sad" hidden="true" value={{userEmotions.sad}}>
        <input id="angry" hidden="true" value={{userEmotions.angry}}>
        <input id="neutral" hidden="true" value={{userEmotions.neutral}}>
        <input id="fearful" hidden="true" value={{userEmotions.fearful}}>
        <input id="surprised" hidden="true" value={{userEmotions.surprised}}>
        <input id="disgusted" hidden="true" value={{userEmotions.disgusted}}>
        <div class="col-sm-1"></div>
        <div class="col-sm-3 ">
            <canvas id="pieChart" width="200" height="200"></canvas>
        </div>
        <div class="col-sm-1"></div>
        <div class="col-sm-3">
            <canvas id="polarAreaChart" width="200" height="200"></canvas>
        </div>
        <div class="col-sm-1"></div>
        <div class="col-sm-3">
            <canvas id="barChart" width="200" height="200"></canvas>
        </div>
    </div>
    <div class="row h-25">
        <br>
    </div>
    <div class="row h-25">
        <div class="col-sm-3 center-align">
            <button id="pieButton" class="btn btn-primary active" data-bs-toggle="button" type="button">Pie Chart</button>
        </div>
        <div class="col-sm-1"></div>
        <div class="col-sm-3 center-align">
            <button id="polarButton" class="btn btn-primary active" data-bs-toggle="button" type="button">Polar Chart</button>
        </div>
        <div class="col-sm-1"></div>
        <div class="col-sm-3 center-align">
            <button id="barButton" class="btn btn-primary active" data-bs-toggle="button" type="button">Bar Chart</button>
        </div>
    </div>
</div>
```

Figure 10-2-4 creation of 7 hidden input text fields & 3 canvases

Later, in JavaScript, I get the values of all the hidden input text fields and all the canvases by their ID. I create a chart to lay on top of each canvas, where I label all emotions and set the value of those labels to be the value of the related hidden input text field. In Figure 10-2-5, I am creating a pie chart. The pie chart is configured with all the necessary information, such as labels and datasets. The labels are different emotions that can be detected. I’m setting the value of these labels by getting the value of the related hidden input text field. I wanted to display each emotion detected out of 100 per cent, so I created a variable called ‘sum’ that is equal to the sum of all the hidden text input field values. Later within the chart function, I am dividing the value of the emotion by the sum of all emotions and multiplying by 100 to get the percentage. I am also rounding up the percentage to make it readable to the user. Within the chart function, I am also setting the colour of each label along with its data entry to make the chart visually appealing.

```
var pie = new Chart(pieChart, {
    type: 'doughnut',
    data: {
        labels: [
            'Happy',
            'Sad',
            'Angry',
            'Surprised',
            'Neutral',
            'Fearful',
            'Disgusted'
        ],
        datasets: [{
            data: [Math.round((happy/sum)*100), Math.round((sad/sum)*100), Math.round((angry/sum)*100),
                Math.round((surprised/sum)*100),Math.round((neutral/sum)*100),Math.round((fearful/sum)*100)
                ,Math.round((disgusted/sum)*100)],
            borderColor: 'rgb(218, 207, 255)',
            backgroundColor: [
                'rgb(33, 200, 180)',
                'rgb(54, 162, 235)',
                'rgb(35, 43, 86)',
                'rgb(255, 234, 132)',
                'rgb(170, 77, 235)',
                'rgb(255, 6, 86)',
                'rgb(4, 253, 6)'
            ],
            hoverOffset: 4
        }]
    }
});
```

Figure 10-2-5 Pie chart function

10.3 Colours, Illustrations, Logo and Icons

I decided to use light, mellow colours throughout my website. I wanted my background colour to be light on the eyes while the illustration, logo and text to be more eye-catching. I decided to make my background a light-pinkish colour, which in hex is the value of '#dacfff'. For the logos and headings, I decided to use a dark eye-catching purple, which in hex has a value of '#6a34e2'. The contrast between the two colours fit the theme of the project. I made sure the website was consistent throughout colour wise.

I want to acknowledge that the illustrations throughout the website were taken from unDraw[26]. unDraw provides open-source illustrations for web design for anyone to use free of charge. I would also like to acknowledge that Lolly's logo was taken at random and coloured to suit my own needs from a GitHub repository by the name of Jellyfish-Bot[27]. I would also like to acknowledge that the icons throughout the website were also taken and changed to suit my own needs from LucidCharts[28].

Figure 10-3-1 Lolly Logo

10.4 Styling and Animation

I want to acknowledge that some parts of my SCSS styling code and GSAP code were taken and heavily modified to suit my own project from a developer named Khalid Akram through their GitHub repository[29].

To style my website, I used CSS/SCSS. I wanted the website to have a clean, modern design. SCSS is a scripting language that compiles into CSS. The reason I used SCSS was that the syntax of the language was much easier to code with. I created an SCSS file for each page on the website. By doing this, I allowed myself to have complete control over the style of the page. I could style different elements within a page without worrying about the code affecting all my other pages. Looking at Figure 10-4-1, you can see a snippet of the code I wrote in SCSS to style the 'About' page. I first called a with the name of 'about-list'. Inside the 'about-list', I can select the different HTML components that lay within the 'about-class' to style. Styling pages by components within a class is why SCSS was used within my project. It gave me the freedom to experiment with styling without affecting the styling of my whole codebase. Even tho I am styling a 'ul' element, the element is only styled if it's within a class called 'about-list', meaning my other 'ul' elements in my codebase didn't take that one particular styling effect.

I also am using CSS to style buttons and other minor components. My buttons use the 'bootstrap.css' library, meaning I could only change the colour of the buttons.

```scss
.about-list {
  width: 100%;
  display: flex;
  justify-content: center;
  ul {
    display: flex;
    justify-content: space-between;
    flex-wrap: wrap;
    padding: 0;
    li {
      list-style: none;
      img {
        margin-bottom: 24px;
        margin-left: -10px;
        margin-top: 15px;
        width: 60px;
        height: 60px;
      }
      h5 {
        color: #6a34e2;
        font-weight: 600;
        font-size: 1.2rem;
        margin: 0;
      }
      p {
        line-height: 1.8rem;
        font-weight: 300;
        color: black;
        width: 270px;
        margin: 16px 0 -15px;
        font-size: 1rem;
        @media (max-width: $widescreen) {
          font-size: 0.875rem;
        }
      }
    }
  }
```

Figure 10-3-1 Lolly Logo

I created animations using the ‘gsap.js’ library. GSAP is an HTML5 JavaScript animation library. To animate an element, you need to get that element by its ID in JavaScript, then create a GSAP function and pass it the element you want to animate. Within the GSAP function, you can add parameters to set the duration of animations, where the element will animate from, the opacity during the animation, and many other parameters. GSAP also has built-in ease modes that range between 1 and 3 for different animation speeds. Looking at Figure 10-3-2, you can see two GSAP functions I wrote to animate two images on the ‘about’ page.

```
gsap.from(img1, {
    x:-1000,
    opacity:0,
    duration: 1.5,
    ease: "power3.out",

})

gsap.from(img2, {
    x:1000,
    opacity:0,
    duration: 1.5,
    ease: "power3.out",

})
```

Figure 10-3-2 GSAP animation functions

Looking at Figure 10-3-2, I have two GSAP functions written in JavaScript. The first function animates an image called ‘img1’ from left to right. This is done by adding the variable ‘x’ to allow horizontal animation. You may also add ‘y’ for vertical animation, but I am not using it in this function. I am also setting the opacity to fade in from 0 to 1, meaning the image fades in with the animation. I’m then setting the duration of the animation as well as the ease. I wanted this particular animation to last 1.5 seconds with the ease of ‘power3.out’, which is the highest ease preset within GSAP. I am doing the same animation in the second function, although the image called ‘img2’ will animate from the right. These two functions will run asynchronously.

GSAP also allows for animations to roll out in sequence by the use of ‘timelines’. In Figure 10-3-3, I write a timeline to allow animations to run in succession.

```
const timeLine = gsap.timeline();

timeLine.from(h1Line,{
        delay:1,
        y:80,
        opacity:0,
        duration:0.8,
        ease: 'power3.out',
        stagger: {
            amount:0.2
        }
    }).from(contentP,{
    y:-40,
    duration: 0.8,
    opacity: 0,
    ease: "power3.out",
    stagger: {
        amount:0.2
        }
    })
```

Figure 10-3-3 GSAP timeline function

The timeline In Figure 10-3-3 is created to allow the heading on the 'Lolly' page first to animate, followed by the subheading. To do this, I first create a timeline. This can be done by creating a variable called 'timeLine' and assigning it to a function called 'gsap. timeline()'. The timeline function is very similar to the function seen in Figure 10-3-2. The only difference is that I am now creating the function using the 'timeLine' variable instead of 'gsap'. To enable animation in succession within a timeline, I put '.from' to the end of the first function. This tells the timeline that when the first function has completed, then run the second function.

10.5 SCSS Compiler Issue

Django, by default, does not compile SCSS code. To compile SCSS, I must install a third-party SCSS compiler. I started researching different compilers. I came across an SCSS compiler named 'django_libsass.SassCompiler' and installed the package[30]. Although I had installed an SCSS compiler, Django was still not compiling any SCSS code I wrote. This was an issue, as I wanted to use SCSS to make my website look clean and modern. I researched the issue and came up empty-handed, so I continued researching different compilers. I finally came across an SCSS compiler that worked named 'Django-sass-processer'[31]. I installed the package and configured the compiler in my settings.py file. I wrote some SCSS code to test the compiler, which turned out to work. Although I fixed the issue, I was still curious why the first compiler didn't work for me. I investigated the case further and came across a similar issue online regarding the 'django_libsass.SassCompiler'. After reading about the issue, I concluded that the problem exists because of the version of Django I was running. The 'django_libsass.SassCompiler' package seemed to be outdated and didn't work on the newer versions of Django

11 Ethics

With a project that deals with personal information, especially emotion detection relating to mental health, I had to ensure that the user's data is secure. The user's data is secured within Django's database. Throughout the software industry, user's personal information is sold to data companies. Freedom of privacy does not exist within the software industry anymore. I decided to protect my user's privacy and not share them with anyone else.

Throughout the development of the project, I also used open-source code to create my desired project. I wanted to make sure to reference all the open-source code used throughout my project. I have done this to ensure the creators of the open-source code were being credited for the work they produced.

12 Conclusion

In conclusion, I have produced a machine learning chatbot name Lolly that communicates to a user based on emotions detected. Lolly uses facial recognition to read the users facial expressions as different emotions. I also developed a clean and modern designed website to host Lolly. The website is fully functional and allows users to create profiles and view personal statistics. The combination of the chatbot and website make for a great product prototype. I'm pleased to say the outcome of the project surpassed my expectations.

In the future, I plan to continue developing Lolly and improve its machine learning model. I also plan to make the website open to the public.

13 Appendix

13.1 Programming Languages Used

- Python
- JavaScript
- CSS/SCSS
- HTML
- YAML

13.2 Software Technologies Used

- AWS (Amazon Web Services)
- RASA
- Django
- NodeJS

13.3 Software Libraries Used

- Face-api.js
- Chart.js
- Bootstrap.js
- Gsap.js
- jQuary.js
- Bootstrap.css
- Materialize.css
- ScrollMagic.js

13.4 Useful Links

Please click the text to follow the link.

- Project Code
- Project Log
- Project Research & Referencing
- Video Demonstration

14 References

[1] A. Gadermann, C. McAuliffe, and E. Jenkins, "Mental health impact of coronavirus pandemic hits marginalized groups hardest," *The Conversation*, 26-Jul-2020.

[2] "The implications of COVID-19 for mental health and substance use," *Kff.org*, 10-Feb-2021. [Online]. Available: https://www.kff.org/coronavirus-covid-19/issue-brief/the-implications-of-covid-19-for-mental-health-and-substance-use/. [Accessed: 13-May-2021].

[3] "Face Recognition with FaceNet and MTCNN," *Arsfutura.com*. [Online]. Available: https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/. [Accessed: 14-May-2021].

[4] V. Mühler, "face-api.js," *GitHub.com*. [Online]. Available: https://github.com/justadudewhohacks/face-api.js. [Accessed: 02-Mar-2021].

[5] "Face-Detection-JavaScript," *GitHub.com*. [Online]. Available: https://github.com/WebDevSimplified/Face-Detection-JavaScript. [Accessed: 02-Mar-2021].

[6] JS Foundation-js. foundation, "jQuery.ajax()," *Jquery.com*. [Online]. Available: https://api.jquery.com/jquery.ajax/. [Accessed: 14-May-2021].

[7] "Machine Learning," *Sas.com*, 25-Feb-2021. [Online]. Available: https://www.sas.com/en_ie/insights/analytics/machine-learning.html. [Accessed: 13-May-2021].

[8] "Open source conversational AI," *Rasa.com*, 01-Dec-2020. [Online]. Available: https://rasa.com/. [Accessed: 13-May-2021].

[9] "Natural Language Understanding (NLU)," *Twilio.com*. [Online]. Available: https://www.twilio.com/docs/glossary/what-is-natural-language-understanding. [Accessed: 13-May-2021].

[10] "Actions," *Rasa.com*. [Online]. Available: https://rasa.com/docs/rasa/actions/. [Accessed: 13-May-2021].

[11] "Stories," *Rasa.com*. [Online]. Available: https://rasa.com/docs/rasa/stories/. [Accessed: 13-May-2021].

[12] "RASA stories not being followed through different values for slots," *Rasa.com*, 07-Apr-2021. [Online]. Available: https://forum.rasa.com/t/rasa-stories-not-being-followed-through-different-values-for-slots/42167. [Accessed: 14-May-2021].

[13] Wikipedia contributors, "Django (web framework)," *Wikipedia, The Free Encyclopedia*, 06-Apr-2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Django_(web_framework)&oldid=1016342445. [Accessed: 14-May-2021].

[14] T. Ruscica, "Django-Website," *GitHub.com*. [Online]. Available: https://github.com/techwithtim/Django-Website. [Accessed: 20-Apr-2021].

[15] "Installation — django-crispy-forms 1.11.1 documentation," *Readthedocs.io*. [Online]. Available: https://django-crispy-forms.readthedocs.io/en/latest/install.html. [Accessed: 15-May-2021].

[16] "Django Middleware," *Javatpoint.com*. [Online]. Available: https://www.javatpoint.com/django-middleware. [Accessed: 15-May-2021].

[17] A. Rodland and ARODLAND, "Crypt::PBKDF2," *Metacpan.org*. [Online]. Available: https://metacpan.org/pod/Crypt::PBKDF2. [Accessed: 15-May-2021].

[18] Wikipedia contributors, "PBKDF2," *Wikipedia, The Free Encyclopedia*, 05-May-2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=PBKDF2&oldid=1021596608. [Accessed: 15-May-2021].

[19] Wikipedia contributors, "SHA-2," *Wikipedia, The Free Encyclopedia*, 28-Apr-2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SHA-2&oldid=1020253249. [Accessed: 15-May-2021].

[20] "How can I enable CORS on Django REST Framework," *Stackoverflow.com*. [Online]. Available: https://stackoverflow.com/questions/35760943/how-can-i-enable-cors-on-django-rest-framework. [Accessed: 29-Apr-2021].

[21] M. Otto, J. Thornton, and Bootstrap contributors, "Bootstrap," *Getbootstrap.com*. [Online]. Available: https://getbootstrap.com/. [Accessed: 18-Apr-2021].

[22] "Materialize," *Materializecss.com*. [Online]. Available: https://materializecss.com/. [Accessed: 02-Jan-2021].

[23] "GSAP - GreenSock," *Greensock.com*. [Online]. Available: https://greensock.com/gsap/. [Accessed: 04-2021].

[24] "Chart.js," *Chartjs.org*. [Online]. Available: https://www.chartjs.org/. [Accessed: 20-Apr-2021].

[25] J. Gaikwad, *Github.com*. [Online]. Available: https://github.com/JiteshGaikwad/Chatbot-Widget. [Accessed: 02-Jan-2021].

[26] "unDraw," *Undraw.co*. [Online]. Available: https://undraw.co/. [Accessed: 25-Apr-2021].

[27] "jellyfish-bot - Overview," *GitHub.com*. [Online]. Available: https://github.com/jellyfish-bot. [Accessed: 22-Apr-2021].

[28] "LucidCharts," *Lucidchart.com*. [Online]. Available: https://www.lucidchart.com/pages/. [Accessed: 25-Apr-2021].

[29] A. Khalid, "cuberto-home-cursor." [Online]. Available: https://github.com/wrongakram/cuberto-home-cursor. [Accessed: 24-Apr-2021].

[30] "How to use SCSS/SASS in your Django project (Python Way)," *Accordbox.com*. [Online]. Available: https://www.accordbox.com/blog/how-use-scss-sass-your-django-project-python-way/. [Accessed: 16-May-2021].

[31] "django-sass-processor," *Pypi.org*. [Online]. Available: https://pypi.org/project/django-sass-processor/. [Accessed: 16-May-2021].