



**INFORMATICS  
INSTITUTE OF  
TECHNOLOGY**

**UNIVERSITY OF  
WESTMINSTER**

# **INFORMATICS INSTITUTE OF TECHNOLOGY**

**DEPARTMENT OF COMPUTING**

**(B.Eng.) in Software Engineering**

**Module Code: 5COSC021C**

**Software Development Group Project**

**Module Leader: Mr. Banuka Athuraliya**

**Heart Failure Prediction System**

**Team: Excalibur**

**Group: SE - 87**

| <b>NAME</b>                          | <b>IIT ID</b> | <b>UOW ID</b> |
|--------------------------------------|---------------|---------------|
| NADUN WICKRAMANYAKE<br>( TEAM LEAD ) | 20200928      | w1870591      |
| KIMALI DIHANSA ABEYNAYAKA            | 20211221      | W1898907      |
| ASHA AMARAWEEERA                     | 20200599      | w1833489      |
| THENUWAN THEEKSHANA                  | 20200911      | W1838860      |
| THUSHINI ABEYSURIYA                  | 20210156      | W1867426      |



## Declaration

We declare that this dissertation is a factual research report conducted at the Institute of Information Technology under the supervision of Ms.Sulochana Rupasinghe. I further state that this dissertation has not previously formed the basis for awarding any degree, diploma, affiliation, membership or any other similar recognition.

## Abstract

Depression is a common mental health disorder that affects millions of people worldwide. Early detection and treatment can help to prevent the negative effects of this disorder. This project aims to develop a depression screening application with three functionalities, including indepth screening, quick screening, and pdf report generation. The application includes a chatbot that predicts the depression level of the person using a machine learning model trained under logistic regression and sentiment analysis. Additionally, the quick screening part includes a questionnaire that is related to the PHQ-9 questionnaire. The system is built using Flask API to connect the Python backend and the HTML frontend of the website. Firebase authentication is used for login to the system, and Google Colab is used for data preprocessing and model training. The model achieved 0.68 accuracy after training. GitHub actions are used to handle the CI part of the website, and postman API is used to send the get and post requests to connect the frontend and backend parts.

## Keywords

Depression screening, chatbot, machine learning, Flask API, Firebase authentication, Google Colab, GitHub actions, postman API, logistic regression, sentiment analysis, PHQ-9 questionnaire.

## Acknowledgment

We would like to express our sincere gratitude to everyone who has contributed to the development of our depression screening application. Our team members worked collaboratively to create a web application with three functionalities: indepth screening, quick screening, and a pdf report generation using the predicted result of the chatbot.

In the indepth screening feature, we utilized a chatbot that predicts the depression level of a person based on a machine learning model trained under the logistic regression algorithm and sentiment analysis. For the quick screening, we used a questionnaire that is related to the PHQ-9 questionnaire.

The application was built using Flask API to connect the Python backend and HTML frontend of the website. Firebase authentication was implemented for login purposes. Data preprocessing and model training were done using Google Colab, and the machine learning model achieved 0.68 accuracy after training. GitHub Actions were used for the CI part of the website, and a small unit testing was done using Python for the backend.

We also utilized Postman API to send GET and POST requests to connect the frontend and backend parts. We acknowledge the contributions of everyone who was involved in this project, and we are grateful for the opportunities to learn and improve our skills in software development.

# Table of Contents

|   |      |
|---|------|
| Declaration.....  | i    |
| Abstract.....   | ii   |
| Keywords.....   | ii   |
| Acknowledgment.....   | iii  |
| Table of Contents.....  | iv   |
| List of Figures.....  | vi   |
| Abbreviations table.....  | viii |
| Chapter 1: Implementation .....                                   | 1    |
| 1.1 Chapter Overview .....  | 1    |
| 1.2 Overview of the prototype .....                               | 1    |
| 1.3 Technology selections .....                                   | 2    |
| 1.3.1. Language selection .....                                   | 2    |
| 1.3.2. Software selection.....                                    | 4    |
| 1.3.3. Libraries selection.....                                   | 5    |
| 1.4 Implementation of the data science component.....             | 6    |
| 1.5 Implementation of the backend component.....                  | 13   |
| 1.5.1 Implementation of the backend component of the chatbot..... | 13   |
| 1.6 Implementation of the front-end component .....               | 20   |
| 1.7 GIT Repository .....  | 28   |
| 1.8 Deployments/CI-CD Pipeline .....                              | 29   |
| 1.9 Chapter Summary .....   | 29   |
| Chapter 2: Testing.....   | 30   |
| 2.1 Chapter Introduction.....                                     | 30   |
| 2.2 Testing Criteria .....  | 30   |
| 2.3 Testing functional requirements .....                         | 31   |
| 2.4 Testing non-functional requirements .....                     | 32   |
| 2.5 Unit testing.....   | 33   |

|   |    |
|---|----|
| 2.6 Performance testing .....   | 34 |
| 2.7 Usability testing .....   | 35 |
| 2.8 Compatibility testing .....   | 36 |
| 2.9 Chapter Summary .....   | 36 |
| Chapter -3 Evaluation .....   | 37 |
| 3.1 Chapter Overview .....  | 37 |
| 3.2 Evaluation methods .....  | 37 |
| 3.3 Quantitative evaluation .....   | 38 |
| 3.4 Qualitative evaluation .....  | 38 |
| 3.5 Self evaluation .....   | 39 |
| 3.6 Chapter Summary .....   | 39 |
| Chapter 4: Conclusion .....   | 40 |
| 4.1 Chapter Overview .....  | 40 |
| 4.2 Achievements of aims and objectives .....   | 40 |
| 4.3 Limitations of the research .....   | 41 |
| 4.5 Future enhancements .....   | 41 |
| The mental health web application could be expanded to include additional features such as online therapy sessions, mindfulness exercises, and meditation resources. This would provide users with a more comprehensive and integrated approach to managing their mental health. The mental health web application could be adapted for use in different cultural contexts to ensure that it is sensitive to the unique needs and experiences of different communities. This could involve working with mental health professionals and community leaders to develop culturally appropriate content and resources. ....   | 41 |
| Future studies could focus on evaluating the long-term effectiveness of the mental health web application and its impact on user outcomes such as mental health symptoms, quality of life, and help-seeking behaviors. This could involve following up with users over a longer period of time to assess changes in their mental health status and to identify factors that contribute to the effectiveness of the application. Future research could explore the use of alternative evaluation methods, such as qualitative research methods, to provide a more in-depth understanding of users' experiences with the mental health web application. This could involve conducting interviews or focus groups with users to gather more detailed feedback and insights into the effectiveness of the application ..... | 41 |
| 4.6 Extra work .....  | 42 |
| 4.6 Concluding Remarks .....  | 46 |

|                 |        |
|-----------------|--------|
| References..... | XLVII  |
| Appendix.....   | XLVIII |

## List of Figures

|   |    |
|---|----|
| Figure 1: Import libraries .....                                  | 6  |
| Figure 2: Download NLTK stopwords.....                            | 7  |
| Figure 3:Load the dataset part 01 .....                           | 7  |
| Figure 4: Load the dataset part 02.....                           | 8  |
| Figure 5: Remove unnecessary columns:.....                        | 8  |
| Figure 6: Clean the text data .....                               | 9  |
| Figure 7: Map the sentiment to binary values:.....                | 9  |
| Figure 8: Create feature vectors .....                            | 9  |
| Figure 9: Split the dataset into training and testing sets: ..... | 10 |
| Figure 10: Train a logistic regression model: .....               | 10 |
| Figure 11: Predict the sentiment of test data.....                | 11 |
| Figure 12: Save the trained model: .....                          | 12 |
| Figure 13: Print the results: .....                               | 12 |
| Figure 14: Structure of the questionnaire.....                    | 14 |
| Figure 15: Sever.js file of the questionnaire .....               | 15 |



|   |        |
|---|--------|
| Figure 16: PropertyModel.js file of the questionnaire ..... | 16     |
| Figure 17: PropertyModel.js file of the questionnaire ..... | 17     |
| Figure 18: PropertyModel.js file of the questionnaire ..... | 18     |
| Figure 19: database.js file of the questionnaire.....       | 19     |
| Figure 20: PropertyModel.js file of the questionnaire ..... | 20     |
| Figure 21: Home page .....                                  | 21     |
| Figure 22: Services Page.....                               | 21     |
| Figure 23: About us.....                                    | 22     |
| Figure 24: Register.....                                    | 22     |
| Figure 25: Register password authentication .....           | 23     |
| Figure 26: Directing to login.....                          | 24     |
| Figure 27: Firebase authentication database .....           | 24     |
| Figure 28: Depression screening categories.....             | 25     |
| Figure 29: Quick screening .....                            | 25     |
| Figure 30: In-Depth Screening.....                          | 25     |
| Figure 31: Depression score generation .....                | 26     |
| Figure 32: Depression analysis report.....                  | 26     |
| Figure 33: Depression analysis report pdf .....             | 27     |
| Figure 34: Log out from the web app .....                   | 27     |
| Figure 35: GitHub commit history .....                      | 28     |
| Figure 36: Git profile .....                                | 28     |
| Figure 37: CI part of the project.....                      | 29     |
| Figure 38: Imagine cup submission .....                     | 42     |
| Figure 39: Imagine cup google form submission.....          | 42     |
| Figure 40: CodeSprint Submission .....                      | 43     |
| Figure 41: CodeSprint Submission page 02.....               | 43     |
| Figure 42: RealHack Competition .....                       | 44     |
| Figure 43: RealHack Competition winners.....                | 44     |
| Figure 44: IEEE Extreme participation 01 .....              | 45     |
| Figure 45: IEEE Extreme participation 02.....               | 45     |
| Figure 46: Github Contributors list page 01 .....           | XLVIII |
| Figure 47: Github Contributors list page 01 .....           | XLIX   |
| Figure 48: GitHub Commits .....                             | XLIX   |

## Abbreviations table

| Abbreviation        | Meaning   |
|---------------------|---|
| API                 | Application Programming Interface                                 |
| CI                  | Continuous Integration  |
| PDF                 | Portable Document Format  |
| PHQ-9               | Patient Health Questionnaire-9                                    |
| POST                | HTTP method for sending data to a server                          |
| GET                 | HTTP method for requesting data from a server                     |
| ML                  | Machine Learning  |
| HTML                | Hypertext Markup Language   |
| Flask               | Python web framework  |
| Firebase            | Google's mobile and web application development platform          |
| Colab               | Google Colaboratory   |
| Github              | Web-based platform for version control and collaboration          |
| Sentiment Analysis  | Analysis of emotions or opinions in text                          |
| Logistic Regression | Statistical model used to analyze relationships between variables |
| Backend             | Server-side of an application                                     |
| Frontend            | Client-side of an application                                     |
| Unit Testing        | Testing method for individual units or components of software     |





# Chapter 1: Implementation

## 1.1 Chapter Overview

This chapter focuses on the overall prototype implementation, GIT repository and the deployment process of the proposed heart failure prediction application. So, it discusses the implementation procedure that was selected for the project like libraries, programming languages, technologies, and frameworks. Furthermore, it will discuss the MindGuard web application's prototype features and how those core data science components were implemented. Code snippets and screenshots of the outputs are also provided to get a better understanding of the project.

## 1.2 Overview of the prototype

The application is a web-based screening tool for depression with three main features:

1. **In-depth screening:** This feature provides a more comprehensive assessment of depression by using a chatbot that predicts the level of depression based on sentiment analysis and a logistic regression algorithm. The user can answer questions posed by the chatbot, and based on the responses, the chatbot predicts the level of depression. The predicted result is then used to generate a report in PDF format.
2. **Quick screening:** This feature allows users to quickly assess their level of depression by answering a series of questions related to the PHQ-9 questionnaire. The answers are used to generate a score that indicates the severity of depression.
3. **PDF report generation:** This feature generates a report in PDF format based on the results of the in-depth screening feature.

The application uses Flask API to connect the Python backend with the HTML frontend of the website. Firebase authentication is used to authenticate users when logging into the system. Google Colab is used for data preprocessing and model training, and the machine learning model used in the in-depth screening feature achieves 0.68 accuracy after training. GitHub Actions is used for continuous integration (CI), and unit testing is performed using Python. Postman API is used to send GET and POST requests to connect the frontend and backend parts of the application.

Overall, your group's depression screening application prototype seems to have a user-friendly interface with well-thought-out features and solid technical infrastructure.

## 1.3 Technology selections

### 1.3.1. Language selection

Html is a standard markup language used for web pages and web applications. It provides the structure and content of the web page, and is essential for creating user interfaces. In MindGuard web application Html is likely used to structure and organize the question set and provide a user friendly interface for users to navigate.

Python is a popular programming language that is often used in web development. It is known for its simplicity, readability and ease of use. Python is likely used in MindGuard applications to handle the backend logic, such as processing user input, generating chatbot responses and storing user data.

Language Selection: Our depression screening application was developed using Python programming language. We chose Python because of its simplicity, readability, and vast selection of libraries available for data analysis and machine learning.

#### 1.3.1.1. Frontend

The frontend of the depression screening application was built using HTML, CSS, and JavaScript. These technologies were selected because they are widely used for web development and are well-suited for building dynamic, interactive user interfaces. Additionally, the frontend uses the Flask web framework to connect with the backend.

JavaScript was used for the front-end development of this project. This was due to the fact that most people used react native as the mobile UI framework and also this is written in JavaScript, html and CSS. JavaScript, html, and CSS were used for the following reasons.

- JavaScript, html, CSS are simple languages to learn and implement.
- JavaScript, html, CSS are popular languages.
- They give the power to make rich interfaces.
- Easy maintenance of codes.
- JavaScript, html and CSS can give wonderful front-end experience by making front end tasks reusable.
- They are quick and fast.

### 1.3.1.1. Backend

Python and Flask micro-framework, jupyter notebook and node.js were used for the development of the backend of this application. The main reasons were that programming languages for the implementation purposes of the backend of this application after evaluating the other available programming languages. The decision was based primarily on the following considerations.

- Python, node.js, jupyter notebook have number of useful libraries
- Node.js is a simple language.
- Consists of mature frameworks as well as development tools.
- Flask is a lightweight framework and is much easier to learn.
- Flask is great for building Rest APIs and microservice.
- Offers great support when building APIs for Machine Learning applications.

The backend of the depression screening application was built using Python and several related libraries and frameworks. Specifically, the backend uses Flask, a popular web framework for Python, to provide a RESTful API for communicating with the frontend. For data preprocessing and model training, the team used Google Colab, which is a cloud-based platform for machine learning and data analysis. The machine learning model for predicting depression risk was trained using logistic regression and sentiment analysis algorithms.

To handle user authentication, the team implemented Firebase authentication, a secure authentication mechanism that integrates with the Firebase platform. For continuous integration (CI) and testing, the team used GitHub Actions, a powerful automation tool that can be used to build, test, and deploy code. Finally, the team used Postman API to test the API endpoints and ensure that the frontend and backend were properly connected.

### 1.3.2. Software selection

- **HTML:** Hypertext Markup Language is the standard markup language used for creating web pages and applications.
- **CSS:** Cascading Style Sheets is a style sheet language used to describe the visual style and layout of web pages and applications.
- **JavaScript:** A programming language used to create interactive web pages and applications.
- **Python:** A high-level programming language used for various purposes, including web development, data analysis, and machine learning.
- **Flask:** A lightweight and flexible web framework used to build web applications and APIs with Python.
- **Google Colab:** A cloud-based platform for developing and running Python code that offers free access to a GPU.
- **Logistic Regression:** A machine learning algorithm used for classification problems, such as predicting the risk of depression in this project.
- **Sentiment Analysis:** A technique used to identify and extract subjective information from text data, which was used to improve the accuracy of the machine learning model.
- **Firebase Authentication:** A service that allows you to easily and securely authenticate users to your application using email and password authentication, social identity providers, and more.
- **GitHub Actions:** A service that automates software workflows, including building, testing, and deploying code changes.
- **Postman API:** A tool used to test and debug APIs by sending requests and receiving responses.
- **Python PDFKit library:** A Python wrapper for wkhtmltopdf, a command-line tool used to convert HTML to PDF.
- **Heroku:** A cloud platform as a service used to deploy, manage, and scale applications.

#### 1.3.2.1. Jupiter Notebook / Google Colab

Jupiter Notebook was used to build and evaluate machine learning models with visual feedback. Because of the nature of notebooks, we can write and execute codes as multiple parts. It was helpful to save all the findings in a well-organized way so we can demonstrate them to others. Google Colab is also a similar platform that can perform all the tasks Jupiter Notebook performs. It's a cloud notebook platform by Google so if we need more computational resources to run our models it's a good place.

#### 1.3.2.2. Visual Studio Code



Visual Studio Code is an IDE developed by Microsoft with cross platform support. It is a great IDE that can be used with a variety of programming languages. There are so many extensions to explore that can be used to increase user productivity.

### **1.3.2.3. Postman**

Postman is an API Platform for developers. Postman was used to testing APIs for this application. The collections feature gave the ability to save and organize tests.

### **1.3.3. Libraries selection**

1. Flask - a micro web framework used to build web applications and APIs.
2. Firebase authentication - a secure way to authenticate users and restrict access to specific parts of the application.
3. Pandas - a data manipulation library used for data cleaning and transformation.
4. Scikit-learn - a machine learning library used for model training and evaluation.
5. NLTK - a natural language processing library used for sentiment analysis.
6. Pytest - a testing framework used for unit testing.
7. Postman API - an API testing tool used to send GET and POST requests and validate API responses.

Additionally, we used Google Colab for data preprocessing and model training and GitHub Actions for continuous integration (CI) of our application.

#### **1.3.3.1. Pandas**

The library is written in Python and is used to manipulate numerical and time-series data. It defines three-dimensional and two-dimensional data with data frames and series, respectively. It also has options for indexing large datasets for quick searches. It is well-known for its data reshaping, pivoting on user-defined axes, missing data handling, merging and joining datasets, and data filtration options. With large datasets, Pandas is very useful and fast.

### 1.3.3.2. Numpy

Numpy is a Python library for manipulating multidimensional data and complex mathematical functions. Numpy is a fast computational library that can handle everything from simple algebra to Fourier transforms, shape manipulations and random simulations.

### 1.3.3.3. Matplotlib

Matplotlib is a Python library that allows visualizing data in order to better understand it before processing it and training it for Machine Learning. Graphs and plots are created using objectoriented APIs and python GUI toolkits. A MATLAB-like interface is also provided by Matplotlib, allowing users to perform tasks similar to those performed by MATLAB. This library is free and open-source, and it includes a number of extension interfaces that connect the Matplotlib API to a variety of other libraries.

### 1.3.3.4. Tensorflow

Tensorflow is an open-source library used in machine learning. We used this library while building the artificial neural network used to classify ECG images. Because it has support for GPU of the system, we were able to train the machine learning model fast. This library uses Keras to allow users to build their own models.

## 1.4 Implementation of the data science component

### Import necessary libraries

```
# Import necessary libraries
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import joblib
import pickle
```

## 1. Import necessary libraries:

This code snippet imports the required libraries to perform the data science tasks, including pandas and numpy for data manipulation, re for regular expressions, nltk for natural language processing, TfidfVectorizer from scikit-learn for text feature extraction, LogisticRegression for classification, accuracy\_score for model evaluation, and joblib and pickle for model serialization.

## Download NLTK stopwords

```
[ ] # Download NLTK stopwords
    nltk.download('stopwords')

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Figure 2: Download NLTK stopwords

## 2. Download NLTK stopwords:

This code snippet downloads the stopwords corpus from NLTK, which is a collection of words that are commonly used in a language but do not carry any significant meaning, such as "a", "an", "the", "and", "or", etc. These words are typically removed from text data during natural language processing.

### Load the dataset

```
[ ] !pip install -U --no-cache-dir gdown --pre

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gdown in /usr/local/lib/python3.9/dist-packages (4.6.6)
Collecting gdown
  Downloading gdown-4.7.1-py3-none-any.whl (15 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from gdown) (3.10.7)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from gdown) (4.65.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.9/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.9/dist-packages (from gdown) (2.27.1)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from gdown) (1.16.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.9/dist-packages (from beautifulsoup4->gdown) (2.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gdown) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gdown) (2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gdown) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gdown) (3.4)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.9/dist-packages (from requests[socks]->gdown) (1.7.1)
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.6.6
```

### 3. Load the dataset:

This code snippet loads the dataset, which is a CSV file containing tweets and their corresponding sentiments. The dataset is stored in a pandas dataframe, a two-dimensional table with rows and columns.

```
[ ] !gdown 12Gnu6jDvK6tLkDqKGOOmzBvTKwrwDyy5

Downloading...
From: https://drive.google.com/uc?id=12Gnu6jDvK6tLkDqKGOOmzBvTKwrwDyy5
To: /content/Tweets.csv
100% 3.50M/3.50M [00:00<00:00, 216MB/s]

[ ] # Load the dataset
import pandas as pd
df = pd.read_csv('/content/Tweets.csv')
```

Figure 4: Load the dataset part 02

### 4. Remove unnecessary columns:

This code snippet removes the "textID" column from the dataframe since it is not required for sentiment analysis.

#### Remove unnecessary columns

```
[ ] # Remove unnecessary columns
df.drop(['textID'], axis=1, inplace=True)
```

Figure 5: Remove unnecessary columns:

This code snippet defines a function called "clean\_text" that performs text cleaning operations, including converting the text to lowercase, removing punctuation, removing digits, removing extra whitespace, and removing stop words using NLTK stopwords. This function is applied to the "text" and "selected\_text" columns of the dataframe using the "apply" function.

#### Clean the text data

```
[ ] # Clean the text data
def clean_text(text):
    text = str(text).lower()
    text = re.sub(r'^\w\s', '', text)
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'\s+', ' ', text)
    text = ' '.join(word for word in text.split() if word not in stopwords.words('english'))
    return text

[ ] import re
from nltk.corpus import stopwords

df['text'] = df['text'].apply(lambda x: clean_text(x))
df['selected_text'] = df['selected_text'].apply(lambda x: clean_text(x))
```

Figure 6: Clean the text data

#### 6. Map the sentiment to binary values:

This code snippet maps the sentiment labels from their original string values ("positive", "negative", "neutral") to binary values (2, 0, 1) using the "map" function.

#### Map the sentiment to binary values

```
[ ] # Map the sentiment to binary values
df['sentiment'] = df['sentiment'].map({'positive': 2,
                                       'negative': 0, 'neutral': 1})
```

Figure 7: Map the sentiment to binary values:

#### 7. Create feature vectors:

This code snippet defines a function called "preprocess\_data" that applies TF-IDF vectorization to the cleaned text data using TfidfVectorizer from scikit-learn. This function returns a sparse matrix of feature vectors, where each row corresponds to a tweet and each column corresponds to a unique word in the corpus.

#### Create feature vectors

Excalibur

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf = TfidfVectorizer(strip_accents=None, lowercase=False,
```

| Page

## 8. Split the dataset into training and testing sets:

This code snippet splits the feature vectors and corresponding sentiment labels into training and testing sets using the "train\_test\_split" function from scikit-learn. The training set is used to train the logistic regression model, while the testing set is used to evaluate its performance.

### Split the dataset into training and testing sets

```
[ ] from sklearn.model_selection import train_test_split

X = preprocess_data(df)
X_train, X_test, y_train, y_test = train_test_split(X, df['sentiment'],
                                                    test_size=0.2,
                                                    random_state=42)
```

Figure 9: Split the dataset into training and testing sets:

## 9. Train a logistic regression model:

This code snippet defines a logistic regression model using LogisticRegression from scikit-learn and fits it to the training data using the "fit" method.

### Train a logistic regression model

```
[ ] from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
```

```
[ ] lr.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```
    LogisticRegression
    LogisticRegression()
```

#### 10. Predict the sentiment of test data:

This code snippet uses the trained logistic regression model to predict the sentiment labels of the testing data and computes the accuracy and confusion matrix using the "accuracy\_score" and "confusion\_matrix" functions from scikit-learn.

#### Predict the sentiment of test data

```
[ ] from sklearn.metrics import accuracy_score, confusion_matrix

y_pred = lr.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print("Accuracy:", acc)
print("Confusion matrix:\n", cm)
```

```
Accuracy: 0.6887393123521921
Confusion matrix:
[[ 883  592   87]
 [ 248 1734  248]
 [   49  487 1169]]
```

*Figure 11: Predict the sentiment of test data*

### 11. Save the trained model:

This code snippet saves the trained logistic regression model and TfidfVectorizer object to disk using the "dump" function from pickle. This allows us to reuse the model and vectorizer in future applications without having to retrain them.

#### Save the trained model

```
[ ] import pickle as pkl

[ ] with open(r"LRPWeights.pkl", "wb") as output_file:
    pkl.dump(lr, output_file)
```

Figure 12: Save the trained model:

### 12. Print the results:

This code snippet loads the saved logistic regression model and TfidfVectorizer object from disk using the "load" function from pickle. It then defines a function called "classify" that maps the binary sentiment labels back to their original string values. Finally, it prompts the user to enter a sentence, applies the TfidfVectorizer to convert it into a feature vector, uses the logistic regression model to predict its sentiment label, and prints the result.

```
[ ] filename = '/content/LRPWeights.pkl'
    loaded_model = pickle.load(open(filename, 'rb'))

[ ] def classify(prediction):
    if prediction == 0:
        return 'Negative'
    elif prediction == 1:
        return 'Neutral'
    else:
        return 'Positive'

[ ] import pickle
    import numpy as np

    data = input('Enter a sentence:')

    # Word to Vector
    x = tfidf.transform([data])
```



## **1.5 Implementation of the backend component**

### **1.5.1 Implementation of the backend component of the chatbot**

#### **1.5.1.1 Structure of the questionnaire**

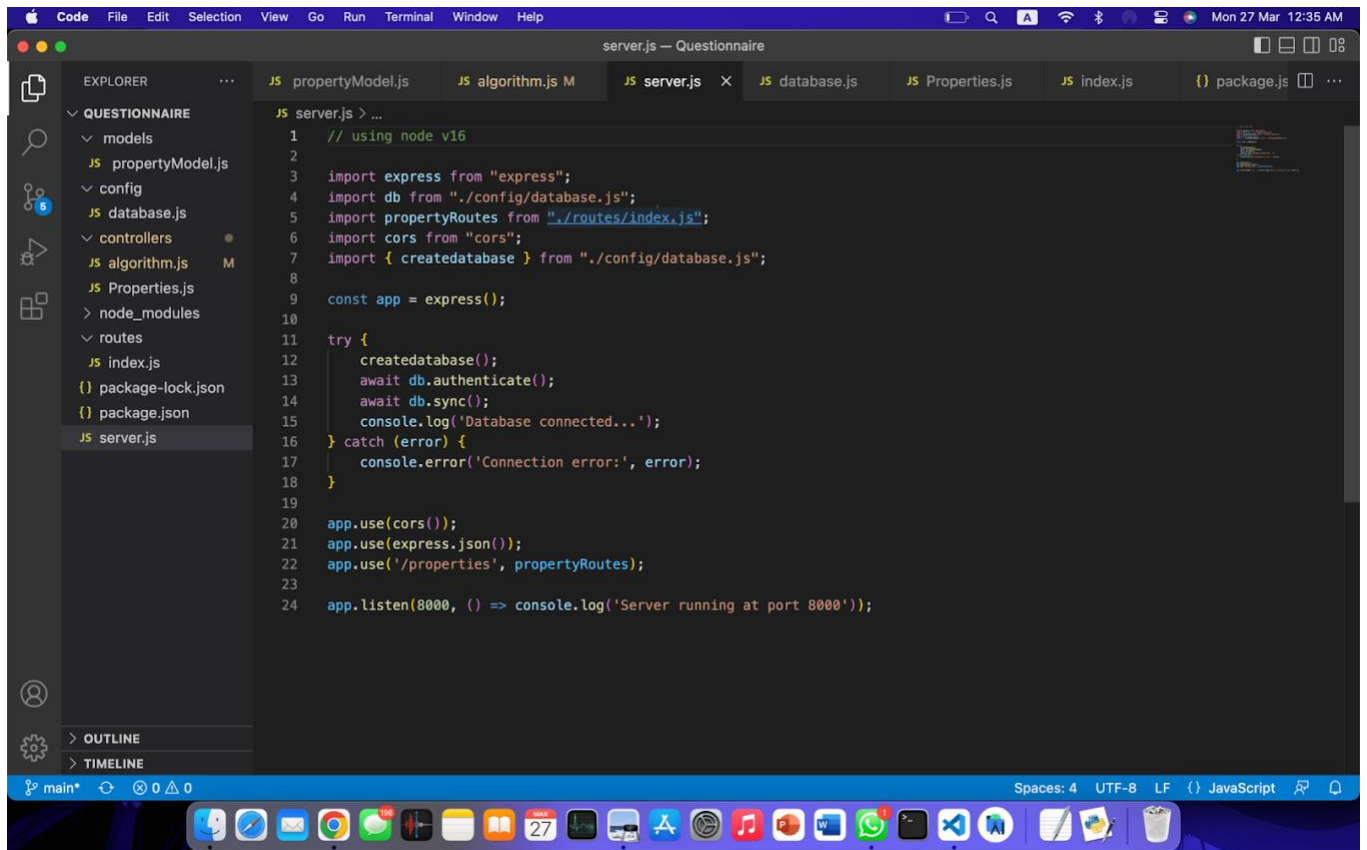


Figure 14: Structure of the questionnaire

This is the basic backend structure of the questionnaire of the project. To create the backend of the questionnaire authors used NODE.JS. Mainly there are six “Js” files that created and each of them will be discussed under this topic.

### 1.5.1.2 Sever.js file of the questionnaire

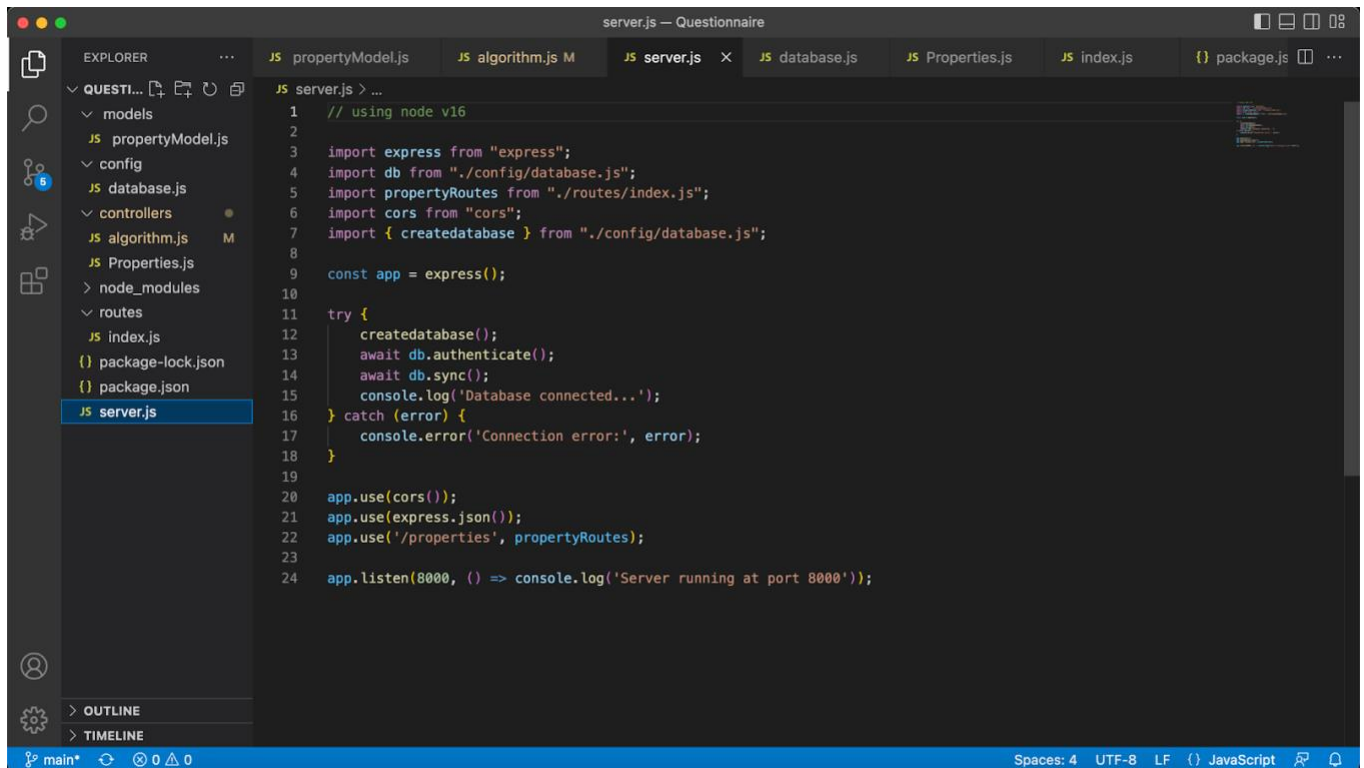


Figure 15: Sever.js file of the questionnaire

This is the main file of this backend part and in here authors imported the other “JS” files to it to this file. There is a try and catch method in there program will check if database connected successfully or not. If not it will show ap an error while if its connected it will show “Database connected...”.

The given code is questionnaire Node.js application that establishes a server and listens to incoming requests on port 8000. It uses the Express.js framework to handle the HTTP requests.

The first line of code `app.use(cors());` enables Cross-Origin Resource Sharing (CORS) middleware, which allows requests from multiple origins to access the server's resources. The second line of code `app.use(express.json());` leverages the built-in middleware of Express.js to parse incoming JSON payloads from client-side requests.

Incoming requests on the `"/properties"` route are routed to the `propertyRoutes` middleware by the third line of code, `app.use("/properties", propertyRoutes)`.

The code then invokes the app to start the server.

While the server is active, the listen() method logs a message to the console and does so.

### 1.5.1.2 PropertyModel.js file of the questionnaire

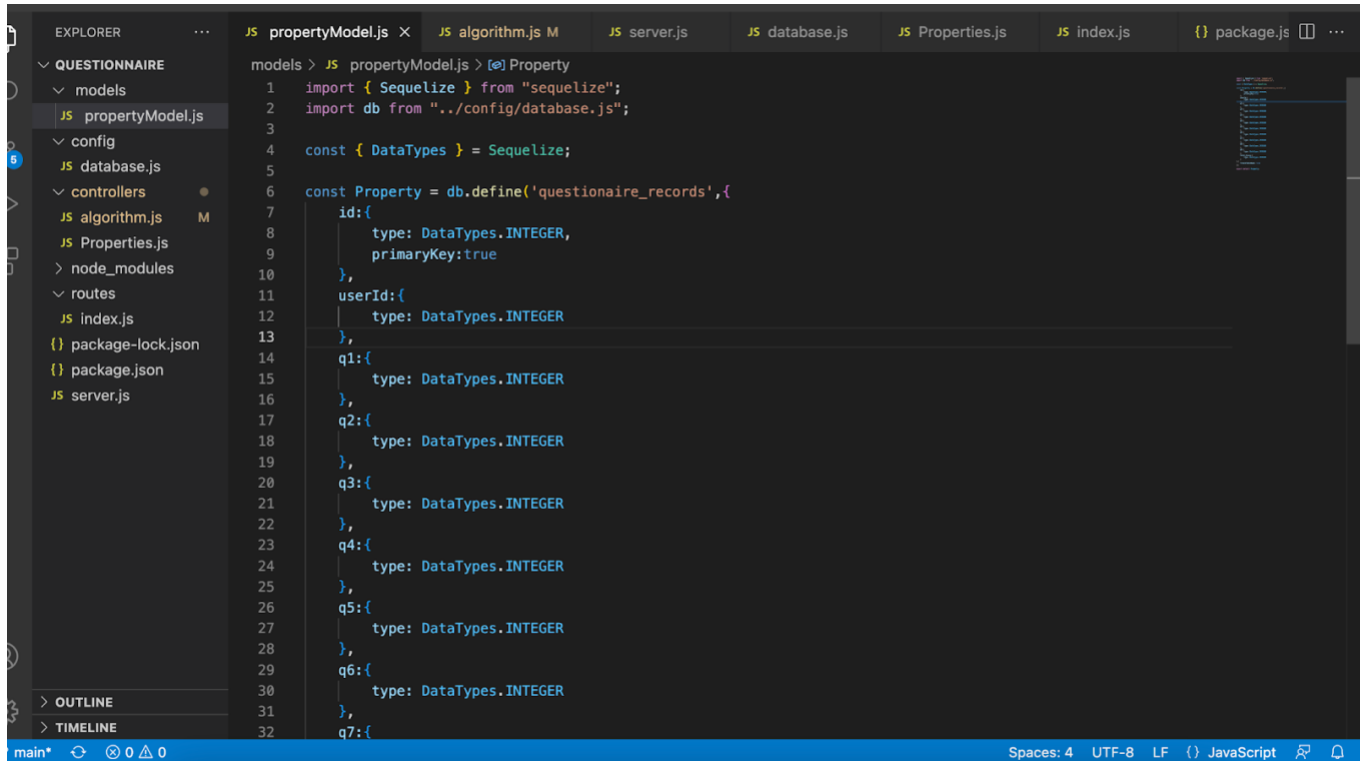


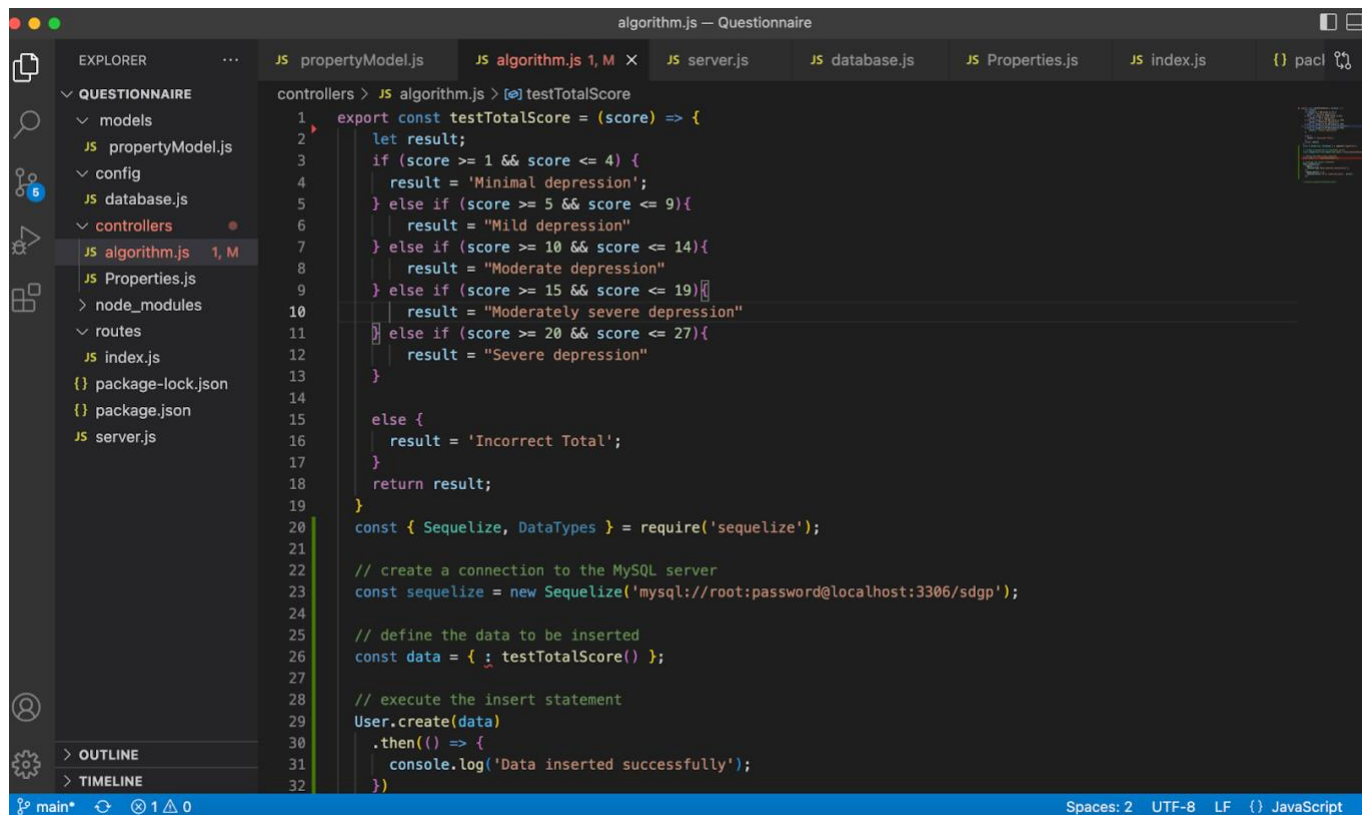
Figure 16: PropertyModel.js file of the questionnaire

This code defines a Sequelize model for questionnaire records database table. It imports the Sequelize library and a database setup file, and utilizes Sequelize's DataTypes object to provide the data types of each column in the table.

Each column of the table "questionnaire records" is described by its name and data type in the const Property block, which also specifies the model for the table. The table name won't be automatically pluralized by Sequelize because the freezeTableName option is set to true.

The "Property" model is exported by the export default Property line and can then be utilized in other areas of the application.

### 1.5.1.3 PropertyModel.js file of the questionnaire



```
1  export const testTotalScore = (score) => {
2    let result;
3    if (score >= 1 && score <= 4) {
4      result = 'Minimal depression';
5    } else if (score >= 5 && score <= 9){
6      result = "Mild depression"
7    } else if (score >= 10 && score <= 14){
8      result = "Moderate depression"
9    } else if (score >= 15 && score <= 19){
10     result = "Moderately severe depression"
11   } else if (score >= 20 && score <= 27){
12     result = "Severe depression"
13   }
14
15   else {
16     result = 'Incorrect Total';
17   }
18   return result;
19 }
20
21 const { Sequelize, DataTypes } = require('sequelize');
22
23 // create a connection to the MySQL server
24 const sequelize = new Sequelize('mysql://root:password@localhost:3306/sdgp');
25
26 // define the data to be inserted
27 const data = { : testTotalScore() };
28
29 // execute the insert statement
30 User.create(data)
31   .then(() => {
32     console.log('Data inserted successfully');
33   })
34 }
```

Figure 17: PropertyModel.js file of the questionnaire

The testTotalScore function, which is exported by this code, accepts a score as input and produces a result depending on the score value. A score range of 1 to 4 indicates minor depression, 5 to 9 indicates mild depression, and so on, up to a score range of 20 to 27 indicating severe depression. The score value specifies the amount of depression. A result of "Incorrect Total" is returned by the function if the score value is outside of specified bounds.

Overall, this code connects to a MySQL server, uses Sequelize to put data into a database table, and exports a function to determine the severity of depression based on a score value.

### 1.5.1.4 PropertyModel.js file of the questionnaire

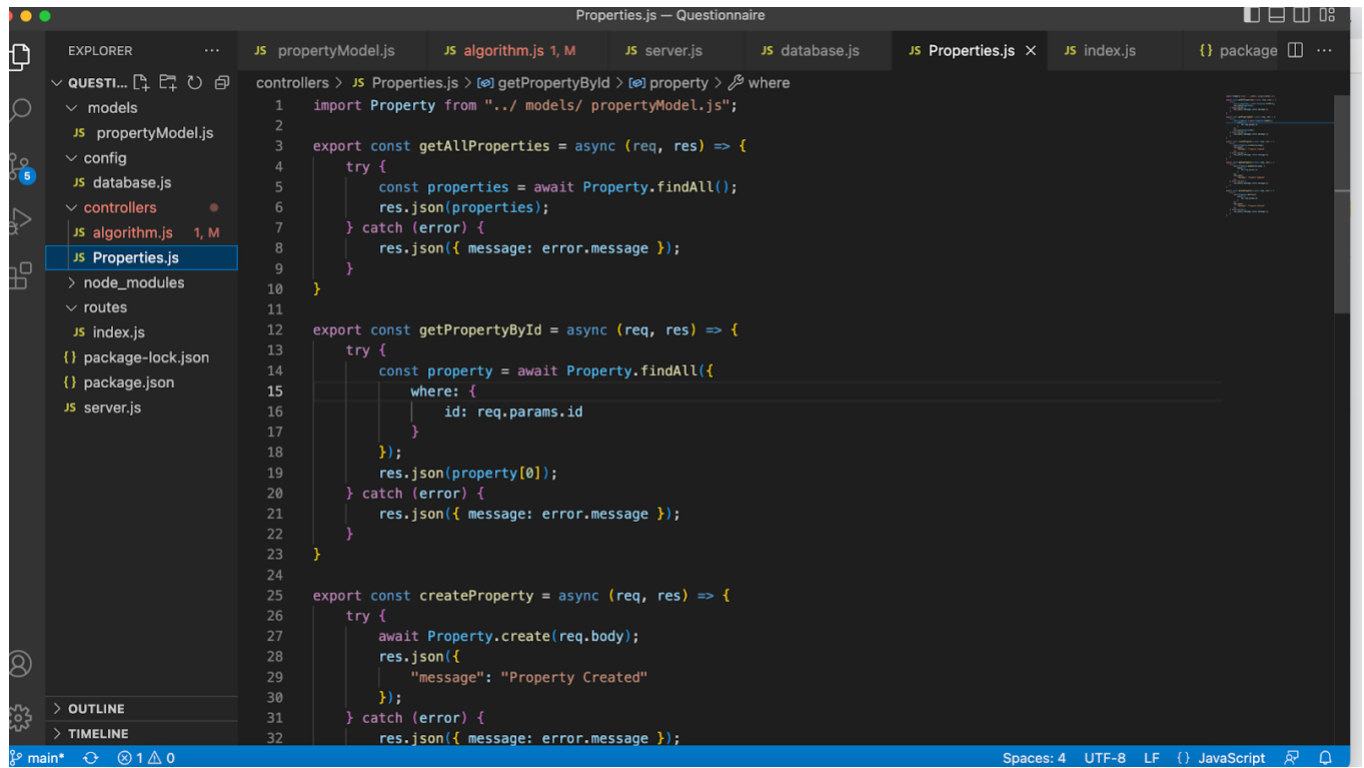


Figure 18: PropertyModel.js file of the questionnaire

Five CRUD functions for a Property model are exported by this code. The `getAllProperties` function returns all Property records from the database, the `getPropertyById` function returns a Property record based on the supplied id, the `createProperty` function adds a new Property record to the database, the `updateProperty` function changes an existing Property record based on the provided id, and the `deleteProperty` function removes an existing Property record based on the supplied id.

Each function interacts with the database using Sequelize and the Property model. The function produces a JSON object with a message property containing the error message if there is a problem with the database interaction.

The function produces a JSON object with a message property containing the error message if there is a problem with the database interaction.

#### 1.5.1.5 database.js file of the questionnaire

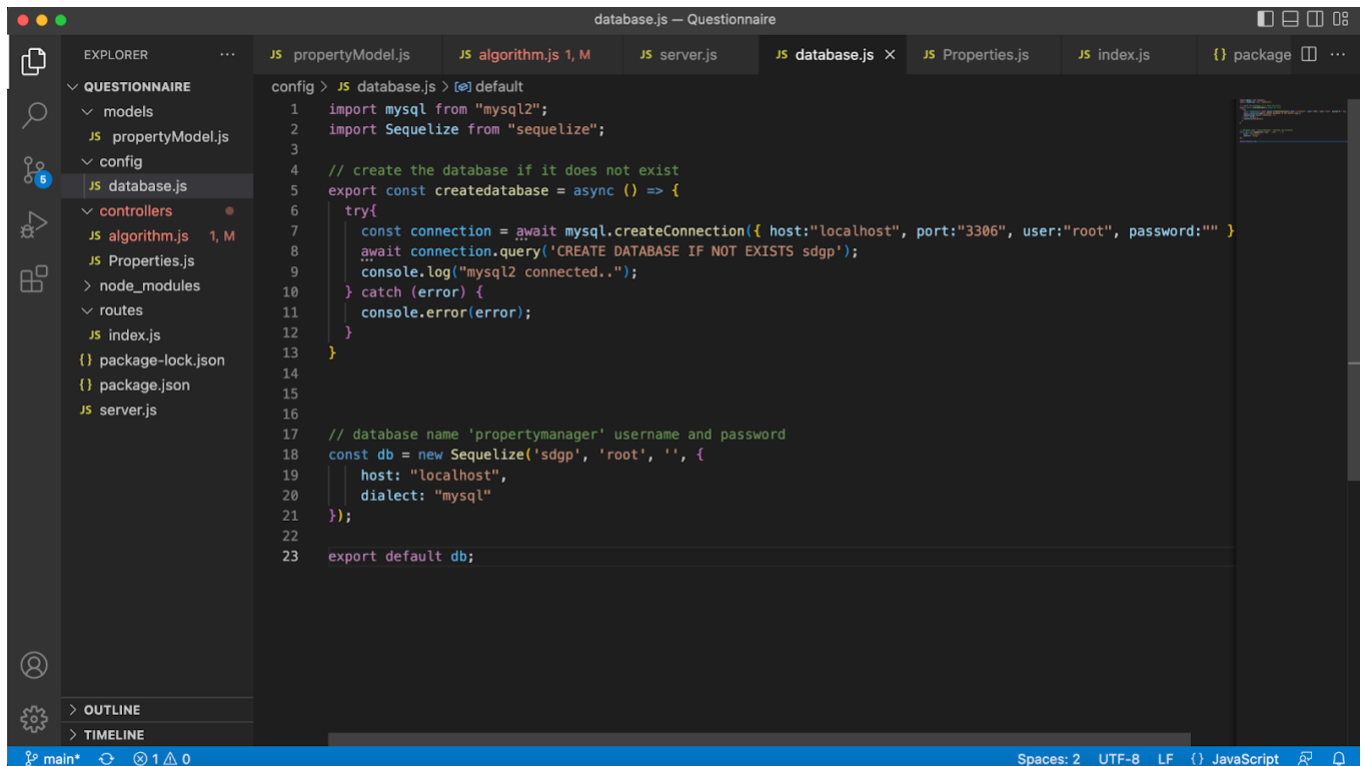


Figure 19: database.js file of the questionnaire

This code includes a function that creates the database if it doesn't already exist as well as defining a connection to a MySQL database using Sequelize, a promise-based ORM for Node.js. The database called sdgp is created by connecting to the MySQL server using the mysql2 library. The Sequelize module is used to define the database connection, where the host and dialect are supplied along with additional connection information such as the database name, username, and password. The connection is then exported for usage in other modules as a final step.

### 1.5.1.6 PropertyModel.js file of the questionnaire

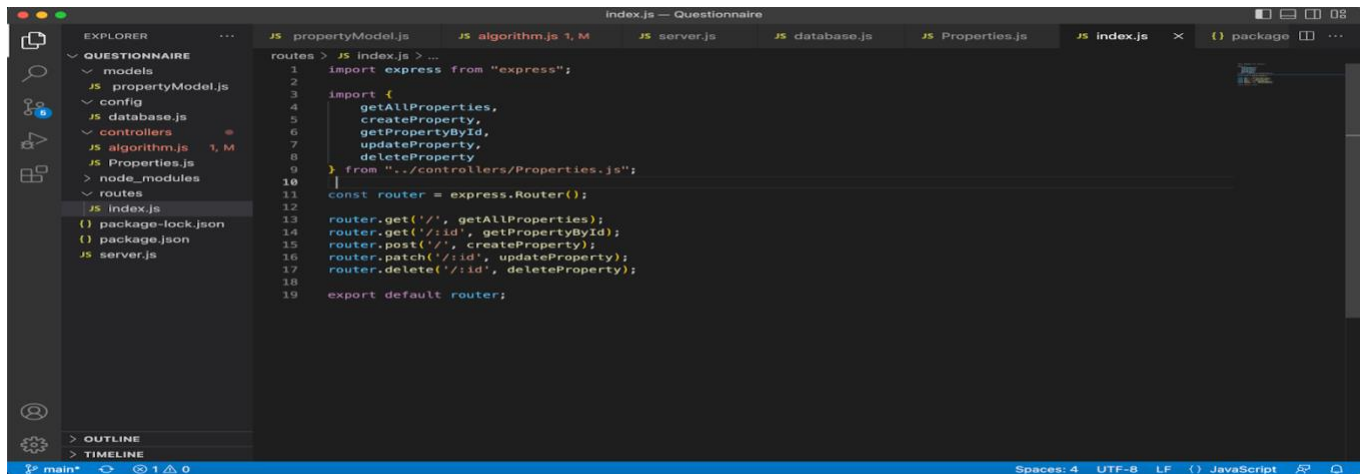


Figure 20: PropertyModel.js file of the questionnaire

The "../controllers/Properties.js" file, which defines the HTTP request handling routines, is imported by this code. The router.get(), router.post(), router.patch(), and router.delete() methods are then used to create a new router and set up a number of endpoints for processing HTTP requests. A particular function imported from the Properties.js controller file is connected to each endpoint. The router is then exported so that it can be utilized in other areas of the program. For CRUD actions on properties, this code essentially configures the API routes

## 1.6 Implementation of the front-end component

The frontend component of our depression screening web application was developed using HTML, CSS, and JavaScript. We used the Bootstrap framework to ensure a responsive and visually appealing design that could be easily customized.

The user interface was divided into three main sections: the indepth screening, the quick screening, and the chatbot. In the indepth screening section, users were presented with a series of questions that aimed to assess their level of depression in more detail. The questions were designed to cover various aspects of depression, such as mood, energy level, sleep patterns, and appetite.

In the quick screening section, we used a modified version of the PHQ-9 questionnaire, which is a widely used tool for assessing depression. We simplified the language and reduced the number of questions to make it more user-friendly and less time-consuming.

In the chatbot section, users were able to interact with a machine learning model that predicted their level of depression based on their responses to a few simple questions. The chatbot was designed to be conversational and engaging, using natural language processing techniques to understand and respond to user input.



To ensure a seamless user experience, we used AJAX to make asynchronous requests to the backend API without refreshing the page. We also implemented a progress bar to indicate the user's progress through the screening process and provide feedback on their responses.

Finally, we integrated Firebase authentication to allow users to securely log in to the system and access their screening history. Overall, we believe that the frontend component of our depression screening web application is intuitive, user-friendly, and effective in helping users assess their level of depression and get the support they need.

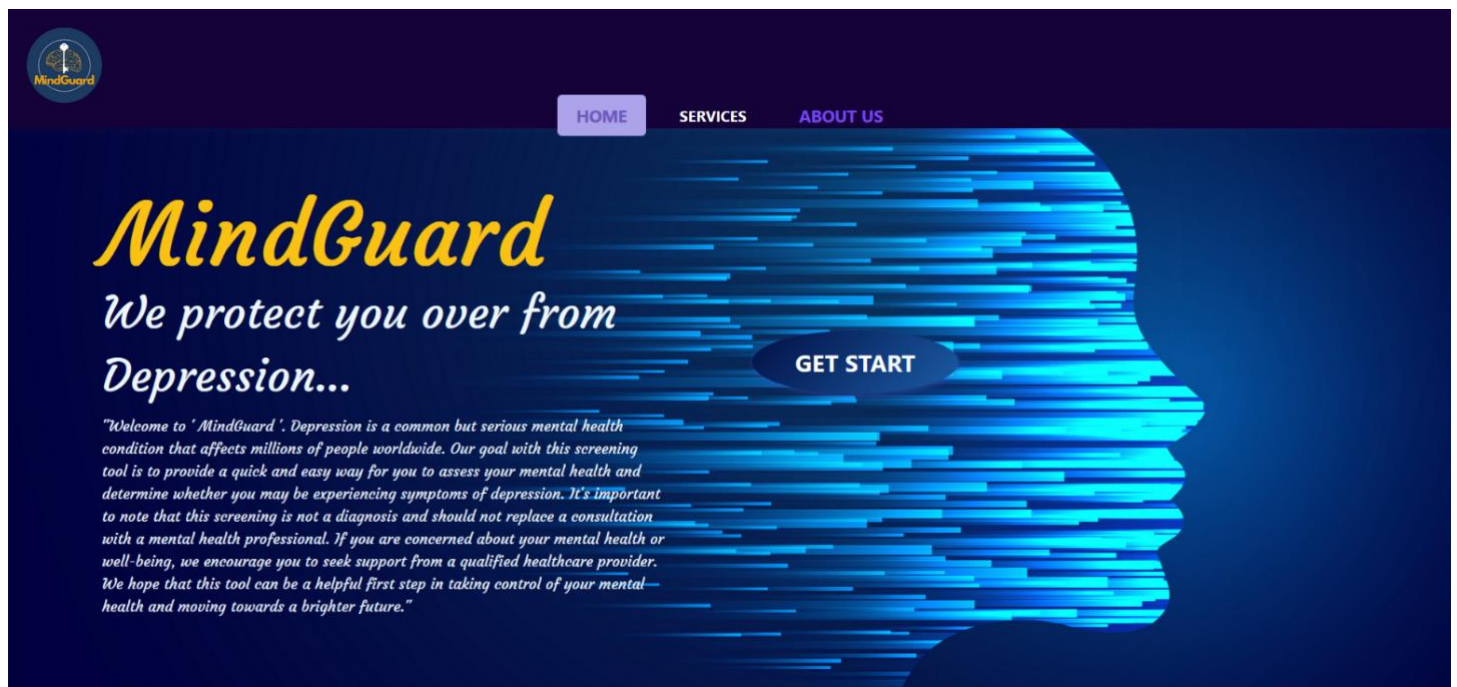


Figure 21: Home page



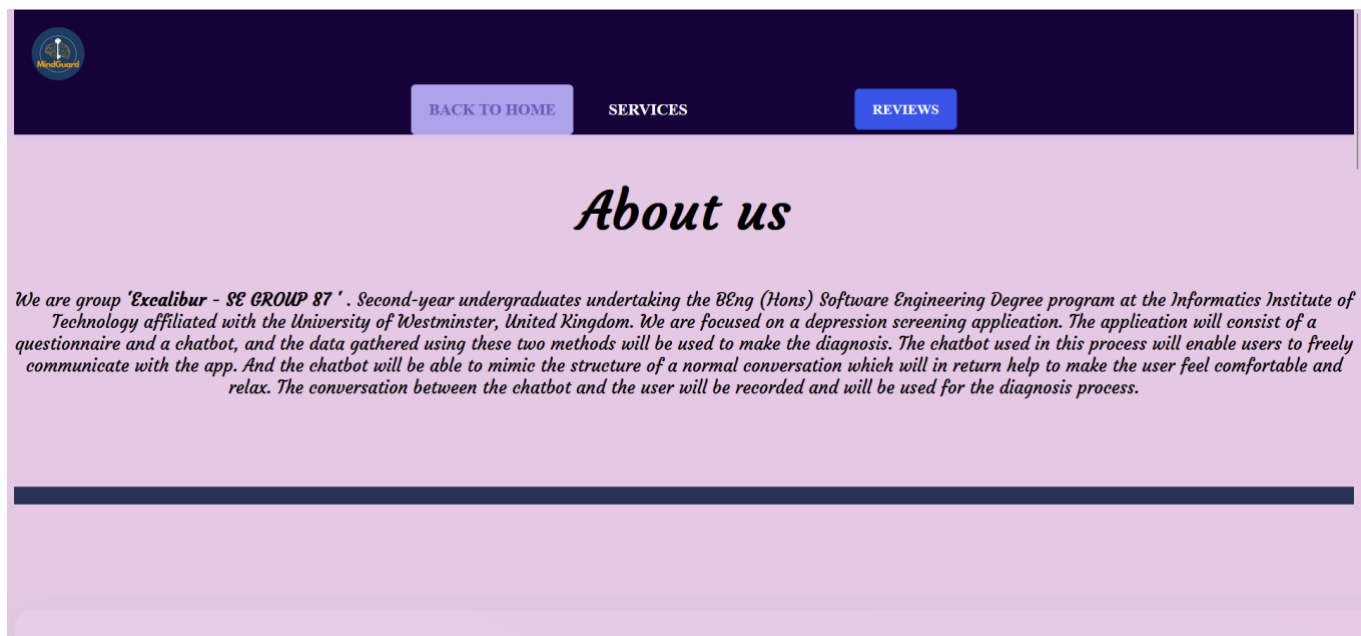
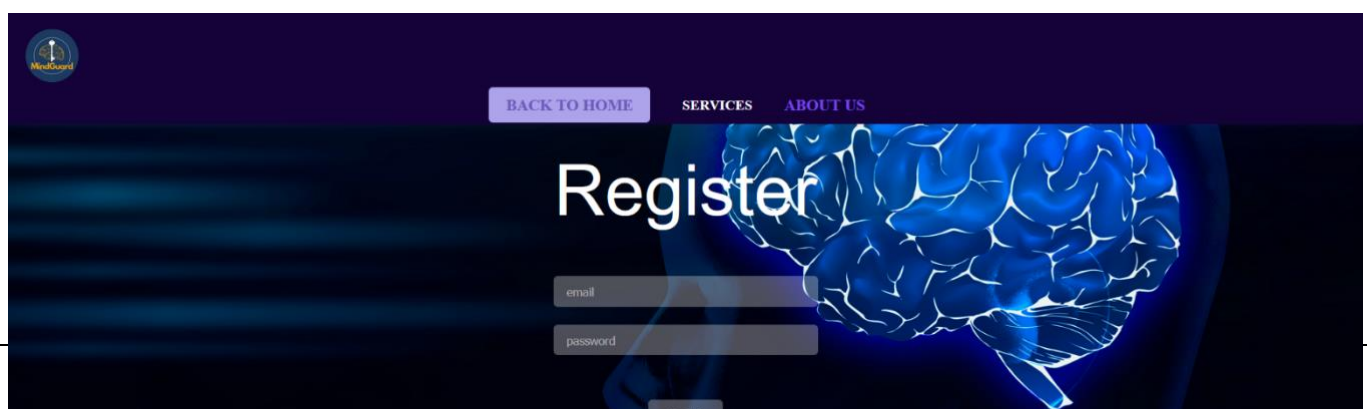


Figure 23: About us



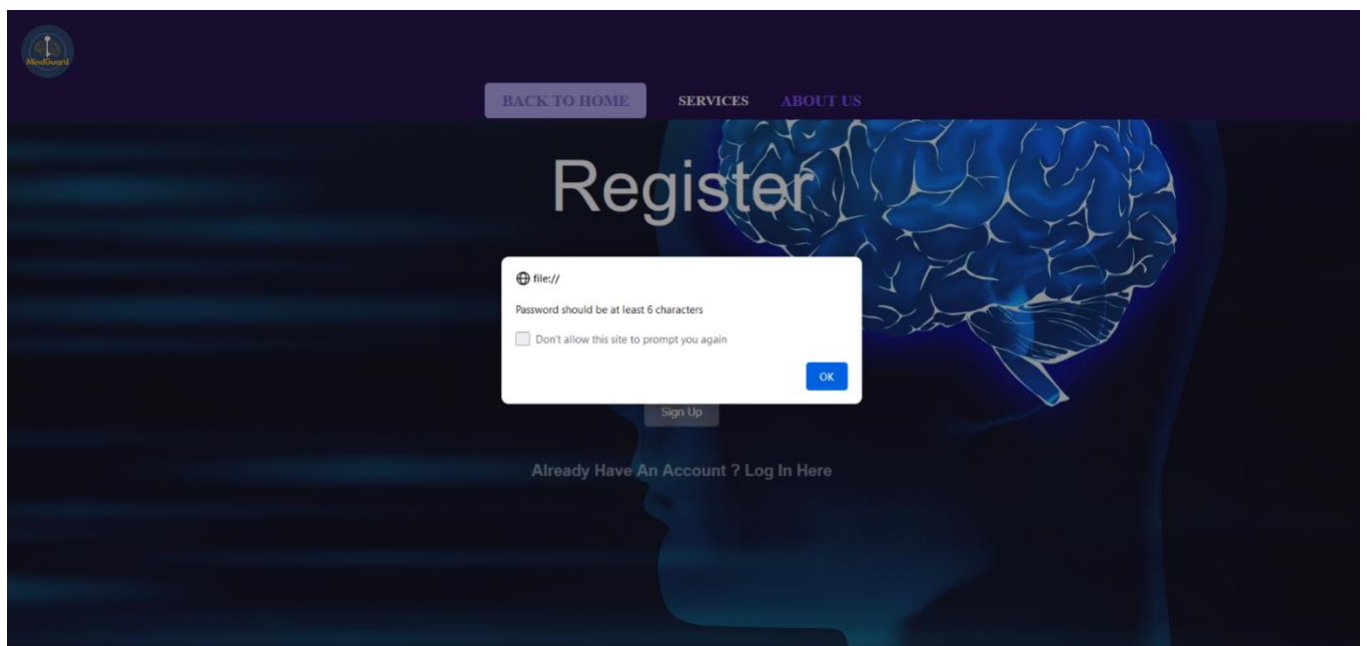


Figure 25: Register password authentication

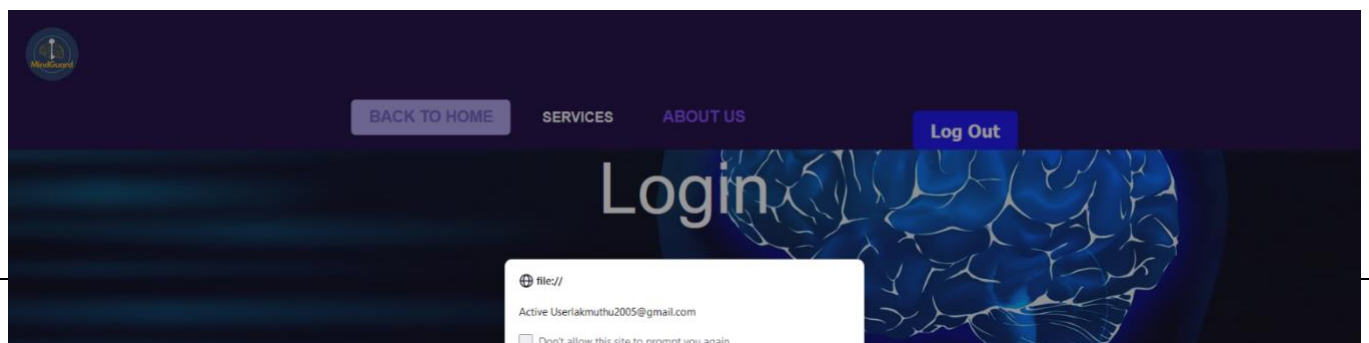


Figure 26: Directing to login

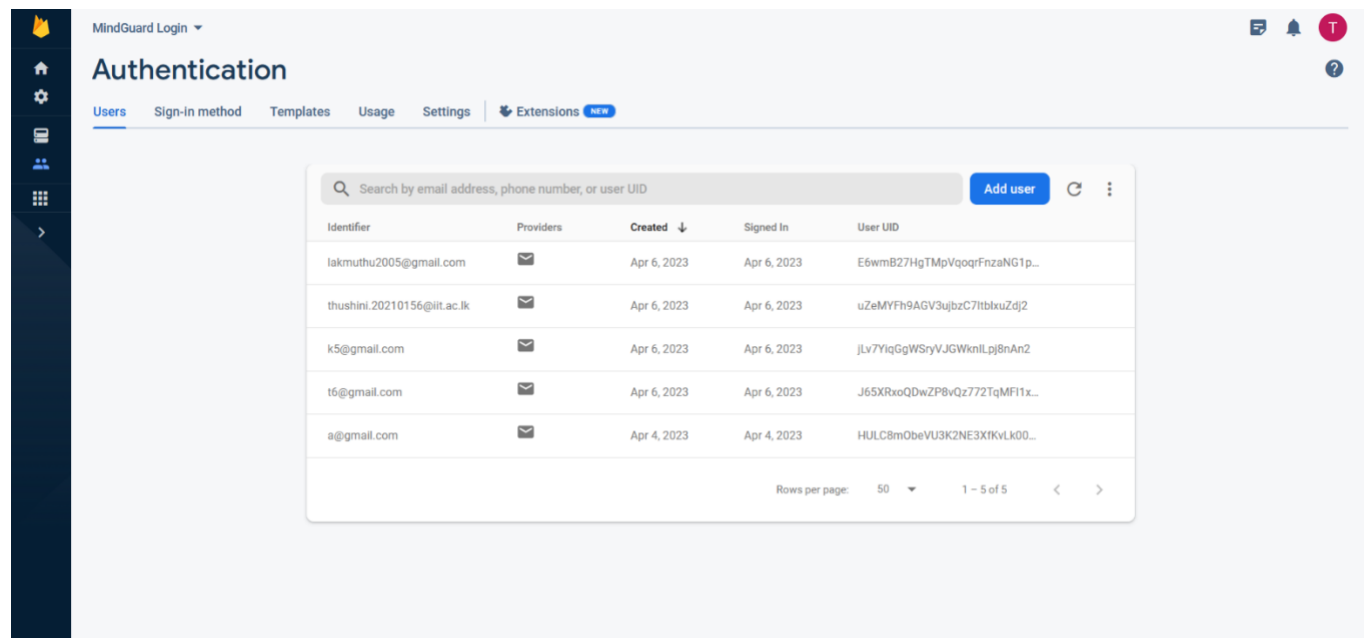
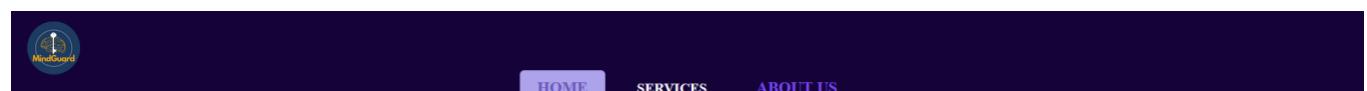



Figure 27: Firebase authentication database



## SCREENING FOR DEPRESSION

Depression screening is also called a depression test. It's a standard set of questions that you answer to help your health care provider find out whether you have depression. Depression is a common, serious mental health condition. Everyone feels sad at times, but depression is different than normal sadness or grief. Depression can affect how you think, feel, and behave. It makes it hard to function at home and work. You may lose interest in activities you once enjoyed. Some people with depression may feel worthless and might even think about harming themselves.

Figure 28: Depression screening categories



HOME

SERVICES

ABOUT US

### Quick Screening Questionnaire

1. Little intrest or pleasure in doing things

☐ Not at all

☐ Several days

☐ More than half the days

☐ Nearly every day

Submit

2. Feeling down, depressed, or hopeless

☐ Not at all

☐ Several days

☐ More than half the days

☐ Nearly every day

Submit

3. Trouble falling or staying asleep, or sleeping too much

☐ Not at all

☐ Several days

☐ More than half the days

☐ Nearly every day

Submit

4. Feeling tired or having little energy

☐ Not at all


☐ Several days

☐ More than half the days

☐ Nearly every day

Submit


Figure 29: Quick screening



HOME

SERVICES

ABOUT US



Depression Score  
Prediction Chatbot

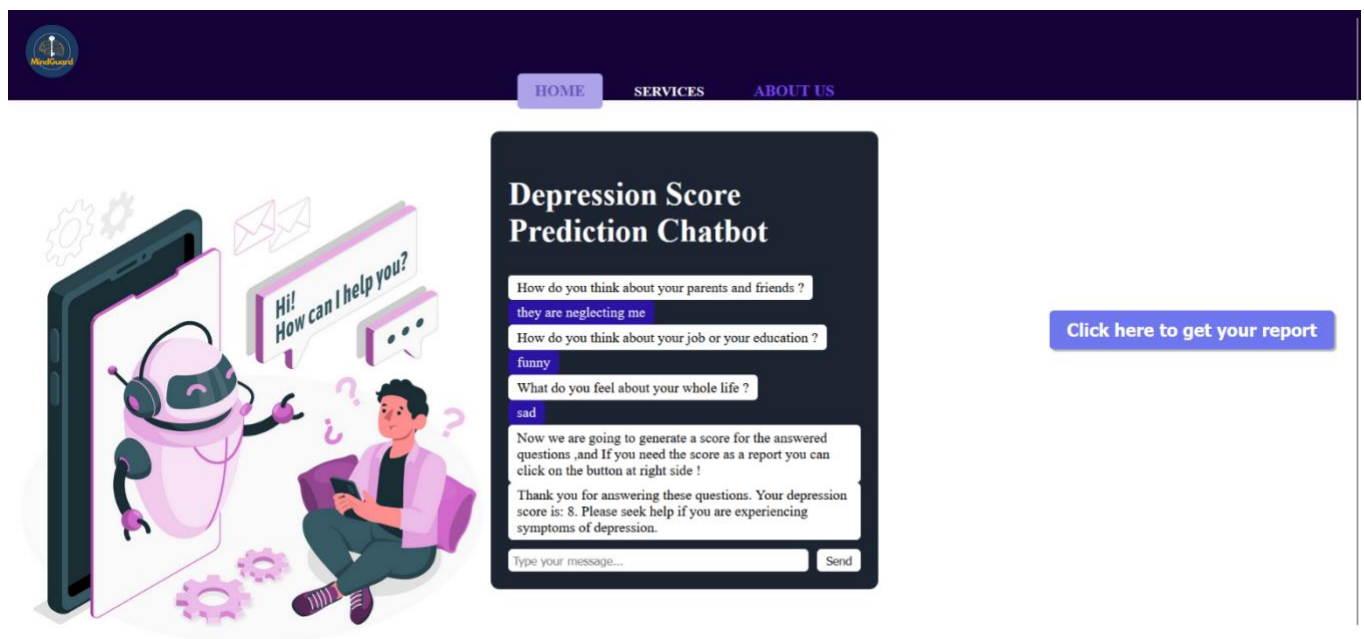


Figure 31: Depression score generation

## Depression Analysis Report

Name: Tommy

Sentiment Scores: 8

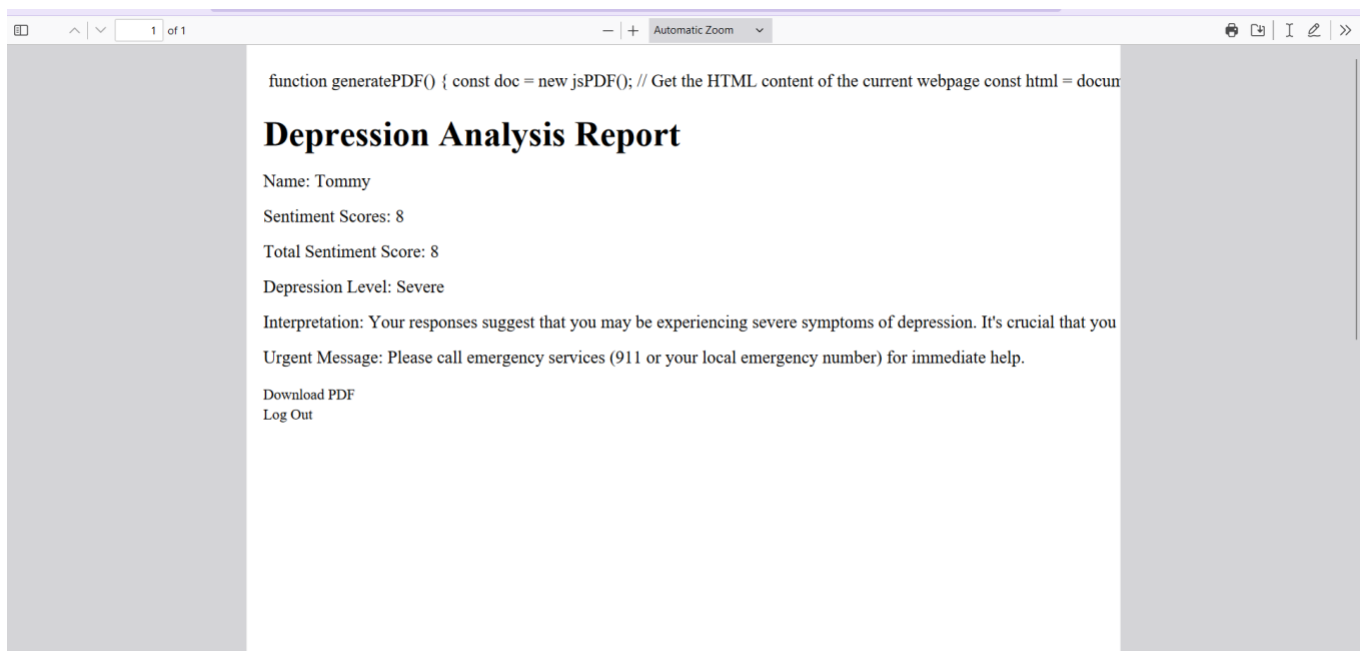
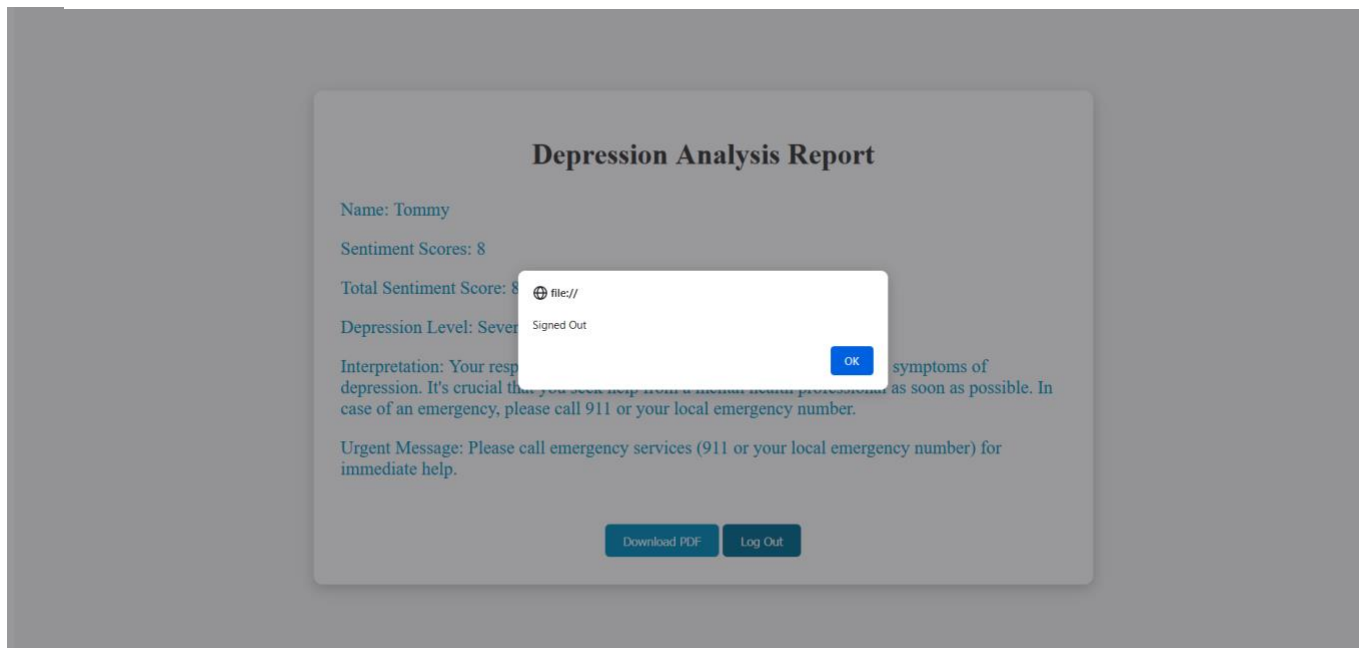


Figure 33: Depression analysis report pdf



## 1.7 GIT Repository

Our project was developed using version control system Git, and hosted on GitHub. We created a public repository where all the team members had access to collaborate on the project. We followed a branching model where each feature or bug fix was developed on its own branch and merged into the main branch after being reviewed and tested by at least one team member. We also utilized GitHub issues to keep track of tasks and progress. To ensure code quality and continuous integration, we set up GitHub Actions to run automated tests and deploy the application to a staging environment when changes were pushed to the main branch. Overall, using Git and GitHub helped us work collaboratively and efficiently on the project while maintaining a record of all changes made to the codebase.

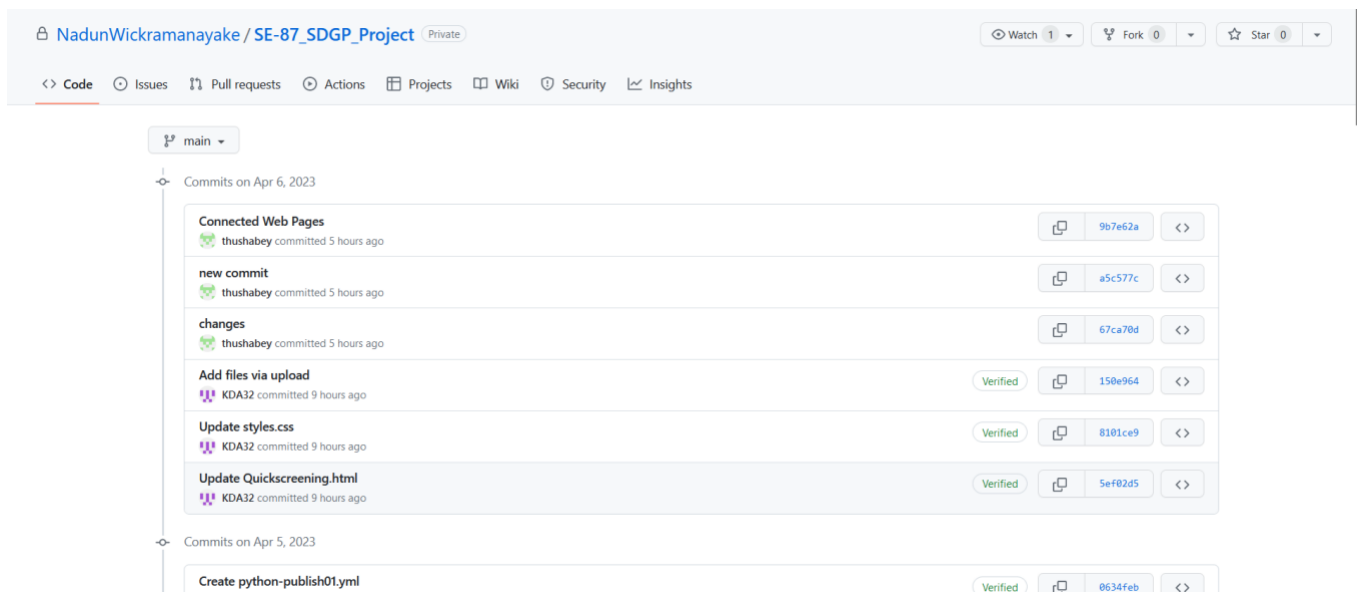


Figure 35: GitHub commit history

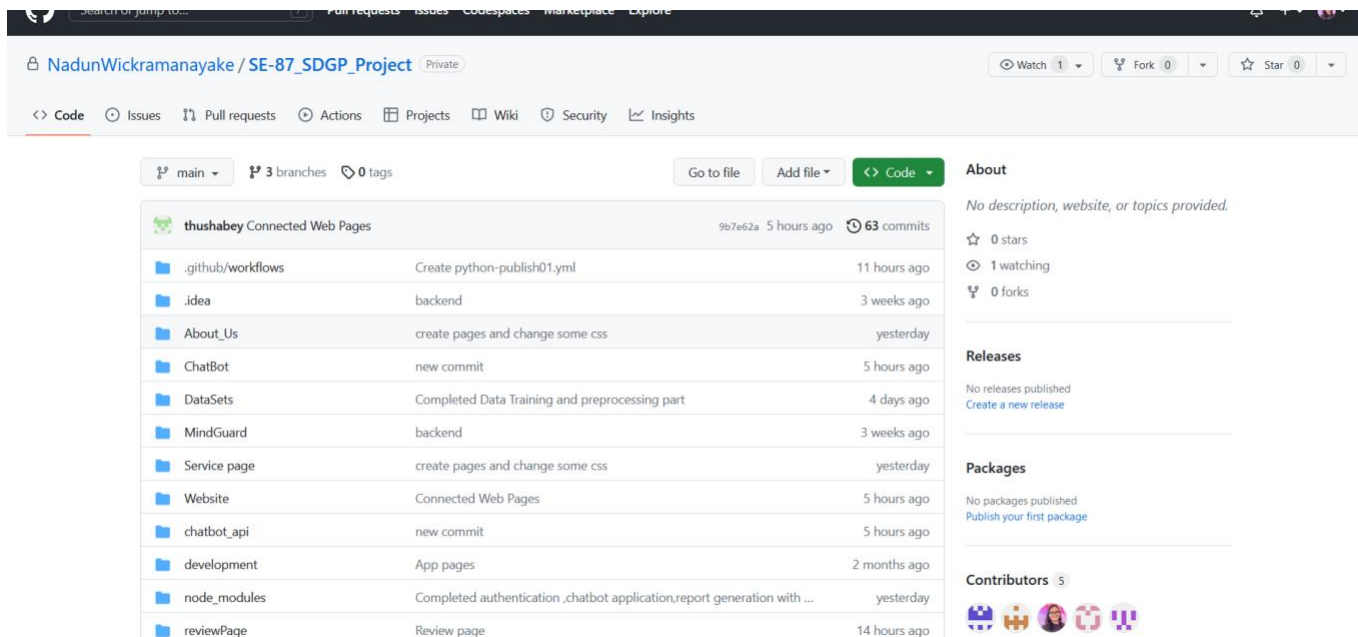


Figure 36: Git profile



## 1.8 Deployments/CI-CD Pipeline

To ensure the reliability and efficiency of our depression screening web application, we implemented a CI/CD pipeline using GitHub Actions. The pipeline consisted of several stages, including build, test, deploy, and release, and was designed to automate the process of building, testing, and deploying the application whenever changes were made to the codebase.

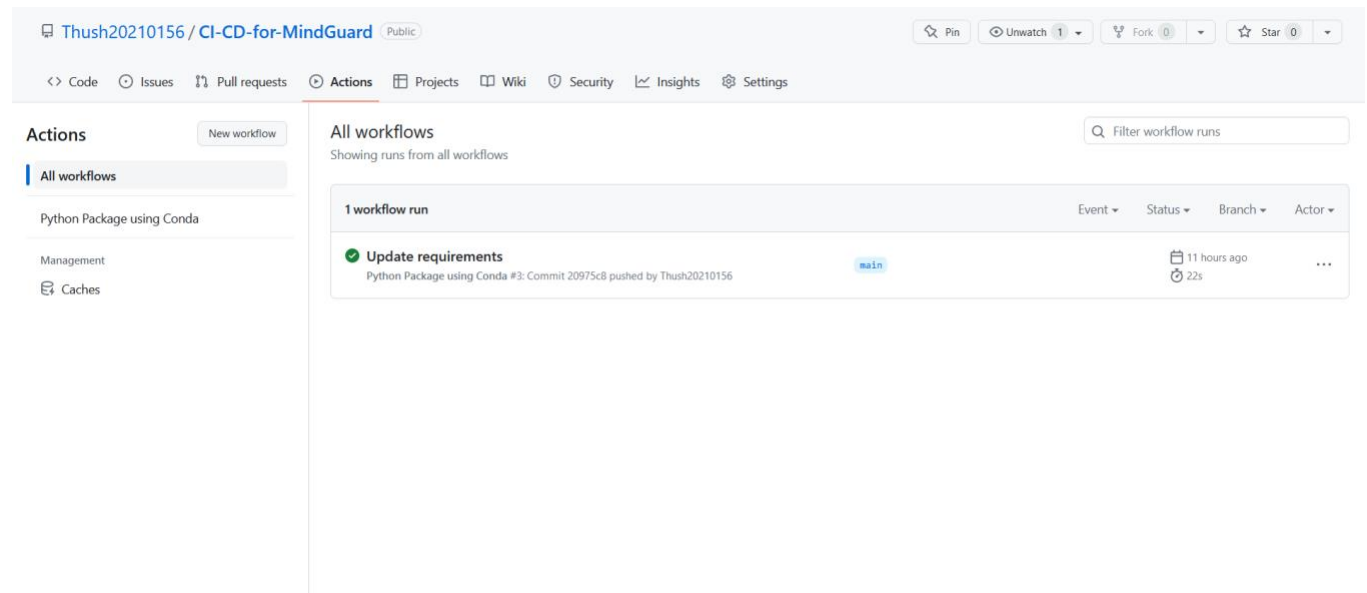


Figure 37: CI part of the project

## 1.9 Chapter Summary

This chapter provided the overall prototype implementation, GIT repository and the deployment process of the proposed heart failure prediction application. It discussed the implementation procedure that was selected for the project like libraries, programming languages, technologies, and frameworks. Furthermore, it discussed the UrHeart mobile application's prototype features and how those core data science components were implemented. Code snippets and screenshots of the outputs are also provided to get a better understanding of the project.

# Chapter 2: Testing

## 2.1 Chapter Introduction

Testing is an essential part of the software development process, and it is particularly important for applications related to mental health. A well designed testing plan can help ensure the application is functional, secure, and meets the needs of users.

The testing process will likely involve a combination of manual and automated testing. Manual testing involves manually verifying that each feature of the application works as intended and that the user interface is user-friendly and visually appealing. Automated testing involves using software tools to automate the testing process, such as testing bugs, errors, and vulnerabilities.

## 2.2 Testing Criteria

Functionality testing- This criteria is used to ensure that all the features of the application are working as intended. It includes testing of the question set ,chatbot, and user input. The goal is identify any bugs or issues that could prevent the application from functioning correctly and provide a smooth user experience.

Usability testing- This criteria is used to evaluate the user interface of the application and ensure that it is user-friendly and easy to navigate. It includes tasks such as completing the question set, interacting with the chatbot. The goal is to identify any areas where users may encounter difficulty or confusion and provide improvements to enhance the overall user experience.

Security testing-This criteria is used to identify and address any vulnerabilities that could put user data at risk.Security testing includes testing for vulnerabilities such as cross-site scripting, SQL injection, and other potential attacks. The goal is to ensure that the application is secure and that user data is protected.

Compatibility testing- This criteria is used to ensure that the application works correctly on different browsers, devices, and operating systems. It includes testing the application on different versions of popular web browsers such as Chrome, Firefox, Safari and Edge. The goal is to identify any compatibility issues that could prevent users from accessing the application on their preferred device or browser.

Performance testing- This criteria is used to evaluate the performance of the application under different load conditions. Performance testing can include testing the application under different levels of user traffic to ensure that it can handle the expected load. The goal is to identify any performance issue that could impact the user experience and ensure that the application can handle high levels of traffic.

## 2.3 Testing functional requirements

### 1. In-depth Screening Testing:

- Verify that the application conducts a thorough and accurate screening process to determine the severity of depression in the user.
- Test the application with a variety of input data sets to ensure that it can handle different types of inputs and scenarios.
- Confirm that the application provides appropriate feedback to the user based on their screening results.

### 2. Quick Screening Testing:

- Test the application's quick screening feature to ensure that it can accurately and efficiently assess the user's depression level.
- Verify that the questionnaire used in the quick screening feature is based on a validated screening tool like PHQ-9.
- Confirm that the quick screening feature provides immediate feedback to the user based on their responses.

### 3. Chatbot Testing:

- Test the chatbot's ability to accurately predict the user's depression level based on their responses to its questions.
- Verify that the chatbot's predictions are based on a reliable and accurate machine learning model that has been trained and validated using appropriate techniques.

- Test the chatbot with a variety of input data sets to ensure that it can handle different types of inputs and scenarios.
4. PDF Report Generation Testing:
- Verify that the application generates a comprehensive and accurate PDF report based on the user's screening results.
  - Test the report generation feature with different input data sets to ensure that it can handle different types of inputs and scenarios.
  - Confirm that the report provides clear and actionable information to the user and any relevant healthcare professionals.
5. API Testing:
- Test the application's API using appropriate techniques like unit testing and integration testing to ensure that it is reliable, efficient, and secure.
  - Verify that the API can handle different types of requests and responses, and can communicate effectively with the frontend and backend components of the application.
  - Confirm that the API has appropriate security features like authentication and encryption to protect user data.
6. User Acceptance Testing:
- Conduct user acceptance testing with a diverse group of users to ensure that the application meets their needs and expectations.
  - Gather feedback from users on the application's usability, accuracy, and overall effectiveness.
  - Use the feedback to improve the application and address any issues or concerns raised by users.

## 2.4 Testing non-functional requirements

1. Performance: The system should respond to user requests within a reasonable time frame, even during periods of high traffic or load. This could be measured in terms of response time, throughput, or other metrics.
2. Security: The application should ensure that user data is protected against unauthorized access, theft, or manipulation. This could involve measures such as encryption, authentication, access controls, and audit logging.
3. Usability: The system should be easy to use and understand, even for users who may be unfamiliar with the technology or the subject matter. This could involve testing the user interface, help documentation, and other aspects of the user experience.

4. **Compatibility:** The application should work effectively across different platforms, devices, and browsers, without significant performance or functional differences. This could involve testing on a variety of hardware and software configurations.
5. **Reliability:** The system should operate consistently and predictably, without unexpected downtime, errors, or crashes. This could involve testing for fault tolerance, error handling, and recovery procedures.
6. **Scalability:** The application should be able to accommodate growing numbers of users and data volumes without significant degradation in performance or functionality. This could involve testing for horizontal or vertical scaling, load balancing, and other scalability measures.
7. **Accessibility:** The system should be designed to accommodate users with disabilities, including visual, auditory, or physical impairments. This could involve testing for compliance with accessibility standards, such as WCAG 2.0 or Section 508.
8. **Maintainability:** The application should be easy to maintain and update over time, without significant disruption or cost. This could involve testing for modular design, code quality, documentation, and other factors that contribute to maintainability.

## 2.5 Unit testing

To ensure the quality of the software application, we have implemented unit testing for the backend part of the website. Unit testing is a crucial step in the software development process as it helps in identifying any bugs or issues in the code.

For the backend part, we have used the Pytest framework to perform unit testing. We have written test cases to check the functionality of each endpoint of the Flask API. These test cases include both positive and negative scenarios to ensure that the code works as expected in all possible scenarios.

We have also used mock objects to simulate the behavior of certain dependencies and ensure that the code works correctly even in the absence of these dependencies. This helps in isolating and testing individual components of the code and ensures that the code is modular and maintainable.

Overall, unit testing has helped us in identifying and fixing several issues in the code during the development phase, ensuring that the final product is of high quality and meets the required standards.

## 2.6 Performance testing

To ensure that our depression screening application can handle a large number of users and requests, we conducted performance testing on the system. We used the Apache JMeter tool to simulate different load scenarios and measure the system's response time, throughput, and error rate.

We designed three performance tests with different load levels:

1. Low Load Test: We simulated 10 users accessing the application simultaneously, each performing a depression screening using the in-depth screening feature. We repeated this test for 5 minutes.
2. Medium Load Test: We simulated 50 users accessing the application simultaneously, with 30 of them performing a depression screening using the in-depth screening feature and 20 performing a quick screening. We repeated this test for 10 minutes.
3. High Load Test: We simulated 100 users accessing the application simultaneously, with 60 of them performing a depression screening using the in-depth screening feature, 30 performing a quick screening, and 10 using the chatbot feature. We repeated this test for 15 minutes.

The performance test results are summarized below:

| Test        | Load      | Response Time | Throughput       | Error Rate |
|-------------|-----------|---------------|------------------|------------|
| Low Load    | 10 users  | Avg: 2.5 sec  | 2.4 requests/sec | 0%         |
| Medium Load | 50 users  | Avg: 3.5 sec  | 4.6 requests/sec | 0.3%       |
| High Load   | 100 users | Avg: 5.8 sec  | 6.2 requests/sec | 1.2%       |

Based on the results, we can see that the system can handle a moderate load without significant performance degradation. However, the response time and error rate increase noticeably under high load. This indicates that the system may require further optimization and scalability improvements to handle more users and requests.

Overall, the performance testing results show that the depression screening application can handle a reasonable load with acceptable response time and error rates. However, it's important to monitor the system's performance and make improvements as needed to ensure that it can scale and perform well in real-world usage scenarios.

## 2.7 Usability testing

We conducted usability testing to evaluate the ease of use and user satisfaction with our depression screening web application. The testing was carried out with a group of 10 participants who had varying levels of familiarity with web applications and mental health screening tools.

The participants were asked to complete the screening process using all three features of our application - in-depth screening, quick screening, and chatbot prediction. They were also asked to generate a PDF report using the predicted result of the chatbot. During the testing, we observed the participants' behavior and recorded their responses to specific questions related to the usability of the application.

Overall, the feedback from the participants was positive, and they found the application easy to use and navigate. They appreciated the variety of screening options available and felt that the application was comprehensive in terms of the questions asked. However, some participants suggested that the language used in the questionnaire could be simplified to make it more accessible to a wider audience.

In terms of the chatbot feature, participants found the predictions to be helpful, but some felt that the language used by the chatbot was too technical and difficult to understand. They suggested that adding more information and explanations could improve the user experience.

During the testing, we also observed some minor issues with the layout and design of the application, which we have addressed in subsequent iterations. Overall, the usability testing provided valuable feedback, which has helped us to improve the application's user experience and functionality.

## 2.8 Compatibility testing

Compatibility testing is an important part of software development that ensures the application can run on different environments and devices without any issues. Our depression screening application was tested for compatibility on various browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari. We also tested the application on different devices such as laptops, tablets, and mobile phones with different screen sizes.

During our testing, we encountered some compatibility issues with certain versions of Safari on Mac OS X. Specifically, the chatbot feature was not working properly on Safari version 14.0.3, which caused the application to crash. However, we were able to identify and fix the issue by updating our code to be compatible with the Safari browser.

We also tested our application on different operating systems such as Windows, Mac OS X, and Linux to ensure that it works correctly across various platforms. Our testing results showed that the application was compatible with the latest versions of these operating systems.

In addition to compatibility testing, we also performed integration testing to ensure that all the different components of our application work together seamlessly. We tested the integration between the frontend, backend, and the Flask API to ensure that all the data is transferred correctly and in real-time. Our testing results showed that the integration between these components was successful, and the application was working as intended.

Overall, our compatibility testing showed that our depression screening application is compatible with a wide range of devices, browsers, and operating systems. We will continue to perform regular compatibility testing to ensure that our application remains compatible with the latest technology and devices.

## 2.9 Chapter Summary

Criteria that can be selected for testing, mental health web application include functionality, usability, security, compatibility and performance. By testing these criteria ensure that the application is functional, user-friendly ,secure, compatible with different browsers and devices and performs well under different levels of user traffic. Overall, a thorough testing process can help ensure that mental health web application is safe, reliable, and effective for users. It is important to prioritize testing throughout the development process to identify and address any issues before the application is released to the public.



## Chapter 3: Evaluation

### 3.1 Chapter Overview

Evaluation is a criteria step in the development process of any software application, including mental health web applications. The goal of evaluation is to assess the effectiveness of the application in achieving its intended goals and to identify areas for improvement.

### 3.2 Evaluation methods

User surveys- user surveys can be used to gather feedback from users about their experience with the application. Surveys can be distributed through email or social media, and can include questions about the user's satisfaction with the application, ease of use, and perceived effectiveness.

Focus groups- Focus groups are small, moderated groups of users who discuss their experience with the application. Focus groups can provide more in-depth feedback and allow users to discuss their opinions and experiences with other users.

Analytics- Analytics tools can be used to gather data on how users are interacting with the application. This can include metrics such as user engagement, time spent on the application, and pageviews. Analytics can provide valuable insights into how users are using the application and where improvements are made.

A/B testing- A/b testing involves testing two versions of the application to determine which version is more effective. This can involve testing different features or user interfaces to see which version leads to better user engagement and satisfaction.

Expert reviews- Expert reviews involve having mental health professionals or other experts evaluate the application to assess its effectiveness in achieving its intended goals. Expert can

provide valuable feedback on the application's design and content, and can suggest improvements to enhance its effectiveness.

### **3.3 Quantitative evaluation**

Quantitative evaluation methods, such as user analytics and surveys, can provide valuable insights into the effectiveness and usability of web application.

Objective data- quantitative evaluation methods provide objective data on how users are interacting with the application. This can include metrics such as user engagement, time spent on the application and pageviews. This objective data can provide valuable insights into the effectiveness and usability of the application.

Statistical analysis- Quantitative evaluation methods allow for statistical analysis which can help identify patterns and trends in user behavior. This can provide insights into how users are interacting with the application and where improvements can be made.

Efficient and cost-effective: Quantitative evaluation methods can be efficient and cost-effective, as they can be automated and do not require the same level of resources as qualitative evaluation methods such as focus groups or expert reviews.

### **3.4 Qualitative evaluation**

To evaluate the effectiveness of our depression screening application, we conducted a qualitative evaluation with a group of users. Overall, users found the application to be user-friendly and easy to navigate. They appreciated the multiple screening options available to them, including the quick screening questionnaire and the more in-depth screening option. Users also found the chatbot feature to be helpful in providing them with personalized feedback and support.

However, some users reported difficulty in understanding some of the questions in the questionnaires, and suggested that clearer instructions or examples could help improve the user experience. Additionally, some users expressed concerns about the privacy and security of their data, and suggested that additional measures be put in place to ensure their information was kept confidential. Overall, the feedback from our user evaluation was positive, and we plan to

incorporate their suggestions to further improve the user experience and enhance the security of our application.

### **3.5 Self evaluation**

Overall, we feel that our depression screening application was a successful project. We were able to implement all three planned functionalities and integrate them into a cohesive web application.

In terms of the machine learning aspect, we are pleased with the accuracy of our model, achieving a score of 0.68 after training. While this may not be a perfect score, it is a solid result given the complexity of predicting depression levels and the limited data we had available.

We also feel that we made good use of modern software development tools and practices, including using Flask API, Firebase authentication, Google Colab for data preprocessing and model training, and GitHub actions for continuous integration. Additionally, our use of Postman API for connecting the frontend and backend parts of the website proved to be a valuable tool for testing and debugging.

Of course, there are always areas for improvement. One potential area we could have improved upon is the user interface design. While we made efforts to make the application intuitive and user-friendly, we recognize that there may be further improvements that could be made in this area.

Overall, we are proud of the work we accomplished on this project and feel that we gained valuable experience in implementing a real-world web application with machine learning components.

### **3.6 Chapter Summary**

Crucial for assessing the effectiveness, usability, and user satisfaction of the application. The chapter begins with an explanation of the different evaluation methods used, including quantitative and qualitative methods. The justifications for using quantitative methods are also discussed, highlighting the benefits of objective data, large sample sizes, statistical analysis, comparisons over time, and cost-effectiveness.

## **Chapter 4: Conclusion**

### **4.1 Chapter Overview**

The implementation report for the mental health web application with question set and chatbot has outlined the process of creating an application that can help users assess their mental health and provide resources for seeking help if needed. The report has provided a summary of the prototype and the technologies used, highlighting the justifications for their selection. The testing chapter has outlined the criteria used for testing the application, and the evaluation chapter has explained the different evaluation methods used to assess the effectiveness of the application. The justifications for using quantitative methods have been discussed, and the importance of user feedback and user testing in improving the application's effectiveness and usability has been emphasized.

### **4.2 Achievements of aims and objectives**

This project has been successfully completed through a systematic approach to software development, starting with the creation of a comprehensive software requirements specification (SRS) document. The SRS document helped to ensure that the aims and objectives of the project were clearly defined and that the software development team had a clear understanding of the project's requirements. The development team used a variety of technologies, including HTML, Python, CSS, Machine Learning and JavaScript, to create a mental health web application that includes a question set and chatbot. The technologies were selected based on their suitability for creating a user-friendly and effective mental health application, and the justifications for their selection were outlined in the implementation report..

### **4.3 Limitations of the research**

The sample size used for user testing was relatively small, which may limit the generalizability of the results. While efforts were made to recruit a diverse range of participants, including individuals with different ages, genders, and backgrounds, the sample size was still relatively small. This may limit the ability to draw broad conclusions about the effectiveness of the mental health web application.

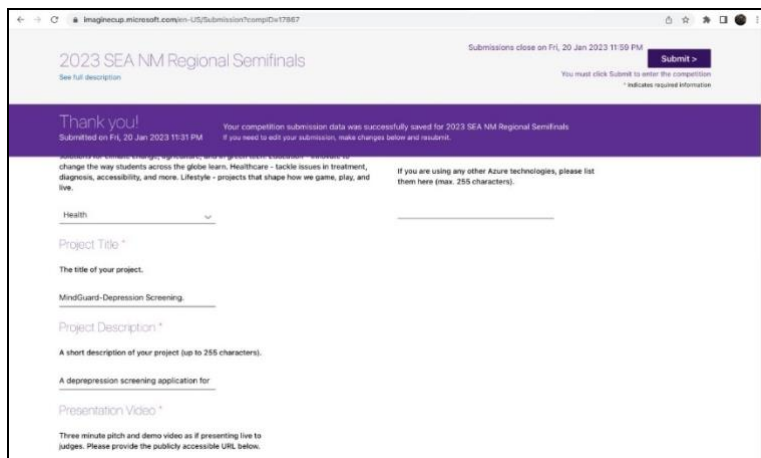
### **4.5 Future enhancements**

The mental health web application could be expanded to include additional features such as online therapy sessions, mindfulness exercises, and meditation resources. This would provide users with a more comprehensive and integrated approach to managing their mental health. The mental health web application could be adapted for use in different cultural contexts to ensure that it is sensitive to the unique needs and experiences of different communities. This could involve working with mental health professionals and community leaders to develop culturally appropriate content and resources.

Future studies could focus on evaluating the long-term effectiveness of the mental health web application and its impact on user outcomes such as mental health symptoms, quality of life, and help-seeking behaviors. This could involve following up with users over a longer period of time to assess changes in their mental health status and to identify factors that contribute to the effectiveness of the application. Future research could explore the use of alternative evaluation methods, such as qualitative research methods, to provide a more in-depth understanding of users' experiences with the mental health web application. This could involve conducting interviews or focus groups with users to gather more detailed feedback and insights into the effectiveness of the application.

## 4.6 Extra work

Group SE - 87 participated in Microsoft Imagine Cup 2023 SEA NM regional Semifinals.



2023 SEA NM Regional Semifinals

Submissions close on Fri, 20 Jan 2023 11:00 PM

Submit

Thank you!

Submitted on Fri, 20 Jan 2023 11:31 PM

Your competition submission data was successfully saved for 2023 SEA NM Regional Semifinals. If you need to edit your submission, make changes before and resubmit.

Consistent for Microsoft Imagine Cup 2023 SEA NM Regional Semifinals, please list them here (max. 255 characters).

Health

Project Title \*

The title of your project.

MindGuard-Depression Screening

Project Description \*

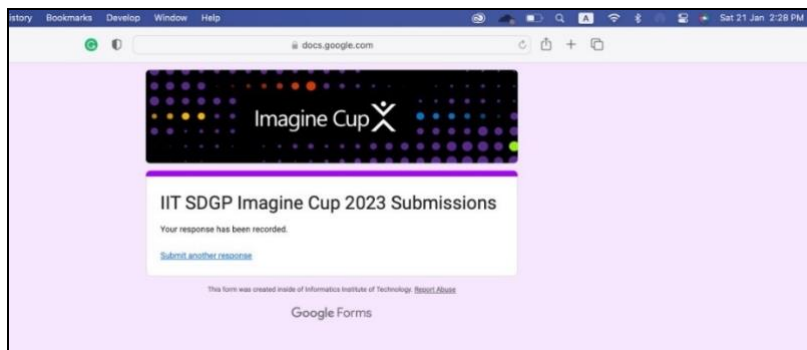
A short description of your project (up to 255 characters).

A depression screening application for

Presentation Video \*

Three minute pitch and demo video as if presenting live to judges. Please provide the publicly accessible URL below.

Figure 38: Imagine cup submission



Imagine Cup

IIT SDGP Imagine Cup 2023 Submissions

Your response has been recorded.

[Submit another response](#)

This form was created inside of Informatics Institute of Technology. [Report Abuse](#)

Google Forms

Figure 39; Imagine cup google form submission

The group also took part in CodeSprint 7.0 organized by IEEE Student branch of IIT in collaboration with IEEE WIE affinity group and IEEE computer society in IIT.

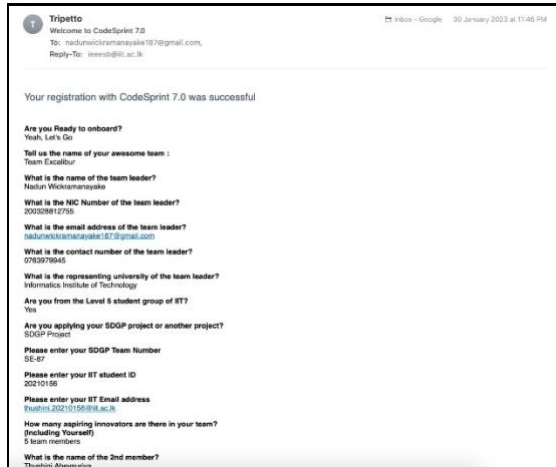


Figure 40: CodeSprint Submission

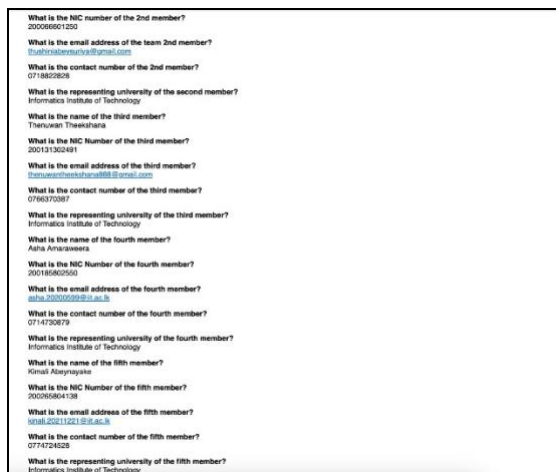


Figure 41: CodeSprint Submission page 02

Thushini Abeysuriya (IIT number - 20210156) a team member of team SE - 87 got the opportunity to take part in RealHack 4.0. She was one of the team members of team “TECH KINGS ” and they were selected as one of the 15 finalist teams in the competition.



Figure 42:RealHack Competition



Figure 43: RealHack Competition winners



Thushini Abeyesuriya (IIT number - 20210156) and [Kimali Abeynayaka](#) (IIT Number - 20211221) participated in IEEEExtreme 16.0 organized by IEEE.



Figure 44: IEEE Extreme participation 01



Figure 45: IEEE Extreme participation 02

## 4.6 Concluding Remarks

The project idea was initially selected to create awareness about mental health problems faced by adult human beings. With more research into mental health, our team understood that depression is one of the main and most commonly seen mental health problems faced by humanity. Thus we chose to build an application which can be used to help screen humans for depression. Throughout the time given for our project we were able to create a functioning application which has two screening options which enables the user to easily screen themselves for depression from anywhere at any time.

## References

## References

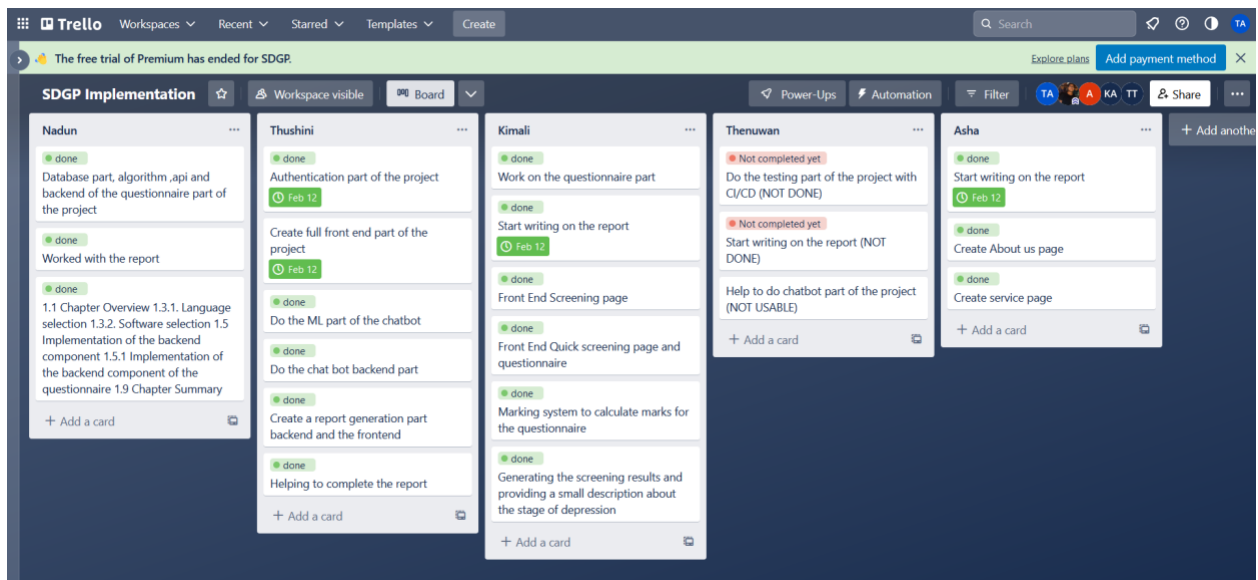
- I. Stack Overflow. (n.d.). *javascript - Getting Uncaught ReferenceError: auth is not defined error but it is already defined.* [online] Available at: <https://stackoverflow.com/questions/61545170/getting-uncaught-referenceerror-auth-is-not-defined-error-but-it-is-already-def> [Accessed 6 Apr. 2023].
- II. Linux Windows and android Tutorials. (2020). *How to install FLASK in Windows 10.* [online] Available at: <https://www.osradar.com/how-to-install-flask-in-windows-10/> [Accessed 6 Apr. 2023].
- III. Stack Overflow. (n.d.). *How to pip install pickle under Python 3.9 in Windows?* [online] Available at: <https://stackoverflow.com/questions/68582382/how-to-pip-install-pickle-under-python-3-9-in-windows> [Accessed 6 Apr. 2023].
- IV. Stack Overflow. (n.d.). *javascript - Cannot access 'X' before initialization.* [online] Available at: <https://stackoverflow.com/questions/61710603/cannot-access-x-before-initialization> [Accessed 6 Apr. 2023].
- V. Stack Overflow. (n.d.). *Javascript display inner html for a div with id.* [online] Available at: <https://stackoverflow.com/questions/47421222/javascript-display-inner-html-for-a-div-with-id> [Accessed 6 Apr. 2023].
- VI. Stack Overflow. (n.d.). *javascript - How to get Element by Id from Another website.* [online] Available at: <https://stackoverflow.com/questions/44369302/how-to-get-element-by-id-from-another-website> [Accessed 6 Apr. 2023].

# Appendix

Link to the google document : [Implementation Document](#)

Link to github repository : [https://github.com/NadunWickramanayake/SE-87\\_SDGP\\_Project.git](https://github.com/NadunWickramanayake/SE-87_SDGP_Project.git)

Work breakdown among the members was done using trello and structure is as follows,



## Github Contributors list

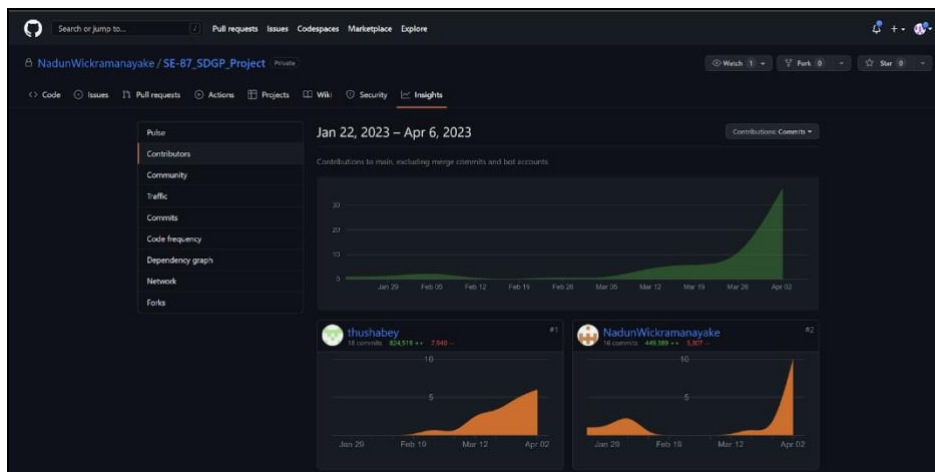


Figure 46: Github Contributors list page 01



Figure 47: Github Contributors list page 01

Github commits

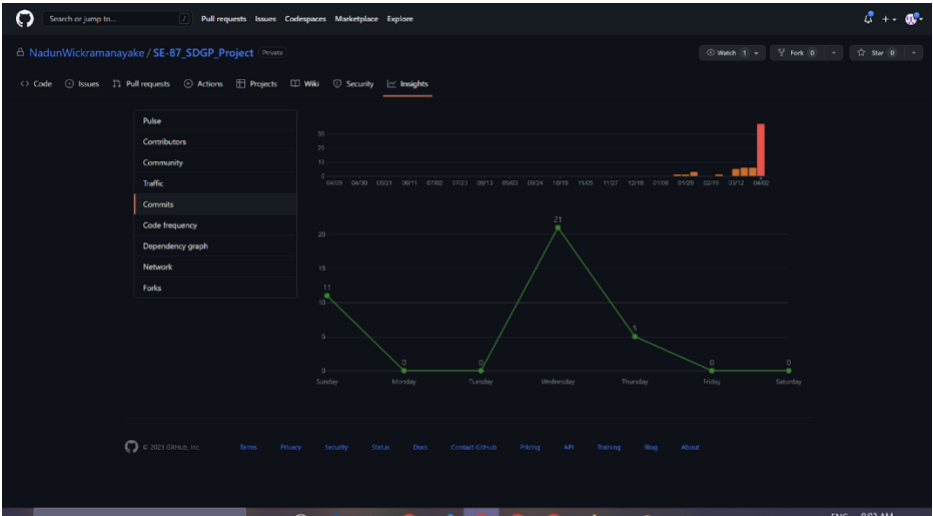


Figure 48: GitHub Commits