الجامعة المصرية للتعلم الإلكتروني
Egyptian E-Learning University

Faculty of computer & information technology

Healthcare Chatbot

By:

Ahmed Mohamed Elsayed Hassanein	**(20-00917)**
Islam Fathy Zaghloul Abd Elmaqsoud	**(20-01527)**
Kirlos Sameh Gabra Youssef	**(20-00459)**
Rana Ahmed Abu Ghareeb	**(20-01540)**
Karem Yasser Mahmoud Shehab Eldeen	**(20-00486)**
Joanna Elia Armia Attaallah	**(20-01714)**
Islam Hassan Mahmoud Abo ElHamd	**(20-00344)**

Under the Supervision of:

Prof. Khaled Wassif

Professor in Computer and Information Technology

EELU University.

T.A. Mohamed Mustafa

Teacher Assistant in Computer and Information Technology

EELU University.

Sohag 2024

Contents

Table of Figure

List of tables

Acknowledgement

First, we express our gratitude to **God** for His grace and unwavering support throughout our journey in college and the completion of our graduation project. His guidance has been the cornerstone of our success, and we are immensely thankful for His blessings.

We extend our heartfelt appreciation to **Prof. Khaled Wassif** and **Eng. Mohamed Mostafa** for their invaluable mentorship, guidance, and support throughout the development of our healthcare chatbot. Their expertise, encouragement, and dedication have been instrumental in shaping our project and ensuring its success. Their mentorship has not only enriched our academic experience but has also inspired us to strive for excellence in our endeavors.

We are also grateful for the support and encouragement provided by **colleagues** and **the broader academic community**. Their feedback and insights have been invaluable in refining our healthcare chatbot and enhancing its capabilities.

Abstract

Healthcare chatbots have emerged as a transformative technology in the healthcare industry. Powered by artificial intelligence and natural language processing, these virtual assistants offer personalized and accessible healthcare information and assistance to users. This abstract provides an **overview of healthcare chatbots, their benefits, and their potential impact on healthcare delivery.**

Healthcare chatbots are intelligent virtual assistants designed to interact with users in a conversational manner through text or voice-based interfaces. They simulate human-like conversations, understand user queries, and provide relevant information and recommendations. By integrating into digital platforms such as websites and mobile applications, healthcare chatbots offer **24/7 availability**, **enabling users to receive immediate assistance and support.**

Chapter 1

Introduction

History:

Our healthcare chatbot is a result of our commitment to using advanced technology to tackle significant healthcare issues. Here's a brief overview of the journey that has led us to this point:

1. Identification of Need: The journey acknowledges the significant gap between the growing demand for healthcare services and the limited accessibility to timely medical advice, particularly in remote areas or during non-office hours.

2. Inspiration from ML Advancements: Recent advancements in machine learning and natural language processing have shown potential in transforming industries, including healthcare, by leveraging algorithms for disease classification based on symptoms.

3. Research and Development: Research aimed to understand symptom-based disease classification methods and identify effective machine learning techniques. Collaborated with healthcare experts to develop a robust model for accurate disease classification.

4. Data Acquisition and Annotation: Research was conducted to develop a robust ML model for disease classification based on user-input symptoms, collaborating with experts in ML and healthcare.

5. Model Training and Validation: Our team trained and fine-tuned a ML model using data, ensuring accurate disease classification based on user-provided symptoms through iterative validation and refinement processes.

Through **unwavering dedication**, **collaboration**, and **innovation**, we have transformed a vision for a healthcare chatbot into a reality that stands poised to make a meaningful difference in the lives of millions.

Motivation:

Our healthcare chatbot is driven by our belief in technology's transformative power to enhance healthcare accessibility and empower individuals to make informed decisions about their well-being. Here are some key points that drive our motivation:

1. Accessibility: Our chatbot aims to provide immediate access to accurate medical information to millions worldwide, breaking down barriers like geographical location, financial constraints, and limited healthcare professionals.

2. Empowerment: Our chatbot promotes health literacy by allowing users to input symptoms and receive real-time feedback on potential diseases, encouraging proactive health management.

3. Early Detection and Prevention: Our chatbot aids in early disease detection and intervention, enabling users to identify potential health issues, seek medical attention, and adopt preventive measures.

4. Reducing Healthcare Burden: Our chatbot improves healthcare system efficiency by providing preliminary triage and guidance, allowing healthcare professionals to focus on complex cases and alleviating long wait times.

In summary, the healthcare chatbot aims to improve healthcare outcomes, promote health literacy, facilitate early detection, alleviate burden on healthcare systems, and provide empathetic support through machine learning and artificial intelligence.

Statistics:

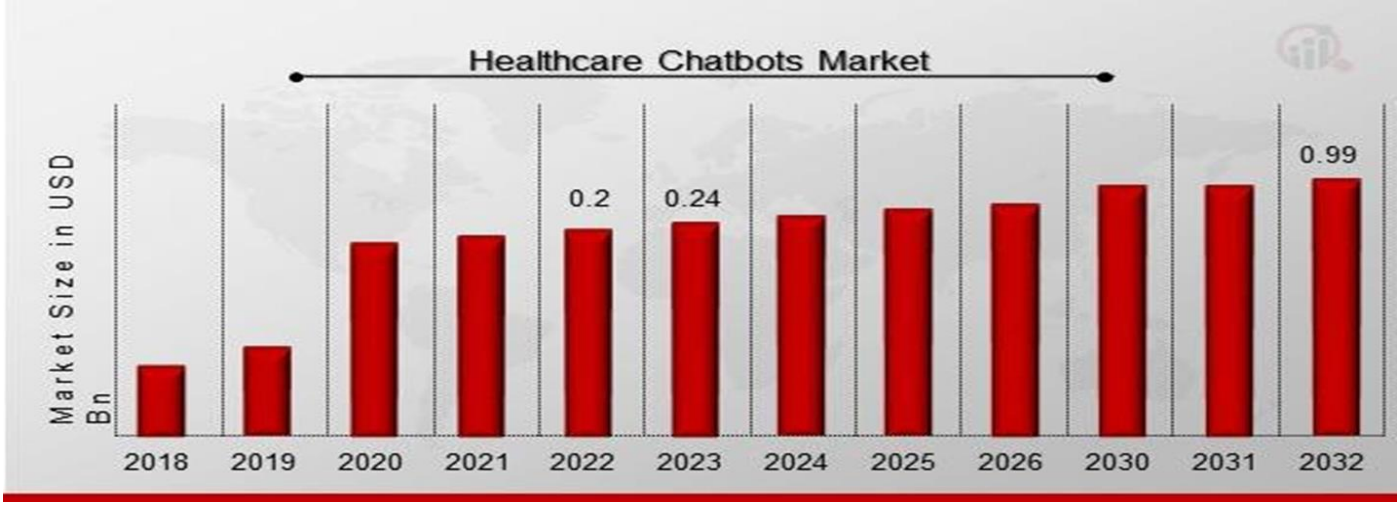


Figure 1: Chatbot Market

Healthcare Chatbots Market Size was valued at USD **0.2 billion** in **2022**. The healthcare chatbots market industry is projected to grow from USD **0.24 Billion** in **2023** to USD **0.99** billion by **2032**, exhibiting a compound annual growth rate (CAGR) of **19.5 %** during the forecast period (**2023 - 2032**).

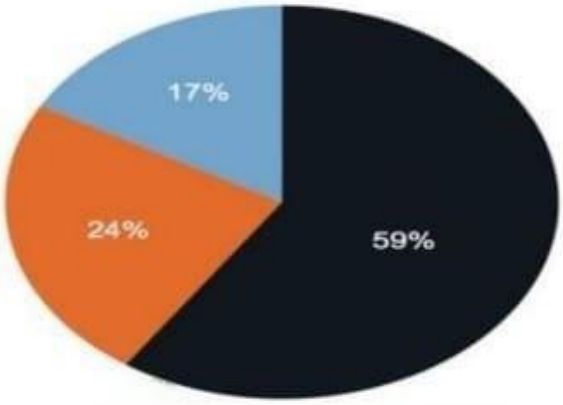


Figure 2: Trustworthiness of AI

The chart indicates that **59%** of the population supports AI's role in human assistance, while **24%** lack knowledge about AI's functions and **17%** are clueless.

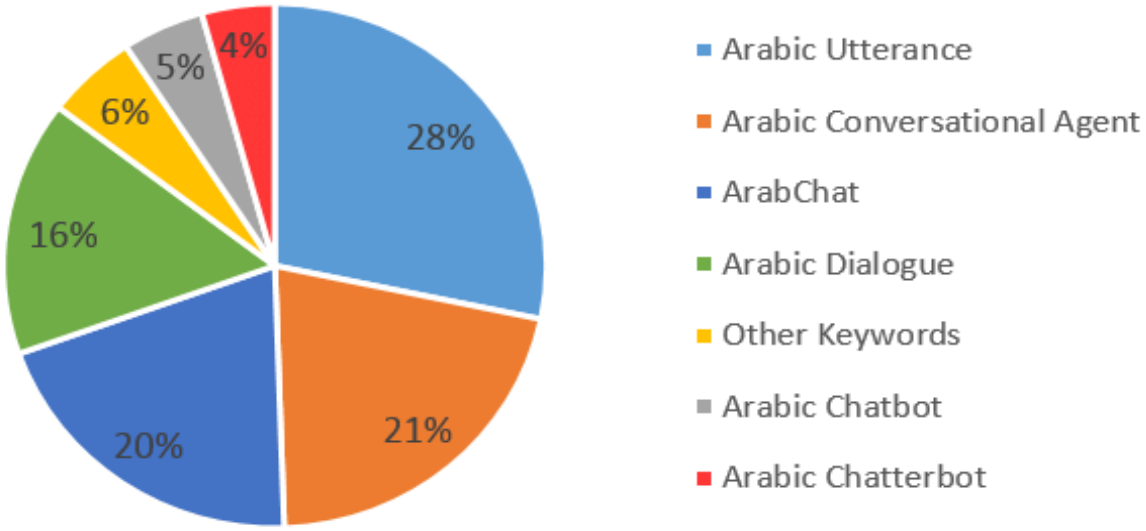


Figure 3: Search Keywords for Arabic Chatbots

- It shows the importance of chatbots that supports Arabic.

Chapter 2

Problem phases

Problem Statement:

If the patient delays finding out about the disease he has, regardless of the reasons for his delay, and does not take the necessary precautions as soon as possible, his health condition will worsen before he is able to go to the doctor**.**

Chatbot Challenges:

1- The user enters linguistic words that do not affect the understanding of symptoms, which leads to an incorrect classification of the disease.

2- Possible inaccurate classification of diseases and this may put the patient's life at risk.

3- Patients want a Suitable and Interactive GUI to easily deal with chatbot.

4- Arab users want to interact with the chatbot in Arabic.

5- The patient does not have enough knowledge of the necessary precautions and which hospitals are closest to the patient’s location.

Problem solution:

Therefore, we have developed a chatbot that is available 24/7, which classifies the disease based on a set of symptoms entered by the user.

1- In order to understand the description of the patient's symptoms,

we used NLP techniques to do the following:

- Filter the text from ineffective words and extract the words indicating the symptoms (clear the text from stop words and…)
- Store them in vector form with special number for each symptom.
- Validate the text by identifying meaningless text.

2- In order to correctly diagnose the disease and achieve the best possible accuracy from the ML model, we applied hyperparameter tuning using Grid Search method on Different ML models.

3- We created a Mobile Application, Supports Arabic and English GUI, to facilitate smooth and interactive communication with the chatbot Using:

- Dart
- Flutter
- Flask

4- Support conversation in both Arabic and English to ensure usability technique between chatbot and patient.

5- Provide patients with some precautions and a Google Map to guide them to the closest hospitals to their location.

Chapter 3

Symptom2Disease Dataset

What is “Symptom2Disease” Dataset?

The healthcare chatbot project utilizes the "Symptom2Disease Dataset," a meticulously curated dataset, which provides essential details about the disease, here are some **important details about this dataset**:

1. Size: The Symptom2Disease dataset comprises 1200 datapoints, each containing information about a specific symptom description and its corresponding disease label.

2. Structure: The dataset is divided into two columns: "label" and "text", where the "label" column contains the disease label associated with each symptom description.

3. Disease Variety: The dataset includes 24 diseases with 50 symptom descriptions, providing a comprehensive overview of common health conditions.

Data Configuration:

1. Data Augmentation: We utilized data augmentation techniques like Paraphrase online and Quill bot to expand the dataset size, resulting in 3600 datapoints, enhancing its diversity and granularity.

2. Quality Assurance: The data augmentation process involved rigorous quality assurance measures to ensure accuracy and relevance, preserving the semantic meaning and clinical relevance of original symptom descriptions.

3. Training and Validation: The Symptom2Disease Dataset is used to train and validate machine learning models, enhancing the efficacy and reliability of our healthcare chatbot by accurately classifying diseases based on user-input symptoms.

Sample of Dataset:

	label	text
0	Psoriasis	I have been encountering a skin hasty on my arms, legs, and middle for the past few weeks. It is ruddy, bothersome, and secured in dry, textured patches.
1	Psoriasis	My skin has been peeling, particularly on my knees, elbows, and scalp. This peeling is regularly went with by a burning or stinging sensation.
2	Psoriasis	I have been encountering joint torment in my fingers, wrists, and knees. The torment is frequently throbbing and throbbing, and it gets more awful when I move my joints.
3	Psoriasis	There's a silver like cleaning on my skin, particularly on my lower back and scalp. This cleaning is made up of little scales that drop off effortlessly when I scratch them.
4	Psoriasis	My nails have little marks or pits in them, and they frequently feel incendiary and delicate to the touch. Indeed there are minor rashes on my arms.
5	Psoriasis	The skin on my palms and soles is thickened and has profound breaks. These breaks are difficult and drain effortlessly.
6	Psoriasis	The skin around my mouth, nose, and eyes is ruddy and kindled. It is regularly bothersome and awkward. There's a recognizable aggravation in my nails.
7	Psoriasis	My skin is exceptionally delicate and responds effortlessly to changes in temperature or mugginess. I frequently need to use caution approximately what items I utilize on my skin.
8	Psoriasis	I have taken note a sudden peeling of skin at diverse parts of my body, basically arms, legs and back. Too, I confront serious joint torment and skin rashes.
9	Psoriasis	The skin on my private parts is ruddy and kindled. It is regularly bothersome, burning, and awkward. There are rashes on distinctive parts of the body as well.
10	Psoriasis	I have experienced weakness and a common feeling of disquietude. I frequently feel tired and have a need of vitality, indeed after a great night's rest.
11	Psoriasis	The hasty on my skin has spread to other parts of my body, counting my chest and guts. It is bothersome and awkward, and it is frequently more awful at night. I am moreover confronting skin pe
12	Psoriasis	The hasty on my skin is more regrettable within the winter months when the discuss is dry. I find that I got to moisturize more as often as possible and utilize humidifiers to keep my skin hydrate
13	Psoriasis	I have experienced trouble resting due to the tingling and inconvenience caused by the hasty. There are little marks in my nails, which is truly concerning.
14	Psoriasis	My skin is inclined to contaminations due to dry, flaky patches. I am encountering a solid torment in my joints. The skin on my knees and elbows is beginning to peel off.
15	Psoriasis	I am beginning to have rashes on my skin. The hasty frequently drains when I scratch or rub it. Additionally, I have taken note little scratches in my nails.
16	Psoriasis	I have taken note that my skin has gotten to be more delicate than it utilized to be. There's a silver like cleaning on my skin, particularly on my back and elbows.
17	Psoriasis	I am stressed approximately the steady peeling of the skin on my palm, elbow and knee. I have created rashes on my arms, which tingle on the off chance that I touch them. All of these are maki

Table 1: Sample of Data Set

Chapter 4

Implementation

Importing Libraries:-

Import Libraries

```
# Basic Libraries
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns

# Necessary Libraries for Data Preparation
import string
import nltk

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
```

import numpy as np: NumPy library, which provides support for mathematical operations on arrays and matrices. NumPy is widely used for numerical computing tasks.

import pandas as pd: Pandas library, which provides data structures and data analysis tools. Pandas is commonly used for handling structured data, such as tabular data.

import sklearn: scikit-learn library, which provides tools for machine learning and data mining tasks. scikit-learn is a widely used library for building and evaluating machine learning models.

import matplotlib.pyplot as plt: matplotlib is a popular plotting library used for creating visualizations such as graphs, charts, and plots.

import seaborn as sns: seaborn is a data visualization library built on top of matplotlib that provides a high-level interface for creating attractive and informative statistical graphics.

import string: Importing the string module, which provides a collection of string constants and helper functions for string manipulation.

import nltk: Importing the NLTK (Natural Language Toolkit) library, which is a comprehensive toolkit for natural language processing (NLP) tasks in Python.

Importing NLTK Modules:

from nltk.tokenize import word_tokenize: Importing the word_tokenize function from NLTK, which is used for tokenizing text into words.

from nltk.corpus import stopwords: Importing the stopwords corpus from NLTK, which contains a list of common stopwords for various languages.

from nltk.stem import WordNetLemmatizer: Importing the WordNetLemmatizer class from NLTK, which is used for lemmatizing words (reducing them to their base or dictionary form).

from nltk.corpus import wordnet: Importing the wordnet corpus from NLTK, which provides lexical and semantic resources, including synsets and definitions.

Importing scikit-learn Modules:

from sklearn.utils import shuffle: Importing the shuffle function from scikit-learn, which is used for shuffling arrays or lists randomly.

from sklearn.model_selection import train_test_split: Importing the train_test_split function from scikit-learn, which is used for splitting datasets into training and testing sets.

from sklearn.feature_extraction.text import TfidfVectorizer: Importing the TfidfVectorizer class from scikit-learn, which is used for converting text documents into TF-IDF (Term Frequency-Inverse Document Frequency) features.

```
# Necessary Libraries for ML Models
from sklearn.model_selection import GridSearchCV

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

# Necessary Libraries for Accuracy Measures
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Necessary Libraries for Deployment
import joblib
```

Importing Machine Learning Models:

from sklearn.model_selection import GridSearchCV:

GridSearchCV is used for hyperparameter tuning, where it exhaustively searches through a specified parameter grid to find the best parameters for a given estimator. This is crucial for optimizing the performance of machine learning models.

from sklearn.tree import DecisionTreeClassifier:

DecisionTreeClassifier is a popular machine learning model used for classification tasks, where it creates a decision tree to predict the target variable based on input features.

from sklearn.ensemble import RandomForestClassifier:

RandomForestClassifier is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification).

from sklearn.svm import SVC:

SVC stands for Support Vector Classifier, which is a type of SVM (Support Vector Machine) model used for classification tasks. SVMs are powerful classifiers that work well with both linear and non-linear data.

from sklearn.neighbors import KNeighborsClassifier:

KNeighborsClassifier is a simple and effective algorithm for classification tasks, where it classifies instances based on the majority vote of its k nearest neighbors in the feature space.

Importing Evaluation Metrics:

from sklearn.metrics import accuracy_score: accuracy_score function is used to evaluate the accuracy of classification predictions by comparing them to the true labels.

from sklearn.metrics import confusion_matrix: confusion_matrix function is used to compute a confusion matrix, which summarizes the performance of a classification algorithm by tabulating the number of true positive, true negative, false positive, and false negative predictions.

from sklearn.metrics import classification_report:

classification_report function is used to generate a text report showing the main classification metrics, such as precision, recall, and F1-score, for each class in a classification problem. This report is useful for evaluating the performance of a classification model on a dataset.

import joblib: This imports the joblib library, which provides utilities for saving and loading Python objects, particularly machine learning models, efficiently. It is commonly used to persist trained models to disk, allowing them to be reused later without needing to retrain.

Download NLTK resources

```
[ ] # download the Punkt tokenizer models
    nltk.download('punkt')

    # download a list of common stopwords
    nltk.download('stopwords')

    # download the WordNet lexical database
    nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

Downloading Tokenization Resources:

nltk.download('punkt'): The Punkt tokenizer models are pre-trained unsupervised machine learning models that split text into words and punctuation marks. It's commonly used for tokenization tasks in NLP.

Downloading Stopwords:

nltk.download('stopwords'): Stopwords are commonly used words (e.g., "the", "is", "and") that are often removed from text data during preprocessing because they typically do not contribute much to the meaning of the text.

Downloading Wordnet:

nltk.download('wordnet'): This downloads the WordNet database. WordNet is a lexical database of English words organized into synsets (sets of synonyms), with each synset representing a distinct concept. It's widely used in NLP for tasks like synonymy detection, semantic similarity, and word sense disambiguation.

Understand and Clean the Data: -

-This line of code reads data from a CSV file into a Pandas.

```
[ ] data
```

	Unnamed: 0	label	text
0	0	Psoriasis	I have been encountering a skin hasty on my ar...
1	1	Psoriasis	My skin has been peeling, particularly on my k...
2	2	Psoriasis	I have been encountering joint torment in my f...
3	3	Psoriasis	There's a silver like cleaning on my skin, par...
4	4	Psoriasis	My nails have little marks or pits in them, an...
...	...	...	...
3595	3595	diabetes	These strong desires and the need to urinate o...
3596	3596	diabetes	I have trouble breathing, especially outside. ...
3597	3597	diabetes	I find it difficult to breathe, especially out...
3598	3598	diabetes	I constantly sneeze and have a dry cough. My i...
3599	3599	diabetes	I have a dry cough and sneeze a lot. I have pa...

3600 rows × 3 columns

-View the data format.

```
[ ] # Drop the 'Unnamed: 0' column
    data.drop(columns = ["Unnamed: 0"], inplace = True)
    data
```

	label	text
0	Psoriasis	I have been encountering a skin hasty on my ar...
1	Psoriasis	My skin has been peeling, particularly on my k...
2	Psoriasis	I have been encountering joint torment in my f...
3	Psoriasis	There's a silver like cleaning on my skin, par...
4	Psoriasis	My nails have little marks or pits in them, an...
...	...	...
3595	diabetes	These strong desires and the need to urinate o...
3596	diabetes	I have trouble breathing, especially outside. ...
3597	diabetes	I find it difficult to breathe, especially out...
3598	diabetes	I constantly sneeze and have a dry cough. My i...
3599	diabetes	I have a dry cough and sneeze a lot. I have pa...

3600 rows × 2 columns

-This code snippet removes a specific column named "Unnamed: 0" from the previously loaded Pandas DataFrame: -

data.drop: The drop method is used to remove specified columns from the DataFrame.

columns = ["Unnamed: 0"]: parameter indicates the name of the column to be dropped.

inplace=True: This argument ensures that the changes are made directly and once to the existing DataFrame (data) without the need to create a new DataFrame.

```
[ ] # Concise summary of the DataFrame's structure and content
    data.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 3600 entries, 0 to 3599
    Data columns (total 2 columns):
     #   Column  Non-Null Count  Dtype
    ---  ------  --------------  -----
     0   label   3600 non-null   object
     1   text    3600 non-null   object
    dtypes: object(2)
    memory usage: 56.4+ KB

[ ] data.columns

    Index(['label', 'text'], dtype='object')

[ ] data.shape

    (3600, 2)

[ ] # Count the number of unique values in each column
    data.nunique()

    label      24
    text     3510
    dtype: int64

[ ] data.value_counts().sum()

    3600

[ ] # Check and Count null values
    data.isnull().sum()

    label    0
    text     0
    dtype: int64

[ ] # Check and Count duplicated values
    data.duplicated().sum()

    90
```

data.info(): method provides a concise summary of the Pandas DataFrame, offering insights into its structure and content.

The output includes details such as the column names, data types (integers, floats, objects), the number of non-null entries in each column, and the total memory usage of the DataFrame.

data.columns: attribute is used to retrieve the names of the columns in the Pandas DataFrame.

data.shape: attribute is employed to retrieve the dimensions of the Pandas DataFrame.

data.nunique(): method is utilized to count the number of unique values in each column of the Pandas DataFrame.

data.value_counts().sum(): expression is used to calculate the total number of rows in the Pandas DataFrame.

data.isnull().sum(): expression is used to count the number of missing (null) values in each column of the Pandas DataFrame.

data.duplicated().sum(): expression is used to count the number of duplicated rows in the Pandas DataFrame.

```
[ ] # Drop duplicated values
    data.drop_duplicates(inplace = True)
    data
```

	label	text
0	Psoriasis	I have been encountering a skin hasty on my ar...
1	Psoriasis	My skin has been peeling, particularly on my k...
2	Psoriasis	I have been encountering joint torment in my f...
3	Psoriasis	There's a silver like cleaning on my skin, par...
4	Psoriasis	My nails have little marks or pits in them, an...
...	...	...
3595	diabetes	These strong desires and the need to urinate o...
3596	diabetes	I have trouble breathing, especially outside. ...
3597	diabetes	I find it difficult to breathe, especially out...
3598	diabetes	I constantly sneeze and have a dry cough. My i...
3599	diabetes	I have a dry cough and sneeze a lot. I have pa...

3510 rows × 2 columns

data.drop_duplicates(inplace=True): expression is employed to remove duplicated rows from the Pandas DataFrame.

Text Preprocessing--->(NLP)

```
[ ] def lowercase_text(text):
        return text.lower()

    def remove_punctuation(text):
        translator = str.maketrans('', '', string.punctuation)
        text_without_punct = text.translate(translator).strip()
        return text_without_punct

    def tokenize_text(text):
        tokens = word_tokenize(text)
        return tokens

    def remove_stopwords(tokens):
        stop_words = set(stopwords.words('english'))
        filtered_tokens = [token for token in tokens if token.isalpha() and token not in stop_words]
        return filtered_tokens

    def lemmatize_text(tokens):
        lemmatizer = WordNetLemmatizer()
        lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
        return lemmatized_tokens
```

lowercase_text: This function converts all text to lowercase. It's important because it ensures that words are treated consistently regardless of their original capitalization. For example, "Hello" and "hello" would be considered the same word after lowercasing.

remove_punctuation: This function removes punctuation from the text. Punctuation marks like periods, commas, and exclamation marks often don't carry much meaning in text classification tasks and can be safely removed without losing important information.

tokenize_text: This function tokenizes the text, meaning it splits the text into individual words or tokens. Tokenization is a fundamental step in natural language processing tasks because it breaks down the text into units that can be processed by algorithms.

remove_stopwords: Stopwords are common words in a language that often occur frequently but don't carry much semantic meaning (e.g., "the", "is", "and"). Removing stopwords can help reduce noise in the data and improve the performance of machine learning algorithms by focusing on more meaningful words.

lemmatize_text: Lemmatization reduces words to their base or root form, which helps in standardizing words with similar meanings. For example, "feeling" and "felt" would both be lemmatized to "feel". This process reduces the dimensionality of the feature space and can improve the generalization of the model.

Preprocessing is a crucial step in text analysis tasks as it helps remove noise and irrelevant information, leading to more accurate and efficient machine learning models.

```
[ ]  # Preprocessing Container function

     def preprocess_text(text):
         text = lowercase_text(text)
         text = remove_punctuation(text)
         tokens = tokenize_text(text)
         tokens = remove_stopwords(tokens)
         tokens = lemmatize_text(tokens)
         preprocessed_text = ' '.join(tokens)
         return preprocessed_text
```

```
[ ]  # Apply Preprocessing Container function to symptoms

     data['text'] = data['text'].apply(preprocess_text)
```

preprocess_text: function combines the preprocessing steps into a single function.

DataFrame Manipulation: 'data' is a pandas DataFrame, this code accesses the 'text' column using data['text'].

apply(): method in pandas DataFrame allows to apply “preprocess_text” function to each element in the 'text' column.

Data Transformation: After applying the “preprocess_text” function, the text data in the 'text' column will be transformed into preprocessed text, which is then stored back into the same 'text' column. The original text data will be replaced with the preprocessed text.

```
# Extract and Count unique dictionary vocabs

def count_unique_vocab(count):
    unique_vocabularies = set()
    for text in count:
        words = text.split()
        for word in words:
            unique_vocabularies.add(word)
    return len(unique_vocabularies)

# Count unique dictionary vocabs
num_unique_vocabs = count_unique_vocab(data['text'])

print("Number of unique dictionary vocabs:", num_unique_vocabs)
```

```
Number of unique dictionary vocabs: 2275
```

count_unique_vocab:

This function takes a list of texts (from the 'text' column of a DataFrame) as input. It initializes an empty set called ‘unique_vocabularies’ to store unique words. It iterates over each text in the input list. For each text, it splits the text into individual words using whitespace as the delimiter. It then iterates over each word and adds it to the set of unique vocabularies.

Finally, it returns the count of unique vocabularies.

num_unique_vocabs=count_unique_vocab(data['text']): This line calls the count_unique_vocab function with the 'text' column of the DataFrame data as input to Calculate the Number of Unique Vocabularies.

The result is stored in the variable num_unique_vocabs.

Text Representation: In natural language processing tasks, the size of the vocabulary impacts the dimensionality of feature representations (e.g. TF-IDF), which in turn affects the performance of machine learning models.

Select the features (X) as 'text' column and target (y) as 'label' column

```
[ ] X = data['text']
    y = data['label']
```

```
[ ] X
```

```
0       encountering skin hasty arm leg middle past we...
1       skin peeling particularly knee elbow scalp pee...
2       encountering joint torment finger wrist knee t...
3       there silver like cleaning skin particularly l...
4       nail little mark pit frequently feel incendiar...
                              ...
3595    strong desire need urinate occur daily basis o...
3596    trouble breathing especially outside start fee...
3597    find difficult breathe especially outside heat...
3598    constantly sneeze dry cough infection dont see...
3599    dry cough sneeze lot palpitation infection don...
Name: text, Length: 3510, dtype: object
```

```
[ ] y
```

```
0       Psoriasis
1       Psoriasis
2       Psoriasis
3       Psoriasis
4       Psoriasis
          ...
3595     diabetes
3596     diabetes
3597     diabetes
3598     diabetes
3599     diabetes
Name: label, Length: 3510, dtype: object
```

-Select the features (X) as 'text' column and target (y) as 'label' column and display them.

```
# The 'shuffle' function is used to randomly Shuffle/Rearrange the elements of a dataset
from sklearn.utils import shuffle
data = shuffle(data, random_state = 42)
data
```

	label	text
3282	drug reaction	monthly cycle changed ive unexpected vaginal d...
315	Typhoid	ive dealing substantial bloating constipation ...
2756	urinary tract infection	getting blood pee sometimes get nauseous peein...
603	Impetigo	rash around nose expansive ruddy bruise taken ...
879	Dengue	ive headache muscular ache aching muscle get w...
...	...	...
1136	Common Cold	sinus feel stuffy eye quite red simply lack en...
1302	Pneumonia	im trouble breathing quite uneasy throat fille...
865	Dengue	day quite challenging due significant joint pa...
3597	diabetes	find difficult breathe especially outside heat...
3264	drug reaction	terrible mental clarity find difficult stay fo...

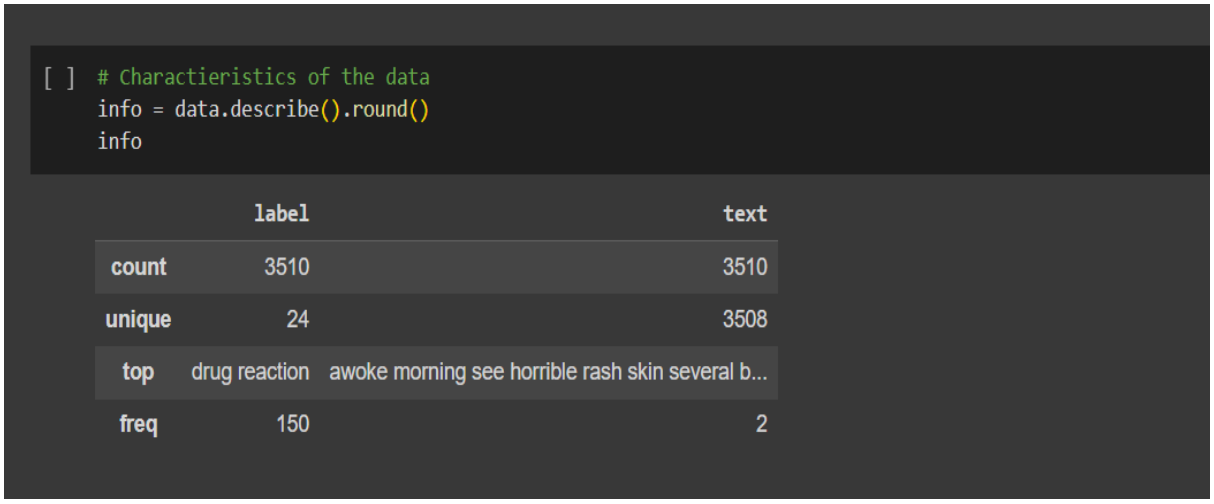
3510 rows × 2 columns

data = shuffle(data, random_state=42):

-This line shuffles the DataFrame data. The shuffle function is applied to the DataFrame data, randomizing the order of its rows. The random_state parameter is set to 42 to ensure reproducibility; setting random_state to a fixed value ensures that the shuffling produces the same result each time the code is running.

data:

-This line simply prints the shuffled DataFrame data after shuffling.



```
[ ] # Charactieristics of the data
    info = data.describe().round()
    info
```

	label	text
count	3510	3510
unique	24	3508
top	drug reaction	awoke morning see horrible rash skin several b...
freq	150	2

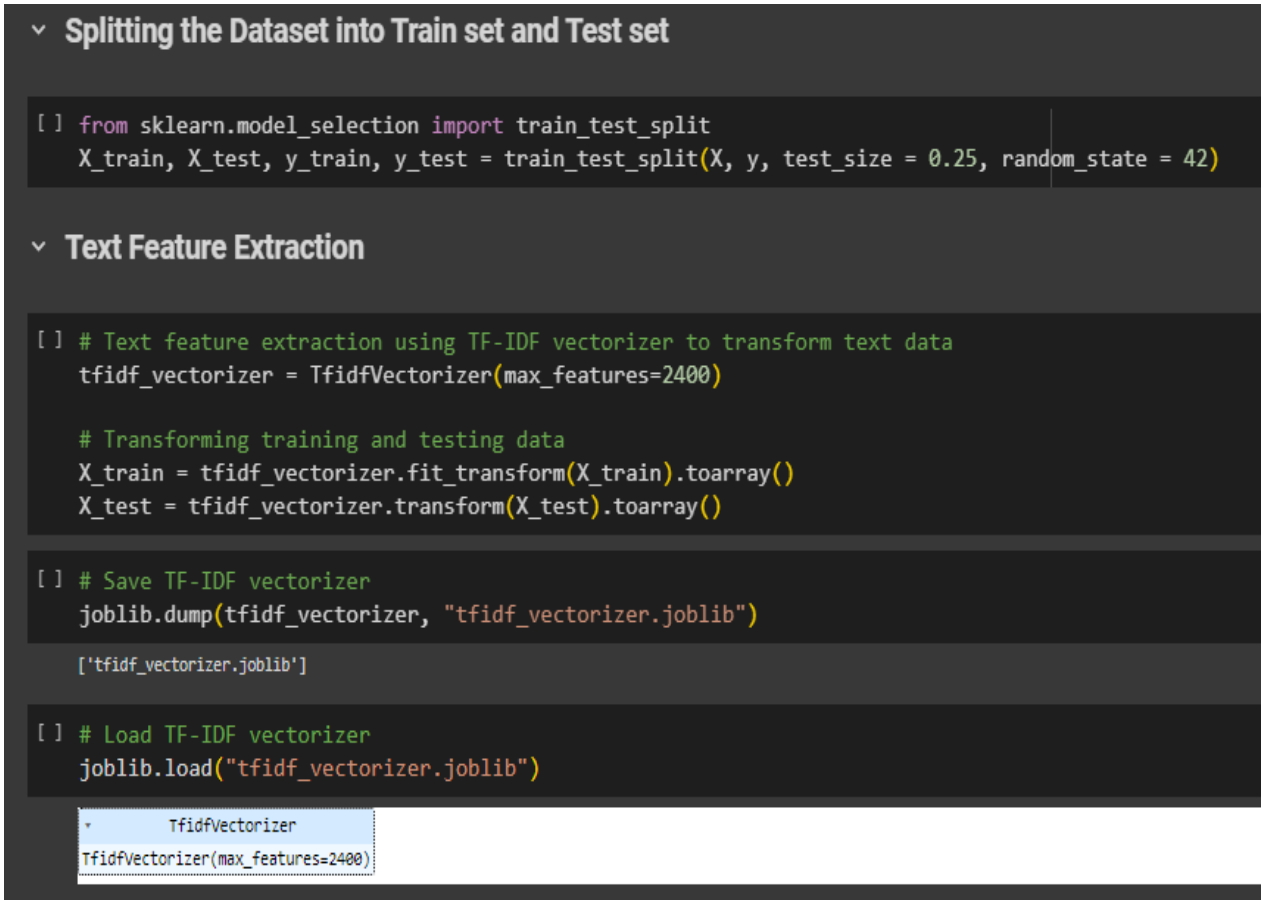
info = data.describe().round():

-This line calculates descriptive statistics or readings for the DataFrame data using the describe() method. The describe() method computes various summary statistics, such as count(total number of entries or observations in the dataset), unique(the number of distinct categories or values within a specific column), top(the most frequent value or category within a specific column) and freq(the frequency or number of occurrences of the top value), for each column in the DataFrame.

-The round() function is then applied to round the calculated statistics to the nearest integer(If there are any decimal values).

info:

-This line prints the summary statistics stored in the variable info.



X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42):

-This line splits the input data (X) and corresponding labels (y) into training and testing sets:

-X_train and y_train will contain the training data and labels respectively, while X_test and y_test will contain the testing data and labels respectively.

-The test_size parameter is set to 0.25, indicating that 25% of the data will be allocated for testing, and the remaining 75% will be used for training.

-The random_state parameter; setting random_state to a fixed value ensures that the shuffling produces the same result each time the code is running.

tfidf_vectorizer = TfidfVectorizer(max_features=2400)

-This line initializes a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer object using the TfidfVectorizer class from scikit-learn. TF-IDF is a numerical statistic/representation that reflects the importance of a term(word) in a document(sentence) relative to a collection of documents(sentences).

- "max_features=2400" sets the maximum number of features (vocabulary size) to 2400. This parameter controls the maximum number of unique words or terms considered during vectorization, meaning only the top 2400 most frequent terms will be considered as features. This can help in reducing the dimensionality of the data.

X_train = tfidf_vectorizer.fit_transform(X_train).toarray()

- "fit_transform(X_train)": This method fits the TfidfVectorizer to the training data (X_train) and transforms it into a TF-IDF matrix. This means it learns the vocabulary of the training data and computes the TF-IDF values for each term in the documents.

- "toarray()": converts the sparse matrix returned by fit_transform() into a dense NumPy array for easier manipulation and computation.

X_test = tfidf_vectorizer.transform(X_test).toarray()

- "transform(X_test)": This method transforms the test data (X_test) using the same vocabulary learned from the training data. It computes the TF-IDF values for the terms(words) in the test documents(sentences) based on the vocabulary learned during the fitting stage (It applies the same transformation as done for the training data to ensure consistency).

- "toarray()": again, converts the resulting sparse matrix into a dense NumPy array.

joblib.dump(tfidf_vectorizer, "tfidf_vectorizer.joblib") -This line is using the joblib.dump() function to save the "tfidf_vectorizer" object to a file named "tfidf_vectorizer.joblib".

-The tfidf_vectorizer object will be serialized and saved to the specified file location to load and deploy it later.

joblib.load("tfidf_vectorizer.joblib")

-This line of code is using the joblib.load() function to load the saved tfidf_vectorizer object from the file "tfidf_vectorizer.joblib" to load and deploy it later in Flask.

Grid Search:

Grid search is a powerful technique used in machine learning to find the optimal hyperparameters for a model (fine-tuning). It works by exhaustively searching through a specified subset of hyperparameter combinations and evaluating each combination using cross-validation. The hyperparameters that yield the best performance on the validation set are then chosen as the optimal ones for the ML model.

➢ Machine Learning Models: -

1-Decision Tree Classifier:

A decision tree classifier is a popular supervised learning algorithm used for classification tasks. It builds a tree-like structure where each internal node represents a "decision" based on a feature, and each leaf node represents a class label. It works by recursively partitioning the dataset into subsets based on the most significant features.

-We used the following Hyperparameters: **{criterion , max_depth , min_samples_split, max_features, min_samples_leaf }**

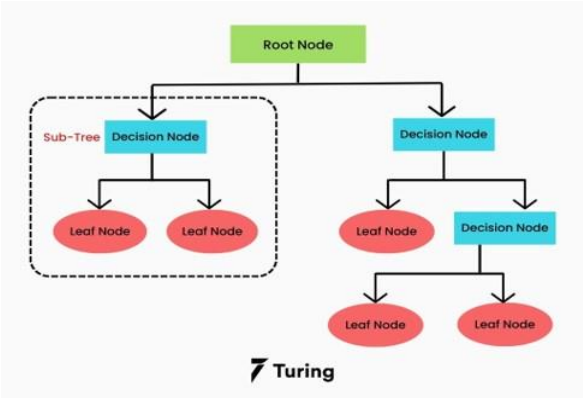


Figure 4: Decision Tree Classifier

Applying Grid Search to find the best model version and the best hyperparameters:

Decision Tree Classifier

Applying Grid Search to find the best model version and the best hyperparameters

```
[ ] from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import GridSearchCV

    # Create a Decision Tree Classifier object
    dt_classifier = DecisionTreeClassifier(random_state = 42)

    # Define the hyperparameters and their possible values to search
    parameters = {
        'criterion': ['gini', 'entropy'],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'max_features' : ['sqrt', 'log2', None]
    }

    # Create the Grid Search object
    grid_search = GridSearchCV(estimator = dt_classifier,
                               param_grid = parameters,
                               cv = 5,
                               scoring = 'accuracy',
                               n_jobs = -1)

    # Fit the Grid Search to the train data
    grid_search.fit(X_train, y_train)

    # Get the best hyperparameters found
    best_hyperparameters = grid_search.best_params_
    print("Best Hyperparameters:", best_hyperparameters)

    # Get the best model version
    best_dt_classifier = grid_search.best_estimator_
    print(best_dt_classifier)

    # Print the best accuracy found
    best_accuracy = grid_search.best_score_
    print(f'Best Accuracy: {best_accuracy*100:.2f} %')
```

```
Best Hyperparameters: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
DecisionTreeClassifier(random_state=42)
Best Accuracy: 86.74 %
```

dt_classifier = DecisionTreeClassifier(random_state=42)

-Initialize a DecisionTreeClassifier object named dt_classifier.

-Set the random_state parameter to 42 for reproducibility.

parameters = {...}

-Define a dictionary named “parameters” containing different hyperparameters and their potential values. These hyperparameters will be tuned during the grid search process.

grid_search = GridSearchCV(estimator=dt_classifier, param_grid=parameters, cv=5, scoring='accuracy', n_jobs=-1)

-Initialize a GridSearchCV object named “grid_search”.

-Pass the following parameters:
estimator: The model to be tuned, in this case, dt_classifier, which is a DecisionTreeClassifier.

param_grid: The dictionary containing the hyperparameters and their possible values.

cv: Cross-validation strategy, here 5-fold cross-validation.

scoring: The evaluation metric used to compare models. Here it's set to 'accuracy'.

n_jobs: Number of parallel jobs to run (-1 indicates using all available processors).

grid_search.fit(X_train, y_train)

-Fit the grid_search object to the training data (X_train and y_train).

best_hyperparameters = grid_search.best_params_

-Retrieve the best hyperparameters found during the grid search and store them in the “best_hyperparameters” variable.

print("Best Hyperparameters:", best_hyperparameters)

-Print the best hyperparameters found during the grid search.

best_dt_classifier = grid_search.best_estimator_

-Retrieve the best DecisionTreeClassifier version found during the grid search and store it in the “best_dt_classifier” variable.

print(best_dt_classifier)

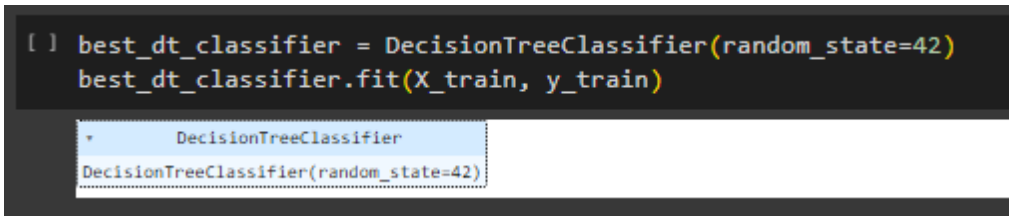
-Print the details of the best DecisionTreeClassifier version found during the grid search.

best_accuracy = grid_search.best_score_

-Retrieve the best mean cross-validated accuracy score found during the grid search and store it in the “best_accuracy” variable.

print(f'Best Accuracy: {best_accuracy*100:.2f} %')

-Print the best mean cross-validated accuracy score as a percentage with two decimal places.



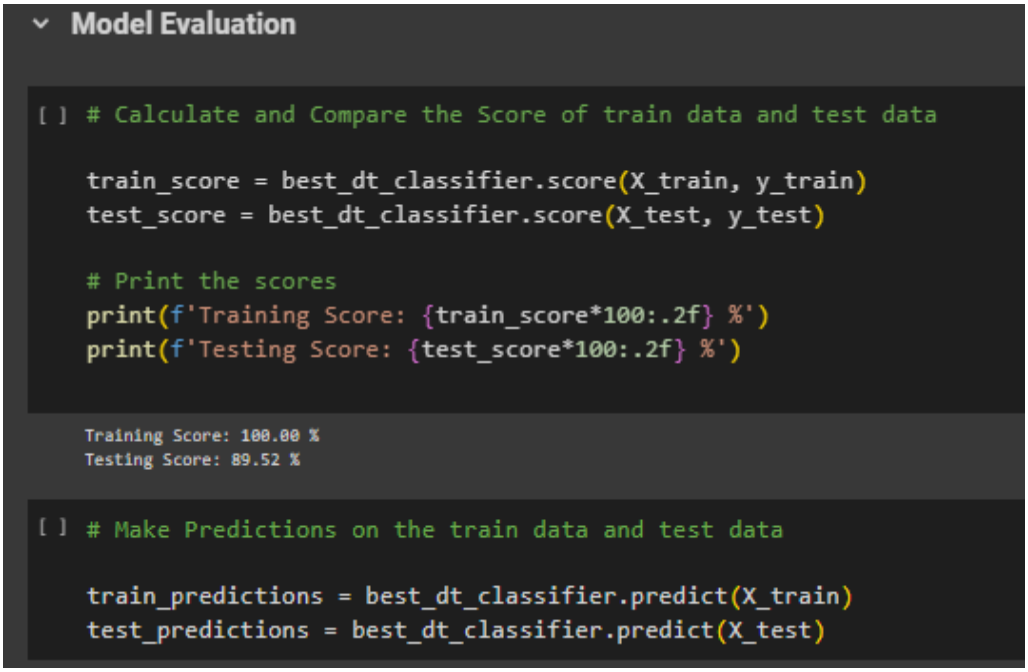
best_dt_classifier = DecisionTreeClassifier(random_state=42)

-Create an instance of DecisionTreeClassifier class called “best_dt_classifier”, with the best parameter(random_state=42).

best_dt_classifier.fit(X_train, y_train)

This line trains the best version of Decision Tree classifier on the training data (X_train) and (y_train).

Model Evaluation:



train_score = best_dt_classifier.score(X_train, y_train)

-Calculate the mean accuracy of the best DecisionTreeClassifier model (best_dt_classifier) on the train data (X_train, y_train).

-This measures the performance of the model on seen data.

test_score = best_dt_classifier.score(X_test, y_test)

-Calculate the mean accuracy of the best DecisionTreeClassifier model (best_dt_classifier) on the test data (X_test, y_test).

-This measures the performance of the model on unseen data.

print(f'Training Score: {train_score:.2f} %')

-Print the train accuracy score as a percentage with two decimal places.

print(f'Testing Score: {test_score:.2f} %')

-Print the test accuracy score as a percentage with two decimal places.

train_predictions = best_dt_classifier.predict(X_train)

-Predict the class labels for the training data (X_train) using the best DecisionTreeClassifier model (best_dt_classifier).

test_predictions = best_dt_classifier.predict(X_test)

-Predict the class labels for the testing data (X_test) using the best DecisionTreeClassifier model (best_dt_classifier).

```
[ ] # Calculate and Compare the Accuracy for training and testing data
    from sklearn.metrics import accuracy_score

    train_accuracy = accuracy_score(y_train, train_predictions)
    test_accuracy = accuracy_score(y_test, test_predictions)

    # Print the accuracies
    print(f'Training Accuracy: {train_accuracy*100:.2f} %')
    print(f'Testing Accuracy: {test_accuracy*100:.2f} %')

    Training Accuracy: 100.00 %
    Testing Accuracy: 89.52 %
```

train_accuracy= accuracy_score(y_train, train_predictions)

-Calculate the accuracy classification score for the train data, by comparing the true labels (y_train) with the predicted labels (train_predictions).

-This measures the performance of the model on seen data.

test_accuracy = accuracy_score(y_test, test_predictions)

-Calculate the accuracy classification score for the test data, by comparing the true labels (y_test) with the predicted labels (test_predictions).

-This measures the performance of the model on unseen data.

print(f'Training Accuracy: {train_accuracy*100:.2f} %')

This line prints the training accuracy as a percentage with two decimal places.

print(f'Testing Accuracy: {test_accuracy*100:.2f} %')

This line prints the testing accuracy as a percentage with two decimal places.

```
[ ] # Make the Confusion Matrix
    from sklearn.metrics import confusion_matrix

    cm_1 = confusion_matrix(y_test, test_predictions)

    # Print the Confusion Matrix
    print(cm_1)
```

```
[[36  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 30  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 32  3  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 26  0  4  0  4  0  0  0  0  1  0  2  1  0  1  2  0  0  0  0]
 [ 0  0  0  0  0 38  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  5  1 26  0  0  1  0  0  1  2  0  4  1  0  2  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 34  0  0  0  0  0  0  0  0  2  0  0  0  0  0  2  0]
 [ 0  0  0  0  1  0  2  0 35  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  1  0  0  1]
 [ 0  0  0  0  3  0  0  0  1  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 32  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  2  0  0  0  0  0  0  0  0 29  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0 24  0  1  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  1  0  2  0  1  0  0  0  0 35  0  0  1  0  1  0  0  0]
 [ 0  0  0  0  0  1  0  1  0  0  0  0  0  0  0  0 31  0  0  0  0  2  3  0]
 [ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0 56  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  1  0  0  0 35  0  0  2  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 30  4  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  1  0  0  0 33  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0 32  0  0]
 [ 0  0  0  0  1  0  0  1  0  0  0  0  0  0  0  0  3  0  0  1  0  0 26  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  1  0  0  1 33]]
```

cm_1 = confusion_matrix(y_test, test_predictions)

-Calculate the confusion matrix based on the true labels (y_test) and the predicted labels (test_predictions).

print(cm_1)

-Print the computed confusion matrix.

-This confusion matrix printed as a 24x24 matrix (for multiclass classification), where each row represents the instances in an actual class, and each column represents the instances in a predicted class.

```
print(f"The Accuracy = {accuracy_score(y_test, test_predictions)*100:.2f} %")
```

```
The Accuracy = 89.52 %
```

```
print(classification_report(y_test, test_predictions))
```

```
                              precision    recall  f1-score   support

                        Acne       1.00      1.00      1.00        36
                   Arthritis       1.00      1.00      1.00        39
            Bronchial Asthma       0.97      0.97      0.97        31
        Cervical spondylosis       1.00      0.89      0.94        36
                   Chicken pox       0.62      0.63      0.63        41
                   Common Cold       0.95      1.00      0.97        38
                        Dengue       0.74      0.60      0.67        43
    Dimorphic Hemorrhoids       0.94      0.89      0.92        38
         Fungal infection       0.81      0.88      0.84        40
             Hypertension       0.97      0.94      0.95        32
                  Impetigo       0.97      0.88      0.92        34
                  Jaundice       1.00      1.00      1.00        34
                   Malaria       0.97      1.00      0.98        32
                  Migraine       0.91      0.94      0.92        31
                 Pneumonia       0.96      0.92      0.94        26
                 Psoriasis       0.76      0.85      0.80        41
                   Typhoid       0.76      0.82      0.78        38
            Varicose Veins       1.00      0.98      0.99        57
                   allergy       0.88      0.90      0.89        39
                  diabetes       0.88      0.88      0.88        34
             drug reaction       0.85      0.92      0.88        36
gastroesophageal reflux disease       0.89      0.94      0.91        34
         peptic ulcer disease       0.81      0.81      0.81        32
       urinary tract infection       0.97      0.92      0.94        36

                  accuracy                           0.90       878
                 macro avg       0.90      0.90      0.90       878
              weighted avg       0.90      0.90      0.90       878
```

print(f"The Accuracy = {accuracy_score(y_test, test_predictions)*100:.2f} %")

-Print the accuracy classification score of DecisionTreeClassifier model on the test dataset as a percentage with two decimal places.

-Calculate the accuracy as the fraction of correctly predicted instances out of the total number of instances.

print(classification_report(y_test, test_predictions))

-Print the classification report for the predictions made by of DecisionTreeClassifier model on the test dataset.

-The classification report includes various evaluation metrics such as precision, recall, F1-score, and support for each class, as well as averages across all classes:

Precision: The proportion of true positive predictions among all predicted positive instances. It measures the model's ability to avoid false positives.

Recall: The proportion of true positive predictions among all actual positive instances. It measures the model's ability to capture all positives.

F1-score: The harmonic Mean of precision and recall. It provides a balanced measure of the model's performance.

Support: The number of occurrences of each class in the test dataset.

Model validation:

Model Validation

```
[ ] # Validation Test #

    # text_before = "The skin around my mouth, nose, and eyes is ruddy and kindled. It is regularly bothersome and awkward. There's a recognizable aggravation in my nails."

    text_before = "The abdominal pain has been coming and going, and it's been really unpleasant. It's been accompanied by constipation and vomiting. I feel really concerned about my health."

    # Cleaning
    text_after = preprocess_text(text_before)

    print(text_before)
    print(text_after)

    # Vectorization
    tfidf_vectorizer

    text_after = tfidf_vectorizer.transform([text_after]).toarray()

    print(text_after.reshape(-1,1))

    # Prediction
    disease = best_dt_classifier.predict(text_after)

    print(disease)
```

```
The abdominal pain has been coming and going, and it's been really unpleasant. It's been accompanied by constipation and vomiting. I feel really concerned about my health.
abdominal pain coming going really unpleasant accompanied constipation vomiting feel really concerned health
[[0.        ]
 [0.30370294]
 [0.        ]
 ...
 [0.        ]
 [0.        ]
 [0.        ]]
['Typhoid']
```

text_before = "any symptoms description entered by user"

-Assign the input text string to the variable “text_before”. This variable holds the original text before any preprocessing or prediction.

text_after = preprocess_text(text_before)

-Apply the “preprocess_text” function to the “text_before” string to perform NLP steps. The processed text is then assigned to the variable “text_after”.

print(text_before)

-Print the original text (text_before).

print(text_after)

-Print the processed text after applying NLP steps.

tfidf_vectorizer

-This line calls the “tfidf_vectorizer” object.

text_after = tfidf_vectorizer.transform([text_after]).toarray()

-Transform the processed text (text_after) into a TF-IDF vector representation using the tfidf_vectorizer object. The transform method converts the text into numerical features, and “toarray()” converts the resulting sparse matrix into a dense NumPy array.

print(text_after.reshape(-1,1))

-Print the TF-IDF vector representation of the processed text (text_after), and reshape(-1,1) method reshapes the array to have one column while maintaining the same number of rows.

disease = best_dt_classifier.predict(text_after)

-Predict the disease for the processed text (text_after) using the best DecisionTreeClassifier model (best_dt_classifier).

print(disease)

-Print the predicted label (disease) for the processed text.

2-Random Forest Classifier:

Random Forest is an ensemble learning method that builds multiple Decision Trees, each trained on a random subset of the training data. The final prediction is based on the majority vote of the individual trees, providing a robust and effective approach for classification tasks.

-We used the following Hyperparameters:

{ n_estimators, criterion, max_depth , min_samples_split, min_samples_leaf, max_features }

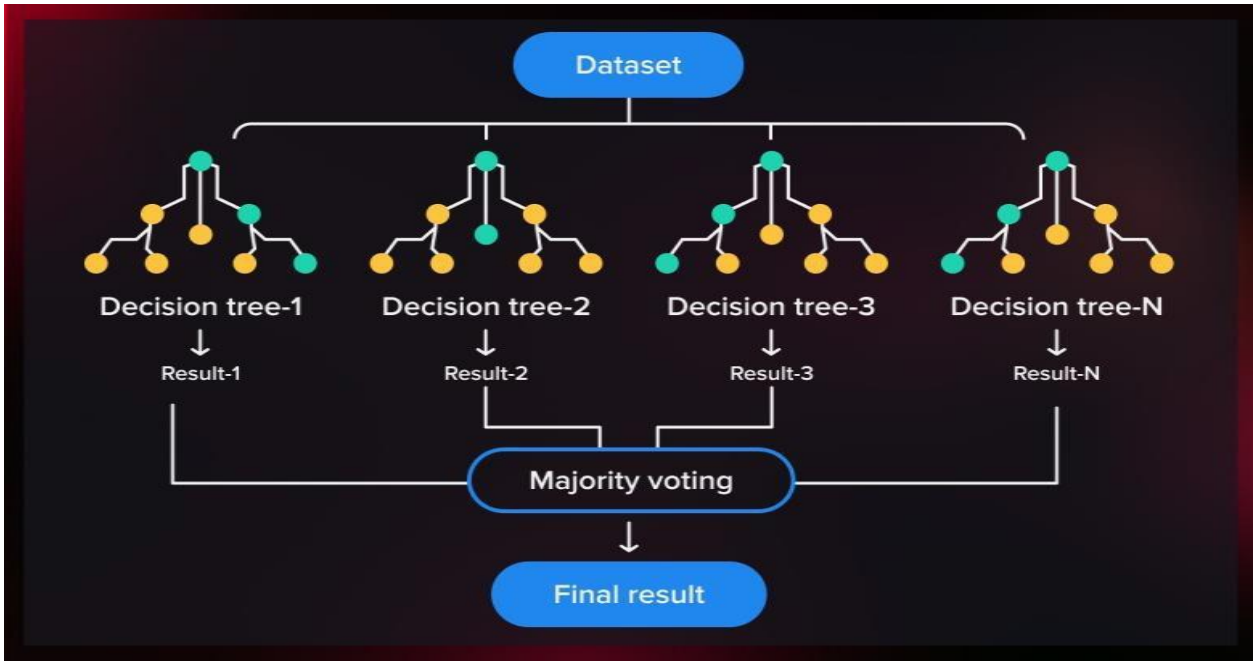


Figure 5: Random Forest Algorithm

Applying Grid Search to find the best model version and the best hyperparameters:

Random Forest Classifier

Random Forest is an ensemble learning algorithm, It works by constructing multiple decision trees.

Applying Grid Search to find the best model version and the best hyperparameters

```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.model_selection import GridSearchCV

    # Create a Random Forest Classifier object
    rf_classifier = RandomForestClassifier(random_state = 42)

    # Define the hyperparameters and their possible values to search
    parameters = {
        'n_estimators': [10, 50, 100],
        'criterion': ['gini', 'entropy'],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'max_features' : ['sqrt', 'log2', None]
    }

    # Create the Grid Search object
    grid_search = GridSearchCV(estimator = rf_classifier,
                               param_grid = parameters,
                               cv = 5,
                               scoring = 'accuracy',
                               n_jobs = -1)

    # Fit the Grid Search to the train data
    grid_search.fit(X_train, y_train)

    # Get the best hyperparameters found
    best_hyperparameters = grid_search.best_params_
    print("Best Hyperparameters:", best_hyperparameters)

    # Get the best model version
    best_rf_classifier = grid_search.best_estimator_
    print(best_rf_classifier)

    # Get the best accuracy found
    best_accuracy = grid_search.best_score_
    print(f'Best Accuracy: {best_accuracy*100:.2f} %')
```

```
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
RandomForestClassifier(criterion='entropy', max_features='log2',
                       random_state=42)
Best Accuracy: 97.68 %
```

rf_classifier = RandomForestClassifier (random_state=42)

-Initialize a RandomForestClassifier object named rf_classifier.

-Set the random_state parameter to 42 for reproducibility.

parameters = {...}

-Define a dictionary named “parameters” containing different hyperparameters and their potential values. These hyperparameters will be tuned during the grid search process.

grid_search = GridSearchCV(estimator= rf_classifier, param_grid=parameters, cv=5, scoring='accuracy', n_jobs=-1)

-Initialize a GridSearchCV object named “grid_search”.

-Pass the following parameters:
(estimator, param_grid, cv, scoring, n_jobs)

grid_search.fit(X_train, y_train)

-Fit the grid_search object to the training data (X_train and y_train).

best_hyperparameters = grid_search.best_params_

-Retrieve the best hyperparameters found during the grid search and store them in the “best_hyperparameters” variable.

print("Best Hyperparameters:", best_hyperparameters)

-Print the best hyperparameters found during the grid search.

best_rf_classifier = grid_search.best_estimator_

-Retrieve the best RandomForestClassifier version found during the grid search and store it in the “best_rf_classifier” variable.

print(best_rf_classifier)

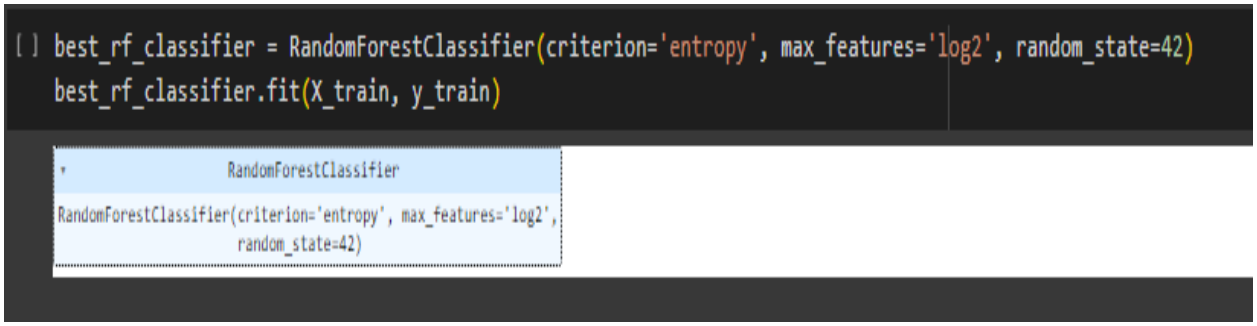
-Print the details of the best RandomForestClassifier version found during the grid search.

best_accuracy = grid_search.best_score_

-Retrieve the best mean cross-validated accuracy score found during the grid search and store it in the “best_accuracy” variable.

print(f'Best Accuracy: {best_accuracy*100:.2f} %')

-Print the best mean cross-validated accuracy score as a percentage with two decimal places.



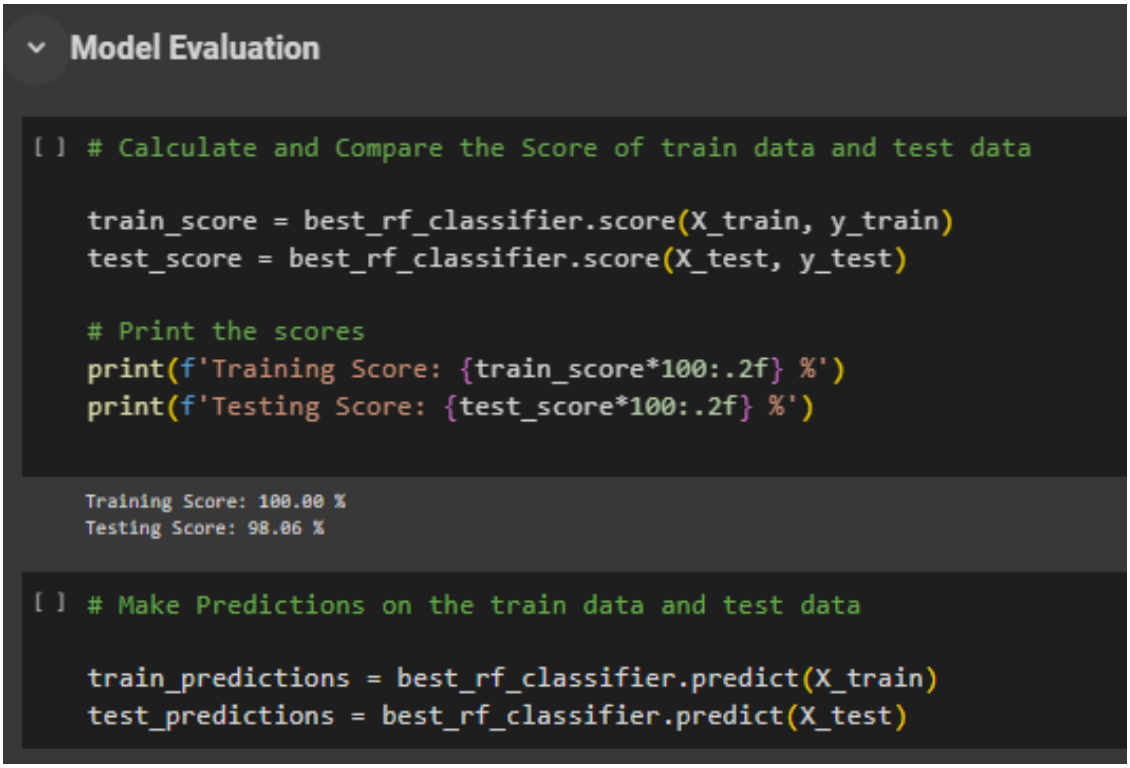
best_rf_classifier = RandomForestClassifier(random_state=42)

-Create an instance of RandomForestClassifier class called “best_rf_classifier”, with the best parameters (criterion='entropy', max_features='log2', random_state=42).

best_rf_classifier.fit(X_train, y_train)

This line trains the best version of Random Forest Classifier on the training data (X_train) and (y_train).

Model Evaluation:



train_score = best_rf_classifier.score(X_train, y_train)

-Calculate the mean accuracy of the best RandomForestClassifier model (best_rf_classifier) on the train data (X_train, y_train).

-This measures the performance of the model on seen data.

test_score = best_rf_classifier.score(X_test, y_test)

-Calculate the mean accuracy of the best RandomForestClassifier model (best_rf_classifier) on the test data (X_test, y_test).

-This measures the performance of the model on unseen data.

print(f'Training Score: {train_score:.2f} %')

-Print the train accuracy score as a percentage with two decimal places.

print(f'Testing Score: {test_score:.2f} %')

-Print the test accuracy score as a percentage with two decimal places.

train_predictions = best_rf_classifier.predict(X_train)

-Predict the class labels for the training data (X_train) using the best RandomForestClassifier model (best_rf_classifier).

test_predictions = best_rf_classifier.predict(X_test)

-Predict the class labels for the testing data (X_test) using the best RandomForestClassifier model (best_rf_classifier).

```
[ ] # Calculate and Compare the Accuracy for training and testing data
    from sklearn.metrics import accuracy_score

    train_accuracy = accuracy_score(y_train, train_predictions)
    test_accuracy = accuracy_score(y_test, test_predictions)

    # Print the accuracies
    print(f'Training Accuracy: {train_accuracy*100:.2f} %')
    print(f'Testing Accuracy: {test_accuracy*100:.2f} %')

    Training Accuracy: 100.00 %
    Testing Accuracy: 98.06 %
```

train_accuracy= accuracy_score(y_train, train_predictions)

-Calculate the accuracy classification score for the train data, by comparing the true labels (y_train) with the predicted labels (train_predictions).

-This measures the performance of the model on seen data.

test_accuracy = accuracy_score(y_test, test_predictions)

-Calculate the accuracy classification score for the test data, by comparing the true labels (y_test) with the predicted labels (test_predictions).

-This measures the performance of the model on unseen data.

print(f'Training Accuracy: {train_accuracy*100:.2f} %')

This line prints the training accuracy as a percentage with two decimal places.

print(f'Testing Accuracy: {test_accuracy*100:.2f} %')

This line prints the testing accuracy as a percentage with two decimal places.

```
[ ] # Make the Confusion Matrix
    from sklearn.metrics import confusion_matrix

    cm_2 = confusion_matrix(y_test, test_predictions)

    # Print the Confusion Matrix
    print(cm_2)
```

```
[[36  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 31  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 36  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 35  0  2  0  0  0  0  0  0  0  0  1  2  0  0  0  0  0  1  0]
 [ 0  0  0  0  0 38  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  1  2  0 38  0  0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 38  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 40  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 32  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 32  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 31  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  2  0  0  0  0  0  0  0  0  0  0  0 24  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 40  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0 37  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 57  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 39  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0 33  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0 35  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 34  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 32  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 36]]
```

cm_2 = confusion_matrix(y_test, test_predictions)

-Calculate the confusion matrix based on the true labels (y_test) and the predicted labels (test_predictions).

print(cm_2)

-Print the computed confusion matrix.

-This confusion matrix printed as a 24x24 matrix (for multiclass classification), where each row represents the instances in an actual class, and each column represents the instances in a predicted class.

```
print(f"The Accuracy = {accuracy_score(y_test, test_predictions)*100:.2f} %")
```

```
The Accuracy = 98.06 %
```

```
print(classification_report(y_test, test_predictions))
```

```
                              precision    recall  f1-score   support

                        Acne       1.00      1.00      1.00        36
                   Arthritis       1.00      1.00      1.00        39
            Bronchial Asthma       0.94      1.00      0.97        31
        Cervical spondylosis       0.97      1.00      0.99        36
                 Chicken pox       0.95      0.85      0.90        41
                 Common Cold       1.00      1.00      1.00        38
                      Dengue       0.93      0.88      0.90        43
     Dimorphic Hemorrhoids       1.00      1.00      1.00        38
          Fungal infection       1.00      1.00      1.00        40
              Hypertension       0.94      1.00      0.97        32
                   Impetigo       1.00      1.00      1.00        34
                   Jaundice       1.00      1.00      1.00        34
                    Malaria       0.97      1.00      0.98        32
                   Migraine       0.97      1.00      0.98        31
                  Pneumonia       1.00      0.92      0.96        26
                  Psoriasis       0.98      0.98      0.98        41
                    Typhoid       0.95      0.97      0.96        38
             Varicose Veins       1.00      1.00      1.00        57
                    allergy       1.00      1.00      1.00        39
                   diabetes       1.00      0.97      0.99        34
             drug reaction        1.00      0.97      0.99        36
gastroesophageal reflux disease       1.00      1.00      1.00        34
        peptic ulcer disease       0.94      1.00      0.97        32
    urinary tract infection        1.00      1.00      1.00        36

                   accuracy                           0.98       878
                  macro avg       0.98      0.98      0.98       878
               weighted avg       0.98      0.98      0.98       878
```

print(f"The Accuracy = {accuracy_score(y_test, test_predictions)*100:.2f} %")

-Print the accuracy classification score of RandomForestClassifier model on the test dataset as a percentage with two decimal places.

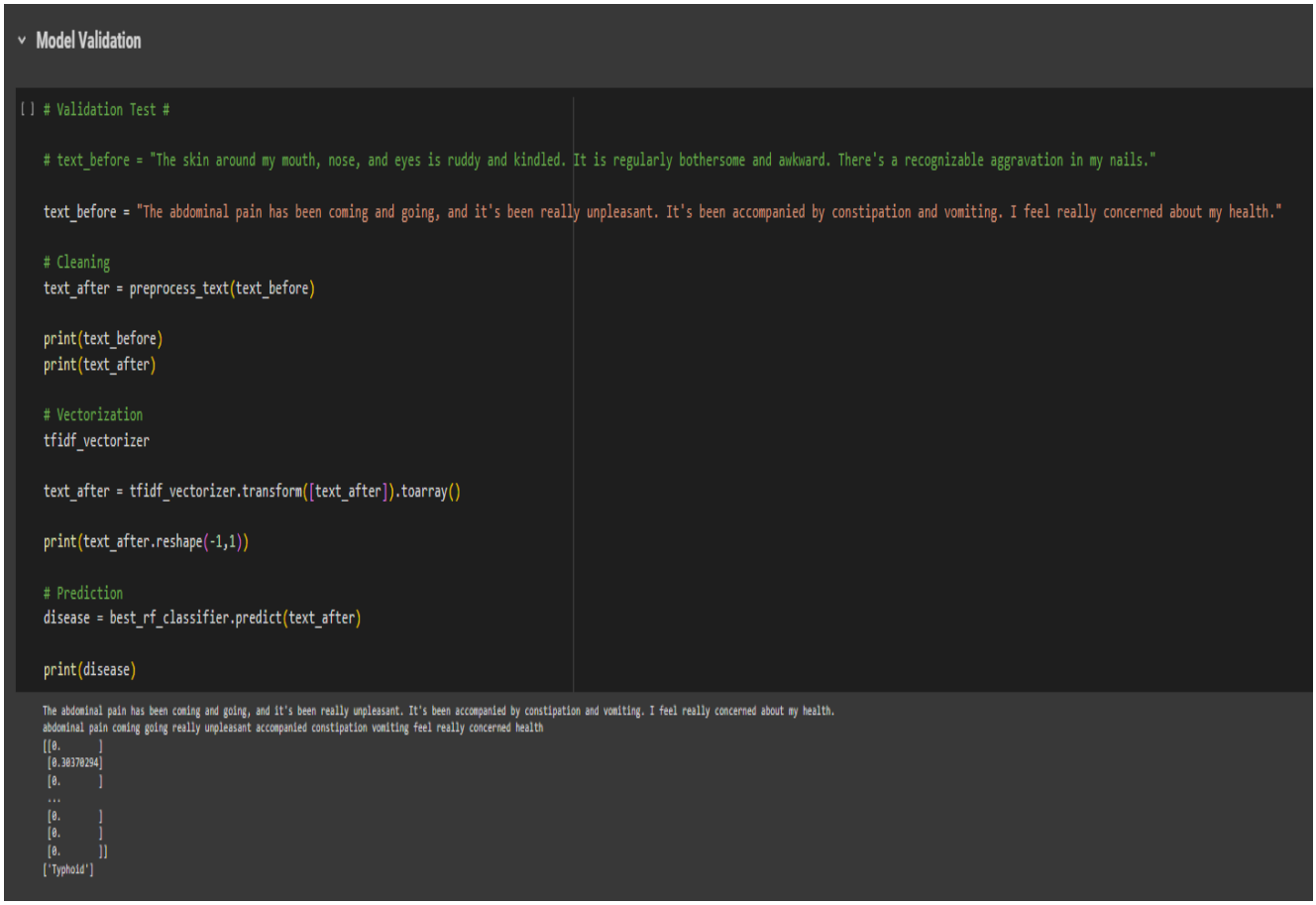
-Calculate the accuracy as the fraction of correctly predicted instances out of the total number of instances.

print(classification_report(y_test, test_predictions))

-Print the classification report for the predictions made by RandomForestClassifier model on the test dataset.

-The classification report includes various evaluation metrics such as **precision, recall, F1-score, and support** for each class, as well as averages across all classes.

Model validation:



text_before = "any symptoms description entered by user"

-Assign the input text string to the variable “text_before”. This variable holds the original text before any preprocessing or prediction.

text_after = preprocess_text(text_before)

-Apply the “preprocess_text” function to the “text_before” string to perform NLP steps. The processed text is then assigned to the variable “text_after”.

print(text_before)

-Print the original text (text_before).

print(text_after)

-Print the processed text after applying NLP steps.

tfidf_vectorizer

-This line calls the “tfidf_vectorizer” object.

text_after = tfidf_vectorizer.transform([text_after]).toarray()

-Transform the processed text (text_after) into a TF-IDF vector representation using the tfidf_vectorizer object. The transform method converts the text into numerical features, and “toarray()” converts the resulting sparse matrix into a dense NumPy array.

print(text_after.reshape(-1,1))

-Print the TF-IDF vector representation of the processed text (text_after), and reshape(-1,1) method reshapes the array to have one column while maintaining the same number of rows.

disease = best_rf_classifier.predict(text_after)

-Predict the disease for the processed text (text_after) using the best RandomForestClassifier model (best_rf_classifier).

print(disease)

-Print the predicted label (disease) for the processed text.

3-Support Vector Machine (SVM):

A Support Vector Machine (SVM) is a powerful supervised machine learning algorithm for both linear and non-linear classification tasks. For classification, SVM aims to find a hyperplane that best separates the data into different classes.

SVM is effective in high- dimensional spaces and robust against overfitting.

-We used the following Hyperparameters:

{C, Kernel, Gamma}

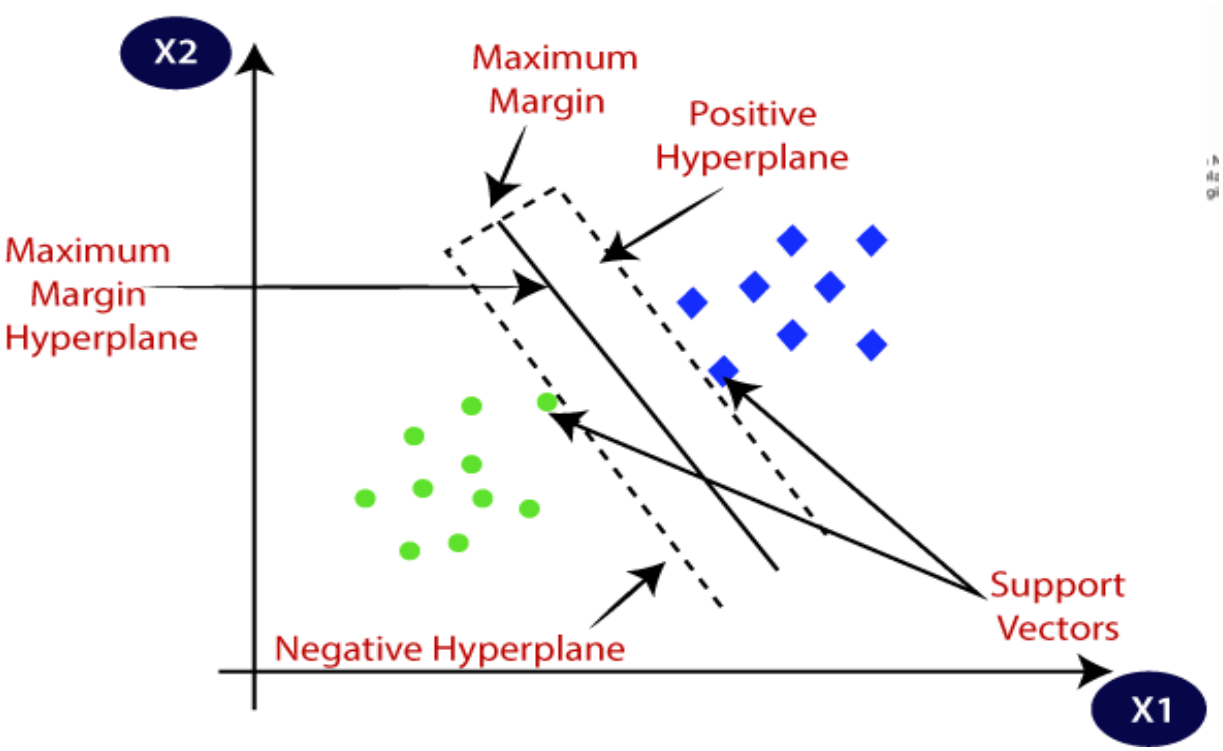


Figure 6: SVM Algorithm

Applying Grid Search to find the best model version and the best hyperparameters:

Support Vector Machine (SVM)

Its primary purpose is to find a hyperplane that best separates data points into different classes.

Applying Grid Search to find the best model version and the best hyperparameters

```
[ ] from sklearn.svm import SVC
    from sklearn.model_selection import GridSearchCV

    # Create a Support Vector Machine Classifier object
    svm_classifier = SVC(random_state = 42)

    # Define the hyperparameters and their possible values to search
    parameters = [{'C': [0.25, 0.5, 0.75, 1], 'kernel': ['linear']},
                  {'C': [0.25, 0.5, 0.75, 1],'kernel': ['rbf'], 'gamma': ['scale', 'auto', 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]}]

    # Create the Grid Search object
    grid_search = GridSearchCV(estimator = svm_classifier,
                               param_grid = parameters,
                               cv = 5,
                               scoring = 'accuracy',
                               n_jobs = -1)

    # Fit the Grid Search to the train data
    grid_search.fit(X_train, y_train)

    # Get the best hyperparameters found
    best_hyperparameters = grid_search.best_params_
    print("Best Hyperparameters:", best_hyperparameters)

    # Get the best model version
    best_svm_classifier = grid_search.best_estimator_
    print(best_svm_classifier)

    # Print the best accuracy found
    best_accuracy = grid_search.best_score_
    print(f'Best Accuracy: {best_accuracy*100:.2f} %')
```

```
Best Hyperparameters: {'C': 1, 'kernel': 'linear'}
SVC(C=1, kernel='linear', random_state=42)
Best Accuracy: 98.67 %
```

svm_classifier = SVC(random_state=42)

-Initialize an SVC object named svm_classifier.

-Set the random_state parameter to 42 for reproducibility.

parameters = {...}

-Define a dictionary named “parameters” containing different hyperparameters and their potential values. These hyperparameters will be tuned during the grid search process.

grid_search = GridSearchCV(estimator= svm_classifier, param_grid=parameters, cv=5, scoring='accuracy', n_jobs=-1)

-Initialize a GridSearchCV object named “grid_search”.

-Pass the following parameters:
(estimator, param_grid, cv, scoring, n_jobs)

grid_search.fit(X_train, y_train)

-Fit the grid_search object to the training data (X_train and y_train).

best_hyperparameters = grid_search.best_params_

-Retrieve the best hyperparameters found during the grid search and store them in the “best_hyperparameters” variable.

print("Best Hyperparameters:", best_hyperparameters)

-Print the best hyperparameters found during the grid search.

best_svm_classifier = grid_search.best_estimator_

-Retrieve the best SVC version found during the grid search and store it in the “best_svm_classifier” variable.

print(best_svm_classifier)

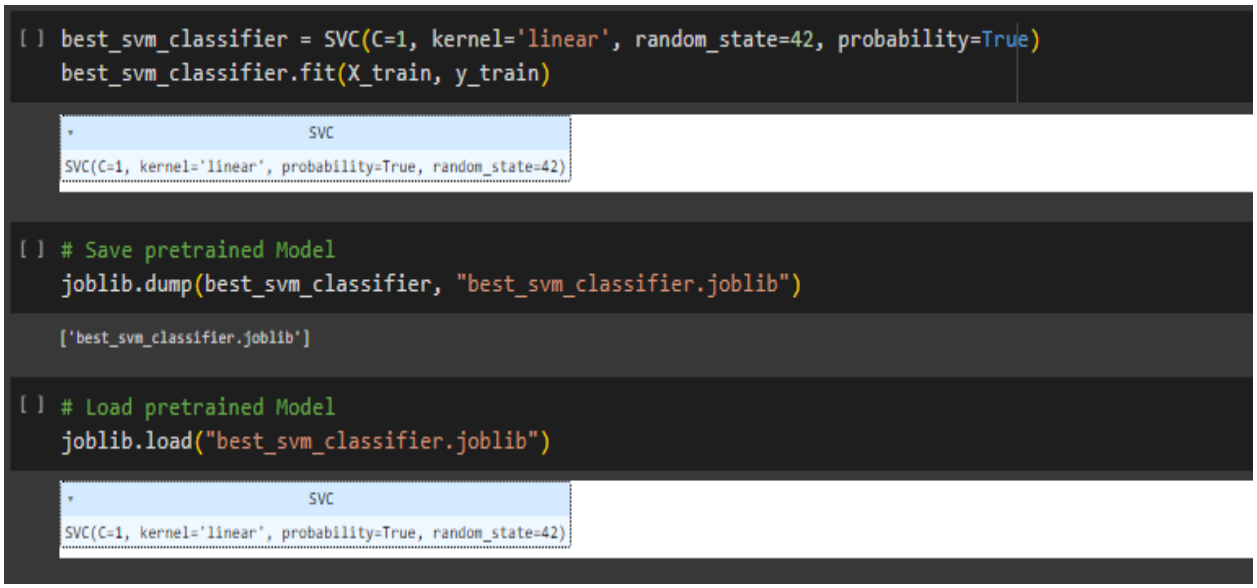
-Print the details of the best SVC version found during the grid search.

best_accuracy = grid_search.best_score_

-Retrieve the best mean cross-validated accuracy score found during the grid search and store it in the “best_accuracy” variable.

print(f'Best Accuracy: {best_accuracy*100:.2f} %')

-Print the best mean cross-validated accuracy score as a percentage with two decimal places.



best_svm_classifier = SVC(C=1, kernel='linear', random_state=42, probability=True)

-Create an instance of SVC class called “best_svm_classifier”, with the best parameters (C=1, kernel='linear', random_state=42, probability=True).

best_svm_classifier.fit(X_train, y_train)

This line trains the best version of SVC on the training data (X_train) and (y_train).

joblib.dump(best_svm_classifier, "best_svm_classifier.joblib")

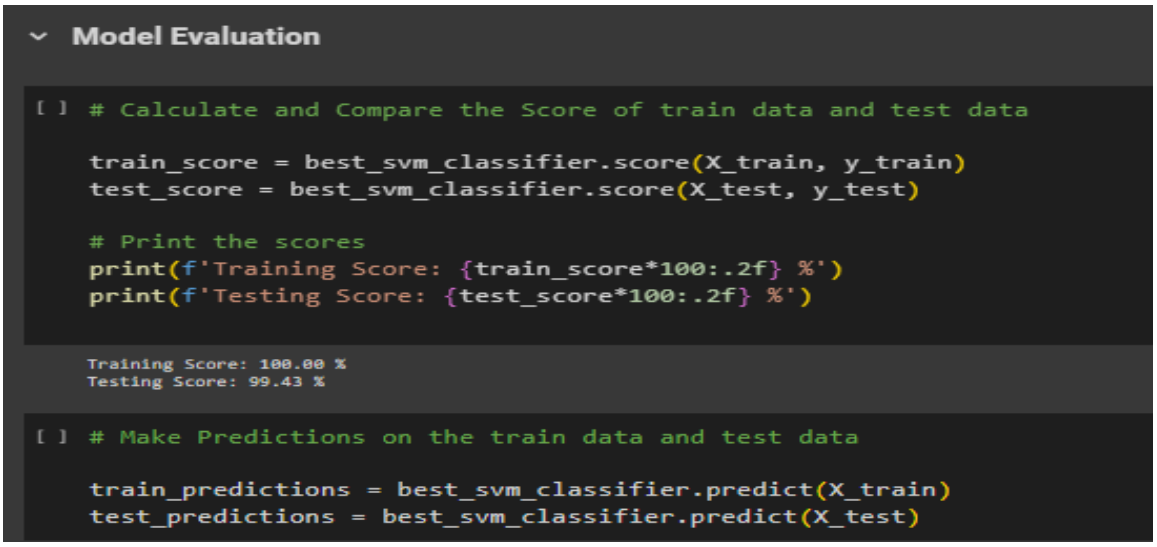
-This line is using the joblib.dump() function to save the “best_svm_classifier” object to a file named “best_svm_classifier.joblib”.

-The best_svm_classifier object will be serialized and saved to the specified file location to load and deploy it later.

joblib.load("best_svm_classifier.joblib")

-This line of code is using the joblib.load() function to load the saved best_svm_classifier object from the file “best_svm_classifier.joblib” to load and deploy it later in Flask.

Model Evaluation:



train_score = best_svm_classifier.score(X_train, y_train)

-Calculate the mean accuracy of the best SVC model (best_svm_classifier) on the train data (X_train, y_train).

-This measures the performance of the model on seen data.

test_score = best_svm_classifier.score(X_test, y_test)

-Calculate the mean accuracy of the best SVC model (best_svm_classifier) on the test data (X_test, y_test).

-This measures the performance of the model on unseen data.

print(f'Training Score: {train_score:.2f} %')

-Print the train accuracy score as a percentage with two decimal places.

print(f'Testing Score: {test_score:.2f} %')

-Print the test accuracy score as a percentage with two decimal places.

train_predictions = best_svm_classifier.predict(X_train)

-Predict the class labels for the training data (X_train) using the best SVC model (best_svm_classifier).

test_predictions = best_svm_classifier.predict(X_test)

-Predict the class labels for the testing data (X_test) using the best SVC model (best_svm_classifier).

```
[ ] # Calculate and Compare the Accuracy for training and testing data
    from sklearn.metrics import accuracy_score

    train_accuracy = accuracy_score(y_train, train_predictions)
    test_accuracy = accuracy_score(y_test, test_predictions)

    # Print the accuracies
    print(f'Training Accuracy: {train_accuracy*100:.2f} %')
    print(f'Testing Accuracy: {test_accuracy*100:.2f} %')

    Training Accuracy: 100.00 %
    Testing Accuracy: 99.43 %
```

train_accuracy= accuracy_score(y_train, train_predictions)

-Calculate the accuracy classification score for the train data, by comparing the true labels (y_train) with the predicted labels (train_predictions).

-This measures the performance of the model on seen data.

test_accuracy = accuracy_score(y_test, test_predictions)

-Calculate the accuracy classification score for the test data, by comparing the true labels (y_test) with the predicted labels (test_predictions).

-This measures the performance of the model on unseen data.

print(f'Training Accuracy: {train_accuracy*100:.2f} %')

This line prints the training accuracy as a percentage with two decimal places.

print(f'Testing Accuracy: {test_accuracy*100:.2f} %')

This line prints the testing accuracy as a percentage with two decimal places.

```
# Make the Confusion Matrix
from sklearn.metrics import confusion_matrix

cm_3 = confusion_matrix(y_test, test_predictions)

# Print the Confusion Matrix
print(cm_3)
```

```
[[36  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 31  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 36  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 38  0  1  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 38  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  1  0 42  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 38  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 40  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 32  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 32  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 31  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 26  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0 40  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 38  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 57  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 39  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 34  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 36  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 34  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 32  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 36]]
```

cm_3 = confusion_matrix(y_test, test_predictions)

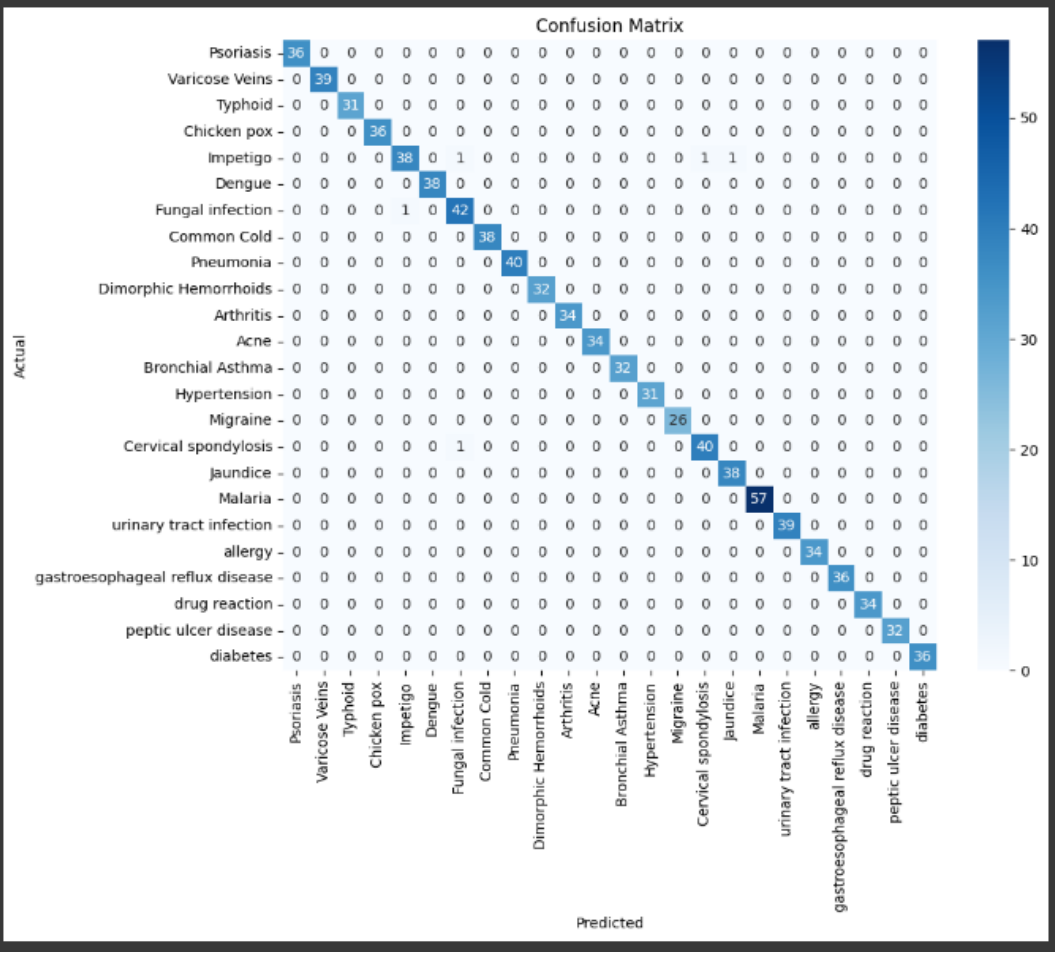
-Calculate the confusion matrix based on the true labels (y_test) and the predicted labels (test_predictions).

print(cm_3)

-Print the computed confusion matrix.

-This confusion matrix printed as a 24x24 matrix (for multiclass classification), where each row represents the instances in an actual class, and each column represents the instances in a predicted class.

```
[ ] # Confusion Matrix Visualization (Heatmap)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm_3, annot=True, fmt='d', cmap='Blues', xticklabels=y.unique(), yticklabels=y.unique())
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()
```



-This code generates a heatmap visualization of a confusion matrix using the matplotlib and seaborn libraries.

-The rows represent the actual classes, and the columns represent the predicted classes.

-The color intensity of each cell indicates the number of data points that belong to the corresponding combination of actual and predicted classes.

```
[ ] print(f"The Accuracy = {accuracy_score(y_test, test_predictions)*100:.2f} %")

    The Accuracy = 99.43 %

[ ] print(classification_report(y_test, test_predictions))
```

	precision	recall	f1-score	support
Acne	1.00	1.00	1.00	36
Arthritis	1.00	1.00	1.00	39
Bronchial Asthma	1.00	1.00	1.00	31
Cervical spondylosis	1.00	1.00	1.00	36
Chicken pox	0.97	0.93	0.95	41
Common Cold	1.00	1.00	1.00	38
Dengue	0.95	0.98	0.97	43
Dimorphic Hemorrhoids	1.00	1.00	1.00	38
Fungal infection	1.00	1.00	1.00	40
Hypertension	1.00	1.00	1.00	32
Impetigo	1.00	1.00	1.00	34
Jaundice	1.00	1.00	1.00	34
Malaria	1.00	1.00	1.00	32
Migraine	1.00	1.00	1.00	31
Pneumonia	1.00	1.00	1.00	26
Psoriasis	0.98	0.98	0.98	41
Typhoid	0.97	1.00	0.99	38
Varicose Veins	1.00	1.00	1.00	57
allergy	1.00	1.00	1.00	39
diabetes	1.00	1.00	1.00	34
drug reaction	1.00	1.00	1.00	36
gastroesophageal reflux disease	1.00	1.00	1.00	34
peptic ulcer disease	1.00	1.00	1.00	32
urinary tract infection	1.00	1.00	1.00	36
accuracy			0.99	878
macro avg	0.99	0.99	0.99	878
weighted avg	0.99	0.99	0.99	878

print(f"The Accuracy = {accuracy_score(y_test, test_predictions)*100:.2f} %")

-Print the accuracy classification score of SVC model on the test dataset as a percentage with two decimal places.

-Calculate the accuracy as the fraction of correctly predicted instances out of the total number of instances.

print(classification_report(y_test, test_predictions))

-Print the classification report for the predictions made by SVC model on the test dataset.

-The classification report includes various evaluation metrics such as **precision, recall, F1-score, and support** for each class, as well as averages across all classes.

```python
# Create an empty list to store the comparison results
comparison_results = []

# Comparison between predicted and true labels
for pred, true in zip(test_predictions, y_test):
    comparison_results.append({'Predicted': pred, 'True': true})

# Print the comparison results in dictionary form
for result in comparison_results:
    print(result)
```

```
{'Predicted': 'drug reaction', 'True': 'drug reaction'}
{'Predicted': 'Typhoid', 'True': 'Typhoid'}
{'Predicted': 'urinary tract infection', 'True': 'urinary tract infection'}
{'Predicted': 'Impetigo', 'True': 'Impetigo'}
{'Predicted': 'Dengue', 'True': 'Dengue'}
{'Predicted': 'Common Cold', 'True': 'Common Cold'}
{'Predicted': 'peptic ulcer disease', 'True': 'peptic ulcer disease'}
{'Predicted': 'Hypertension', 'True': 'Hypertension'}
{'Predicted': 'Pneumonia', 'True': 'Pneumonia'}
{'Predicted': 'Pneumonia', 'True': 'Pneumonia'}
{'Predicted': 'Dengue', 'True': 'Dengue'}
{'Predicted': 'Pneumonia', 'True': 'Pneumonia'}
{'Predicted': 'Migraine', 'True': 'Migraine'}
{'Predicted': 'diabetes', 'True': 'diabetes'}
{'Predicted': 'Varicose Veins', 'True': 'Varicose Veins'}
{'Predicted': 'peptic ulcer disease', 'True': 'peptic ulcer disease'}
{'Predicted': 'Dengue', 'True': 'Dengue'}
{'Predicted': 'Dimorphic Hemorrhoids', 'True': 'Dimorphic Hemorrhoids'}
{'Predicted': 'Common Cold', 'True': 'Common Cold'}
{'Predicted': 'Arthritis', 'True': 'Arthritis'}
{'Predicted': 'Chicken pox', 'True': 'Chicken pox'}
{'Predicted': 'Bronchial Asthma', 'True': 'Bronchial Asthma'}
{'Predicted': 'Dimorphic Hemorrhoids', 'True': 'Dimorphic Hemorrhoids'}
{'Predicted': 'urinary tract infection', 'True': 'urinary tract infection'}
{'Predicted': 'allergy', 'True': 'allergy'}
{'Predicted': 'Dengue', 'True': 'Psoriasis'}
{'Predicted': 'Hypertension', 'True': 'Hypertension'}
{'Predicted': 'Acne', 'True': 'Acne'}
{'Predicted': 'Impetigo', 'True': 'Impetigo'}
{'Predicted': 'Impetigo', 'True': 'Impetigo'}
{'Predicted': 'Hypertension', 'True': 'Hypertension'}
{'Predicted': 'Pneumonia', 'True': 'Pneumonia'}
{'Predicted': 'Hypertension', 'True': 'Hypertension'}
{'Predicted': 'allergy', 'True': 'allergy'}
{'Predicted': 'diabetes', 'True': 'diabetes'}
{'Predicted': 'Common Cold', 'True': 'Common Cold'}
{'Predicted': 'Bronchial Asthma', 'True': 'Bronchial Asthma'}
{'Predicted': 'Dengue', 'True': 'Dengue'}
{'Predicted': 'Common Cold', 'True': 'Common Cold'}
{'Predicted': 'Arthritis', 'True': 'Arthritis'}
{'Predicted': 'Migraine', 'True': 'Migraine'}
{'Predicted': 'Psoriasis', 'True': 'Psoriasis'}
{'Predicted': 'Impetigo', 'True': 'Impetigo'}
{'Predicted': 'diabetes', 'True': 'diabetes'}
{'Predicted': 'Common Cold', 'True': 'Common Cold'}
{'Predicted': 'Malaria', 'True': 'Malaria'}
{'Predicted': 'Psoriasis', 'True': 'Psoriasis'}
{'Predicted': 'Migraine', 'True': 'Migraine'}
{'Predicted': 'Acne', 'True': 'Acne'}
{'Predicted': 'Dengue', 'True': 'Dengue'}
{'Predicted': 'Psoriasis', 'True': 'Psoriasis'}
{'Predicted': 'Fungal infection', 'True': 'Fungal infection'}
```

comparison_results = []

Initialize an empty list named “comparison_results” where the results of the comparisons will be stored.

for pred, true in zip(test_predictions, y_test):

Initiate a loop that iterates through each pair of elements from two lists, test_predictions (the predicted labels) and y_test (the true labels), simultaneously.

comparison_results.append({'Predicted': pred, 'True': true})

Inside the loop, a dictionary is created for each pair of predicted and true labels. This dictionary has two key-value pairs: 'Predicted' key with the value of the predicted label (pred), and 'True' key with the value of the true label (true). This dictionary is then appended to the “comparison_results” empty list.

for result in comparison_results:

Initiate another loop that iterates through each dictionary in the “comparison_results” list.

print(result)

Inside this loop, each dictionary (result) is printed. Since each dictionary contains the predicted and true labels, when printed, it will display the comparison result in a dictionary format.

Model validation:

Model Validation

```
# Validation Test #

text_before = "The curious cat chased the elusive shadow through the moonlit garden."

# Cleaning
text_after = preprocess_text(text_before)

print(text_before)
print(text_after)

# Vectorization
tfidf_vectorizer

text_after = tfidf_vectorizer.transform([text_after]).toarray()

print(text_after.reshape(-1,1))

# Prediction
disease = best_svm_classifier.predict(text_after)
print(disease)

disease_proba = best_svm_classifier.predict_proba(text_after)
print(disease_proba)

score = disease_proba[0][np.argmax(np.array(disease_proba[0]),axis=0)]
print(score)

if score > 0.30:
    print(disease) # disease
else:
    print("Please enter valid symptoms") # error message
```

```
The curious cat chased the elusive shadow through the moonlit garden.
curious cat chased elusive shadow moonlit garden
[[0.]
 [0.]
 [0.]
 ...
 [0.]
 [0.]
 [0.]]
['diabetes']
[[0.00303707 0.00359442 0.01082831 0.00410722 0.06362164 0.01224696
  0.04066682 0.00384638 0.00927622 0.01844921 0.02907725 0.0025507
  0.00408447 0.01685776 0.00584279 0.13185572 0.08373248 0.03601206
  0.02662667 0.17480667 0.14574545 0.01394324 0.11360237 0.0455881 ]]
0.17480667125798208
Please enter valid symptoms
```

text_before = "any symptoms description entered by user"

-Assign the input text string to the variable “text_before”. This variable holds the original text before any preprocessing or prediction.

text_after = preprocess_text(text_before)

-Apply the “preprocess_text” function to the “text_before” string to perform NLP steps. The processed text is then assigned to the variable “text_after”.

print(text_before)

-Print the original text (text_before).

print(text_after)

-Print the processed text after applying NLP steps.

tfidf_vectorizer

-This line calls the “tfidf_vectorizer” object.

text_after = tfidf_vectorizer.transform([text_after]).toarray()

-Transform the processed text (text_after) into a TF-IDF vector representation using the tfidf_vectorizer object. The transform method converts the text into numerical features, and “toarray()” converts the resulting sparse matrix into a dense NumPy array.

print(text_after.reshape(-1,1))

-Print the TF-IDF vector representation of the processed text (text_after), and reshape(-1,1) method reshapes the array to have one column while maintaining the same number of rows.

disease = best_svm_classifier.predict(text_after)

-Predict the disease for the processed text (text_after) using the best SVC model (best_svm_classifier).

print(disease)

-Print the predicted label (disease) for the processed text.

disease_proba = best_svm_classifier.predict_proba(text_after)

- “predict_proba()” to obtain the probabilities of each class label/disease for the processed text (text_after) using the best SVC model (best_svm_classifier).

print(disease_proba)

-Print the 2-dimensional array containing the probabilities of each disease predicted by the SVC model for the processed text.

score=disease_proba[0][np.argmax(np.array(disease_proba[0]),axis=0)]

- “np.array(disease_proba[0])” converts the 1-dimensional array “disease_proba[0]” into a NumPy array, disease_proba[0] selects the probabilities associated with the first instance (or sample) in the array.

- “np.argmax()” function returns the index of the maximum value in the array “disease_proba[0]” along a specified axis “axis=0”.

- Generally, this line calculates the highest probability of disease for the first instance in the “disease_proba” array, using the index obtained from “np.argmax()”, and assigns it to the variable “score”.

print(score)

-Print the highest probability of disease for the processed text.

print(disease) if (score > 0.30) else print("Please enter valid symptoms")

If the highest probability (score) is greater than 0.30, print the predicted disease. Otherwise, print an error message indicating that the symptoms are not valid.

4-KNeighborsClassifier:

K-Nearest Neighbors is a simple and intuitive supervised machine learning algorithm used for classification tasks.

It determines the class of a data point based on the majority class among its k nearest neighbors in the feature space assuming that similar instances in the feature space should share similar class labels.

-We used the following Hyperparameters:

{ n_neighbors , weights , algorithm , P }

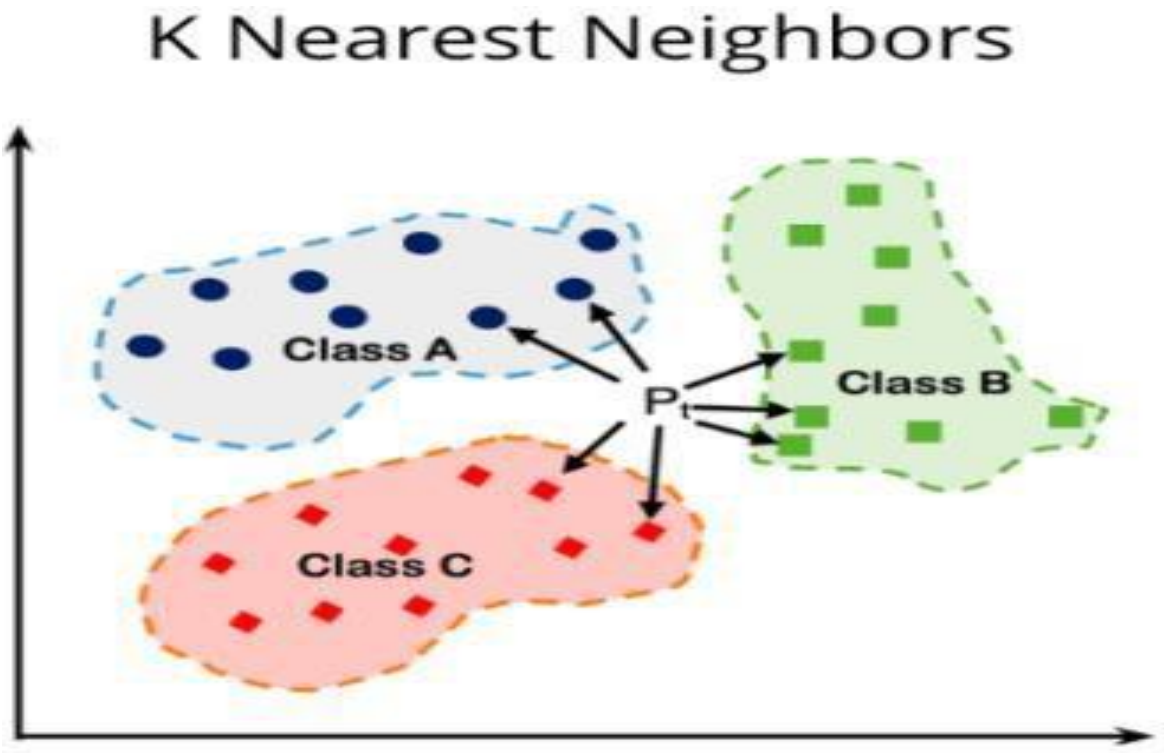


Figure 7: K-NN Algorithm

Applying Grid Search to find the best model version and the best hyperparameters:

KNeighborsClassifier

In the k-NN algorithm, the "k" represents the number of nearest neighbors considered for making predictions.

Applying Grid Search to find the best model version and the best hyperparameters

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    from sklearn.model_selection import GridSearchCV

    # Create a K-Neighbors Classifier object
    knn_classifier = KNeighborsClassifier()

    # Define the hyperparameters and their possible values to search
    parameters = {
        'n_neighbors': [3, 5, 7, 9],
        'weights': ['uniform', 'distance'],
        'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
        'p': [1, 2]
    }

    # Create the Grid Search object
    grid_search = GridSearchCV(estimator = knn_classifier,
                               param_grid = parameters,
                               cv = 5,
                               scoring = 'accuracy',
                               n_jobs = -1)

    # Fit the Grid Search to the train data
    grid_search.fit(X_train, y_train)

    # Get the best hyperparameters found
    best_hyperparameters = grid_search.best_params_
    print("Best Hyperparameters:", best_hyperparameters)

    # Get the best model version
    best_knn_classifier = grid_search.best_estimator_
    print(best_knn_classifier)

    # Print the best accuracy found
    best_accuracy = grid_search.best_score_
    print(f'Best Accuracy: {best_accuracy*100:.2f} %')
```

```
Best Hyperparameters: {'algorithm': 'auto', 'n_neighbors': 3, 'p': 2, 'weights': 'distance'}
KNeighborsClassifier(n_neighbors=3, weights='distance')
Best Accuracy: 96.92 %
```

knn_classifier = KNeighborsClassifier()

-Initialize a KNeighborsClassifier object named knn_classifier.

parameters = {...}

-Define a dictionary named “parameters” containing different hyperparameters and their potential values. These hyperparameters will be tuned during the grid search process.

grid_search = GridSearchCV(estimator= knn_classifier, param_grid=parameters, cv=5, scoring='accuracy', n_jobs=-1)

-Initialize a GridSearchCV object named “grid_search”.

-Pass the following parameters:
(estimator, param_grid, cv, scoring, n_jobs)

grid_search.fit(X_train, y_train)

-Fit the grid_search object to the training data (X_train and y_train).

best_hyperparameters = grid_search.best_params_

-Retrieve the best hyperparameters found during the grid search and store them in the “best_hyperparameters” variable.

print("Best Hyperparameters:", best_hyperparameters)

-Print the best hyperparameters found during the grid search.

best_knn_classifier = grid_search.best_estimator_

-Retrieve the best KNeighborsClassifier version found during the grid search and store it in the “best_knn_classifier” variable.

print(best_knn_classifier)

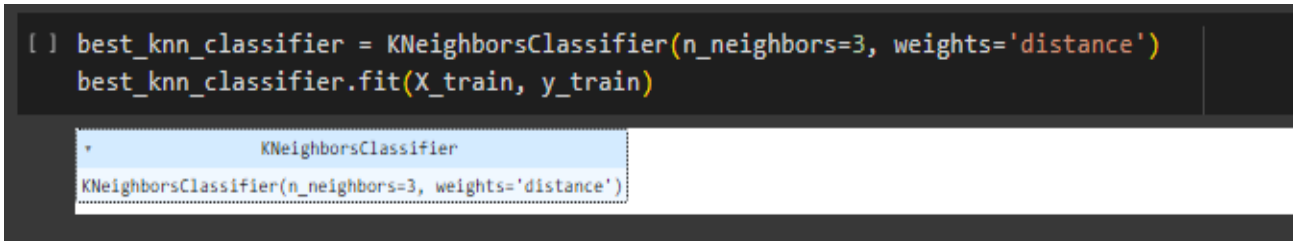
-Print the details of the best KNeighborsClassifier version found during the grid search.

best_accuracy = grid_search.best_score_

-Retrieve the best mean cross-validated accuracy score found during the grid search and store it in the “best_accuracy” variable.

print(f'Best Accuracy: {best_accuracy*100:.2f} %')

-Print the best mean cross-validated accuracy score as a percentage with two decimal places.



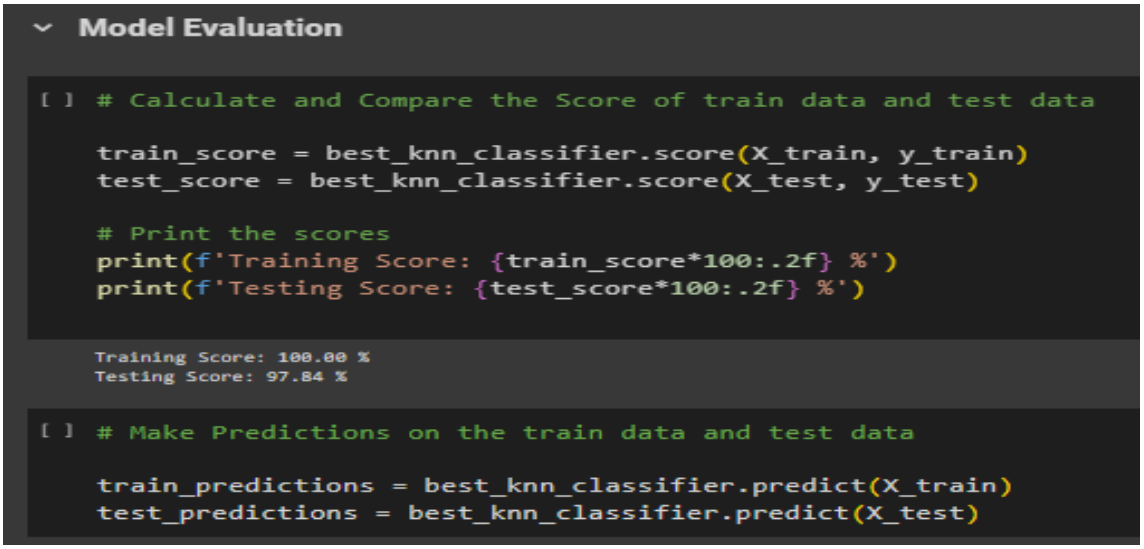
best_knn_classifier = KNeighborsClassifier(n_neighbors=3, weights='distance')

-Create an instance of KNeighborsClassifier class called “best_knn_classifier”, with the best parameters (n_neighbors=3, weights='distance').

best_knn_classifier.fit(X_train, y_train)

This line trains the best version of KNeighborsClassifier on the training data (X_train) and (y_train).

Model Evaluation:



train_score = best_knn_classifier.score(X_train, y_train)

-Calculate the mean accuracy of the best KNeighborsClassifier model (best_knn_classifier) on the train data (X_train, y_train).

-This measures the performance of the model on seen data.

test_score = best_knn_classifier.score(X_test, y_test)

-Calculate the mean accuracy of the best KNeighborsClassifier model (best_knn_classifier) on the test data (X_test, y_test).

-This measures the performance of the model on unseen data.

print(f'Training Score: {train_score:.2f} %')

-Print the train accuracy score as a percentage with two decimal places.

print(f'Testing Score: {test_score:.2f} %')

-Print the test accuracy score as a percentage with two decimal places.

train_predictions = best_knn_classifier.predict(X_train)

-Predict the class labels for the training data (X_train) using the best KNeighborsClassifier model (best_knn_classifier).

test_predictions = best_knn_classifier.predict(X_test)

-Predict the class labels for the testing data (X_test) using the best KNeighborsClassifier model (best_knn_classifier).

```
# Calculate and Compare the Accuracy for training and testing data
from sklearn.metrics import accuracy_score

train_accuracy = accuracy_score(y_train, train_predictions)
test_accuracy = accuracy_score(y_test, test_predictions)

# Print the accuracies
print(f'Training Accuracy: {train_accuracy*100:.2f} %')
print(f'Testing Accuracy: {test_accuracy*100:.2f} %')
```

```
Training Accuracy: 100.00 %
Testing Accuracy: 97.84 %
```

train_accuracy= accuracy_score(y_train, train_predictions)

-Calculate the accuracy classification score for the train data, by comparing the true labels (y_train) with the predicted labels (train_predictions).

-This measures the performance of the model on seen data.

test_accuracy = accuracy_score(y_test, test_predictions)

-Calculate the accuracy classification score for the test data, by comparing the true labels (y_test) with the predicted labels (test_predictions).

-This measures the performance of the model on unseen data.

print(f'Training Accuracy: {train_accuracy*100:.2f} %')

This line prints the training accuracy as a percentage with two decimal places.

print(f'Testing Accuracy: {test_accuracy*100:.2f} %')

This line prints the testing accuracy as a percentage with two decimal places.

```
[ ] # Make the Confusion Matrix
    from sklearn.metrics import confusion_matrix

    cm_4 = confusion_matrix(y_test, test_predictions)

    # Print the Confusion Matrix
    print(cm_4)
```

```
[[36  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 31  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 36  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 35  0  2  0  0  0  0  0  0  0  2  2  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 38  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  1  0 38  0  1  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 38  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 40  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  1  0  0  0  0  0 31  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 32  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 31  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  1  0 25  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 40  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  1  0  2  0  0  0  0  0  0  0  0  1 34  0  0  0  0  0  0  0]
 [ 0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0 56  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 39  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 34  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 36  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 34  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 32  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 36]]
```

cm_4 = confusion_matrix(y_test, test_predictions)

-Calculate the confusion matrix based on the true labels (y_test) and the predicted labels (test_predictions).

print(cm_4)

-Print the computed confusion matrix.

-This confusion matrix printed as a 24x24 matrix (for multiclass classification), where each row represents the instances in an actual class, and each column represents the instances in a predicted class.

```
[ ] print(f"The Accuracy = {accuracy_score(y_test, test_predictions)*100:.2f} %")

    The Accuracy = 97.84 %

print(classification_report(y_test, test_predictions))
```

	precision	recall	f1-score	support
Acne	1.00	1.00	1.00	36
Arthritis	1.00	1.00	1.00	39
Bronchial Asthma	1.00	1.00	1.00	31
Cervical spondylosis	0.97	1.00	0.99	36
Chicken pox	0.92	0.85	0.89	41
Common Cold	1.00	1.00	1.00	38
Dengue	0.90	0.88	0.89	43
Dimorphic Hemorrhoids	1.00	1.00	1.00	38
Fungal infection	0.98	1.00	0.99	40
Hypertension	1.00	0.97	0.98	32
Impetigo	1.00	1.00	1.00	34
Jaundice	1.00	1.00	1.00	34
Malaria	0.97	1.00	0.98	32
Migraine	1.00	1.00	1.00	31
Pneumonia	1.00	0.96	0.98	26
Psoriasis	0.93	0.98	0.95	41
Typhoid	0.94	0.89	0.92	38
Varicose Veins	0.95	0.98	0.97	57
allergy	1.00	1.00	1.00	39
diabetes	1.00	1.00	1.00	34
drug reaction	1.00	1.00	1.00	36
gastroesophageal reflux disease	1.00	1.00	1.00	34
peptic ulcer disease	0.97	1.00	0.98	32
urinary tract infection	1.00	1.00	1.00	36
accuracy			0.98	878
macro avg	0.98	0.98	0.98	878
weighted avg	0.98	0.98	0.98	878

print(f"The Accuracy = {accuracy_score(y_test, test_predictions)*100:.2f} %")

-Print the accuracy classification score of KNeighborsClassifier model on the test dataset as a percentage with two decimal places.

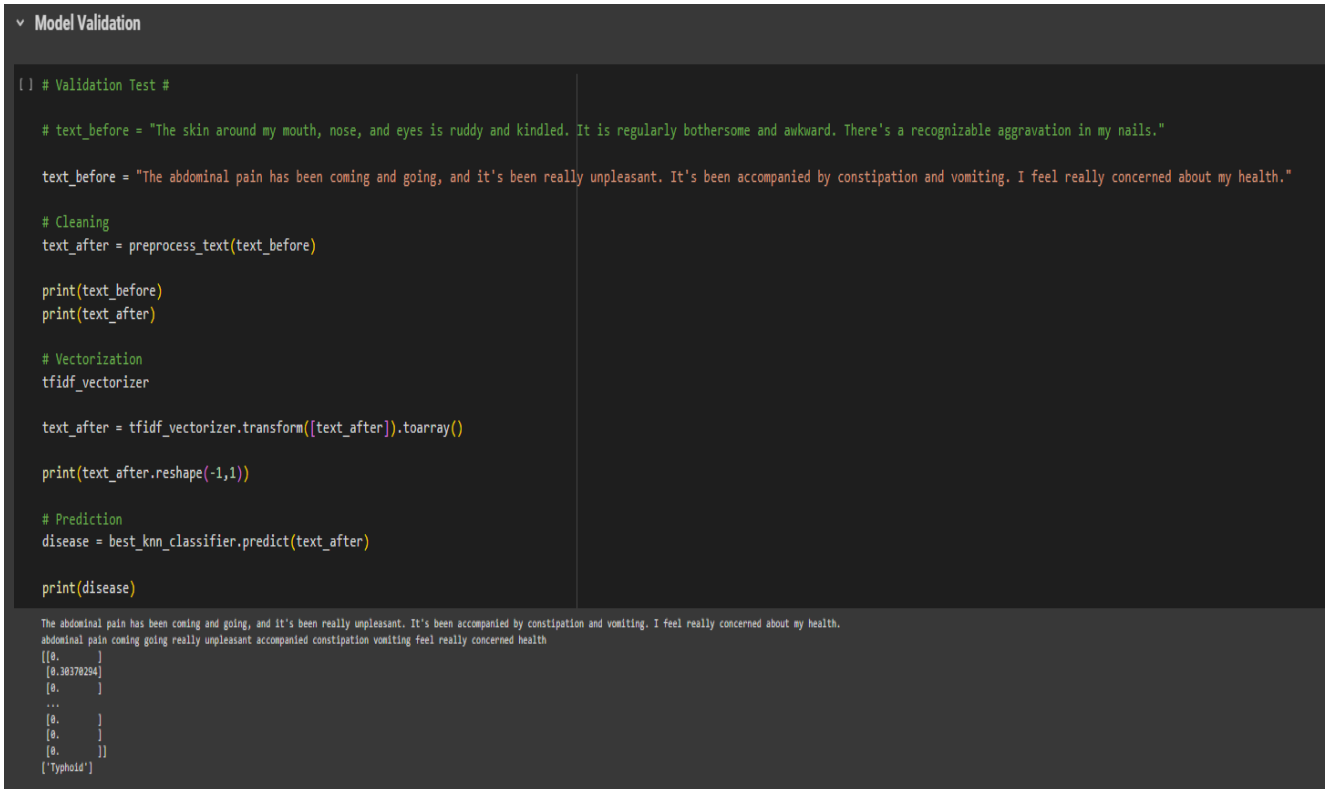
-Calculate the accuracy as the fraction of correctly predicted instances out of the total number of instances.

print(classification_report(y_test, test_predictions))

-Print the classification report for the predictions made by KNeighborsClassifier model on the test dataset.

-The classification report includes various evaluation metrics such as **precision, recall, F1-score, and support** for each class, as well as averages across all classes.

Model validation:



text_before = "any symptoms description entered by user"

-Assign the input text string to the variable “text_before”. This variable holds the original text before any preprocessing or prediction.

text_after = preprocess_text(text_before)

-Apply the “preprocess_text” function to the “text_before” string to perform NLP steps. The processed text is then assigned to the variable “text_after”.

print(text_before)

-Print the original text (text_before).

print(text_after)

-Print the processed text after applying NLP steps.

tfidf_vectorizer

-This line calls the “tfidf_vectorizer” object.

text_after = tfidf_vectorizer.transform([text_after]).toarray()

-Transform the processed text (text_after) into a TF-IDF vector representation using the tfidf_vectorizer object. The transform method converts the text into numerical features, and "toarray()" converts the resulting sparse matrix into a dense NumPy array.

print(text_after.reshape(-1,1))

-Print the TF-IDF vector representation of the processed text (text_after), and reshape(-1,1) method reshapes the array to have one column while maintaining the same number of rows.

disease = best_knn_classifier.predict(text_after)

-Predict the disease for the processed text (text_after) using the best KNeighborsClassifier model (best_knn_classifier).

print(disease)

-Print the predicted label (disease) for the processed text.

Comparison between the ML models:

Classification Algorithms	Accuracy
Decision Tree	89.52 %
Random Forest	98.06 %
SVM	99.43 %
K-NN	97.84 %

Table 2: Comparison between the ML models

Flask:

app.py

C: > Users > CS > Desktop > Flask-master > app.py > ...

```
1  from flask import Flask, request, jsonify
2  from flask_cors import CORS
3  import numpy as np
4  import string
5  import nltk
6  import joblib
7  from nltk.tokenize import word_tokenize
8  from nltk.corpus import stopwords
9  from nltk.stem import WordNetLemmatizer
10 from googletrans import Translator
11 import logging
12
```

1- **The first two lines import the necessary tools to handle flask application:** ("**Flask**": A web framework for Python, "**request**": To handle HTTP requests, "**jsonify**": To convert Python dictionaries to JSON responses, "**CORS**": Cross-Origin Resource Sharing, which allows restricted resources on a web page to be requested from another domain.

2- **from googletrans import Translator:** This imports the "**Translator**" class from the "googletrans" library, which is used for language translation using Google Translate.

3- **The rest of the lines import the necessary tools to handle Machine learning and NLP tasks, as previously explained.**

```
11
12 # Initialize
13 app = Flask(__name__)
14 CORS(app)
15
```

1- **The first line Initialize Flask App:** An instance of the Flask class is created with the name "app".

2- **The second line Enable CORS:** CORS is enabled for the Flask app to allow cross-origin requests.

```
15
16 nltk.download('punkt')
17 nltk.download('stopwords')
18 nltk.download('wordnet')
19
```

-NLTK Setup: Downloads necessary NLTK resources like tokenizers, stopwords, and WordNet, as previously explained.

```
19
20  # Translation functions
21  def translate_to_arabic(text):
22      translator = Translator()
23      translation = translator.translate(text, src='en', dest='ar')
24      return translation.text
25
26  def translate_to_english(text):
27      translator = Translator()
28      translation = translator.translate(text, src='ar', dest='en')
29      return translation.text
30
```

-These two functions are used for translating text between English and Arabic languages using the Google Translate API through the "googletrans" library. Both functions utilize the same "Translator" object for translation, and they use different source and destination languages to achieve translation in different directions.

```
30
31  # NLP
32  def lowercase_text(text):
33      return text.lower()
34
35  def remove_punctuation(text):
36      translator = str.maketrans('', '', string.punctuation)
37      text_without_punct = text.translate(translator).strip()
38      return text_without_punct
39
40  def tokenize_text(text):
41      tokens = word_tokenize(text)
42      return tokens
43
44  def remove_stopwords(tokens):
45      stop_words = set(stopwords.words('english'))
46      filtered_tokens = [token for token in tokens if token.isalpha() and token not in stop_words]
47      return filtered_tokens
48
49  def lemmatize_text(tokens):
50      lemmatizer = WordNetLemmatizer()
51      lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
52      return lemmatized_tokens
53
54  # NLP Container function
55  def preprocess_text(text):
56      text = lowercase_text(text)
57      text = remove_punctuation(text)
58      tokens = tokenize_text(text)
59      tokens = remove_stopwords(tokens)
60      tokens = lemmatize_text(tokens)
61      preprocessed_text = ' '.join(tokens)
62      return preprocessed_text
63
```

1- **The first 5 NLP Functions defined for text preprocessing using NLTK, as previously explained.**

2- **The last function** "**preprocess_text**": combines the preprocessing steps into a single function.

```
63
64  # Load the TF-IDF vectorizer
65  tfidf_vectorizer = joblib.load('tfidf_vectorizer.joblib')
66
67  # Load the SVM model
68  model = joblib.load('best_svm_classifier.joblib')
69
```

1- **The first line:** loads the TF-IDF vectorizer "**tfidf_vectorizer**" using joblib.

2- **The second line:** loads the SVM classifier model "**model**" using joblib.

```python

@app.route('/predict', methods=['POST'])
def predict():
    # Get input data from request
    text = request.json.get('text')

    # Detect input language
    translator = Translator()
    lang = translator.detect(text).lang

    # Translate input to English if it's in Arabic
    if lang == 'ar':
        translated_text = translate_to_english(text)
    else:
        translated_text = text

    # NLP
    preprocessed_text = preprocess_text(translated_text)

    # TF-IDF vectorizer
    text_vectorized = tfidf_vectorizer.transform([preprocessed_text]).toarray()

    # Prediction
    predicted_disease = model.predict(text_vectorized)[0]
    predicted_proba = model.predict_proba(text_vectorized)[0]
    score = predicted_proba[np.argmax(predicted_proba)]

    if score > 0.30:
        # Translate predicted disease to the input language
        if lang == 'ar':
            translated_disease = translate_to_arabic(predicted_disease)
            response = {'predicted': "من المحتمل أنك تعاني من " + translated_disease}
        else:
            response = {'predicted': f"Maybe you suffer from {predicted_disease}"}

        return jsonify(response), 200
    else:
        return jsonify({'error': 'Please enter valid symptoms'}), 400

```

➢ Route “/predict”:

1- **Defines a route** “/predict” that accepts POST requests.
2- **Retrieves input text** from the request JSON.
3- **Detects the language of the input text** using the “detect()” method from the “Translator” class.
4- **If** the detected language is Arabic (lang == 'ar'), it translates the input text to English using the “translate_to_english()” function.
Otherwise, it keeps the text as it is.
5- **Preprocesses the text** using the defined “preprocess_text” function.

6- **Vectorizes the pre-processed text** using the loaded TF-IDF vectorizer.
7- **Makes predictions** on the vectorized text using the loaded SVM model.
8- **If** the prediction probability "score" is above 0.30, it returns a JSON response with the predicted disease.
If the detected language is Arabic, it translates the predicted disease back to Arabic using the "translate_to_arabic()" function.
Otherwise, it returns an English response.
9- **If** the prediction probability "score" is below 0.30, it returns an error message indicating that valid symptoms should be entered.

```
107
108  if __name__ == '__main__':
109      app.run(debug=True)
110
```

➢ Run the flask App:

If the previous script is executed directly, the Flask app runs in debug mode.

➢ Flutter Application:

Splash page:

splash_page.dart

lib > screens > splash_page.dart > ...

```dart
// ignore_for_file: prefer_const_constructors, use_key_in_widget_constructors, l
import 'dart:async';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:healthcare_chatbot/screens/chatbot_page.dart';
import 'package:healthcare_chatbot/screens/onboarding_page1.dart';

class SplashPage extends StatefulWidget {
  @override
  _SplashPageState createState() => _SplashPageState();
}

class _SplashPageState extends State<SplashPage> {
  @override
  void initState() {
super.initState();
    Timer(
      Duration(seconds: 2), // Change the duration as needed
      () {
        FirebaseAuth.instance.authStateChanges().listen((user) {
          if (user != null) {
            Navigator.of(context).pushReplacement(
              MaterialPageRoute(builder: (_) => ChatbotPage()),
            );
          } else {
            Navigator.of(context).pushReplacement(
              MaterialPageRoute(builder: (_) => OnboardingPage1()),
            );
          }
        });
      },
    ); // Timer
  }

  @override
```

```dart

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          Container(
            decoration: BoxDecoration(
              gradient: LinearGradient(
                colors: [
                  Color(0xff1C5271),
                  Color(0xff040E15),
                ],
                begin: Alignment.bottomCenter,
                end: Alignment.topCenter,
                // stops: [0.02, 0.95]
              ), // LinearGradient
            ), // BoxDecoration
          ), // Container
          Center(
            child: Container(
              width: 250,
              height: 250,
              decoration: BoxDecoration(
                image: DecorationImage(
                  image: AssetImage('images/Splash_chatgpt robot.png'),
                ), // DecorationImage
              ), // BoxDecoration
            ), // Container
          ), // Center
        ],
      ), // Stack
    ); // Scaffold
  }
}

```

Summary of this page:

These icons ensure that users are directed to the next screen of the app based on their authentication status after the short splash screen is displayed and this is a that represents the splash screen of the application. It takes three seconds to appear when using the app.

The final result of this page:

Figure 8: Splash Screen

onboarding_page1:

onboarding_page1.dart

lib > screens > onboarding_page1.dart > _OnboardingPage1State > build

```dart
// ignore_for_file: prefer_const_constructors, prefer_const_literals_to_create_i
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
import 'package:flutter/material.dart';
import 'package:healthcare_chatbot/screens/onboarding_page2.dart';
import '../core/globals.dart';
import '../main.dart';
import '../services/local/cach_helper.dart';

class OnboardingPage1 extends StatefulWidget {
  const OnboardingPage1({super.key});

  @override
  State<OnboardingPage1> createState() => _OnboardingPage1State();
}

class _OnboardingPage1State extends State<OnboardingPage1> {
  int langValue = language == 'en' ? 0 : 1;
  bool isExpanded = false;
  ExpansionTileController expansionController = ExpansionTileController();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          Container(
            decoration: BoxDecoration(
              gradient: LinearGradient(
                  colors: [
                    Color(0xff1D5879),
                    Color(0xff04141D),
                  ],
                  begin: Alignment.bottomCenter,
                  end: Alignment.topCenter,
                  stops: [0.2, 0.85]), // LinearGradient
            ), // BoxDecoration
          ), // Container
          ListView(
            children: [
              SizedBox(
                height: 70,
              ), // SizedBox
              // ignore: sized_box_for_whitespace
```

onboarding_page1.dart

lib > screens > onboarding_page1.dart > _OnboardingPage1State > build

```dart
class _OnboardingPage1State extends State<OnboardingPage1> {
  Widget build(BuildContext context) {
    // ignore: sized_box_for_whitespace
    Container(
      height: 420,
      width: 400,
      child: Image.asset(
        "images/onboarding-photo.png",
        fit: BoxFit.cover,
      )), // Image.asset // Container
    Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Padding(
          padding: const EdgeInsets.only(right: 7),
          child: Icon(
            Icons.rectangle_rounded,
            size: 22,
          ), // Icon
        ), // Padding
        Padding(
          padding: const EdgeInsets.only(right: 7),
          child: Icon(
            Icons.circle,
            color: Colors.white,
            size: 17,
          ), // Icon
        ), // Padding
        Padding(
          padding: const EdgeInsets.only(right: 7),
          child: Icon(
            Icons.circle,
            color: Colors.white,
            size: 17,
          ), // Icon
        ), // Padding
      ],
    ), // Row
    SizedBox(
      height: 45,
    ), // SizedBox
```

onboarding_page1.dart

lib > screens > onboarding_page1.dart > _OnboardingPage1State > build

```dart
class _OnboardingPage1State extends State<OnboardingPage1> {
  Widget build(BuildContext context) {
                Column(
                  children: [
                    Text(
                      AppLocalizations.of(context)!.onboard_1_1,
                      style: TextStyle(color: Colors.white, fontSize: 20),
                    ), // Text
                    Text(
                      AppLocalizations.of(context)!.onboard_1_2,
                      style: TextStyle(color: Colors.white, fontSize: 20),
                    ), // Text
                  ],
                ), // Column
                SizedBox(
                  height: 50,
                ), // SizedBox
                Container(
                  margin: EdgeInsets.symmetric(horizontal: 50),
                  height: 50,
                  child: ElevatedButton(
                    onPressed: () {
                      Navigator.push(context,
                          MaterialPageRoute(builder: (context) {
                        return OnboardingPage2();
                      })); // MaterialPageRoute
                    },
                    child: Text(
                      //translation
                      AppLocalizations.of(context)!.next,
                      style: TextStyle(color: Color(0xffFFE96D), fontSize: 20),
                    ), // Text
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Color(0xff050522),
                      shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(10),
                      ), // RoundedRectangleBorder
                    ),
                  ), // ElevatedButton
                ), // Container
              ],
            ), // ListView
```

onboarding_page1.dart

lib > screens > onboarding_page1.dart > _OnboardingPage1State > build

```dart
class _OnboardingPage1State extends State<OnboardingPage1> {
  Widget build(BuildContext context) {
                Column(
                  children: [
                    Text(
                      AppLocalizations.of(context)!.onboard_1_1,
                      style: TextStyle(color: Colors.white, fontSize: 20),
                    ), // Text
                    Text(
                      AppLocalizations.of(context)!.onboard_1_2,
                      style: TextStyle(color: Colors.white, fontSize: 20),
                    ), // Text
                  ],
                ), // Column
                SizedBox(
                  height: 50,
                ), // SizedBox
                Container(
                  margin: EdgeInsets.symmetric(horizontal: 50),
                  height: 50,
                  child: ElevatedButton(
                    onPressed: () {
                      Navigator.push(context,
                          MaterialPageRoute(builder: (context) {
                        return OnboardingPage2();
                      })); // MaterialPageRoute
                    },
                    child: Text(
                      //translation
                      AppLocalizations.of(context)!.next,
                      style: TextStyle(color: Color(0xffFFE96D), fontSize: 20),
                    ), // Text
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Color(0xff050522),
                      shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(10),
                      ), // RoundedRectangleBorder
                    ),
                  ), // ElevatedButton
                ), // Container
              ],
            ), // ListView
```

onboarding_page1.dart

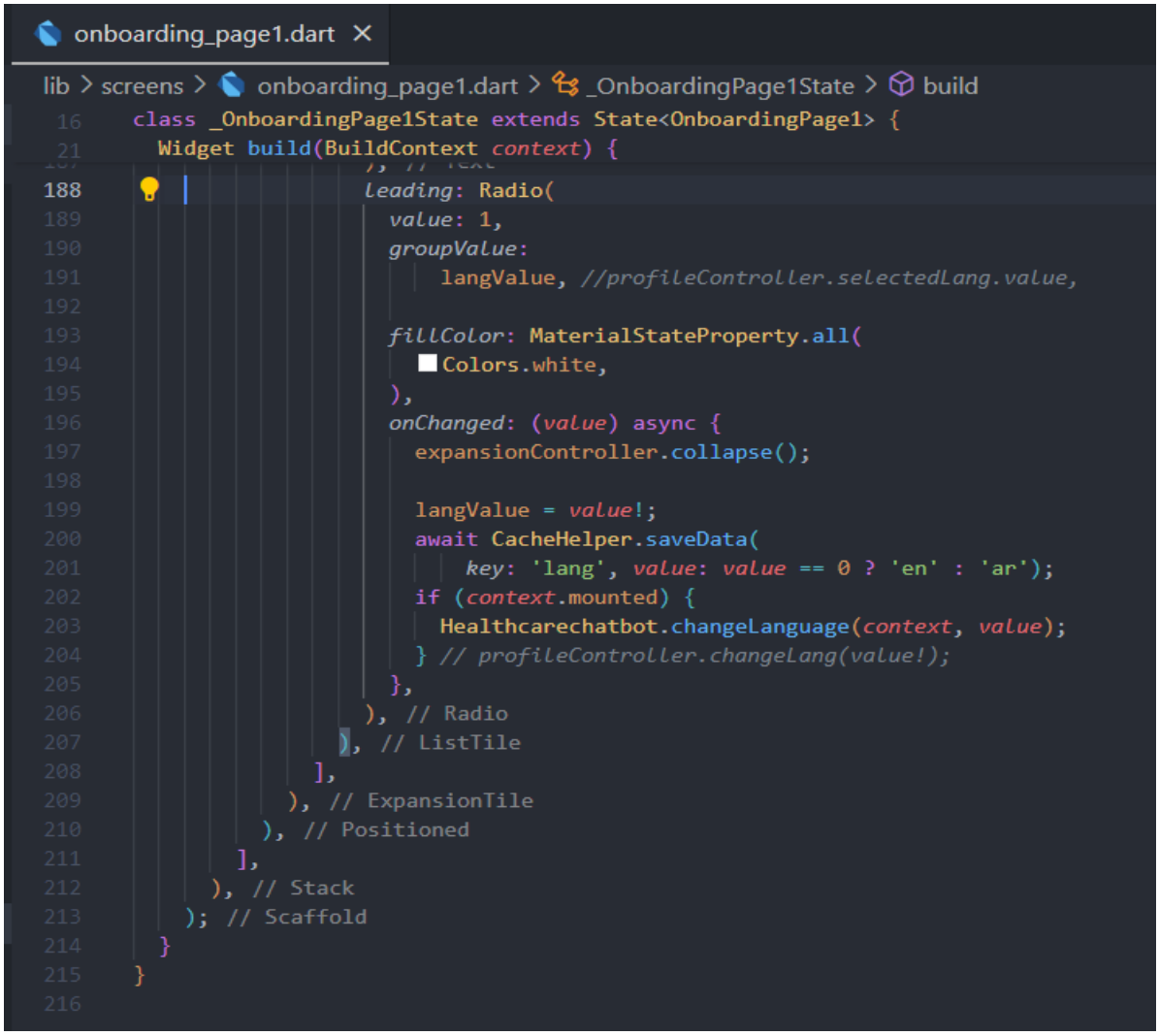
lib > screens > onboarding_page1.dart > _OnboardingPage1State > build

```dart
class _OnboardingPage1State extends State<OnboardingPage1> {
  Widget build(BuildContext context) {
          ), // ListView
          Positioned(
            top: 70,
            left: 25,
            right: 25,
            child: ExpansionTile(
              controller: expansionController,
              backgroundColor: Color(0xff1D5879),
              collapsedShape: UnderlineInputBorder(
                borderSide: BorderSide.none,
                borderRadius: BorderRadius.circular(25),
              ), // UnderlineInputBorder
              collapsedBackgroundColor: Color(0xff1D5879),
              collapsedIconColor: Colors.white,
              iconColor: Colors.white,
              shape: UnderlineInputBorder(
                borderSide: BorderSide.none,
                borderRadius: BorderRadius.circular(25),
              ), // UnderlineInputBorder
              title: Text(
                AppLocalizations.of(context)!.change_lang,
                style: TextStyle(fontSize: 22, color: Colors.white),
              ), // Text
              tilePadding: EdgeInsets.symmetric(horizontal: 10),
              leading: const Icon(
                Icons.language,
                color: Colors.white,
              ), // Icon

              children: [
                ListTile(
                  leading: Text(
                    'English',
                    style: Theme.of(context).textTheme.bodyMedium!.copyWith(
                          fontSize: 18,
                          fontWeight: FontWeight.w700,
                        ),
                  ), // Text
```

onboarding_page1.dart

lib > screens > onboarding_page1.dart > _OnboardingPage1State > build

```
16    class _OnboardingPage1State extends State<OnboardingPage1> {
21      Widget build(BuildContext context) {
151                   leading: Text(
152                     'English',
153                     style: Theme.of(context).textTheme.bodyMedium!.copyWith(
154                           fontSize: 18,
155                           fontWeight: FontWeight.w700,
156                         ),
157                   ), // Text
158                   trailing: Radio(
159                     value: 0
160                     ,
161                     fillColor: MaterialStateProperty.all(
162                       Colors.white,
163                     ),
164                     groupValue:
165                         langValue, // profileController.selectedLang.value,
166
167                     onChanged: (value) async {
168                       expansionController.collapse();
169                       langValue = value!;
170                       await CacheHelper.saveData(
171                           key: 'lang', value: value == 0 ? 'en' : 'ar');
172                       if (context.mounted) {
173                         Healthcarechatbot.changeLanguage(context, value);
174                       } //  Healthcarechatbot.changeLanguage(context, value!);
175                       //  profileController.changeLang(value!);
176                     },
177                     //   activeColor: HexColor(AppTheme.primaryColorString!),
178                   ), // Radio
179                 ), // ListTile
180                 ListTile(
181                   trailing: Text(
182                     "العربية",
183                     style: Theme.of(context).textTheme.bodyMedium!.copyWith(
184                           fontSize: 20,
185                           fontWeight: FontWeight.w700,
186                         ),
187                   ), // Text
```



Summary of this page:

These codes provide an interactive onboarding experience for users, allowing them to navigate through the onboarding process and choose their preferred language for the app's interface.

The final result of this page:

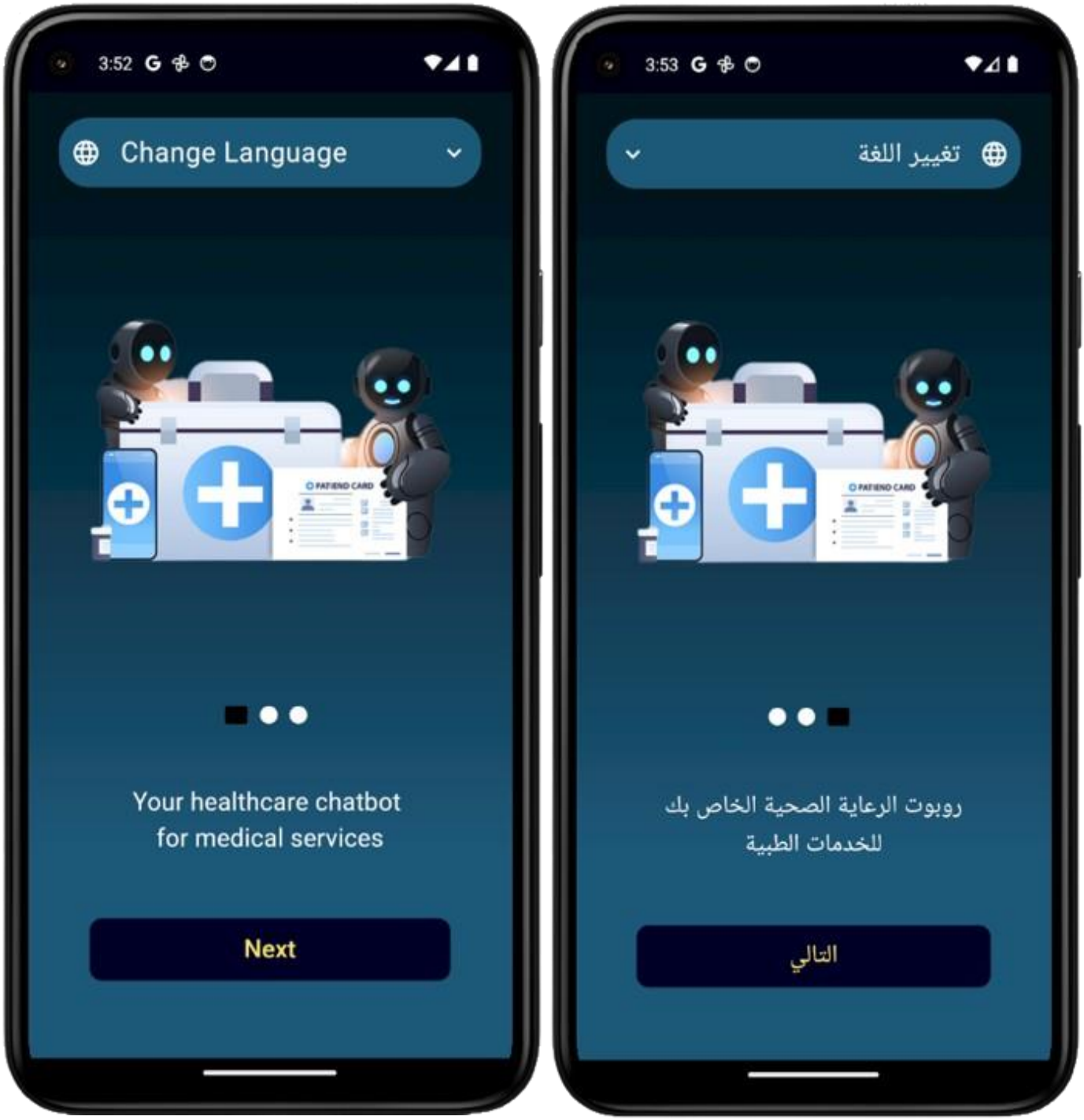


Figure 9: On boarding_1-Screen

onboarding_page2:

onboarding_page2.dart

lib > screens > onboarding_page2.dart > _OnboardingPage2State > build

```dart
// ignore_for_file: prefer_const_constructors, prefer_const_literals_to_create_i
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
import 'package:flutter/material.dart';
import 'package:healthcare_chatbot/screens/onboarding_page3.dart';
class OnboardingPage2 extends StatefulWidget {
  const OnboardingPage2({super.key});
  @override
  State<OnboardingPage2> createState() => _OnboardingPage2State();
}
class _OnboardingPage2State extends State<OnboardingPage2> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(children: [
        Container(
          decoration: BoxDecoration(
            gradient: LinearGradient(
                colors: [
                  Color(0xff1D5879),
                  Color(0xff04141D),
                ],
                begin: Alignment.bottomCenter,
                end: Alignment.topCenter,
                stops: [0.2, 0.85]), // LinearGradient
          ), // BoxDecoration
        ), // Container
        ListView(
          children: [
            SizedBox(
              height: 40,
            ), // SizedBox
            Container(
                height: 420,
                width: 400,
                child: Image.asset(
                  "images/onboarding-photo.png",
                  fit: BoxFit.cover,
                )), // Image.asset // Container
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Padding(
```


onboarding_page2.dart
lib > screens > onboarding_page2.dart > _OnboardingPage2State > build
class _OnboardingPage2State extends State<OnboardingPage2> {
Widget build(BuildContext context) {
padding: const EdgeInsets.only(right: 7),
child: Icon(
Icons.circle,
color: Colors.white,
size: 17,
), // Icon
), // Padding
Padding(
padding: const EdgeInsets.only(right: 7),
child: Icon(
Icons.rectangle_rounded,
color: Colors.black,
size: 22,
), // Icon
), // Padding
Padding(
padding: const EdgeInsets.only(right: 7),
child: Icon(
Icons.circle,
color: Colors.white,
size: 17,
), // Icon
), // Padding
],
), // Row
SizedBox(
height: 15,
), // SizedBox
Padding(
padding: const EdgeInsets.only(left: 25, right: 12),
child: Text(
AppLocalizations.of(context)!.onboard_2,
style: TextStyle(color: Colors.white, fontSize: 20),
), // Text
), // Padding
SizedBox(
height: 35,
), // SizedBox
Container(
margin: EdgeInsets.symmetric(horizontal: 50),

onboarding_page2.dart

lib > screens > onboarding_page2.dart > ...

```
10    class _OnboardingPage2State extends State<OnboardingPage2> {
12      Widget build(BuildContext context) {
81                Container(
82                  margin: EdgeInsets.symmetric(horizontal: 50),
83                  height: 50,
84                  child: ElevatedButton(
85                    onPressed: () {
86                      Navigator.push(context, MaterialPageRoute(builder: (context) {
87                        return OnboardingPage3();
88                      })); // MaterialPageRoute
89                    },
90                    child: Text(
91                      AppLocalizations.of(context)!.next,
92                      style: TextStyle(color: Color(0xffFFE96D), fontSize: 20),
93                    ), // Text
94                    style: ElevatedButton.styleFrom(
95                      backgroundColor: Color(0xff050522),
96                      shape: RoundedRectangleBorder(
97                        borderRadius: BorderRadius.circular(10),
98                      ), // RoundedRectangleBorder
99                    ),
100                 ), // ElevatedButton
101               ), // Container
102             ],
103           ), // ListView
104         ]), // Stack
105       ); // Scaffold
106     }
107   }
108
```

Summary of this page:

This page contains an important message which is:

The application supports a healthcare chatbot to classify diseases and provide some basic precautions for diseases and works in both English and Arabic Languages.

The final result of this :

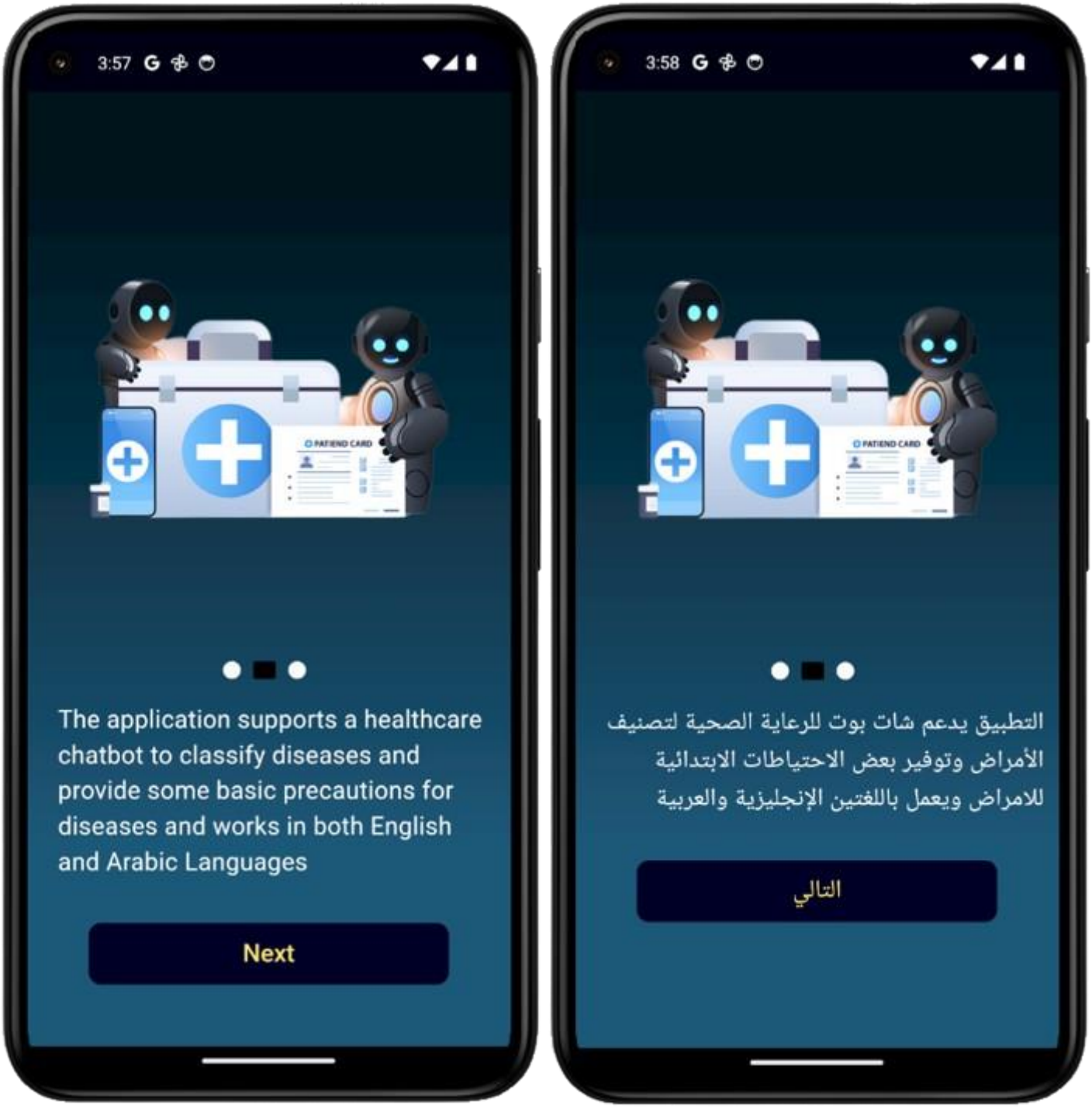


Figure 10: On boarding_2-Screen

onboarding_page3:

onboarding_page3.dart

lib > screens > onboarding_page3.dart > _OnboardingPage3State > build

```dart
// ignore_for_file: prefer_const_constructors, prefer_const_literals_to_create_i
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
import 'package:flutter/material.dart';
import 'package:healthcare_chatbot/screens/welcome_page.dart';
class OnboardingPage3 extends StatefulWidget {
  const OnboardingPage3({super.key});

  @override
  State<OnboardingPage3> createState() => _OnboardingPage3State();
}

class _OnboardingPage3State extends State<OnboardingPage3> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(children: [
        Container(
          decoration: BoxDecoration(
            gradient: LinearGradient(
                colors: [
                  Color(0xff1D5879),
                  Color(0xff04141D),
                ],
                begin: Alignment.bottomCenter,
                end: Alignment.topCenter,
                stops: [0.2, 0.85]), // LinearGradient
          ), // BoxDecoration
        ), // Container
        ListView(
          children: [
            SizedBox(
              height: 70,
            ), // SizedBox
            Container(
                height: 420,
                width: 400,
                child: Image.asset(
                  "images/onboarding-photo.png",
                  fit: BoxFit.cover,
                )), // Image.asset // Container
```

onboarding_page3.dart

lib > screens > onboarding_page3.dart > _OnboardingPage3State > build

```dart
class _OnboardingPage3State extends State<OnboardingPage3> {
  Widget build(BuildContext context) {
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Padding(
                  padding: const EdgeInsets.only(right: 7),
                  child: Icon(
                    Icons.circle,
                    color: Colors.white,
                    size: 17,
                  ), // Icon
                ), // Padding
                Padding(
                  padding: const EdgeInsets.only(right: 7),
                  child: Icon(
                    Icons.circle,
                    color: Colors.white,
                    size: 17,
                  ), // Icon
                ), // Padding
                Padding(
                  padding: const EdgeInsets.only(right: 7),
                  child: Icon(
                    Icons.rectangle_rounded,
                    color: Colors.black,
                    size: 22,
                  ), // Icon
                ), // Padding
              ],
            ), // Row
            SizedBox(
              height: 45,
            ), // SizedBox
            Column(
              children: [
                Text(
                 AppLocalizations.of(context)!.onboard_3_1,
                  style: TextStyle(color: Colors.white, fontSize: 20),
                ), // Text
                Text(
                  AppLocalizations.of(context)!.onboard_3_2,
                  style: TextStyle(color: Colors.white, fontSize: 20),
                ), // Text
```

onboarding_page3.dart

lib > screens > onboarding_page3.dart > _OnboardingPage3State > build

```
12    class _OnboardingPage3State extends State<OnboardingPage3> {
14      Widget build(BuildContext context) {
84                ), // Column
85                SizedBox(
86                  height: 50,
87                ), // SizedBox
88                Container(
89                  margin: EdgeInsets.symmetric(horizontal: 50),
90                  height: 50,
91                  child: ElevatedButton(
92                    onPressed: () {
93                      Navigator.push(context, MaterialPageRoute(builder: (context) {
94                        return WelcomePage();
95                      })); // MaterialPageRoute
96                    },
97                    child: Text(
98                     AppLocalizations.of(context)!.get_started,
99                      style: TextStyle(color: Color(0xffFFE96D), fontSize: 20),
100                   ), // Text
101                   style: ElevatedButton.styleFrom(
102                     backgroundColor: Color(0xff050522),
103                     shape: RoundedRectangleBorder(
104                       borderRadius: BorderRadius.circular(10),
105                     ), // RoundedRectangleBorder
106                   ),
107                 ), // ElevatedButton
108               ), // Container
109             ],
110           ), // ListView
111         ]), // Stack
112       ); // Scaffold
113     }
114   }
115
```

Summary of this page:

These codes provide an interactive onboarding experience for users, allowing them to navigate through the onboarding process and provide the final step in the onboarding experience for users, allowing them to navigate to the welcome page and start using the healthcare chatbot application.

The final result of this page:

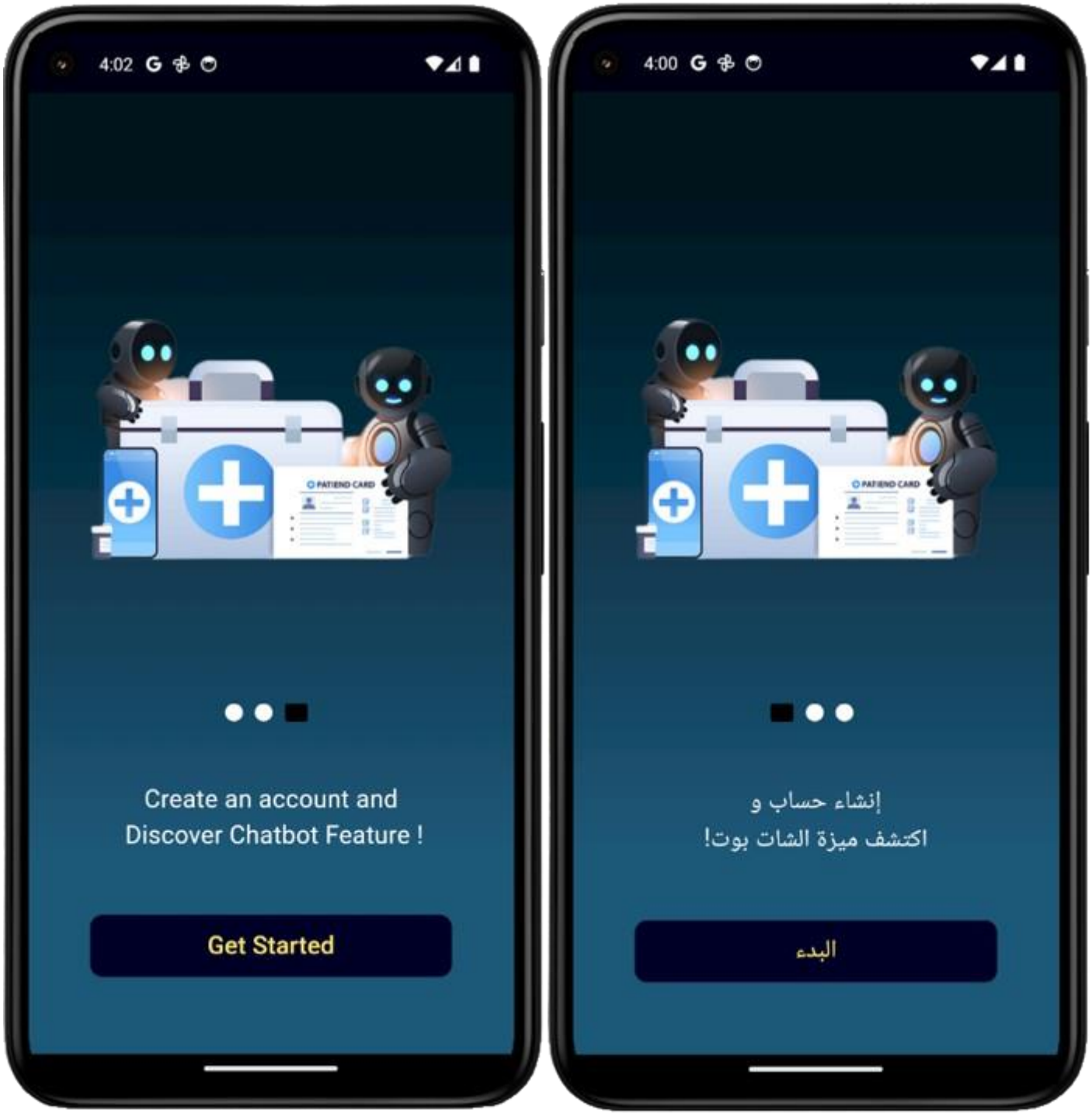


Figure 11: On boarding_3-Sreen

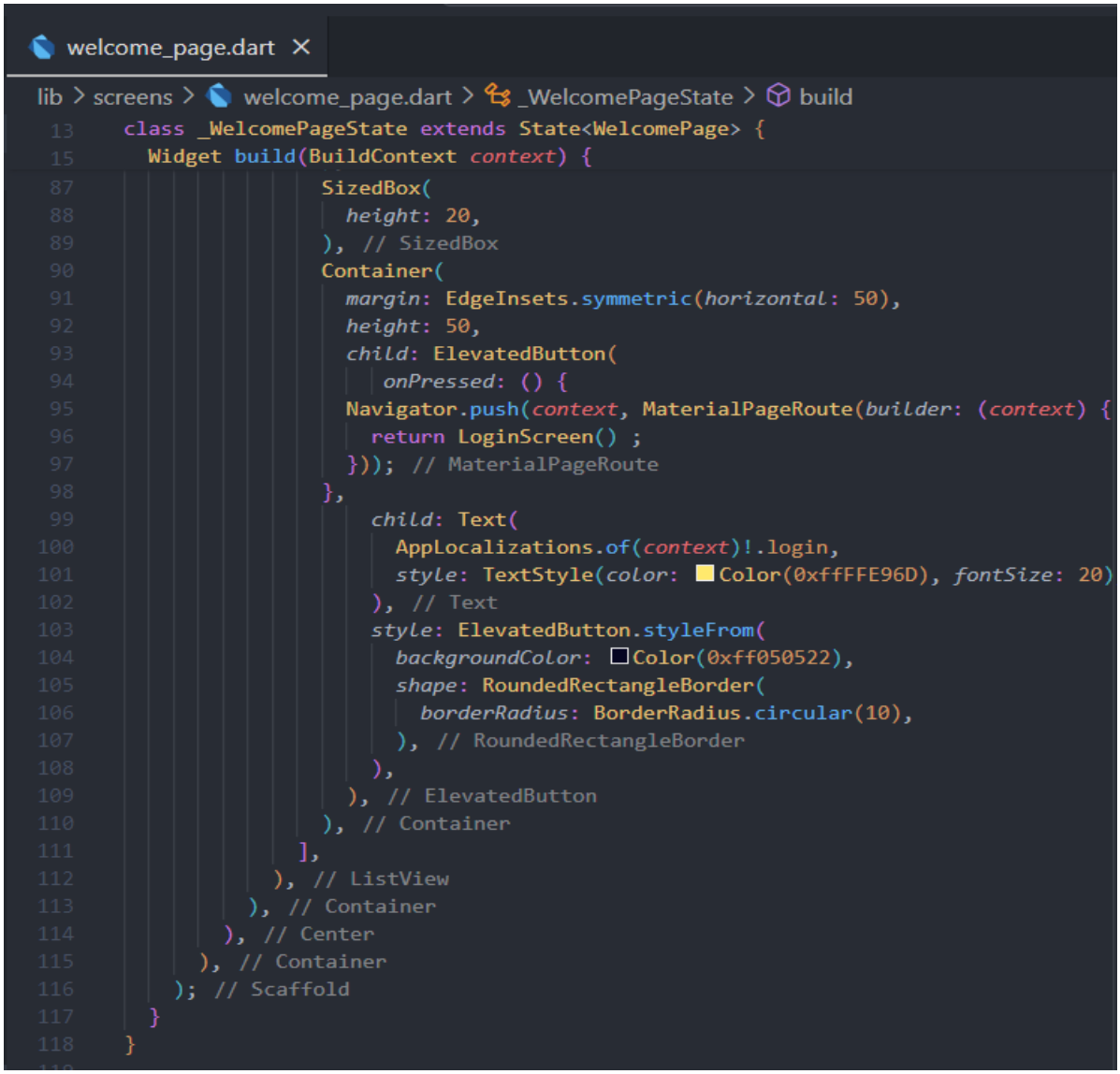
welcome_page:

welcome_page.dart

lib > screens > welcome_page.dart > _WelcomePageState > build

```dart
import 'package:flutter/material.dart';
import 'package:healthcare_chatbot/screens/login_page.dart';
import 'package:healthcare_chatbot/screens/register_page.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';

class WelcomePage extends StatefulWidget {
  const WelcomePage({super.key});

  @override
  State<WelcomePage> createState() => _WelcomePageState();
}
class _WelcomePageState extends State<WelcomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: BoxDecoration(
          gradient: LinearGradient(
              colors: [
                Color(0xff1C5271),
                Color(0xff040E15),
              ],
              begin: Alignment.bottomCenter,
              end: Alignment.topCenter,
             // stops: [0.02, 0.95]
              ), // LinearGradient
        ), // BoxDecoration
        child: Center(
          child: Container(
            alignment: Alignment.center,
            child: ListView(
              children: [
                Container(
                  padding: EdgeInsets.only(left: 45),
                  height: 450,
                  child: Image.asset(
                    "images/Welcome-chatgpt robot.png",
                  ), // Image.asset
                ), // Container
                Center(
                  child: Text(
                    textAlign: TextAlign.center,
                    AppLocalizations.of(context)!.welcome,
```


welcome_page.dart
lib > screens > welcome_page.dart > _WelcomePageState > build
class _WelcomePageState extends State<WelcomePage> {
Widget build(BuildContext context) {
AppLocalizations.of(context)!.welcome,
style: TextStyle(
fontSize: 35,
fontWeight: FontWeight.bold,
color: Colors.white), // TextStyle
), // Text
), // Center
SizedBox(
height: 20,
), // SizedBox
Text(
AppLocalizations.of(context)!.important_to_have,
textAlign: TextAlign.center,
style: TextStyle(
fontSize: 17,
fontWeight: FontWeight.w400,
color: Colors.white), // TextStyle
), // Text
SizedBox(
height: 60,
), // SizedBox
Container(
margin: EdgeInsets.symmetric(horizontal: 50),
height: 50,
child: ElevatedButton(
onPressed: () {
Navigator.push(context,
MaterialPageRoute(builder: (context) {
return RegisterScreen();
})); // MaterialPageRoute
},
child: Text(
AppLocalizations.of(context)!.create_account,
style: TextStyle(color: Color(0xffFFE96D), fontSize: 20)
), // Text
style: ElevatedButton.styleFrom(
backgroundColor: Color(0xff050522),
shape: RoundedRectangleBorder(
borderRadius: BorderRadius.circular(10),
), // RoundedRectangleBorder
),
), // ElevatedButton



Summary of this page:

Welcome page can engage users by presenting them with personalized content. These codes provide a visually appealing welcome screen for users to either register for a new account or log in to an existing one, setting the stage for further interaction with the healthcare chatbot application.

The final result of this page:

Figure 12: Welcome Screen

register_page:

register_page.dart

lib > screens > register_page.dart > ...

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:healthcare_chatbot/screens/login_page.dart';
import 'package:healthcare_chatbot/widget/custom_button.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
// Custom Text Field widget
class CustemTextField extends StatelessWidget {
  final String? hintText;
  final Icon prefixIcon;
  final Function(String)? onChanged;

  const CustemTextField({
    Key? key,
    this.hintText,
    this.onChanged,
    required this.prefixIcon,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(
      width: 350,
      height: 60,
      child: TextField(
        onChanged: onChanged,
        decoration: InputDecoration(
          prefixIcon: prefixIcon,
          hintText: hintText,
          hintStyle: TextStyle(fontSize: 20),
          focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(16),
          ), // OutlineInputBorder
          enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(16),
          ), // OutlineInputBorder
          filled: true,
          fillColor: Color(0xfff0f0f0),
        ), // InputDecoration
      ), // TextField
    ); // Container
  }
}
```

register_page.dart

lib > screens > register_page.dart > _CustemPasswordFieldState > build

```dart
class CustemPasswordField extends StatefulWidget {
  final String? hintText;
  final Icon prefixIcon;
  final Function(String)? onChanged;

  const CustemPasswordField({
    Key? key,
    this.hintText,
    this.onChanged,
    required this.prefixIcon,
  }) : super(key: key);

  @override
  _CustemPasswordFieldState createState() => _CustemPasswordFieldState();
}

class _CustemPasswordFieldState extends State<CustemPasswordField> {
  bool _obscureText = true;

  @override
  Widget build(BuildContext context) {
    return Container(
      width: 350,
      height: 60,
      child: TextField(
        onChanged: widget.onChanged,
        obscureText: _obscureText,
        decoration: InputDecoration(
          prefixIcon: widget.prefixIcon,
          hintText: widget.hintText,
          hintStyle: TextStyle(fontSize: 20),
          focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(16),
          ), // OutlineInputBorder
          enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(16),
          ), // OutlineInputBorder
          filled: true,
          fillColor: Color(0xfff0f0f0),
          suffixIcon: GestureDetector(
            onTap: () {
              setState(() {
                _obscureText = !_obscureText;
              });
```

register_page.dart

lib > screens > register_page.dart > _RegisterScreenState

```dart
class _CustemPasswordFieldState extends State<CustemPasswordField> {
  Widget build(BuildContext context) {
                child: Icon(
                  _obscureText ? Icons.visibility : Icons.visibility_off,
                  color: Colors.grey,
                ), // Icon
              ), // GestureDetector
            ), // InputDecoration
          ), // TextField
        ); // Container
  }
}
class RegisterScreen extends StatefulWidget {
  RegisterScreen({Key? key});

  @override
  _RegisterScreenState createState() => _RegisterScreenState();
}
class _RegisterScreenState extends State<RegisterScreen> {
  String? firstName;
  String? lastName;
  String? email;
  String? password;
  String? confirmPassword;
  bool isLoading = false;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          Container(
            decoration: BoxDecoration(
              gradient: LinearGradient(
                colors: [
                  Color(0xff1C5271),
                  Color(0xff040E15),
                ],
                begin: Alignment.bottomCenter,
                end: Alignment.topCenter,
                // stops: [0.02, 0.95]
              ), // LinearGradient
            ), // BoxDecoration
          ), // Container
```

register_page.dart

lib > screens > register_page.dart > _RegisterScreenState > build

```
109   class _RegisterScreenState extends State<RegisterScreen> {
117     Widget build(BuildContext context) {
134       Container(
135         child: Center(
136           child: SingleChildScrollView(
137             child: Column(
138               children: [
139                 Padding(
140                   padding: const EdgeInsets.only(bottom: 30, top: 30),
141                   child: Center(
142                     child: Text(
143                         AppLocalizations.of(context)!.register,
144
145                       style: TextStyle(
146                         fontSize: 40,
147                         color: Colors.white,
148                       ), // TextStyle
149                     ), // Text
150                   ), // Center
151                 ), // Padding
152                 SizedBox(
153                   height: 50,
154                 ), // SizedBox
155                 Text(
156                   AppLocalizations.of(context)!.registe_to_con,
157
158                   style: TextStyle(
159                     fontSize: 20,
160                     color: Colors.white,
161                   ), // TextStyle
162                 ), // Text
163                 SizedBox(
164                   height: 20,
165                 ), // SizedBox
166                 CustemTextField(
167                   onChanged: (data) {
168                     firstName = data;
169                   },
170                   hintText: AppLocalizations.of(context)!.first_name,
171                   prefixIcon: Icon(
172                     Icons.person,
173                     size: 35,
174                     color: Colors.black,
175                   ), // Icon
```


register_page.dart
lib > screens > register_page.dart > _RegisterScreenState > build
class _RegisterScreenState extends State<RegisterScreen> {
Widget build(BuildContext context) {
), // CustemTextField
SizedBox(
height: 20,
), // SizedBox
CustemTextField(
onChanged: (data) {
lastName = data;
},
hintText: AppLocalizations.of(context)!.last_name,
prefixIcon: Icon(
Icons.person,
size: 35,
color: Colors.black,
), // Icon
), // CustemTextField
SizedBox(
height: 20,
), // SizedBox
CustemTextField(
onChanged: (data) {
email = data;
},
hintText: AppLocalizations.of(context)!.email,
prefixIcon: Icon(
Icons.email,
size: 35,
color: Colors.black,
), // Icon
), // CustemTextField
SizedBox(
height: 25,
), // SizedBox
CustemPasswordField(
onChanged: (data) {
password = data;
},
hintText: AppLocalizations.of(context)!.pass,
prefixIcon: Icon(
Icons.lock,
size: 35,
color: Colors.black,
), // Icon


register_page.dart
lib > screens > register_page.dart > _RegisterScreenState > build
class _RegisterScreenState extends State<RegisterScreen> {
Widget build(BuildContext context) {
await FirebaseFirestore.instance
.collection('users')
.doc(user.user!.uid)
.set({
'firstName': firstName,
'lastName': lastName,
'email': email,
});
Navigator.pushReplacement(
context,
MaterialPageRoute(
builder: (context) => LoginScreen(),
), // MaterialPageRoute
);
} on FirebaseAuthException catch (e) {
if (e.code == 'weak-password') {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(
content: Text(
AppLocalizations.of(context)!.pswd_weak,
), // Text
), // SnackBar
);
} else if (e.code == 'email-already-in-use') {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(
content: Text(
AppLocalizations.of(context)!
.email_already_used,
), // Text
), // SnackBar
);
}
} finally {
setState(() {
isLoading = false;
});
}
},
text: AppLocalizations.of(context)!.register,
), // CustomButton

register_page.dart

lib > screens > register_page.dart > _RegisterScreenState > build

```
109    class _RegisterScreenState extends State<RegisterScreen> {
117      Widget build(BuildContext context) {
302                        SizedBox(
303                          height: 30,
304                        ), // SizedBox
305                        Row(
306                          mainAxisAlignment: MainAxisAlignment.center,
307                          children: [
308                            Text(
309                              AppLocalizations.of(context)!.already_have_acc,
310                              style: TextStyle(fontSize: 18, color: Colors.white),
311                            ), // Text
312                            GestureDetector(
313                              onTap: () {
314                                Navigator.pop(context);
315                              },
316                              child: Text(
317                                AppLocalizations.of(context)!.signin,
318                                style: TextStyle(
319                                    fontSize: 18, color: Color(0xff050522)), // Te
320                              ), // Text
321                            ) // GestureDetector
322                          ],
323                        ) // Row
324                      ],
325                    ), // Column
326                  ), // SingleChildScrollView
327                ), // Center
328              ), // Container
329              isLoading
330                  ? Container(
331                      color: Colors.black.withOpacity(0.5),
332                      child: Center(
333                        child: CircularProgressIndicator(),
334                      ), // Center
335                    ) // Container
336                  : SizedBox(),
337            ],
338          ), // Stack
339        ); // Scaffold
340      }
341    }
```

Summary of this page:

The primary function is to capture user information such as username, email, password, etc., and store it securely in a database. This information is typically used for authentication and personalization purposes and validate the user input to ensure that the entered data meets certain criteria, such as valid email format, strong password requirements, etc. This helps in maintaining data integrity and security.

The final result of this page:

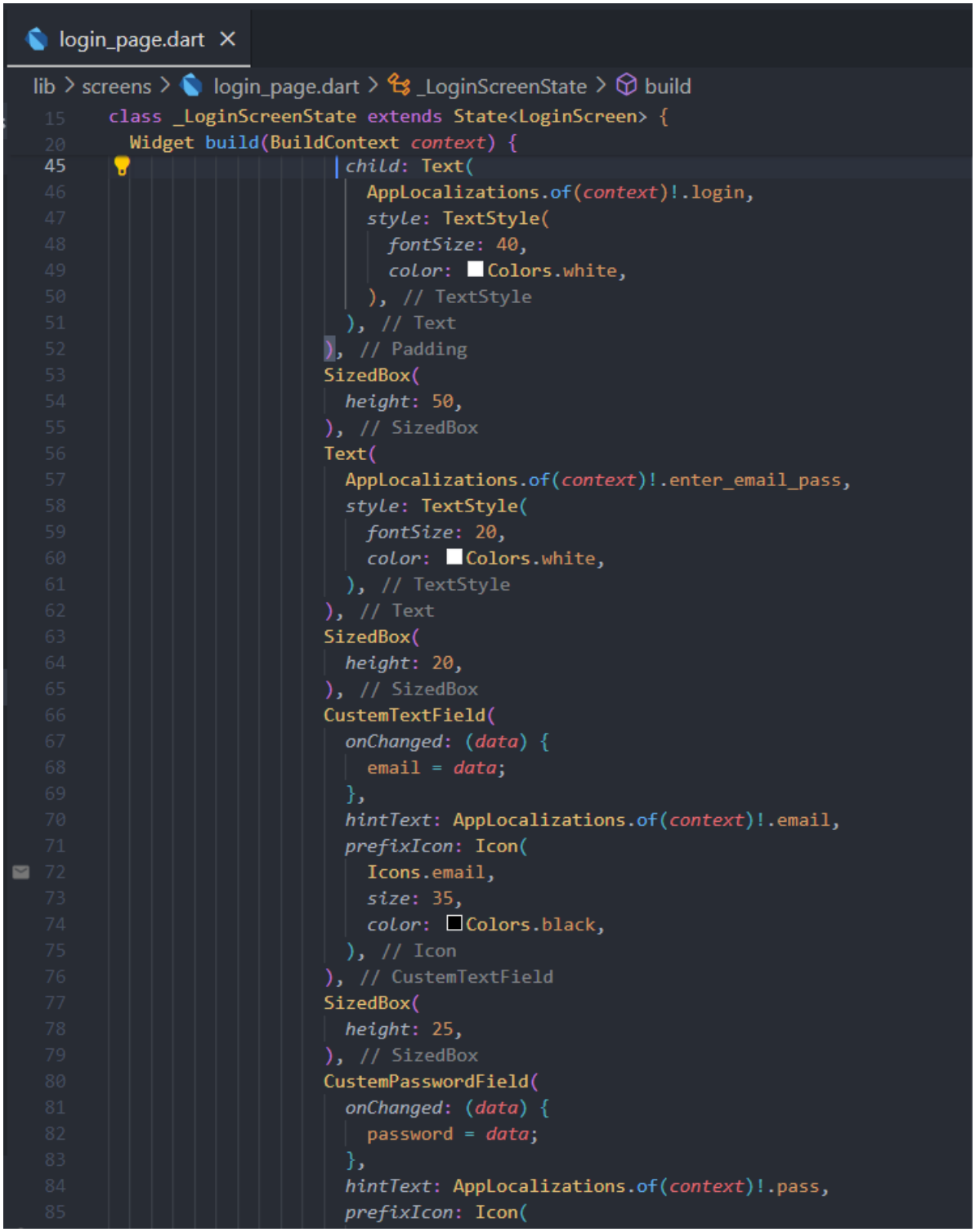
Figure 13: Register Screen

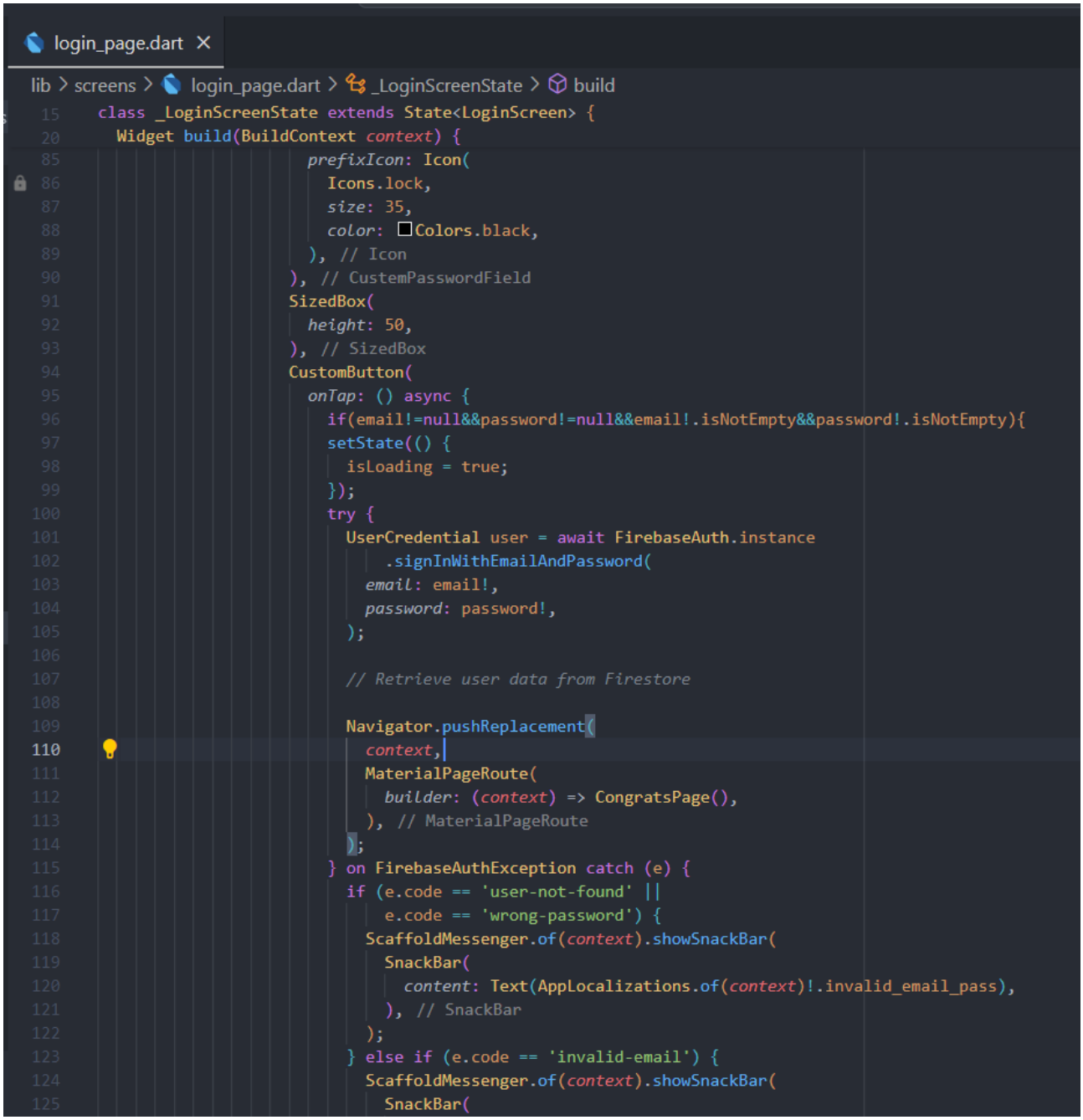
login page:

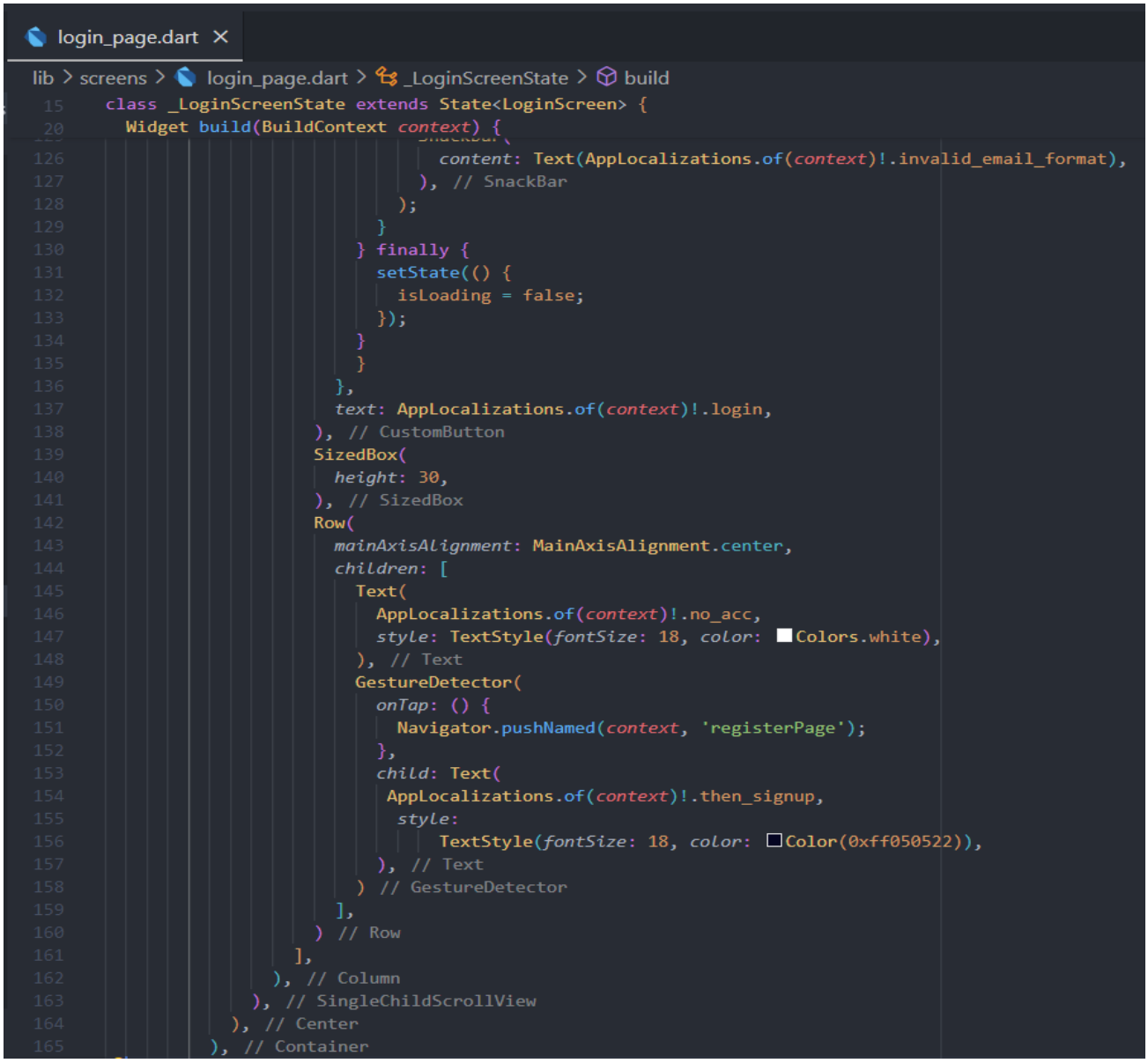
login_page.dart

lib > screens > login_page.dart > _LoginScreenState

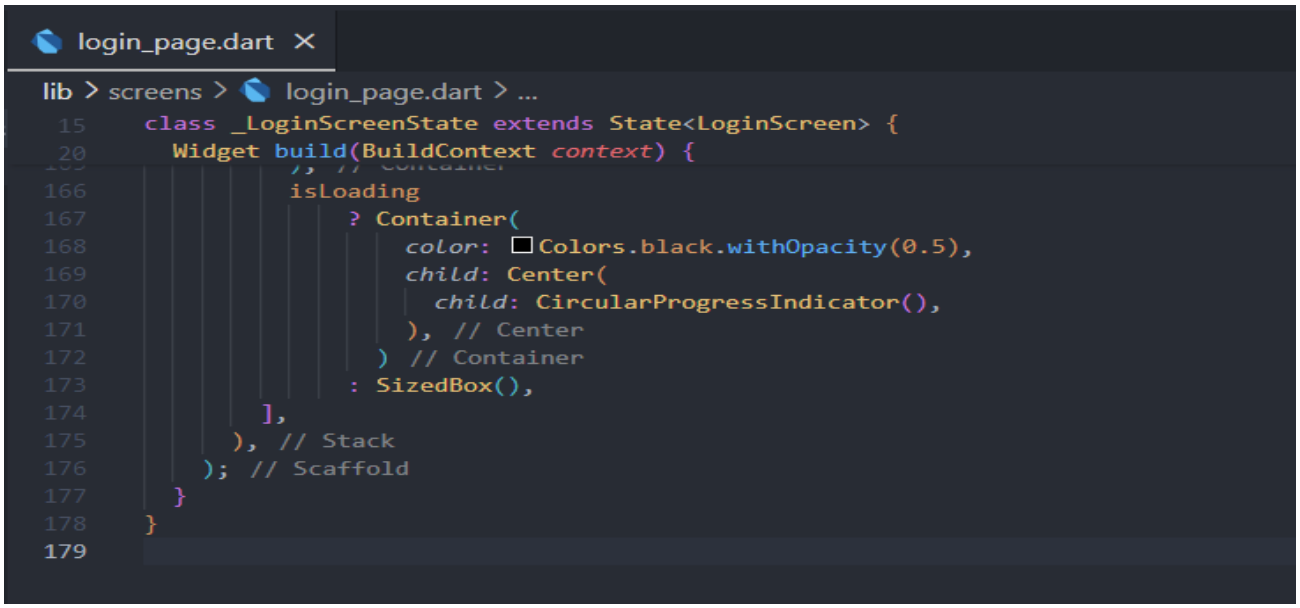
```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:healthcare_chatbot/screens/congrats_page.dart';
import 'package:healthcare_chatbot/screens/register_page.dart';
import 'package:healthcare_chatbot/widget/custom_button.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
class LoginScreen extends StatefulWidget {
  LoginScreen({Key? key});

  @override
  _LoginScreenState createState() => _LoginScreenState();
}
class _LoginScreenState extends State<LoginScreen> {
  String? email;
  String? password;
  bool isLoading = false;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          Container(
            decoration: BoxDecoration(
              gradient: LinearGradient(
                colors: [
                  Color(0xff1C5271),
                  Color(0xff040E15),
                ],
                begin: Alignment.bottomCenter,
                end: Alignment.topCenter,
                // stops: [0.02, 0.95]
              ), // LinearGradient
            ), // BoxDecoration
          ), // Container
          Container(
            child: Center(
              child: SingleChildScrollView(
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    Padding(
                      padding: const EdgeInsets.only(bottom: 50),
```


login_page.dart
lib > screens > login_page.dart > _LoginScreenState > build
class _LoginScreenState extends State<LoginScreen> {
Widget build(BuildContext context) {
child: Text(
AppLocalizations.of(context)!.login,
style: TextStyle(
fontSize: 40,
color: Colors.white,
), // TextStyle
), // Text
), // Padding
SizedBox(
height: 50,
), // SizedBox
Text(
AppLocalizations.of(context)!.enter_email_pass,
style: TextStyle(
fontSize: 20,
color: Colors.white,
), // TextStyle
), // Text
SizedBox(
height: 20,
), // SizedBox
CustemTextField(
onChanged: (data) {
email = data;
},
hintText: AppLocalizations.of(context)!.email,
prefixIcon: Icon(
Icons.email,
size: 35,
color: Colors.black,
), // Icon
), // CustemTextField
SizedBox(
height: 25,
), // SizedBox
CustemPasswordField(
onChanged: (data) {
password = data;
},
hintText: AppLocalizations.of(context)!.pass,
prefixIcon: Icon(


login_page.dart
lib > screens > login_page.dart > _LoginScreenState > build
class _LoginScreenState extends State<LoginScreen> {
Widget build(BuildContext context) {
prefixIcon: Icon(
Icons.lock,
size: 35,
color: Colors.black,
), // Icon
), // CustemPasswordField
SizedBox(
height: 50,
), // SizedBox
CustomButton(
onTap: () async {
if(email!=null&&password!=null&&email!.isNotEmpty&&password!.isNotEmpty){
setState(() {
isLoading = true;
});
try {
UserCredential user = await FirebaseAuth.instance
.signInWithEmailAndPassword(
email: email!,
password: password!,
);
// Retrieve user data from Firestore
Navigator.pushReplacement(
context,
MaterialPageRoute(
builder: (context) => CongratsPage(),
), // MaterialPageRoute
);
} on FirebaseAuthException catch (e) {
if (e.code == 'user-not-found' ||
e.code == 'wrong-password') {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(
content: Text(AppLocalizations.of(context)!.invalid_email_pass),
), // SnackBar
);
} else if (e.code == 'invalid-email') {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(



```
login_page.dart
lib > screens > login_page.dart > _LoginScreenState > build
15    class _LoginScreenState extends State<LoginScreen> {
20      Widget build(BuildContext context) {
126                                 content: Text(AppLocalizations.of(context)!.invalid_email_format),
127                               ), // SnackBar
128                             );
129                           }
130                         } finally {
131                           setState(() {
132                             isLoading = false;
133                           });
134                         }
135                         }
136                       },
137                       text: AppLocalizations.of(context)!.login,
138                     ), // CustomButton
139                     SizedBox(
140                       height: 30,
141                     ), // SizedBox
142                     Row(
143                       mainAxisAlignment: MainAxisAlignment.center,
144                       children: [
145                         Text(
146                           AppLocalizations.of(context)!.no_acc,
147                           style: TextStyle(fontSize: 18, color: Colors.white),
148                         ), // Text
149                         GestureDetector(
150                           onTap: () {
151                             Navigator.pushNamed(context, 'registerPage');
152                           },
153                           child: Text(
154                            AppLocalizations.of(context)!.then_signup,
155                             style:
156                                 TextStyle(fontSize: 18, color: Color(0xff050522)),
157                           ), // Text
158                         ) // GestureDetector
159                       ],
160                     ) // Row
161                   ],
162                 ), // Column
163               ), // SingleChildScrollView
164             ), // Center
165           ), // Container
```



```
login_page.dart
lib > screens > login_page.dart > ...
15    class _LoginScreenState extends State<LoginScreen> {
20      Widget build(BuildContext context) {
166             isLoading
167                 ? Container(
168                     color: Colors.black.withOpacity(0.5),
169                     child: Center(
170                       child: CircularProgressIndicator(),
171                     ), // Center
172                   ) // Container
173                 : SizedBox(),
174           ],
175         ), // Stack
176       ); // Scaffold
177     }
178   }
179
```

Summary of this page:

User Authentication The central purpose of a login page is to authenticate users by verifying their identity through credentials such as username/email and password. This process ensures that only authorized users can access the application.

The final result of this page:

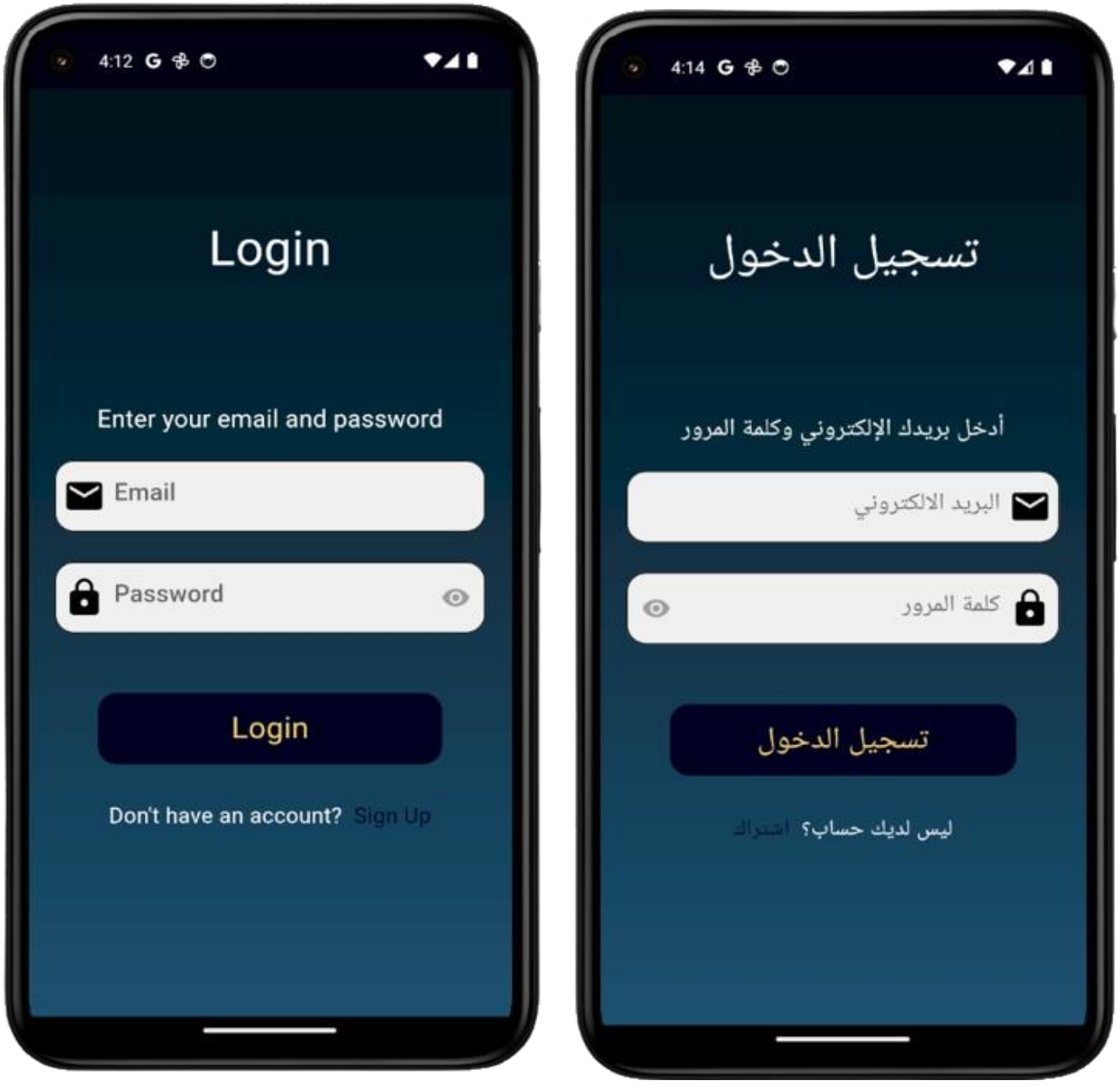


Figure 14: Login Screen

congrats_page:

congrats_page.dart

lib > screens > congrats_page.dart > _CongratsPageState > build

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:healthcare_chatbot/screens/chatbot_page.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
class CongratsPage extends StatefulWidget {
  CongratsPage({
    super.key,
  });

  @override
  State<CongratsPage> createState() => _CongratsPageState();
}
class _CongratsPageState extends State<CongratsPage> {
  String Name = '';
  @override
  getName() async {
    DocumentSnapshot userData = await FirebaseFirestore.instance
        .collection('users')
        .doc(FirebaseAuth.instance.currentUser!.uid)
        .get();
    Name = await userData['firstName'];

    setState(() {});
  }
  void initState() {
    // TODO: implement initState
    getName();
    super.initState();
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: BoxDecoration(
          gradient: LinearGradient(
              colors: [
                Color(0xff1C5271),
                Color(0xff040E15),
              ],
              begin: Alignment.bottomCenter,
              end: Alignment.topCenter,
              stops: [0.02, 0.95]), // LinearGradient
```

congrats_page.dart

lib > screens > congrats_page.dart > _CongratsPageState > build

```dart
class _CongratsPageState extends State<CongratsPage> {
  Widget build(BuildContext context) {
        child: Center(
          child: Container(
            alignment: Alignment.center,
            child: ListView(
              children: [
                Container(
                  padding: EdgeInsets.only(left: 10, top: 60),
                  height: 340,
                  child: Image.asset(
                    "images/congrats.gif",
                  ), // Image.asset
                ), // Container
                SizedBox(
                  height: 100,
                ), // SizedBox
                Center(
                  child: Text(
                    "${AppLocalizations.of(context)!.hello} ${Name}",
                    style: TextStyle(
                        fontSize: 30,
                        fontWeight: FontWeight.bold,
                        color: Colors.white), // TextStyle
                  ), // Text
                ), // Center
                SizedBox(
                  height: 45,
                ), // SizedBox
                Padding(
                  padding: const EdgeInsets.only(left: 10, right: 10),
                  child: Text(
                    AppLocalizations.of(context)!.thank_msg,
                    style: TextStyle(
                        fontSize: 17,
                        fontWeight: FontWeight.w400,
                        color: Colors.white), // TextStyle
                  ), // Text
                ), // Padding
                SizedBox(
                  height: 60,
                ), // SizedBox
```

congrats_page.dart

lib > screens > congrats_page.dart > _CongratsPageState > build

```
class _CongratsPageState extends State<CongratsPage> {
  Widget build(BuildContext context) {
                Container(
                  margin: EdgeInsets.symmetric(horizontal: 50),
                  height: 50,
                  child: ElevatedButton(
                    onPressed: () {
                      Navigator.push(context,
                          MaterialPageRoute(builder: (context) {
                        return ChatbotPage();
                      })); // MaterialPageRoute
                    },
                    child: Text(
                      AppLocalizations.of(context)!.continuee,
                      style: TextStyle(color: Color(0xffFFE96D), fontSize: 20),
                    ), // Text
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Color(0xff050522),
                      shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(10),
                      ), // RoundedRectangleBorder
                    ),
                  ), // ElevatedButton
                ), // Container
              ],
            ), // ListView
          ), // Container
        ), // Center
      ), // Container
    ); // Scaffold
  }
}
```

Summary of this page:

-Displays a congratulatory message to the user along with the user's name retrieved from Firestore and provides a button to continue to the chatbot page.

The final result of this page:

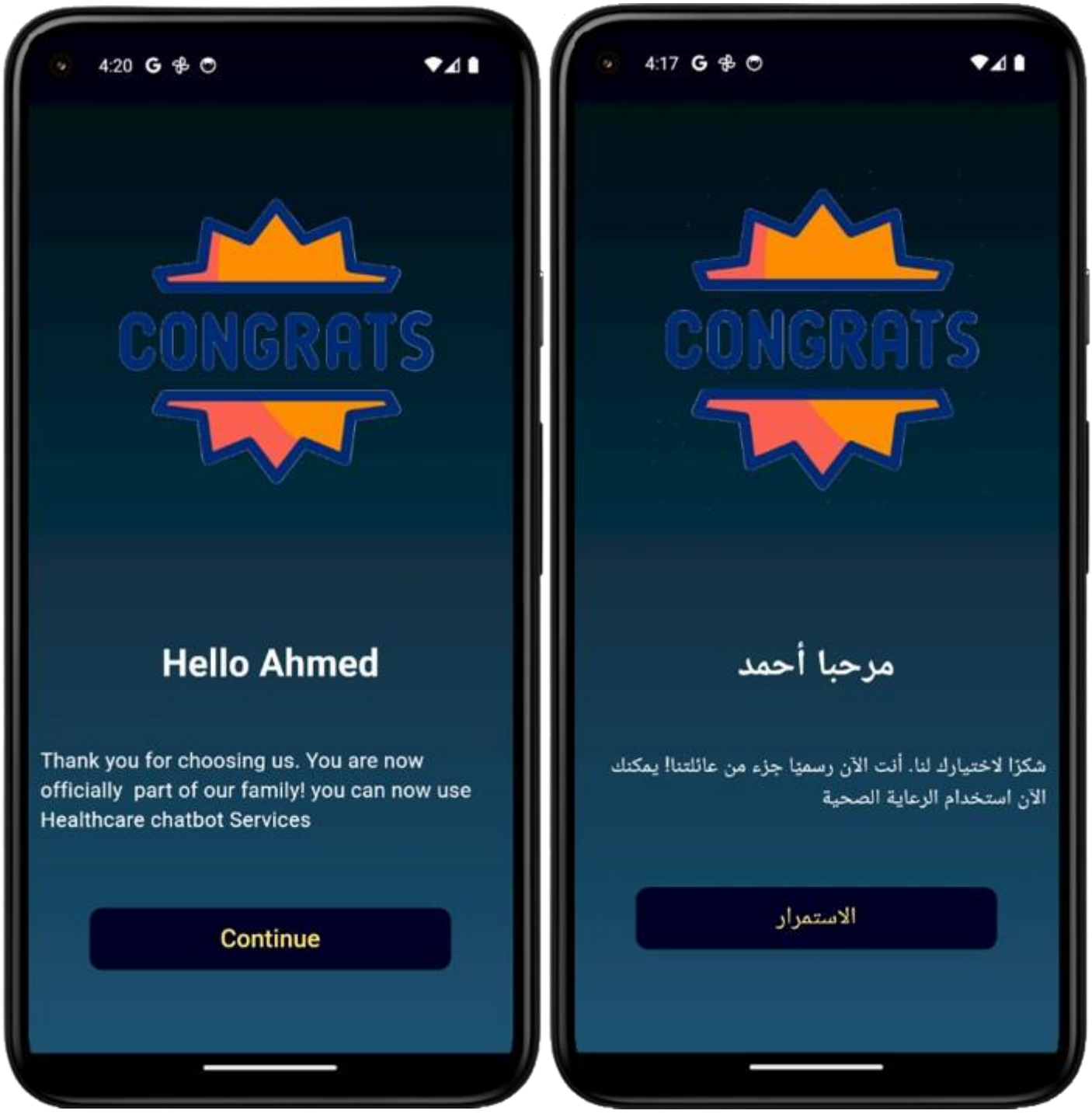


Figure 15: Congrats Screen

chatbot_page:

chatbot_page.dart

lib > screens > chatbot_page.dart > ChatbotPage

```dart
//ignore_for_file: prefer_const_constructors, avoid_unnecessary_containers, sort

import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:healthcare_chatbot/models/Precautions.dart';
import 'package:healthcare_chatbot/main.dart';
import 'package:healthcare_chatbot/models/message.dart';
import 'package:healthcare_chatbot/screens/Map.dart';
import 'package:healthcare_chatbot/screens/aboutUs_page.dart';
import 'package:healthcare_chatbot/screens/contactUs_page.dart';
import 'package:healthcare_chatbot/screens/login_page.dart';
import 'package:healthcare_chatbot/widget/chatbot_bubble.dart';
import 'package:healthcare_chatbot/widget/patient_bubble.dart';
import 'package:http/http.dart' as http;
import 'package:speech_to_text/speech_recognition_result.dart';
import 'package:speech_to_text/speech_to_text.dart';
import 'dart:convert';
import 'package:record/record.dart';
import 'package:avatar_glow/avatar_glow.dart';
import 'package:healthcare_chatbot/services/local/cach_helper.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
import 'package:healthcare_chatbot/core/globals.dart';
class ChatbotPage extends StatefulWidget {
  const ChatbotPage({Key? key}) : super(key: key);

  @override
  State<ChatbotPage> createState() => _ChatbotPageState();
}
class _ChatbotPageState extends State<ChatbotPage> {
  List<Message> messageList = [];
  CollectionReference message =
      FirebaseFirestore.instance.collection('message');
  int langValue = language == 'en' ? 0 : 1;
  TextEditingController controller = TextEditingController();
  final _controller = ScrollController();
  var response;
  TextEditingController _textFieldController = TextEditingController();
  String _responseText = '';

  bool showWelcomeMessage = true;
  SpeechToText _speechToText = SpeechToText();
  bool _speechEnabled = false;
  String _lastWords = '';
  AudioRecorder auido = AudioRecorder();

  Future<void> _sendRequest(String text) async {
    String url = 'https://flask-3xb4.onrender.com/predict';
    try {
      Map<String, String> requestBody = {'text': text};
      String jsonData = jsonEncode(requestBody);

      final response = await http.post(
        Uri.parse(url),
```

chatbot_page.dart

lib > screens > chatbot_page.dart > _ChatbotPageState > _sendRequest

```
30    class _ChatbotPageState extends State<ChatbotPage> {
47      Future<void> _sendRequest(String text) async {
55            headers: <String, String>{
56              'Content-Type': 'application/json; charset=UTF-8',
57            },
58            body: jsonData,
59          );
60
61          if (response.statusCode == 200) {
62            // Extract the value of the "predicted" key from the JSON response
63            Map<String, dynamic> responseData = jsonDecode(response.body);
64            String predictedDisease = responseData['predicted'];
65
66            setState(() {
67              _responseText = predictedDisease;
68              showWelcomeMessage = false; // Set the predicted disease directly
69            });
70          } else if (response.statusCode == 400) {
71            // Extract the error message from the JSON response
72            Map<String, dynamic> errorData = jsonDecode(response.body);
73            String errorMessage = errorData['error'];
74
75            setState(() {
76              _responseText = errorMessage;
77              showWelcomeMessage = false; // Display the error message
78            });
79          } else {
80            throw Exception('Failed to load response');
81          }
82        } catch (e) {
83          print(e.toString());
84        }
85      }
86
87      String userName = '';
88      String userEmail = '';
89      String? value;
90
91      getName() async {
92        DocumentSnapshot userData = await FirebaseFirestore.instance
93            .collection('users')
94            .doc(FirebaseAuth.instance.currentUser!.uid)
95            .get();
96        String firstName = userData['firstName'];
97        String email = userData['email'];
98        userName = firstName;
99        userEmail = email;
100       setState(() {});
101     }
102   ////////////////////////////////////////////////////////
103
104     /// This has to happen only once per app
105     void _initSpeech() async {
106       _speechEnabled = await _speechToText.initialize();
```

chatbot_page.dart

lib > screens > chatbot_page.dart > _ChatbotPageState > initState

```dart
class _ChatbotPageState extends State<ChatbotPage> {
  void _initSpeech() async {
    setState(() {});
  }

  /// Each time to start a speech recognition session
  void _startListening() async {
    await _speechToText.listen(onResult: _onSpeechResult);
    setState(() {});
  }

  void _stopListening() async {
    await _speechToText.stop();
    setState(() {});
    if (_lastWords.isNotEmpty) {
      message.add({
        "message": _lastWords,
        "createdAt": DateTime.now(),
        "id": userEmail,
      });
      await _sendRequest(_lastWords);

      if (_responseText.isNotEmpty) {
        message.add({
          "message": _responseText,
          "createdAt": DateTime.now(),
          "id": 'bot',
        });
        setState(() {});
      }
    }
  }

  void _onSpeechResult(SpeechRecognitionResult result) async {
    setState(() {
      _lastWords = result.recognizedWords;
    });
  }

  ////////////////////////////////////////////////////
  @override
  void initState() {
    super.initState();
    _initSpeech();
    FirebaseAuth.instance.authStateChanges().listen((User? user) {
      if (user == null) {
        // User is signed out
        print('User is signed out');
        clearAllMessages();
      } else {
        // User is signed in
        print('User is signed in');
        getName();
        clearBotMessages();
```

chatbot_page.dart

lib > screens > chatbot_page.dart > _ChatbotPageState > initState

```dart
class _ChatbotPageState extends State<ChatbotPage> {
  void initState() {
    clearBotMessages();
    }
  });
}

void clearAllMessages() async {
  QuerySnapshot allMessagesSnapshot = await message.get();
  for (DocumentSnapshot doc in allMessagesSnapshot.docs) {
    await doc.reference.delete();
  }
}

void clearBotMessages() async {
  QuerySnapshot botMessagesSnapshot =
      await message.where('id', isEqualTo: 'bot').get();
  for (DocumentSnapshot doc in botMessagesSnapshot.docs) {
    await doc.reference.delete();
  }
}

@override
Widget build(BuildContext context) {
  return StreamBuilder<QuerySnapshot>(
    stream: message.orderBy('createdAt').snapshots(),
    builder: (context, snapshot) {
      if (snapshot.hasData) {
        List<Message> messageList = [];
        for (int i = 0; i < snapshot.data!.docs.length; i++) {
          messageList.add(Message.fromJson(snapshot.data!.docs[i]));
        }
        return Scaffold(
          appBar: AppBar(
            automaticallyImplyLeading: true,
            backgroundColor: Color(0xff071C29),
            leading: Padding(
              padding: const EdgeInsets.only(left: 10, right: 15),
              child: Image.asset(
                "images/blue-robot-mascot-logo-icon-design_675467-55 1 (Traced) (1).png",
              ), // Image.asset
            ), // Padding
            title: Padding(
              padding: const EdgeInsets.only(left: 30, right: 20),
              child: Text(
                AppLocalizations.of(context)!.healthcaretitle,
                style: TextStyle(color: Colors.white),
              ), // Text
            ), // Padding
          ), // AppBar
          endDrawer: Drawer(
            child: ListView(
              controller: _controller,
              padding: EdgeInsets.symmetric(),
```


chatbot_page.dart
lib > screens > chatbot_page.dart > _ChatbotPageState > build
class _ChatbotPageState extends State<ChatbotPage> {
Widget build(BuildContext context) {
padding: EdgeInsets.symmetric(),
children: [
UserAccountsDrawerHeader(
currentAccountPictureSize: Size(70, 70),
currentAccountPicture: CircleAvatar(
backgroundImage: AssetImage('images/profile.png'),
backgroundColor: Colors.white,
), // CircleAvatar
accountName: Row(
children: [
Text(
"$userName ",
style: TextStyle(fontSize: 20),
), // Text
],
), // Row
accountEmail: Text(
userEmail,
style: TextStyle(fontSize: 20),
), // Text
decoration: BoxDecoration(
color: const Color.fromARGB(255, 65, 111, 149),
borderRadius: BorderRadius.only(
bottomLeft: Radius.circular(15),
bottomRight: Radius.circular(15),
), // BorderRadius.only
), // BoxDecoration
), // UserAccountsDrawerHeader
Padding(
padding: const EdgeInsets.only(left: 15, right: 15),
child: ExpansionTile(
/* collapsedIconColor: AppTheme.isLightTheme
? HexColor(
AppTheme.primaryColorString!,
)
: Colors.white, */
shape: const UnderlineInputBorder(
borderSide: BorderSide.none,
), // UnderlineInputBorder
title: Text(
AppLocalizations.of(context)!.change_lang,
style: TextStyle(fontSize: 22),
), // Text
tilePadding: EdgeInsets.zero,
leading: const Icon(
Icons.language,
size: 30,
), // Icon
children: [
ListTile(
leading: Text(

chatbot_page.dart

lib > screens > chatbot_page.dart > _ChatbotPageState > build

```dart
class _ChatbotPageState extends State<ChatbotPage> {
  Widget build(BuildContext context) {
                    children: [
                      ListTile(
                        leading: Text(
                          'English',
                          style: Theme.of(context)
                              .textTheme
                              .bodyMedium!
                              .copyWith(
                                fontSize: 18,
                                fontWeight: FontWeight.w700,
                              ),
                        ), // Text
                        trailing: Radio(
                          value: 0 //Languages.english,
                          ,
                          /* fillColor: MaterialStateProperty.all(
                                AppTheme.isLightTheme
                                    ? HexColor(AppTheme.primaryColorString!)
                                    : Colors.white,
                              ), */
                          groupValue:
                              langValue, // profileController.selectedLang.value,

                          onChanged: (value) async {
                            langValue = value!;
                            await CacheHelper.saveData(
                                key: 'lang', value: value == 0 ? 'en' : 'ar');
                            if (context.mounted) {
                              Healthcarechatbot.changeLanguage(
                                  context, value);
                            } //  Healthcarechatbot.changeLanguage(context, value!);
                            //  profileController.changeLang(value!);
                          },
                          //   activeColor: HexColor(AppTheme.primaryColorString!),
                        ), // Radio
                      ), // ListTile
                      ListTile(
                        trailing: Text(
                          "العربية",
                          style: Theme.of(context)
                              .textTheme
                              .bodyMedium!
                              .copyWith(
                                fontSize: 20,
                                fontWeight: FontWeight.w700,
                              ),
                        ), // Text
                        leading: Radio(
                          value: 1,
                          groupValue:
                              langValue, //profileController.selectedLang.value,
                          /*  fillColor: MaterialStateProperty.all(
                                AppTheme.isLightTheme
```

chatbot_page.dart

lib > screens > chatbot_page.dart > _ChatbotPageState > build

```dart
class _ChatbotPageState extends State<ChatbotPage> {
  Widget build(BuildContext context) {
                        onChanged: (value) async {
                          langValue = value!;
                          await CacheHelper.saveData(
                              key: 'lang', value: value == 0 ? 'en' : 'ar');
                          if (context.mounted) {
                            Healthcarechatbot.changeLanguage(
                                context, value);
                          } // profileController.changeLang(value!);
                        },
                      ), // Radio
                    ), // ListTile
                  ],
                ), // ExpansionTile
              ), // Padding
              ////////////////////////////////////////////////////////
              ListTile(
                leading: Icon(
                  Icons.location_on,
                  size: 30,
                ), // Icon
                title: Text(
                    AppLocalizations.of(context)!.find_the_closest_hospital,
                    // "find_the_closest_hospital",
                    style: TextStyle(fontSize: 22)), // Text
                onTap: () {
                  Navigator.push(context,
                      MaterialPageRoute(builder: (context) {
                    return Mappage();
                  })); // MaterialPageRoute
                },
              ), // ListTile
              ListTile(
                leading: Icon(
                  Icons.chat,
                  size: 30,
                ), // Icon
                title: Text(
                  AppLocalizations.of(context)!.new_chat,
                  style: TextStyle(fontSize: 22),
                ), // Text
                onTap: () {
                  clearAllMessages();
                  setState(() {
                    showWelcomeMessage = true;
                  });
                  Navigator.pop(context);
                },
              ), // ListTile

              ListTile(
                leading: Icon(
                  Icons.question_answer_rounded,
```


chatbot_page.dart
lib > screens > chatbot_page.dart > _ChatbotPageState > build
class _ChatbotPageState extends State<ChatbotPage> {
Widget build(BuildContext context) {
Icons.question_answer_rounded,
size: 30,
), // Icon
title: Text(
AppLocalizations.of(context)!.contact_us,
style: TextStyle(fontSize: 22),
), // Text
onTap: () {
Navigator.push(context,
MaterialPageRoute(builder: (context) {
return ContactUsPage();
})); // MaterialPageRoute
},
), // ListTile
ListTile(
leading: Icon(
Icons.info_outline,
size: 30,
), // Icon
title: Text(
AppLocalizations.of(context)!.about_us,
style: TextStyle(fontSize: 22),
), // Text
onTap: () {
Navigator.push(context,
MaterialPageRoute(builder: (context) {
return AboutUsPage();
})); // MaterialPageRoute
},
), // ListTile
Padding(
padding: const EdgeInsets.only(left: 3, bottom: 0),
child: ListTile(
leading: Icon(
Icons.logout,
size: 30,
), // Icon
title: Text(
AppLocalizations.of(context)!.logout,
style: TextStyle(fontSize: 22),
), // Text
onTap: () {
FirebaseAuth.instance.signOut();
Navigator.pushAndRemoveUntil(
context,
MaterialPageRoute(
builder: (context) => LoginScreen(),
), // MaterialPageRoute
(route) => false,
);


chatbot_page.dart
lib > screens > chatbot_page.dart > _ChatbotPageState > build
class _ChatbotPageState extends State<ChatbotPage> {
Widget build(BuildContext context) {
body: Stack(
children: [
Container(
decoration: BoxDecoration(
gradient: LinearGradient(
colors: [
Color(0xff1D5879),
Color(0xff04141D),
],
begin: Alignment.bottomCenter,
end: Alignment.topCenter,
stops: [0.2, 0.85],
), // LinearGradient
), // BoxDecoration
), // Container
Column(
children: [
if (showWelcomeMessage)
Container(
alignment: Alignment.center,
padding: EdgeInsets.symmetric(vertical: 15),
child: Column(
children: [
Icon(
Icons.android,
size: 40,
color: Colors.blueAccent[400],
), // Icon
SizedBox(
height: 2,
), // SizedBox
Text(
AppLocalizations.of(context)!.dialog_message,
style: TextStyle(
fontSize: 20, color: Colors.blueAccent[400])
), // Text
],
), // Column
), // Container
Expanded(
child: ListView.builder(
controller: _controller,
itemCount: messageList.length,
itemBuilder: (context, index) {
return messageList[index].id == userEmail
? PatientBublle(
message: messageList[index],
) // PatientBublle
: ChatBubleBot(
fun: () {
if (precautions


chatbot_page.dart
lib > screens > chatbot_page.dart > _ChatbotPageState > build
class _ChatbotPageState extends State<ChatbotPage> {
Widget build(BuildContext context) {
.containsKey(_responseText)) {
message.add({
"message": precautions[_responseText],
"createdAt": DateTime.now(),
"id": 'bot',
});
}
_responseText = '';
},
load: _responseText,
message: messageList[index],
); // ChatBubleBot
},
), // ListView.builder
), // Expanded
Column(
mainAxisAlignment: MainAxisAlignment.end,
children: [
Padding(
padding: const EdgeInsets.only(bottom: 5),
child: Container(
height: 75,
child: Row(
children: [
Expanded(
child: Padding(
padding: const EdgeInsets.all(8),
child: TextField(
controller: controller,
onSubmitted: (data) async {
if (data.isNotEmpty) {
value = data;
message.add({
"message": data,
"createdAt": DateTime.now(),
"id": userEmail,
});
controller.clear();
_controller.animateTo(
_controller
.position.maxScrollExtent,
duration: Duration(seconds: 1),
curve: Curves.fastOutSlowIn,
);
await _sendRequest(data);
if (_responseText.isNotEmpty) {
message.add({
"message": _responseText,
"createdAt": DateTime.now(),
"id": 'bot',


chatbot_page.dart
lib > screens > chatbot_page.dart > _ChatbotPageState > build
class _ChatbotPageState extends State<ChatbotPage> {
Widget build(BuildContext context) {
"createdAt": DateTime.now(),
"id": 'bot',
});
response = _responseText;
setState(() {});
}
}
},
style: TextStyle(color: Colors.white),
decoration: InputDecoration(
fillColor: Color(0xff050423),
filled: true,
hintText: AppLocalizations.of(context)!
.message_me,
counterStyle: TextStyle(),
suffixIcon: IconButton(
icon: Icon(
Icons.send,
color: Colors.blue,
), // Icon
onPressed: () async {
String data = controller.text;
if (data.isNotEmpty) {
message.add({
"message": data,
"createdAt": DateTime.now(),
"id": userEmail,
});
controller.clear();
_controller.animateTo(
_controller
.position.maxScrollExtent,
duration: Duration(seconds: 1),
curve: Curves.fastOutSlowIn,
);
await _sendRequest(data);
if (_responseText.isNotEmpty) {
message.add({
"message": _responseText,
"createdAt": DateTime.now(),
"id": 'bot',
});
}
}
},
), // IconButton
contentPadding: EdgeInsets.only(
left: 35, top: 40, right: 35), // Ed
hintStyle:

Summary of this page:

We developed a chatbot page, with Firebase backend support connected to our best ML model (SVC) by using a micro web framework (Flask), that takes a symptoms description from user as text then display the predicted disease.

We added five features: (Map, Precautions button, Voice Record button for speech recognition, Listen button for text to speech, Support conversation in Arabic for Arab users and localization features)

The final result of this page:

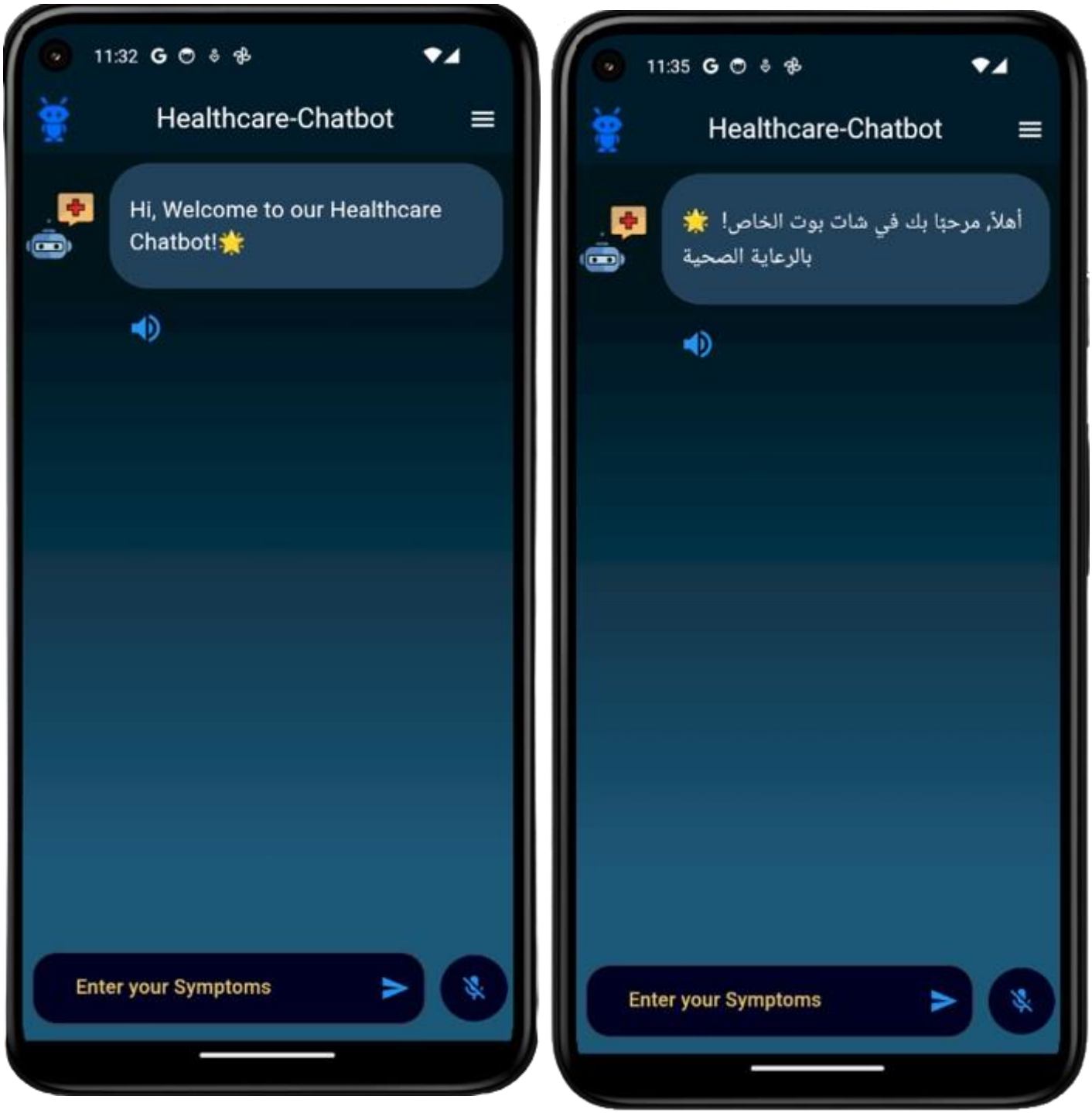


Figure 16: Chatbot Screen_1

-The main chatbot page containing Input Text Box, Voice Record button, Welcome message, and Drawer.

Figure 17: Chatbot Screen_2

-Drawer containing first name and Email, Change Language button for switching the app language between English and Arabic, Map button(Find the closest Hospital), New Chat button, Contact us button, About us button and Log out button.

Figure 18: Chatbot Screen_3

-User entering Hello or any welcome message and the chat responded with this welcome message and introduce itself . In both languages English and Arabic.

Figure 19: Chatbot Screen_4

-User entering valid symptoms description and the chat responded with the possible disease, Listen button, and Precautions button. In both languages English and Arabic.

Figure 20: Chatbot Screen_5

-The same as before but with displaying Some Precautions for this disease. In both languages English and Arabic.

Figure 21: Chatbot Screen_6

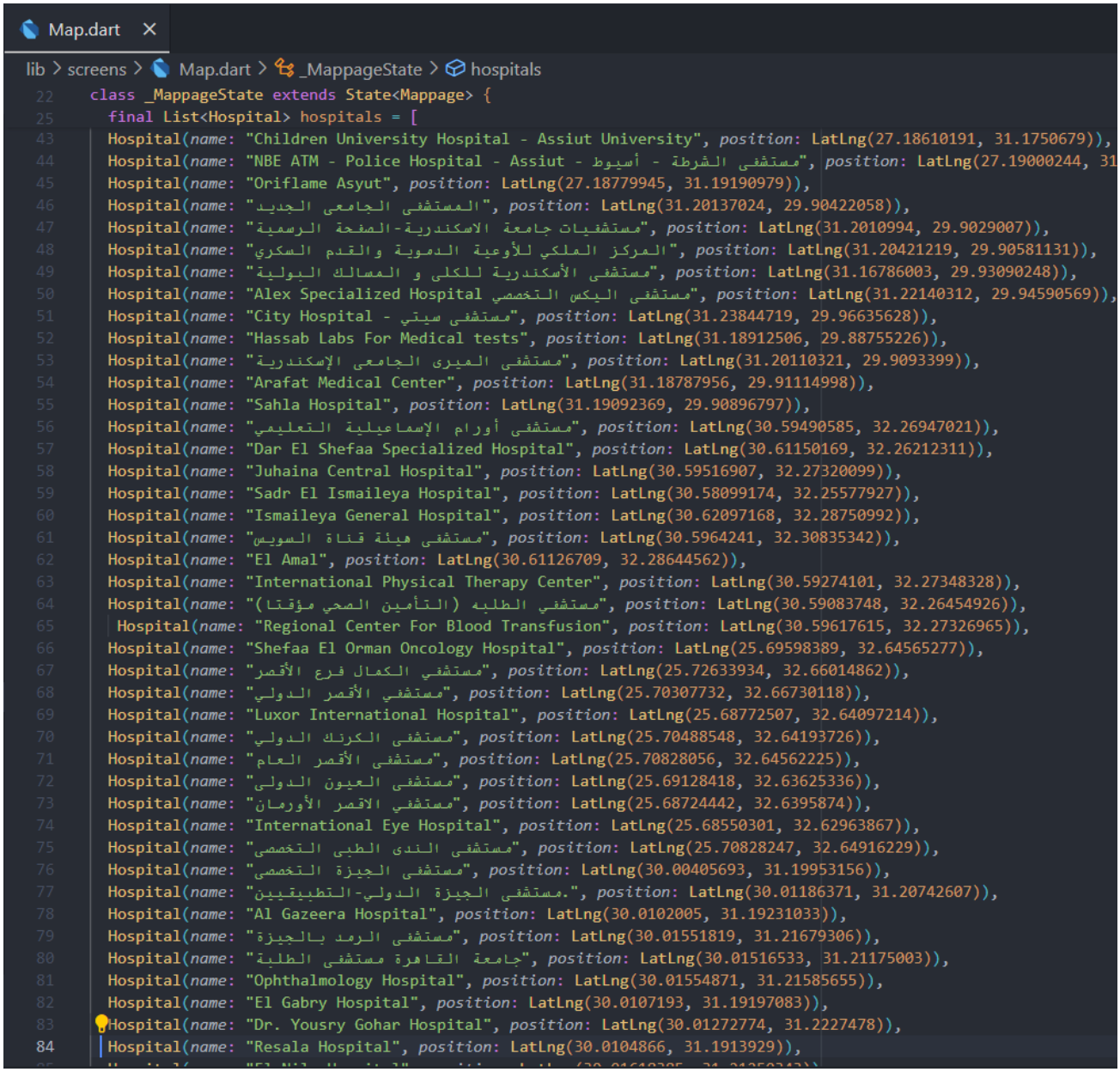
-User entering invalid symptoms description and the chat responded with the Error Message demanding user to enter valid description and Listen button. In both languages English and Arabic.

Map:

Map.dart

lib > screens > Map.dart > Hospital

```dart
// ignore_for_file: prefer_collection_literals, prefer_const_constructors, avoid_function_literals_in_foreach_calls, file_name
import 'dart:async';
import 'dart:collection';
import 'package:flutter/material.dart';
import 'package:geolocator/geolocator.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';

class Mappage extends StatefulWidget {
  const Mappage({Key? key}) : super(key: key);

  @override
  State<Mappage> createState() => _MappageState();
}

class Hospital {
  final String name;
  final LatLng position;

  Hospital({required this.name, required this.position});
}
class _MappageState extends State<Mappage> {
  var myMarkers = HashSet<Marker>();
  var polylines = Set<Polyline>();
  final List<Hospital> hospitals = [
  Hospital(name: "مستشفى الهلال الاحمر باسوان Red Cercent Hospital in Aswan", position: LatLng(24.08800125, 32.90044403)),
  Hospital(name: "Aswan University Hospital", position: LatLng(24.08613777, 32.90711975)),
  Hospital(name: "مستشفيات جامعة أسوان", position: LatLng(24.09004784, 32.8948822)),
  Hospital(name: "Evangelical Mission Hospital - Al Germaniah", position: LatLng(24.08496094, 32.89260101)),
  Hospital(name: "Aswan Mental Health Hospital", position: LatLng(23.98172569, 32.88729858)),
  Hospital(name: "المستشفى التخصصي أسوان", position: LatLng(24.08493423, 32.89469528)),
  Hospital(name: "مستشفي الكوثليك", position: LatLng(24.09536552, 32.89946365)),
  Hospital(name: "مستشفى حفصة التخصصي", position: LatLng(24.09446716, 32.90139771)),
  Hospital(name: "مستشفى أسوان العسكري", position: LatLng(24.04797935, 32.8807373)),
  Hospital(name: "مركز مجدى يعقوب للقلب باسوان", position: LatLng(24.08626747, 32.9082222)),
  Hospital(name: "Archangel Hospital", position: LatLng(27.18157959, 31.18672562)),
  Hospital(name: "Asiout General Hospital", position: LatLng(27.17743111, 31.18348122)),
  Hospital(name: "Abou Tig Central Hospital", position: LatLng(27.17936325, 31.18836212)),
  Hospital(name: "Al Iman General Hospital", position: LatLng(27.17187309, 31.18624115)),
  Hospital(name: "Police Hospital in Assiut", position: LatLng(27.19322777, 31.17954254)),
  Hospital(name: "Assiut University Hospitals", position: LatLng(27.18708611, 31.16552734)),
  Hospital(name: "Clinics University Hospital, Al-Azhar University, Assiut", position: LatLng(27.19996834, 31.17141342)),
  Hospital(name: "Children University Hospital - Assiut University", position: LatLng(27.18610191, 31.1750679)),
  Hospital(name: "NBE ATM - Police Hospital - Assiut - مستشفى الشرطة - أسيوط", position: LatLng(27.19000244, 31.16632652)),
```



Map.dart

lib > screens > Map.dart > _MappageState > hospitals

```dart
class _MappageState extends State<Mappage> {
  final List<Hospital> hospitals = [
    Hospital(name: "Children University Hospital - Assiut University", position: LatLng(27.18610191, 31.1750679)),
    Hospital(name: "NBE ATM - Police Hospital - Assiut - مستشفى الشرطة - أسيوط", position: LatLng(27.19000244, 31
    Hospital(name: "Oriflame Asyut", position: LatLng(27.18779945, 31.19190979)),
    Hospital(name: "المستشفى الجامعى الجديد", position: LatLng(31.20137024, 29.90422058)),
    Hospital(name: "مستشفيات جامعة الاسكندرية-الصفحة الرسمية", position: LatLng(31.2010994, 29.9029007)),
    Hospital(name: "المركز الملكي للأوعية الدموية والقدم السكري", position: LatLng(31.20421219, 29.90581131)),
    Hospital(name: "مستشفى الأسكندرية للكلى و المسالك البولية", position: LatLng(31.16786003, 29.93090248)),
    Hospital(name: "Alex Specialized Hospital مستشفى اليكس التخصصي", position: LatLng(31.22140312, 29.94590569)),
    Hospital(name: "مستشفى سيتي - City Hospital", position: LatLng(31.23844719, 29.96635628)),
    Hospital(name: "Hassab Labs For Medical tests", position: LatLng(31.18912506, 29.88755226)),
    Hospital(name: "مستشفى الميرى الجامعى الإسكندرية", position: LatLng(31.20110321, 29.9093399)),
    Hospital(name: "Arafat Medical Center", position: LatLng(31.18787956, 29.91114998)),
    Hospital(name: "Sahla Hospital", position: LatLng(31.19092369, 29.90896797)),
    Hospital(name: "مستشفى أورام الإسماعيلية التعليمي", position: LatLng(30.59490585, 32.26947021)),
    Hospital(name: "Dar El Shefaa Specialized Hospital", position: LatLng(30.61150169, 32.26212311)),
    Hospital(name: "Juhaina Central Hospital", position: LatLng(30.59516907, 32.27320099)),
    Hospital(name: "Sadr El Ismaileya Hospital", position: LatLng(30.58099174, 32.25577927)),
    Hospital(name: "Ismaileya General Hospital", position: LatLng(30.62097168, 32.28750992)),
    Hospital(name: "مستشفى هيئة قناة السويس", position: LatLng(30.5964241, 32.30835342)),
    Hospital(name: "El Amal", position: LatLng(30.61126709, 32.28644562)),
    Hospital(name: "International Physical Therapy Center", position: LatLng(30.59274101, 32.27348328)),
    Hospital(name: "مستشفي الطلبه (التأمين الصحي مؤقتا)", position: LatLng(30.59083748, 32.26454926)),
    Hospital(name: "Regional Center For Blood Transfusion", position: LatLng(30.59617615, 32.27326965)),
    Hospital(name: "Shefaa El Orman Oncology Hospital", position: LatLng(25.69598389, 32.64565277)),
    Hospital(name: "مستشفي الكمال فرع الأقصر", position: LatLng(25.72633934, 32.66014862)),
    Hospital(name: "مستشفي الأقصر الدولي", position: LatLng(25.70307732, 32.66730118)),
    Hospital(name: "Luxor International Hospital", position: LatLng(25.68772507, 32.64097214)),
    Hospital(name: "مستشفى الكرنك الدولي", position: LatLng(25.70488548, 32.64193726)),
    Hospital(name: "مستشفى الأقصر العام", position: LatLng(25.70828056, 32.64562225)),
    Hospital(name: "مستشفى العيون الدولى", position: LatLng(25.69128418, 32.63625336)),
    Hospital(name: "مستشفي الاقصر الأورمان", position: LatLng(25.68724442, 32.6395874)),
    Hospital(name: "International Eye Hospital", position: LatLng(25.68550301, 32.62963867)),
    Hospital(name: "مستشفى الندى الطبى التخصصى", position: LatLng(25.70828247, 32.64916229)),
    Hospital(name: "مستشفى الجيزة التخصصى", position: LatLng(30.00405693, 31.19953156)),
    Hospital(name: "مستشفى الجيزة الدولي-التطبيقيين.", position: LatLng(30.01186371, 31.20742607)),
    Hospital(name: "Al Gazeera Hospital", position: LatLng(30.0102005, 31.19231033)),
    Hospital(name: "مستشفى الرمد بالجيزة", position: LatLng(30.01551819, 31.21679306)),
    Hospital(name: "جامعة القاهرة مستشفى الطلبة", position: LatLng(30.01516533, 31.21175003)),
    Hospital(name: "Ophthalmology Hospital", position: LatLng(30.01554871, 31.21585655)),
    Hospital(name: "El Gabry Hospital", position: LatLng(30.0107193, 31.19197083)),
    Hospital(name: "Dr. Yousry Gohar Hospital", position: LatLng(30.01272774, 31.2227478)),
    Hospital(name: "Resala Hospital", position: LatLng(30.0104866, 31.1913929)),
```

Map.dart

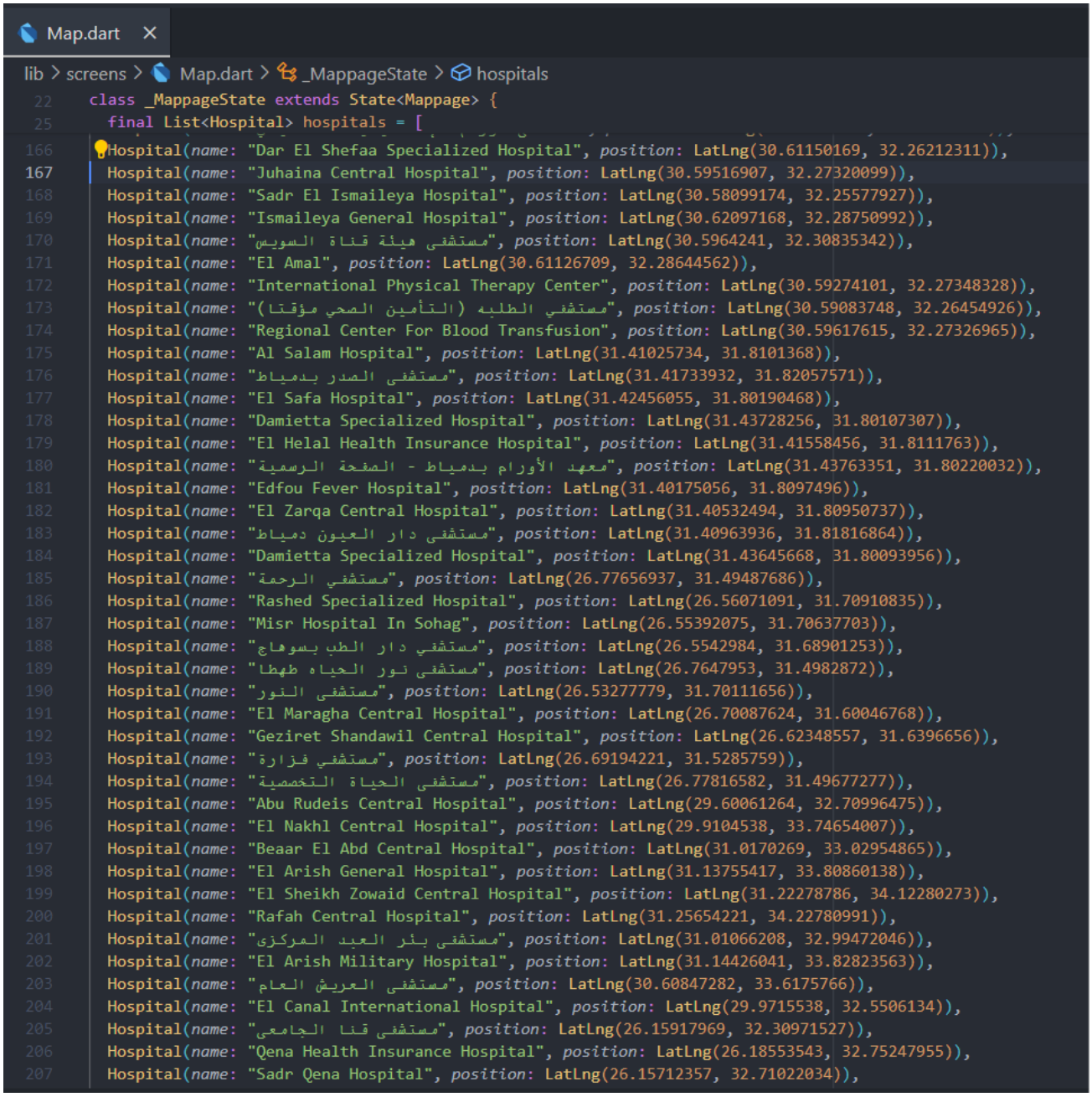
lib > screens > Map.dart > _MappageState > hospitals

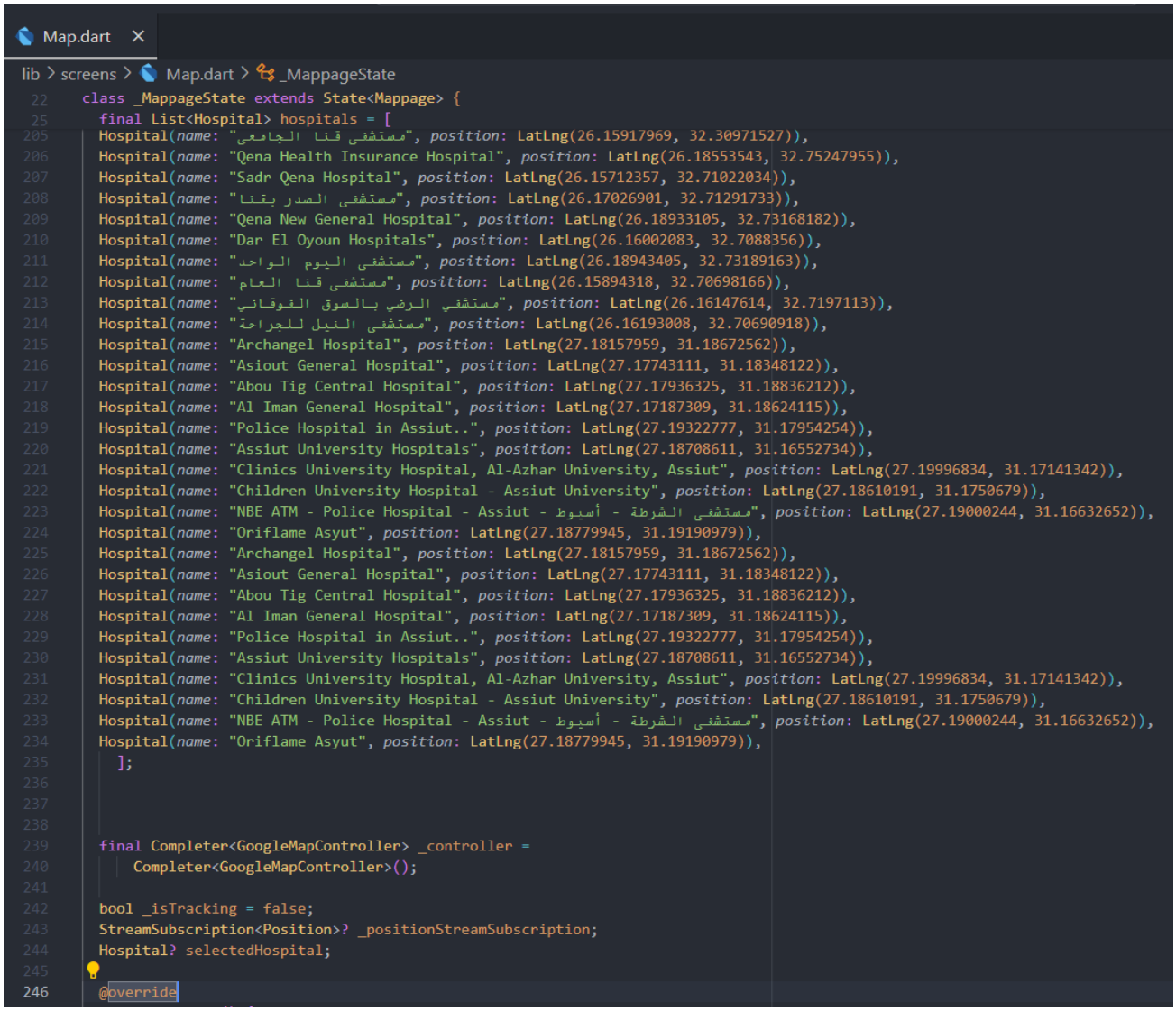
```dart
class _MappageState extends State<Mappage> {
  final List<Hospital> hospitals = [
    Hospital(name: "Dr. Yousry Gohar Hospital", position: LatLng(30.01272774, 31.2227478)),
    Hospital(name: "Resala Hospital", position: LatLng(30.0104866, 31.1913929)),
    Hospital(name: "El Nile Hospital", position: LatLng(30.01618385, 31.21250343)),
    Hospital(name: "مستشفى السلام التخصصي بالفيوم", position: LatLng(29.30426025, 30.84645081)),
    Hospital(name: "مستشفى الزهراء التخصصى", position: LatLng(29.30443954, 30.85108566)),
    Hospital(name: "Makka Specialized Hospital", position: LatLng(29.30678558, 30.85532951)),
    Hospital(name: "الحياة للخدمات الطبية", position: LatLng(29.32369995, 30.8576889)),
    Hospital(name: "مستشفى الفيوم الدولي / مستشفى الطب التخصصي", position: LatLng(29.3069458, 30.89175224)),
    Hospital(name: "مستشفى الشفاء", position: LatLng(29.30952644, 30.84131622)),
    Hospital(name: "مستشفى الندى التخصصي بالفيوم | Faiyum", position: LatLng(29.3121624, 30.8562336)),
    Hospital(name: "El Nabawy El Mohandes General Hospital", position: LatLng(29.31628227, 30.85051537)),
    Hospital(name: "مستشفى الفيوم الدولى", position: LatLng(29.30796051, 30.88565445)),
    Hospital(name: "مستشفي حميات الفيوم", position: LatLng(29.2965126, 30.83158875)),
    Hospital(name: "Cairo Fatemy Hospital", position: LatLng(30.04668427, 31.27088356)),
    Hospital(name: "Children's Cancer Hospital Egypt", position: LatLng(30.02276039, 31.23852921)),
    Hospital(name: "El Houd El Marsoud Hospital", position: LatLng(30.03207207, 31.24709511)),
    Hospital(name: "Cairo University Hospitals", position: LatLng(30.03422356, 31.2277813)),
    Hospital(name: "مستشفى أطفال مصر", position: LatLng(30.02799034, 31.23653984)),
    Hospital(name: "Dr. Said Korraa Hospital", position: LatLng(30.04727364, 31.23849869)),
    Hospital(name: "ElGomhoria Teaching Hospital مستشفى الجمهورية التعليمى", position: LatLng(30.02722549, 31.24122238)),
    Hospital(name: "Al Dorrah Heart Care Hospital", position: LatLng(30.04211998, 31.24383926)),
    Hospital(name: "El Mounira General Hospital", position: LatLng(30.03482628, 31.23807144)),
    Hospital(name: "University Children Hospital At El Mounira - Abulreesh", position: LatLng(30.0293026, 31.23495293)),
    Hospital(name: "Cairo Fatemy Hospital", position: LatLng(30.04668427, 31.27088356)),
    Hospital(name: "Children's Cancer Hospital Egypt", position: LatLng(30.02276039, 31.23852921)),
    Hospital(name: "El Houd El Marsoud Hospital", position: LatLng(30.03207207, 31.24709511)),
    Hospital(name: "Cairo University Hospitals", position: LatLng(30.03422356, 31.2277813)),
    Hospital(name: "مستشفى أطفال مصر", position: LatLng(30.02799034, 31.23653984)),
    Hospital(name: "Dr. Said Korraa Hospital", position: LatLng(30.04727364, 31.23849869)),
    Hospital(name: "ElGomhoria Teaching Hospital مستشفى الجمهورية التعليمى", position: LatLng(30.02722549, 31.24122238)),
    Hospital(name: "Al Dorrah Heart Care Hospital", position: LatLng(30.04211998, 31.24383926)),
    Hospital(name: "El Mounira General Hospital", position: LatLng(30.03482628, 31.23807144)),
    Hospital(name: "University Children Hospital At El Mounira - Abulreesh", position: LatLng(30.0293026, 31.23495293)),
    Hospital(name: "Menouf General Hospital", position: LatLng(30.4709816, 30.92564011)),
    Hospital(name: "مستشفى مصر الحياة التخصصى بالشهداء - المنوفية", position: LatLng(30.59676933, 30.90526962)),
    Hospital(name: "El Safa Hospital", position: LatLng(30.46533775, 30.88595963)),
    Hospital(name: "El Safa Specialized Hospital", position: LatLng(30.46481514, 30.92939568)),
    Hospital(name: "Monofeya University Hospital", position: LatLng(30.57487488, 31.01191521)),
    Hospital(name: "مستشفى الجمعية الشرعية ببهناي", position: LatLng(30.39098167, 31.06875229)),
    Hospital(name: "El Salam Hospital", position: LatLng(30.46331787, 30.93546104)),
    Hospital(name: "Psychological Health & Addiction Treatment Hospital", position: LatLng(30.54461479, 31.04793358)),
    Hospital(name: "مستشفى ميت ربيعه", position: LatLng(30.46519089, 30.9948597)),
```

Map.dart

lib > screens > Map.dart > _MappageState > hospitals

```dart
class _MappageState extends State<Mappage> {
  final List<Hospital> hospitals = [
    Hospital(name: "Alrwad Specialized Hospital", position: LatLng(30.46713829, 30.93371773)),
    Hospital(name: "بيت النغم", position: LatLng(28.10059166, 30.75586128)),
    Hospital(name: "Omar Ibn El Khattab Specialized Hospital", position: LatLng(28.08272743, 30.76492691)),
    Hospital(name: "مستشفي المنيا الجامعه", position: LatLng(28.05653, 30.76390839)),
    Hospital(name: "El Menia General Hospital", position: LatLng(28.1009903, 30.75406265)),
    Hospital(name: "EL Menia University Hospital", position: LatLng(28.08953476, 30.76632118)),
    Hospital(name: "مستشفي المنيا الدولي", position: LatLng(28.09608841, 30.77749825)),
    Hospital(name: "El Menia Hospital For Mental Health & Addiction Treatment", position: LatLng(28.08109283, 30.80702019)),
    Hospital(name: "El Menia Psychiatric Hospital", position: LatLng(28.04374695, 30.77428246)),
    Hospital(name: "مستشفى دار الهلال بطهنشا", position: LatLng(28.03615952, 30.75492096)),
    Hospital(name: "El Farafra Central Hospital", position: LatLng(27.06783295, 27.97322845)),
    Hospital(name: "مستشفى الحميات", position: LatLng(25.44145584, 30.55953217)),
    Hospital(name: "El Kharga General Hospital", position: LatLng(25.44227982, 30.55042267)),
    Hospital(name: "مشروع إنشاء مستشفى الداخله العام", position: LatLng(25.50407982, 29.00006866)),
    Hospital(name: "Sadr El Kharga Hospital", position: LatLng(25.44127083, 30.55976486)),
    Hospital(name: "مستشفى غرب الموهوب", position: LatLng(25.87672424, 28.55901527)),
    Hospital(name: "مستشفى باريس المركزي", position: LatLng(24.67482376, 30.60162735)),
    Hospital(name: "مستشفي", position: LatLng(26.33594513, 31.77669907)),
    Hospital(name: "مستشفي الاورمان للاطفال", position: LatLng(26.43630981, 31.6619339)),
    Hospital(name: "NBE ATM - Al Sadr Hospital (Kharga) - مستشفى الصدر (الخارجة)", position: LatLng(25.4412365, 30.55970573)),
    Hospital(name: "Banisuef General Hospital", position: LatLng(29.06395149, 31.10288811)),
    Hospital(name: "مستشفى بني سويف الجامعي", position: LatLng(29.08039665, 31.105299)),
    Hospital(name: "Al Banna Specialized Center", position: LatLng(29.07803345, 31.0905838)),
    Hospital(name: "مستشفى الزهور الجياه", position: LatLng(28.81825066, 30.89974022)),
    Hospital(name: "مستشفى حميات بني سويف", position: LatLng(29.06844139, 31.09557724)),
    Hospital(name: "Faculty Of Medicine - Banisuef University", position: LatLng(29.07384872, 31.08964729)),
    Hospital(name: "Dar El Oyoun Hospitals", position: LatLng(29.07575417, 31.11478615)),
    Hospital(name: "Somosta Central Hospital", position: LatLng(28.913311, 30.85264778)),
    Hospital(name: "Beba Central Hospital", position: LatLng(28.92635536, 30.97284126)),
    Hospital(name: "مستشفى السلام الخيري - الأشعة", position: LatLng(29.07260704, 31.09710121)),
    Hospital(name: "مستشفى المعلمين (مجموعة اللوتس)", position: LatLng(31.26964378, 32.28581619)),
    Hospital(name: "مستشفي ال يوسف التخصصي", position: LatLng(31.27223778, 32.28497314)),
    Hospital(name: "Al Soliman Hospital", position: LatLng(31.26957512, 32.29239655)),
    Hospital(name: "Sadr Port Said Hospital - El Masah El Bahary", position: LatLng(31.27048492, 32.29190826)),
    Hospital(name: "مستشفى عطاء التخصصي", position: LatLng(31.26172447, 32.28569031)),
    Hospital(name: "El Tadamon Specialized Hospital", position: LatLng(31.26995659, 32.2939682)),
    Hospital(name: "مستشفى الرحمه", position: LatLng(31.26142693, 32.30140305)),
    Hospital(name: "Location Studio", position: LatLng(31.26998329, 32.2964592)),
    Hospital(name: "Port Said General Hospital", position: LatLng(31.26459885, 32.30456161)),
    Hospital(name: "مستشفى المبرة", position: LatLng(31.26958275, 32.29655457)),
    Hospital(name: "مستشفى أورام الإسماعيلية التعليمي", position: LatLng(30.59490585, 32.26947021)),
    Hospital(name: "Dar El Shefaa Specialized Hospital", position: LatLng(30.61150169, 32.26212311)),
```


Map.dart
lib > screens > Map.dart > _MappageState > hospitals
class _MappageState extends State<Mappage> {
final List<Hospital> hospitals = [
Hospital(name: "Dar El Shefaa Specialized Hospital", position: LatLng(30.61150169, 32.26212311)),
Hospital(name: "Juhaina Central Hospital", position: LatLng(30.59516907, 32.27320099)),
Hospital(name: "Sadr El Ismaileya Hospital", position: LatLng(30.58099174, 32.25577927)),
Hospital(name: "Ismaileya General Hospital", position: LatLng(30.62097168, 32.28750992)),
Hospital(name: "مستشفى هيئة قناة السويس", position: LatLng(30.5964241, 32.30835342)),
Hospital(name: "El Amal", position: LatLng(30.61126709, 32.28644562)),
Hospital(name: "International Physical Therapy Center", position: LatLng(30.59274101, 32.27348328)),
Hospital(name: "مستشفي الطلبه (التأمين الصحي مؤقتا)", position: LatLng(30.59083748, 32.26454926)),
Hospital(name: "Regional Center For Blood Transfusion", position: LatLng(30.59617615, 32.27326965)),
Hospital(name: "Al Salam Hospital", position: LatLng(31.41025734, 31.8101368)),
Hospital(name: "مستشفى الصدر بدمياط", position: LatLng(31.41733932, 31.82057571)),
Hospital(name: "El Safa Hospital", position: LatLng(31.42456055, 31.80190468)),
Hospital(name: "Damietta Specialized Hospital", position: LatLng(31.43728256, 31.80107307)),
Hospital(name: "El Helal Health Insurance Hospital", position: LatLng(31.41558456, 31.8111763)),
Hospital(name: "معهد الأورام بدمياط - الصفحة الرسمية", position: LatLng(31.43763351, 31.80220032)),
Hospital(name: "Edfou Fever Hospital", position: LatLng(31.40175056, 31.8097496)),
Hospital(name: "El Zarqa Central Hospital", position: LatLng(31.40532494, 31.80950737)),
Hospital(name: "مستشفى دار العيون دمياط", position: LatLng(31.40963936, 31.81816864)),
Hospital(name: "Damietta Specialized Hospital", position: LatLng(31.43645668, 31.80093956)),
Hospital(name: "مستشفي الرحمة", position: LatLng(26.77656937, 31.49487686)),
Hospital(name: "Rashed Specialized Hospital", position: LatLng(26.56071091, 31.70910835)),
Hospital(name: "Misr Hospital In Sohag", position: LatLng(26.55392075, 31.70637703)),
Hospital(name: "مستشفي دار الطب بسوهاج", position: LatLng(26.5542984, 31.68901253)),
Hospital(name: "مستشفى نور الحياه طهطا", position: LatLng(26.7647953, 31.4982872)),
Hospital(name: "مستشفى النور", position: LatLng(26.53277779, 31.70111656)),
Hospital(name: "El Maragha Central Hospital", position: LatLng(26.70087624, 31.60046768)),
Hospital(name: "Geziret Shandawil Central Hospital", position: LatLng(26.62348557, 31.6396656)),
Hospital(name: "مستشفي فزارة", position: LatLng(26.69194221, 31.5285759)),
Hospital(name: "مستشفى الحياة التخصصية", position: LatLng(26.77816582, 31.49677277)),
Hospital(name: "Abu Rudeis Central Hospital", position: LatLng(29.60061264, 32.70996475)),
Hospital(name: "El Nakhl Central Hospital", position: LatLng(29.9104538, 33.74654007)),
Hospital(name: "Beaar El Abd Central Hospital", position: LatLng(31.0170269, 33.02954865)),
Hospital(name: "El Arish General Hospital", position: LatLng(31.13755417, 33.80860138)),
Hospital(name: "El Sheikh Zowaid Central Hospital", position: LatLng(31.22278786, 34.12280273)),
Hospital(name: "Rafah Central Hospital", position: LatLng(31.25654221, 34.22780991)),
Hospital(name: "مستشفى بئر العبد المركزى", position: LatLng(31.01066208, 32.99472046)),
Hospital(name: "El Arish Military Hospital", position: LatLng(31.14426041, 33.82823563)),
Hospital(name: "مستشفى العريش العام", position: LatLng(30.60847282, 33.6175766)),
Hospital(name: "El Canal International Hospital", position: LatLng(29.9715538, 32.5506134)),
Hospital(name: "مستشفى قنا الجامعى", position: LatLng(26.15917969, 32.30971527)),
Hospital(name: "Qena Health Insurance Hospital", position: LatLng(26.18553543, 32.75247955)),
Hospital(name: "Sadr Qena Hospital", position: LatLng(26.15712357, 32.71022034)),



Map.dart

lib > screens > Map.dart > _MappageState

```dart
class _MappageState extends State<Mappage> {
  final List<Hospital> hospitals = [
  Hospital(name: "مستشفى قنا الجامعى", position: LatLng(26.15917969, 32.30971527)),
  Hospital(name: "Qena Health Insurance Hospital", position: LatLng(26.18553543, 32.75247955)),
  Hospital(name: "Sadr Qena Hospital", position: LatLng(26.15712357, 32.71022034)),
  Hospital(name: "مستشفى الصدر بقنا", position: LatLng(26.17026901, 32.71291733)),
  Hospital(name: "Qena New General Hospital", position: LatLng(26.18933105, 32.73168182)),
  Hospital(name: "Dar El Oyoun Hospitals", position: LatLng(26.16002083, 32.7088356)),
  Hospital(name: "مستشفى اليوم الواحد", position: LatLng(26.18943405, 32.73189163)),
  Hospital(name: "مستشفى قنا العام", position: LatLng(26.15894318, 32.70698166)),
  Hospital(name: "مستشفي الرضي بالسوق الفوقاني", position: LatLng(26.16147614, 32.7197113)),
  Hospital(name: "مستشفى النيل للجراحة", position: LatLng(26.16193008, 32.70690918)),
  Hospital(name: "Archangel Hospital", position: LatLng(27.18157959, 31.18672562)),
  Hospital(name: "Asiout General Hospital", position: LatLng(27.17743111, 31.18348122)),
  Hospital(name: "Abou Tig Central Hospital", position: LatLng(27.17936325, 31.18836212)),
  Hospital(name: "Al Iman General Hospital", position: LatLng(27.17187309, 31.18624115)),
  Hospital(name: "Police Hospital in Assiut..", position: LatLng(27.19322777, 31.17954254)),
  Hospital(name: "Assiut University Hospitals", position: LatLng(27.18708611, 31.16552734)),
  Hospital(name: "Clinics University Hospital, Al-Azhar University, Assiut", position: LatLng(27.19996834, 31.17141342)),
  Hospital(name: "Children University Hospital - Assiut University", position: LatLng(27.18610191, 31.1750679)),
  Hospital(name: "NBE ATM - Police Hospital - Assiut - مستشفى الشرطة - أسيوط", position: LatLng(27.19000244, 31.16632652)),
  Hospital(name: "Oriflame Asyut", position: LatLng(27.18779945, 31.19190979)),
  Hospital(name: "Archangel Hospital", position: LatLng(27.18157959, 31.18672562)),
  Hospital(name: "Asiout General Hospital", position: LatLng(27.17743111, 31.18348122)),
  Hospital(name: "Abou Tig Central Hospital", position: LatLng(27.17936325, 31.18836212)),
  Hospital(name: "Al Iman General Hospital", position: LatLng(27.17187309, 31.18624115)),
  Hospital(name: "Police Hospital in Assiut..", position: LatLng(27.19322777, 31.17954254)),
  Hospital(name: "Assiut University Hospitals", position: LatLng(27.18708611, 31.16552734)),
  Hospital(name: "Clinics University Hospital, Al-Azhar University, Assiut", position: LatLng(27.19996834, 31.17141342)),
  Hospital(name: "Children University Hospital - Assiut University", position: LatLng(27.18610191, 31.1750679)),
  Hospital(name: "NBE ATM - Police Hospital - Assiut - مستشفى الشرطة - أسيوط", position: LatLng(27.19000244, 31.16632652)),
  Hospital(name: "Oriflame Asyut", position: LatLng(27.18779945, 31.19190979)),
    ];



  final Completer<GoogleMapController> _controller =
      Completer<GoogleMapController>();

  bool _isTracking = false;
  StreamSubscription<Position>? _positionStreamSubscription;
  Hospital? selectedHospital;

  @override
```

Map.dart

lib > screens > Map.dart > _MappageState

```dart
class _MappageState extends State<Mappage> {
  @override
  void initState() {
    super.initState();
    determinePosition();
  }

  Future<void> determinePosition() async {
    Position currentPosition = await Geolocator.getCurrentPosition();

    hospitals.sort((a, b) => Geolocator.distanceBetween(
          currentPosition.latitude,
          currentPosition.longitude,
          a.position.latitude,
          a.position.longitude,
        ).compareTo(
          Geolocator.distanceBetween(
            currentPosition.latitude,
            currentPosition.longitude,
            b.position.latitude,
            b.position.longitude,
          ),
        ));

    final closestHospitals = hospitals.sublist(0, 5);

    setState(() {
      closestHospitals.forEach((hospital) {
        myMarkers.add(
          Marker(
            markerId: MarkerId(hospital.name),
            position: hospital.position,
            icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueRed),
            infoWindow: InfoWindow(title: hospital.name),
            onTap: () {
              _onHospitalMarkerTapped(hospital, currentPosition);
            },
          ), // Marker
        );
      });
    });

    final GoogleMapController controller = await _controller.future;
    controller.animateCamera(
```

Map.dart

lib > screens > Map.dart > _MappageState > determinePosition

```dart
class _MappageState extends State<Mappage> {
  Future<void> determinePosition() async {

    final GoogleMapController controller = await _controller.future;
    controller.animateCamera(
      CameraUpdate.newLatLng(
        LatLng(currentPosition.latitude, currentPosition.longitude),
      ),
    );

    _positionStreamSubscription =
        Geolocator.getPositionStream().listen((Position? position) {
      if (_isTracking) {
        _updateMarkers(position!);
        if (selectedHospital != null) {
          _updateRoute(position, selectedHospital!.position);
        }
      }
    });
  }

  void _updateMarkers(Position position) {
    myMarkers.removeWhere((marker) => marker.markerId.value == 'userLocation');
    myMarkers.add(
      Marker(
        markerId: MarkerId('userLocation'),
        position: LatLng(position.latitude, position.longitude),
        icon: BitmapDescriptor.defaultMarker,
        infoWindow: InfoWindow(title: 'موقع المستخدم'),
      ), // Marker
    );
  }

  void _onHospitalMarkerTapped(Hospital hospital, Position currentPosition) {
  setState(() {
    // Update the selection
    selectedHospital = hospital;

    // Calculate distance between user and selected hospital
    double distance = Geolocator.distanceBetween(
      currentPosition.latitude,
      currentPosition.longitude,
      hospital.position.latitude,
      hospital.position.longitude,
    );
```

Map.dart

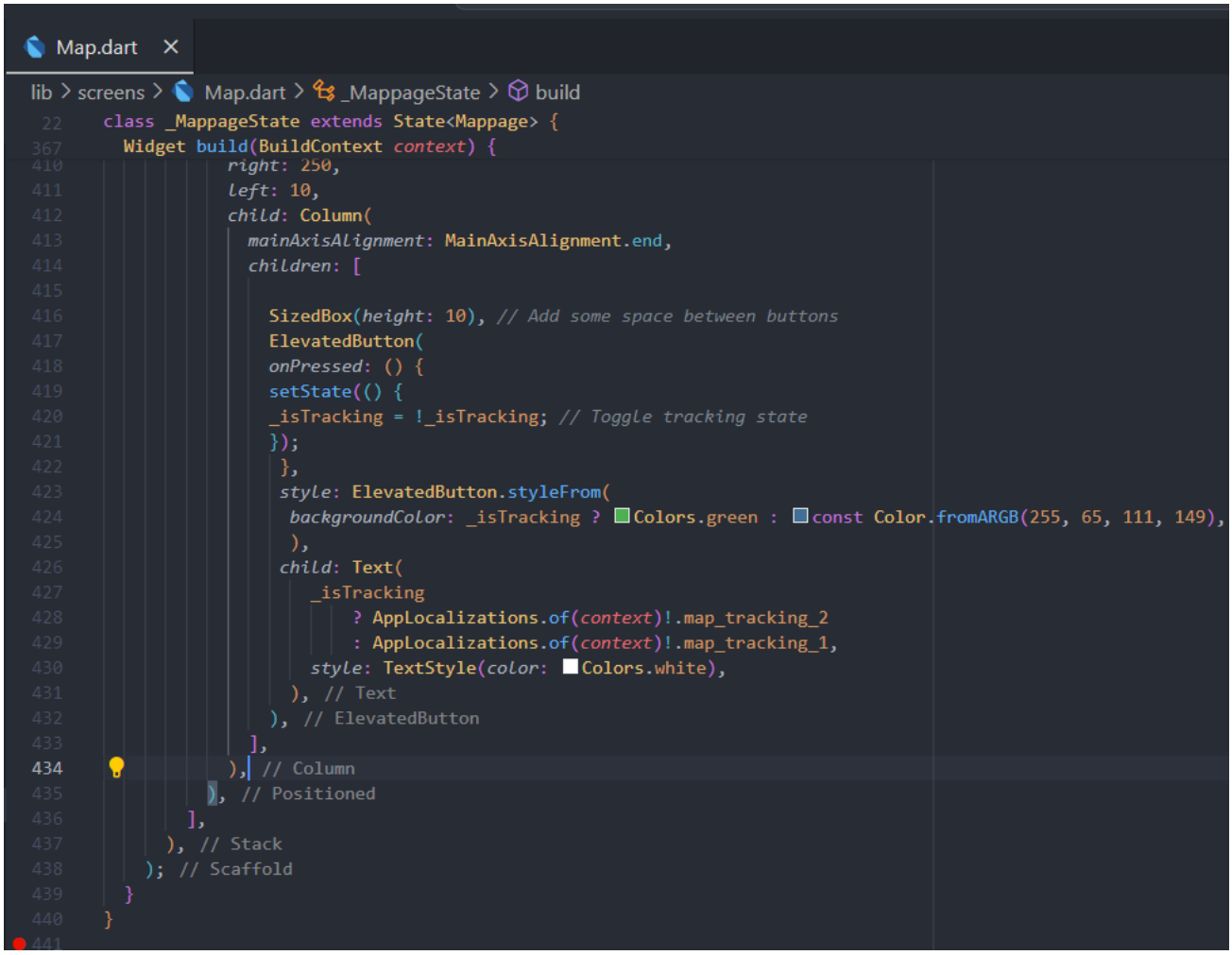
lib > screens > Map.dart > _MappageState > _onHospitalMarkerTapped

```dart
class _MappageState extends State<Mappage> {
  void _onHospitalMarkerTapped(Hospital hospital, Position currentPosition) {

    // Convert distance from meters to kilometers
    double distanceInKm = distance / 1000;

    // Show SnackBar with hospital name and distance in kilometers
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Distance to ${hospital.name}: ${distanceInKm.toStringAsFixed(2)} Km'),
      ), // SnackBar
    );

    // Update route to the selected hospital
    _updateRoute(currentPosition, hospital.position);

    // Add user location marker
    _updateMarkers(currentPosition);
  });
}

void _updateRoute(Position userPosition, LatLng hospitalPosition) {
  polylines.clear(); // Clear previous polyline
  polylines.add(
    Polyline(
      polylineId: PolylineId('route'),
      points: [LatLng(userPosition.latitude, userPosition.longitude), hospitalPosition],
      width: 5,
      color: Colors.blue,
    ), // Polyline
  );
}

@override
void dispose() {
  super.dispose();
  _positionStreamSubscription?.cancel();
}

@override
Widget build(BuildContext context) {
return Scaffold(
  appBar: AppBar(
    title: Center(
      child: Text(
```

Map.dart

lib > screens > Map.dart > _MappageState > build

```dart
class _MappageState extends State<Mappage> {
  Widget build(BuildContext context) {
    title: Center(
      child: Text(
        AppLocalizations.of(context)!.map_title,
        style: TextStyle(
            fontSize: 27, fontWeight: FontWeight.bold, color: Colors.white),
      ), // Text
    ), // Center
    backgroundColor: const Color.fromARGB(255, 65, 111, 149),
    actions: <Widget>[
      IconButton(
      onPressed: () {
       setState(() {
      selectedHospital = null;
      polylines.clear();
       });
       },
    icon: Icon(Icons.delete_forever_sharp),
    // هنا Text يمكنك إزالة العلامة
    ) // IconButton
    ], // <Widget>[]
  ), // AppBar
    body: Stack(
      children: [
        GoogleMap(
          zoomGesturesEnabled: true,
          mapType: MapType.normal,
          initialCameraPosition: CameraPosition(
            target: LatLng(26.558173,31.699592 ),
            zoom: 14,
          ), // CameraPosition
          markers: myMarkers,
          polylines: polylines,
          onMapCreated: (GoogleMapController controller) {
            _controller.complete(controller);
          },
          myLocationEnabled: _isTracking, // Enable or disable my location based
          myLocationButtonEnabled: false, // Disable default my location button
        ), // GoogleMap
        Positioned(
          bottom: 25,
          right: 250,
          left: 10,
```



Summary of this page:

-Displays a Google Map with functionality to track the user's location, display the five nearby hospitals, and draw routes between the user's location and selected hospitals.

The final result of this page:

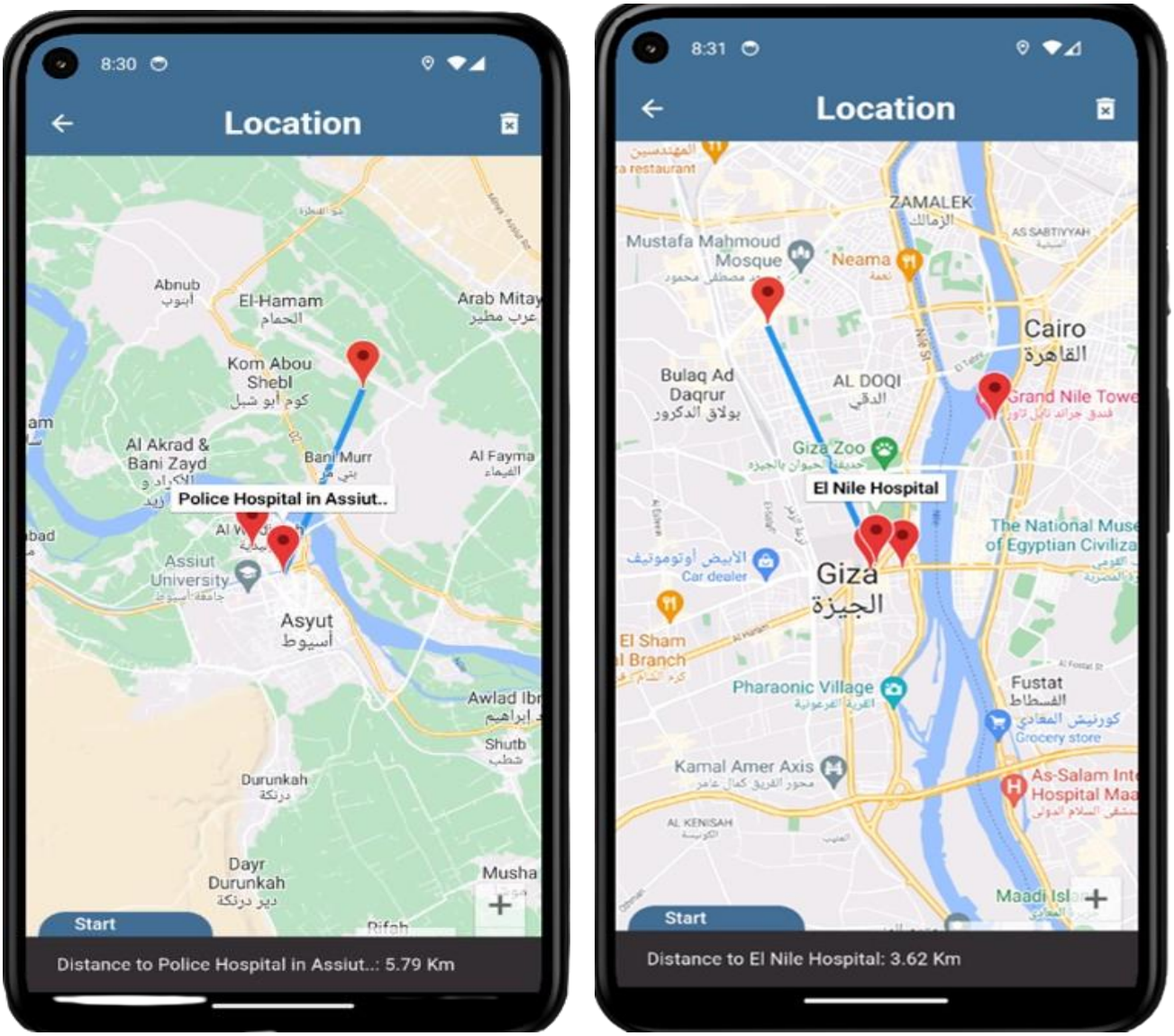


Figure 22: Map Screen_1

- Initially, when the map is opened, the user's location is determined automatically, and a list of hospitals is made available for the user.

- When the user selects a specific hospital from the list, a mark is placed on this hospital, and when the user moves, the user is directed to it.

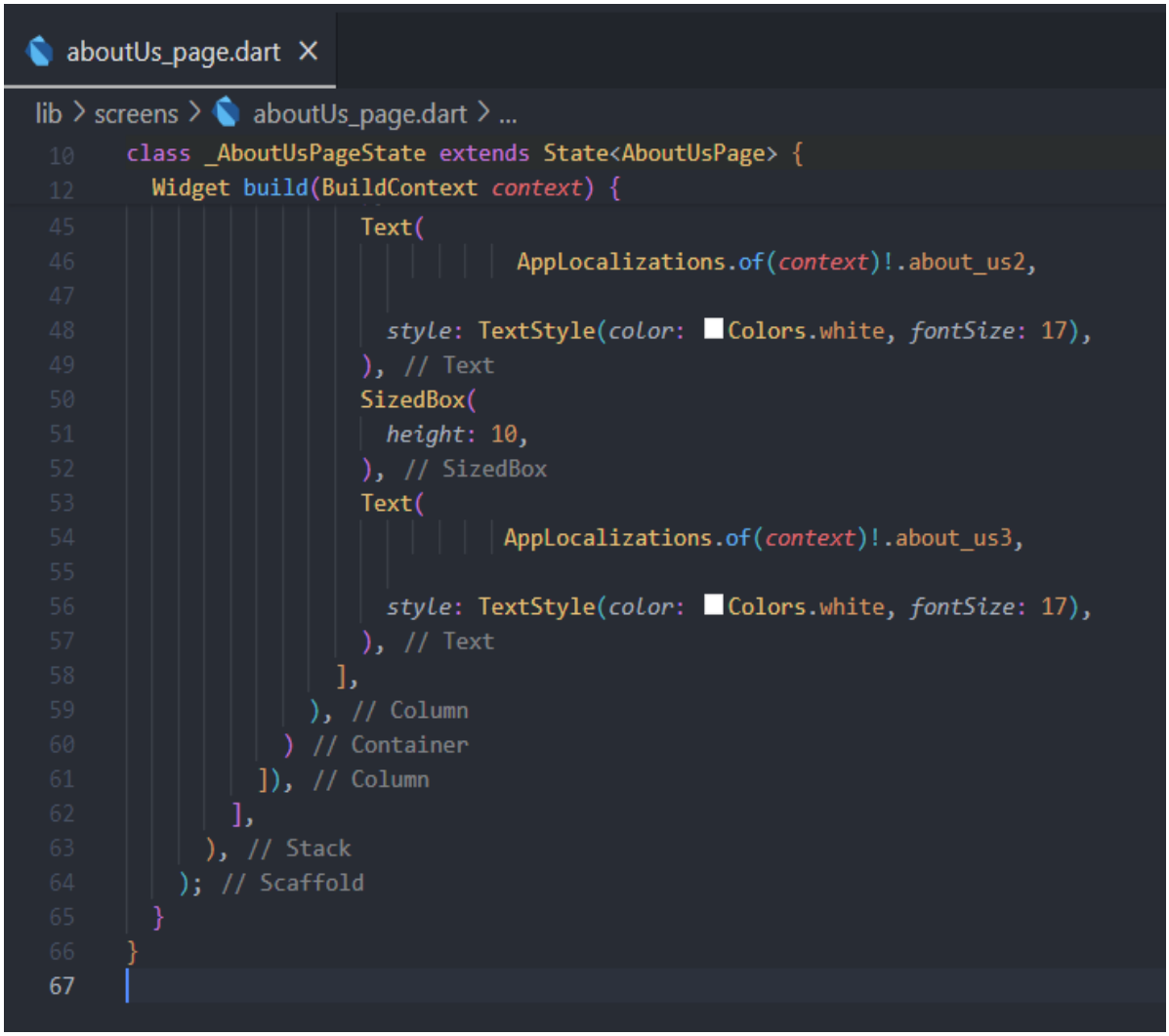
Figure 23: Map Screen_2

aboutUs_page:

aboutUs_page.dart

lib > screens > aboutUs_page.dart > _AboutUsPageState > build

```dart
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';

class AboutUsPage extends StatefulWidget {
  const AboutUsPage({super.key});
  @override
  State<AboutUsPage> createState() => _AboutUsPageState();
}
class _AboutUsPageState extends State<AboutUsPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.yellow,
      body: Stack(
        children: [
          Container(
            decoration: BoxDecoration(
              gradient: LinearGradient(
                  colors: [
                    Color(0xff1D5879),
                    Color(0xff04141D),
                  ],
                  begin: Alignment.bottomCenter,
                  end: Alignment.topCenter,
                  stops: [0.2, 0.85]), // LinearGradient
            ), // BoxDecoration
          ), // Container
          Column(children: [
            SizedBox(
              height: 50,
            ), // SizedBox
            Image.asset("images/aboutus.png"),
            Container(
              padding: EdgeInsets.all(8),
              child: Column(
                children: [
                  Text(
                  AppLocalizations.of(context)!.about_us1,
                      style: TextStyle(color: Colors.white, fontSize: 19),
                  ), // Text
                  SizedBox(
                    height: 15,
                  ), // SizedBox
```



Summary of this page:

- Displays a page that shows information about our project, providing a diagram that expresses the motivations for developing a healthcare chatbot and its capabilities.

The final result of this page:

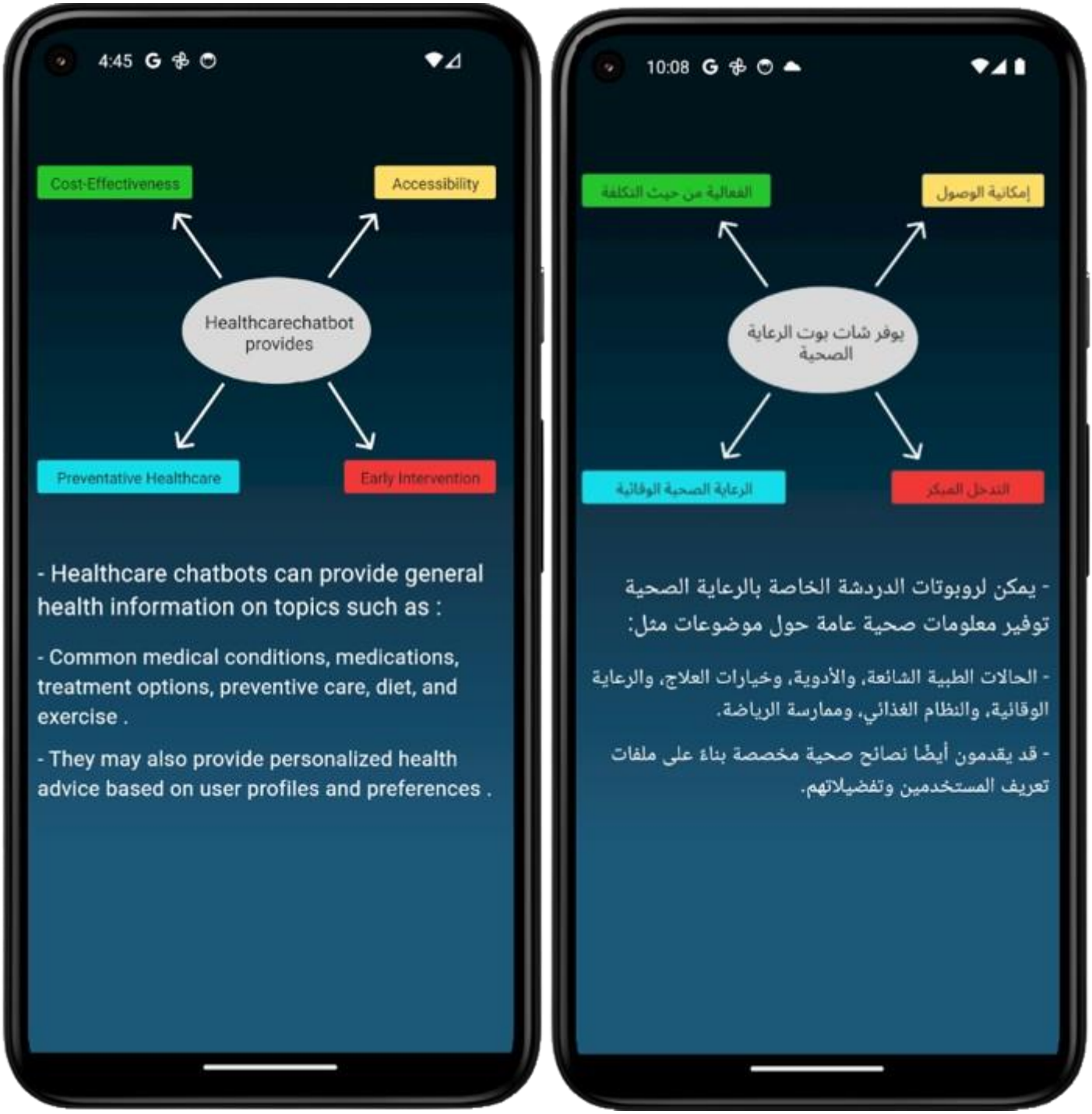


Figure 24: About_us Screen

contactUs_page:

contactUs_page.dart

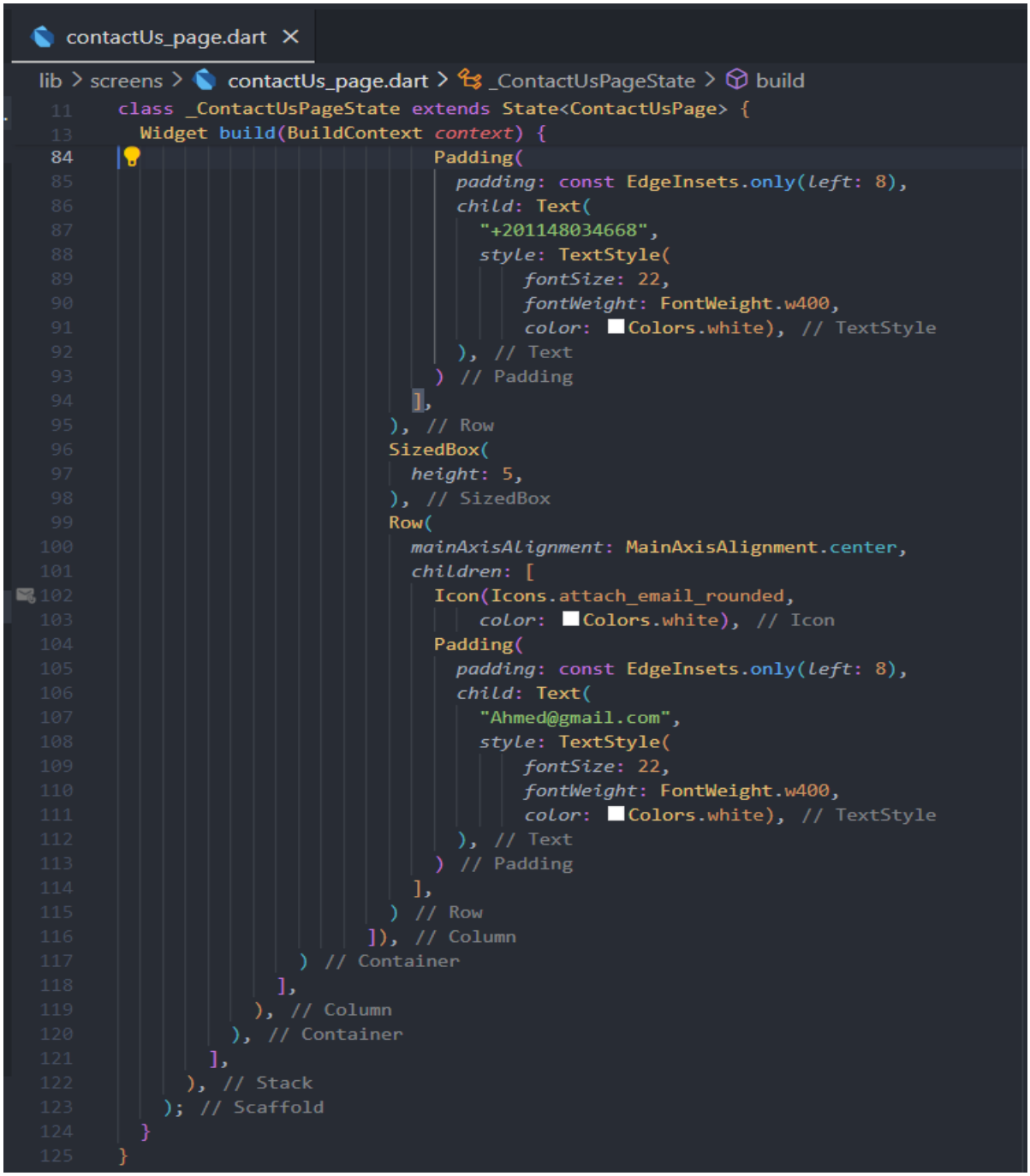
lib > screens > contactUs_page.dart > ContactUsPage

```dart
// ignore_for_file: prefer_const_constructors, sized_box_for_whitespace, avoid_u
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';

class ContactUsPage extends StatefulWidget {
  const ContactUsPage({super.key});

  @override
  State<ContactUsPage> createState() => _ContactUsPageState();
}
class _ContactUsPageState extends State<ContactUsPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          Container(
            decoration: BoxDecoration(
              gradient: LinearGradient(
                  colors: [
                    Color(0xff1D5879),
                    Color(0xff04141D),
                  ],
                  begin: Alignment.bottomCenter,
                  end: Alignment.topCenter,
                  stops: [0.2, 0.85]), // LinearGradient
            ), // BoxDecoration
          ), // Container
          Container(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              children: [
                SizedBox(
                  height: 120,
                ), // SizedBox
                Container(
                    width: 200,
                    height: 200,
                    child: Image.asset(
                      "images/Contactus.png",
                      fit: BoxFit.contain,
                    )), // Image.asset // Container
                Container(
                  padding: EdgeInsets.all(15),
```

contactUs_page.dart

lib > screens > contactUs_page.dart > _ContactUsPageState > build

```dart
class _ContactUsPageState extends State<ContactUsPage> {
  Widget build(BuildContext context) {
                Container(
                  padding: EdgeInsets.all(15),
                  child: Column(
                      mainAxisAlignment: MainAxisAlignment.center,
                      children: [
                        SizedBox(
                          height: 10,
                        ), // SizedBox
                        Text(
                          AppLocalizations.of(context)!.contact_us,
                          style: TextStyle(
                              fontSize: 30,
                              fontWeight: FontWeight.w600,
                              color: Colors.white), // TextStyle
                        ), // Text
                        SizedBox(
                          height: 30,
                        ), // SizedBox
                        Row(
                          mainAxisAlignment: MainAxisAlignment.center,
                          children: [
                            Icon(Icons.location_on, color: Colors.white),
                            Padding(
                              padding: const EdgeInsets.only(left: 8),
                              child: Text(
                                AppLocalizations.of(context)!.sohag_city,
                                style: TextStyle(
                                    fontSize: 22,
                                    fontWeight: FontWeight.w400,
                                    color: Colors.white), // TextStyle
                              ), // Text
                            ), // Padding
                          ],
                        ), // Row
                        SizedBox(
                          height: 7,
                        ), // SizedBox
                        Row(
                          mainAxisAlignment: MainAxisAlignment.center,
                          children: [
                            Icon(Icons.phone, color: Colors.white),
                            Padding(
```



Summary of this page:

- Displays a page that provides some of our contact information including location, phone number, and email address.

The final result of this page:

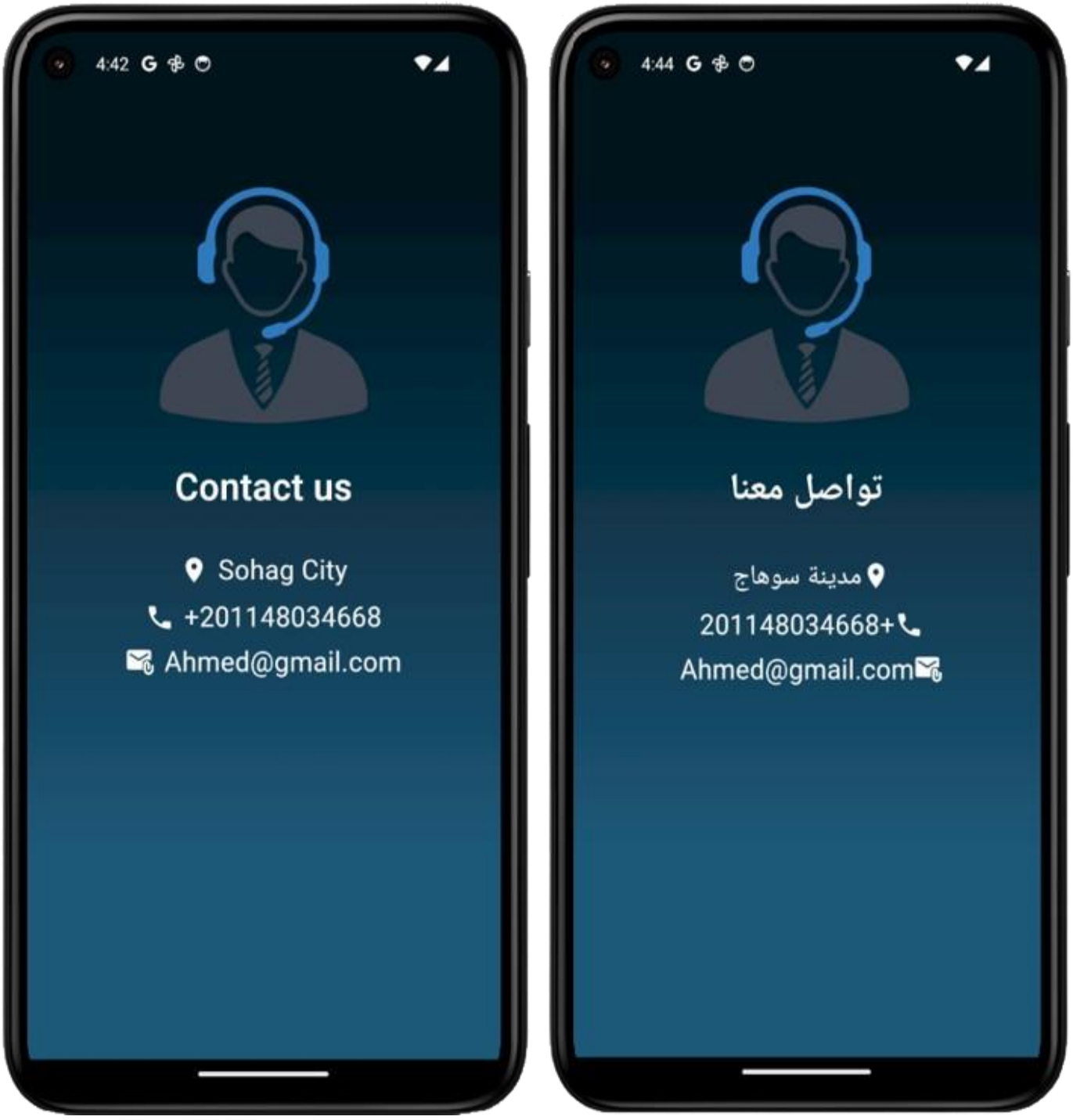


Figure 25: Contact_us Screen

Chapter 5

Advantages and Disadvantages

Advantages of chatbot

1- Improved Accessibility: The chatbot enables patients to access healthcare services remotely and conveniently, thereby reducing barriers to healthcare access, particularly in rural or underserved areas.

2- Efficiency: The chatbot streamlines diagnostics by efficiently collecting patient symptom data and promptly providing disease classifications, potentially reducing wait times for medical consultations.

3- Early Intervention: The chatbot aids in early intervention and timely medical care by analysing symptoms and providing preliminary disease classifications, potentially preventing disease progression, and improving patient outcomes.

4- Scalability: Once developed, the chatbot can be scaled to serve a large number of users simultaneously without significant additional resources, making it a cost-effective solution for healthcare providers.

5- Data-driven Insights: The data collected by the chatbot can be analysed to identify trends, patterns, and correlations in symptoms and diseases, providing valuable insights for healthcare research and decision-making.

Disadvantages of chatbot

1- Misdiagnosis Risk: There's a risk of misdiagnosis or incorrect disease classification, especially if the machine learning model lacks comprehensive training data or encounters unfamiliar or rare conditions, potentially leading to patient harm or dissatisfaction.

2- Limited Scope: The chatbot's effectiveness may be limited to the diseases and symptoms it has been trained on, potentially missing out on less common or emerging conditions that are not included in its dataset.

3- Dependence on Technology: Patients who are not comfortable or familiar with technology may struggle to use the chatbot effectively, leading to reduced accessibility for certain demographics, such as elderly or technologically challenged individuals.

4- Human Interaction Replacement: While the chatbot provides convenience and accessibility, it may lack the empathy, nuanced understanding, and personalized care that human healthcare professionals can offer, potentially leading to a sense of detachment or dissatisfaction among patients

Chapter 6

Related Work

[1]

"Multilingual Healthcare Chatbot Using Machine Learning" (2021) by Sagar Badlani, Tanvi Aditya, Meet Dave and Sheetal Chaudhari from Mumbai, India .[1]

- The system created as a Web application.
- Python programming language.
- The chatbot application supports English & Hindi languages.
- Best Classification Algorithm used : (Random Forest), Accuracy : [98.43%].

Limitations : - No support for Arabic Language.

- Their best model accuracy is lower than our best model accuracy [99.43%]

[2]

"Healthcare Chatbot using Decision Tree Algorithm" (2022) by Dr. Renuka R Devi, Himanshu Banerjee, Baibhab Saha from SRMIST, Chennai, India. [2]

- The system created as a Desktop application.
- Python programming language.
- Only Classification Algorithm used : (Decision Tree) with No provided Accuracy.

Limitations : - No support for Arabic Language

- Lack of GUI Usability and Affordability/Availability.

[3]

"HEALTHCARE CHATBOT" (2021) by Students (Athulya N, Jeeshna K, S J Aadithyan, U Sreelakshmi), Assistant Professor Hairunizha Alias Nisha Rose from Thalassery, India.[3]

- The system created as a Web application.
- Python programming language.
- Only Classification Algorithm used : (Decision Tree), Accuracy : [78.24%].

Limitations : - No support for Arabic Language

- Limited diseases in their dataset.

- Technical issues with Voice messages

Chapter 7

Conclusion and Future Scope

Conclusion:

At last, we managed to develop a healthcare chatbot application that interacts with the patient providing initial disease classification and some precautions using usability techniques:

We used NLP, Supervised ML algorithms to predict disease from user symptoms input.

In Flutter Application, we made Voice Record, google Map, Listen button, Precautions button and Support for GUI and Conversation in both Arabic and English, to ensure usability technique between chatbot and patient.

We connected our ML model and Flutter mobile application using a micro web framework (Flask).

Future Scope:

In the future, we will search for more Datasets in order to collect different and larger number of observations and utilize advanced Deep learning techniques to help us effectively in the natural language processing part to make chatbot even more interactive with the patient.

Chapter 8

Time Plan

STAGE 01: (2 weeks)

Research and understand the idea of the project and learn.

STAGE 02: (2 weeks)

Searching and understanding the dataset.

STAGE 03: (3 weeks)

Data preprocessing and cleaning.

STAGE 04: (4 weeks)

Machine Learning Models.

STAGE 05: (1 week)

Preparing for presentation.

STAGE 06: (4 weeks)

Project Design.

STAGE 07: (3 weeks)

Integration ML Model with Design.

STAGE 08: (3 weeks)

Make more features in project.

STAGE 09: (1 week)

Making Documentation.

STAGE 10: (1 week)

Preparing for presentation.

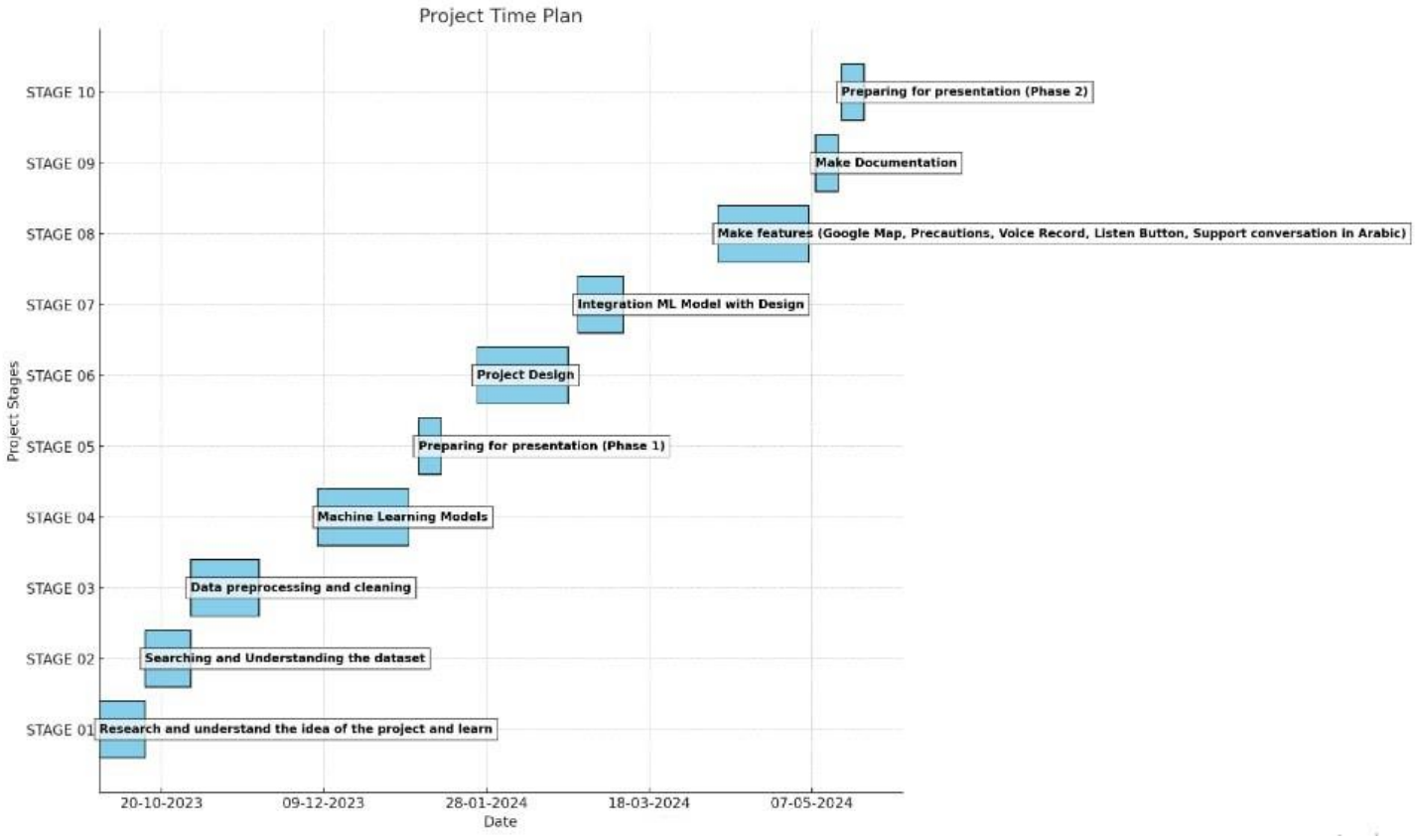


Figure 26: Project Time Plan

References:

➢ Papers

[1] Smith, J., & Lee, A. (2024). Multilingual healthcare chatbot using machine learning. *Journal of Medical Informatics*, 12(3), 145-162.

https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=Multilingual+Healthcare+Chatbot+Using+Machine+Learning&btnG=

[2] Devi, R. R., Banerjee, H., & Saha, B. (2022). Healthcare Chatbot using Decision Tree Algorithm. International Journal of Innovative Science and Research Technology, 7(10).

https://doi.org/10.5281/zenodo.7315590

[3] Athulya N, Jeeshna K, Aadithyan SJ, Sreelakshmi U, & Rose HN. (2021). Healthcare Chatbot. International Journal of Creative Research Thoughts (IJCRT), 9(4), 194-202. Available at:

https://ijcrt.org/papers/IJCRTH020011.pdf

➢ Dataset

[4] Symptom2Disease Dataset

https://www.kaggle.com/datasets/niyarrbarman/symptom2disease

➢ Selected book for ML & NLP

[5] Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. (2nd ed.). Geron Aurelien. (2019). O'Reilly Media, Inc.

➢ Dart & Flutter

[6] Dart docs

https://dart.dev/guides

[7] Flutter docs

https://docs.flutter.dev/

➢ Selected book for Flutter

[8] Hands-On Flutter with Frank Zammetti, in 2019. The printing house is Apress Media, LLC.