Faculty Of Engineering

Computer Engineering Department

Helwan university

Disease Prediction System

Supervised By

Dr: Ahmed Elbadawy

Prepared By

Mohamed Mahrous Abd-Elsalam

Mohamed Hassan Mansour

Abdelrahman Ehab Mohamed

Sally Youssef Saeed

Manar Anwar Farouq

2022

TABLE OF CONTENTS

LIST OF FIGURES

Abstract

Disease Prediction using Machine Learning is a system which predicts the disease based on the information or the symptoms he/she enter into the system and provides the accurate results based on that information. If the patient is not much serious and the user just wants to know the type of disease, he/she has been through. It is a system which provides the user the tips and tricks to maintain the health system of the user and it provides a way to find out the disease using this prediction. Nowadays health industry plays major role in curing the diseases of the patients so this is also some kind of help for the health industry to tell the user and also it is useful for the user in case he/she doesn't want to go to the hospital or any other clinics, so just by entering the symptoms and all other useful information the user can get to know the disease he/she is suffering from and the health industry can also get benefit from this system by just asking the symptoms from the user and entering in the system and in just few seconds they can tell the exact and up to some extent the accurate diseases. This Disease Prediction Using Machine Learning is completely done with the help of Machine Learning algorithms that can predict the disease with high accuracy. and also using the dataset that is available previously by multiple sources.

Keywords

Disease Prediction, Symptoms, Data Science, Data Analysis, Machine Learning, Flutter, API, Mobile App.

1.Introduction

In this project, we are going to use data science techniques to analyze and extract insights from real patients' data and build machine learning models that can predict the patient disease according to his symptoms. and we are going to build a mobile application using flutter that will be the GUI for the user, so at the end of this project we will end up with a mobile app that any user can use and enter his symptoms and see the resulted disease at any time.

1.1. Problem statement

Accurate and on-time analysis of any health-related problem is important for the prevention and treatment of the illness. The traditional of diagnosis may not be sufficient in the case of a serious ailment.

1.2. Objective

Our objective is to develop a medical diagnosis system based on machine learning (ML) algorithms for prediction of any disease that can way help in a more accurate and faster diagnosis than the conventional method. Also, given the circumstances we are living in at the present time due to the Covid-19, this project will be very effective in achieving social distancing and reducing overcrowding in hospitals, which will reduce the risk of infection.

1.3. Motivation

We are very motivated to build this project for many reasons, which are the following:

- This project will serve the medical field.
- A lot of patients data is abundantly available on the internet, which will help us complete the project.
- At the end of this project, we expect to deliver a very intelligent system that can diagnose any patient according to their symptoms.
- Our gain from this project will be a huge experience in different fields which are mobile application, machine learning, data gathering and analysis, APIs and how we can connect them with our application.

1.4. Development Process

- First, We are going to use all the data-gathering techniques(e.g. web scraping, APIs, Free datasets on the internet) to gather real and trusted data about patients that have already been diagnosed before according to their symptoms.
- The second stage will be the most important part of our project, which will be the data cleaning and analysis part, In this part, we will use EDA(Exploratory Data Analysis)Techniques to analyze, visualize, and understand the data (domain knowledge). After that, we are going to clean the data and prepare it to enter into our classification machine learning algorithms for training, and then we are going to evaluate our trained algorithms to select the highest accuracy among them.
- After that, we will deploy our best accuracy trained algorithm as an API, and then, building our mobile application that will connect with this API via the internet and act as the GUI to the user, so simply, the user can enter his symptoms and see the result of his predicted disease at the same time.

1.5. Project Timeline

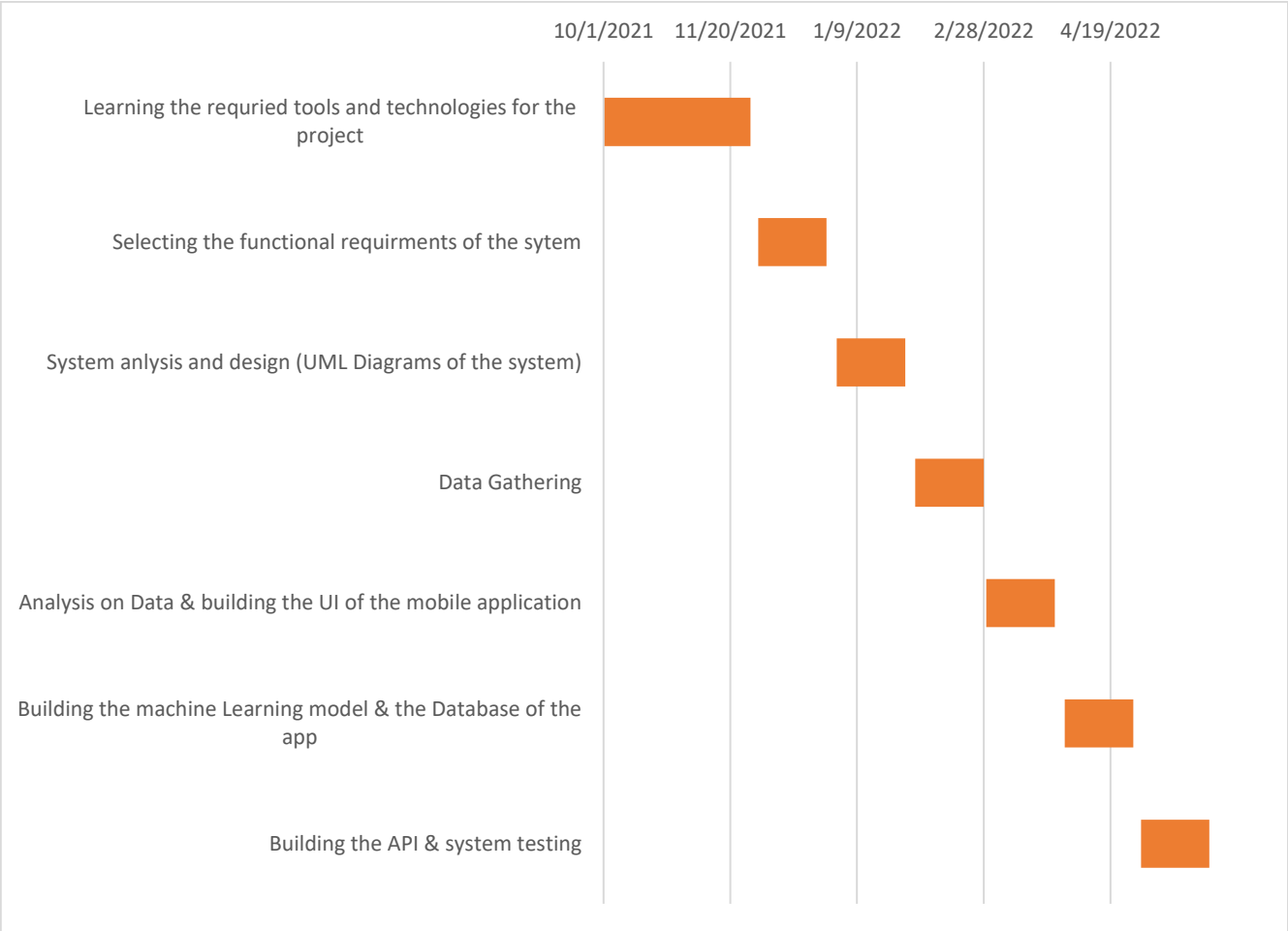


Figure 1: Project Timeline

2.Literature Review

There have been numerous studies done related to predicting the disease using different machine learning techniques and algorithms which can be used by medical institutions.

Here, we are going to review some of these studies done in research papers(that we have read)using the techniques and results used by them.

- MIN CHEN et al, [1] proposed a disease prediction system in his paper where he used machine learning algorithms. In the prediction of disease, he used techniques like CNN-UDRP algorithm, CNN-MDRP algorithm, Naive Bayes, K-Nearest Neighbor, and Decision Tree. This proposed system had an accuracy of 94.8%.
- Sayali Ambekar et al, [2] recommended Disease Risk Prediction and used a convolution neural network to perform the task. In this paper machine learning techniques like CNN-UDRP algorithm, Naive Bayes, and KNN algorithm are used. The system uses structured data to be trained and its accuracy reaches 82% and achieved by using Naïve Bayes.
- Rinkal Keniya, Aman Khakharia, Vruddhi Shah, and Vrushabh Gada, [3] developed a disease prediction system that can predict the disease according to their symptoms using data of real patients from different hospitals and using different machine learning algorithms, and they found that the weighted KNN algorithm gave the best results with accuracy 93.5%.
- Naganna Chetty et al, [4] developed a system that gives improved results for disease prediction and used a fuzzy approach. And used techniques like KNN classifier, Fuzzy c-means clustering, and Fuzzy KNN classifier. In this paper diabetes disease and liver, disorder prediction is done and the accuracy of Diabetes is 97.02% and Liver disorder is 96.13%.

3.Analysis and Requirements

3.1. System Requirements

3.1.1. Enumerated Functional Requirements

REQ-x	Description
Register	The system allows users to register in the system to create an account by entering the following details: • Email • Password • Name • Phone Number • Gender • Birthdate And the system also asks for the chronic diseases status of the user: • Do you have diabetes? • Do you have blood pressure? • Do you have heart disease?
Login	The system allows users to login to their accounts by entering the following details: • Email • Password
Enter symptoms	The system allows users to enter their symptoms
Get predicted disease	The system takes the symptoms from the user and predict the disease
Get disease information	After the system gives the predicted disease. It allows the user to know the full description and precautions related to the disease.
View patient history	The system allows the user to see all his last predicted diseases and their symptoms (user history).
Give feedback	The system allows the user to give feedback to rate the resulted disease.
See doctors nearby	The system allows the user to see the doctors nearby through GPS and Maps.
View profile	The user can see his personal information.

Edit Profile	The user can edit and update his personal information.
Log out	The user can log out from the system.

3.1.2. Enumerated Nonfunctional Requirements

REQ-x	Description
Usability	The system is very easy to use for the user.
Portability	The system is a mobile app, and anyone can install it on his phone and use it from anywhere.
Reliability	The system doesn't fail, and all function of the system should work correctly.
Availability	The system is available for 24 hours 7 days a week.
Security	The system provides a good level of privacy for all the users data.

3.1.3. Functional Requirements Specification

Stakeholders:

- Users
- Hospitals

Actors and Goals:

- Actor: Users (participating), Goal: use the system to get diagnosed.
- Actor: Admin (participating), Goal: has access to the database and maintain the system.

Use Cases:

Use Case Description:

Use Case	Description
Register	The user can register in the system to create an account by entering the following details: • Email • Password • Name • Phone Number • Gender • Birthdate • Do you have diabetes? • Do you have blood pressure? • Do you have heart disease?
Login	The user and the admin can login to their accounts by entering the following details: • Email • Password
Enter symptoms	The user can enter their symptoms to the system.
Get predicted disease	The user knows the diagnosed disease after he enter his symptoms.
Get disease info	After the system gives the predicted disease. the user can know the full description and precautions related to the disease.
View patient history	The user can see all his last predicted diseases and their symptoms (user history).
Give feedback	The user can give feedback to rate the resulted disease.
See doctors nearby	The user can see the doctors nearby through GPS and Maps.
View profile	The user can see his personal information.
Edit Profile	The user can edit and update his personal information.
Log out	The user and the admin can log out from the system.
See patient's details	The admin has access to the database and can see all the details of all users.

See patient's feedback	The admin can see all feedbacks from the users.

Use Case Diagram:

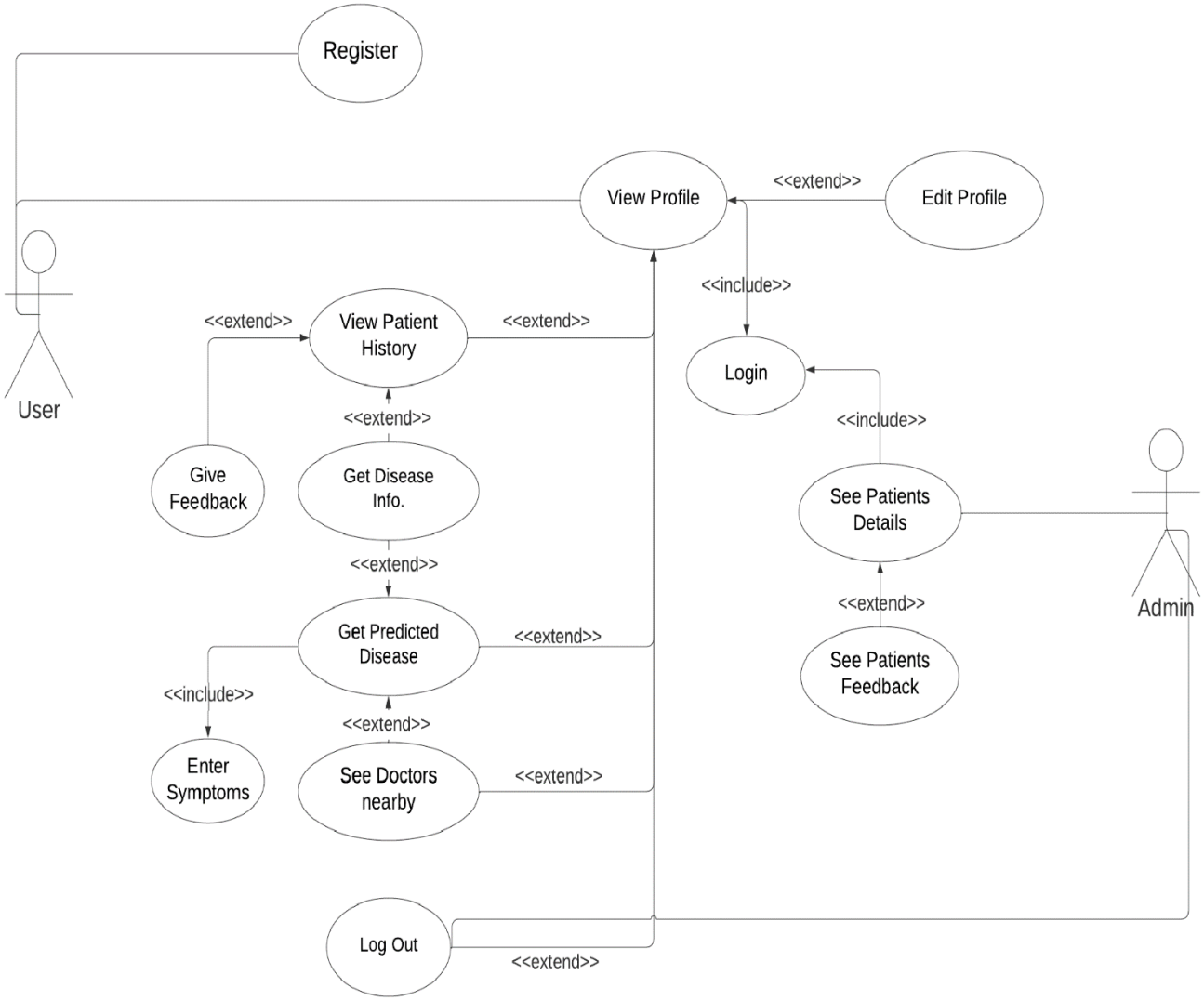


Figure 2: Use case diagram

System Sequence Diagrams:

- Register to the system and search for disease sequence diagram:

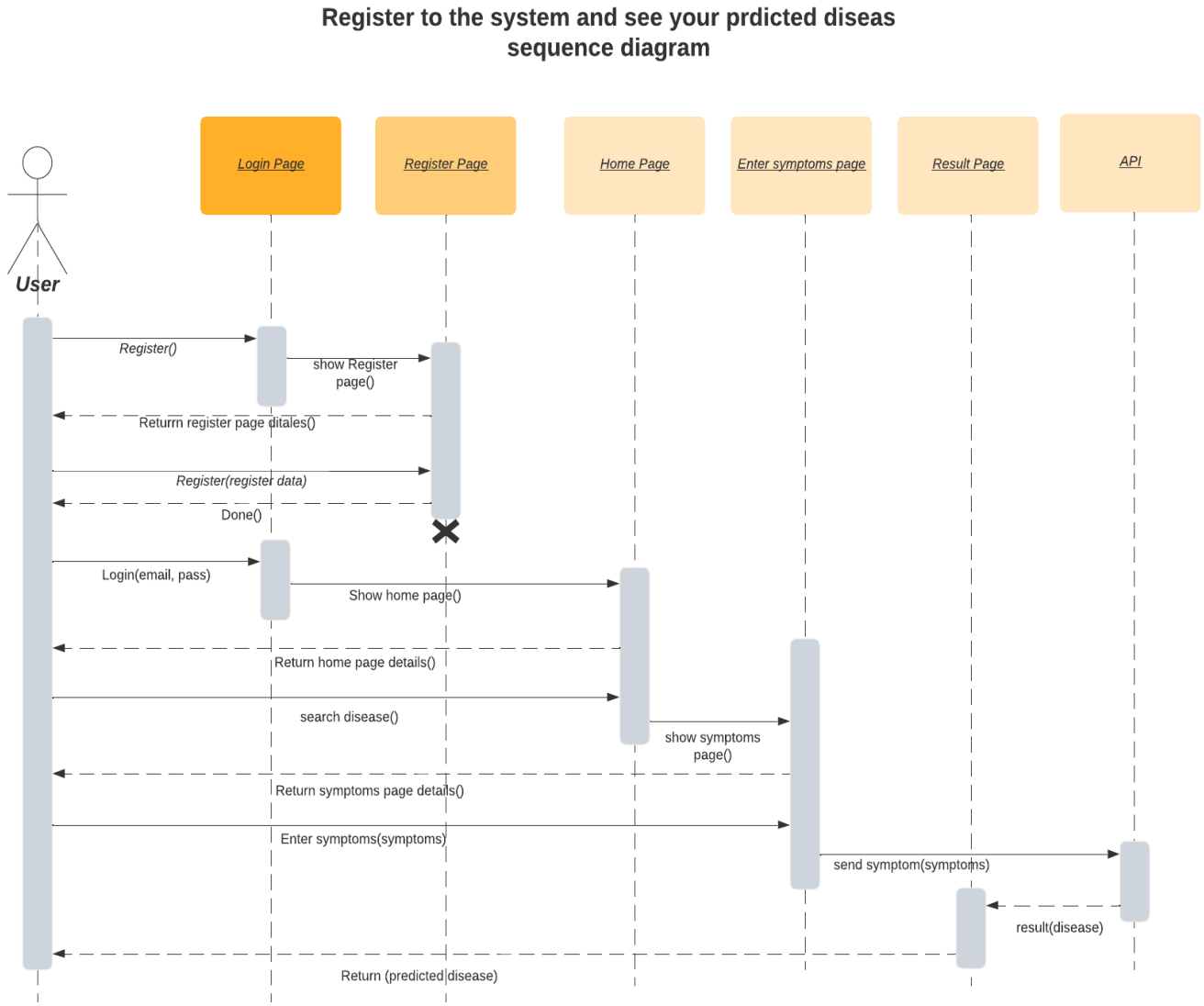


Figure 3: Register to the system and search for disease sequence diagram

- See patient history sequence diagram:

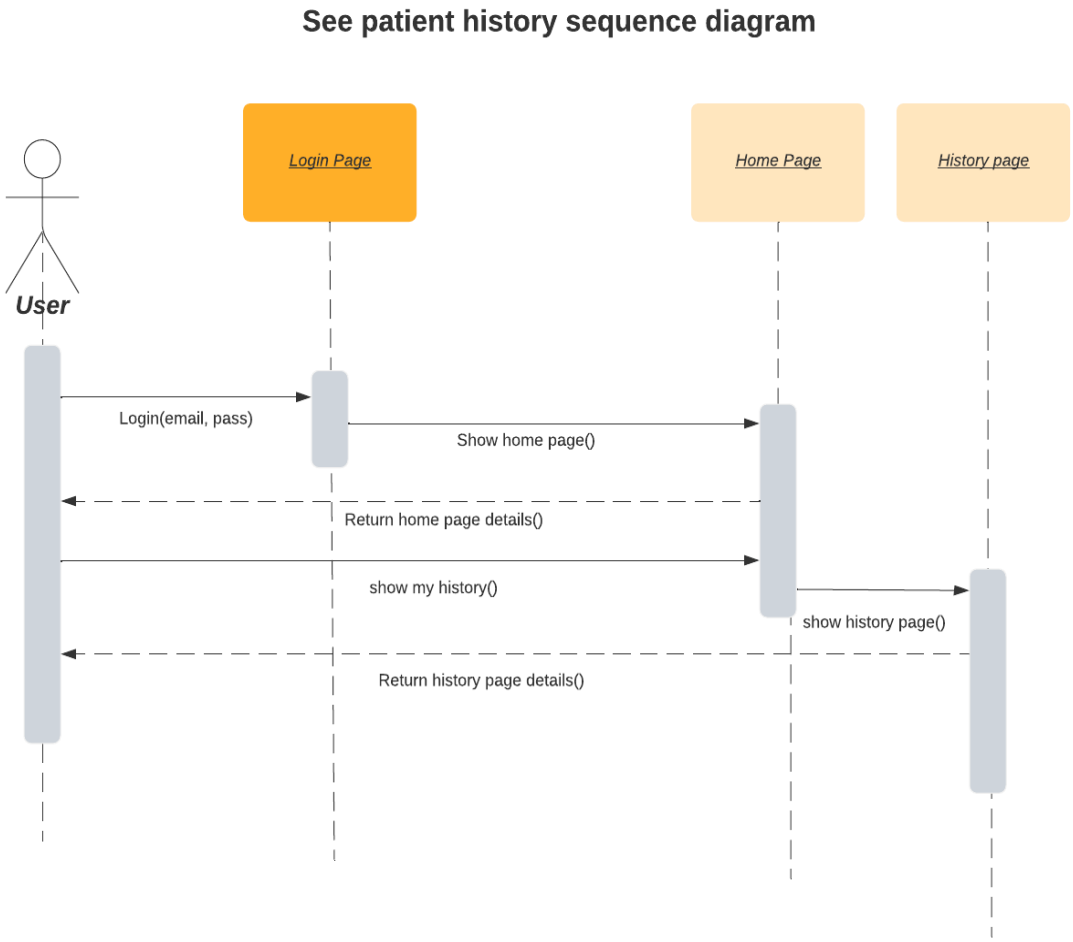


Figure 4: See patient history sequence diagram

- Give feedback sequence diagram:

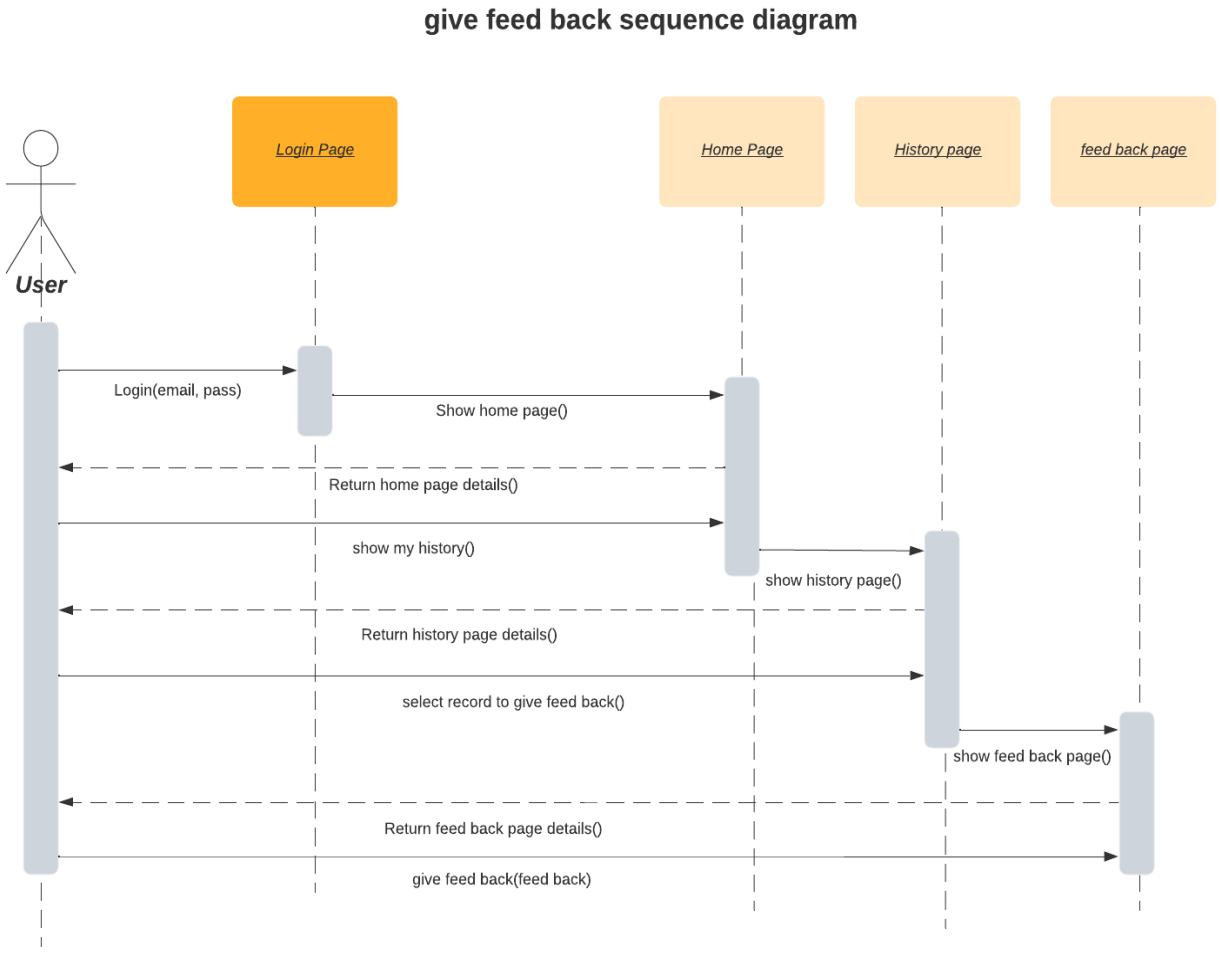


Figure 5: Give feedback sequence diagram

- See doctors nearby sequence diagram:

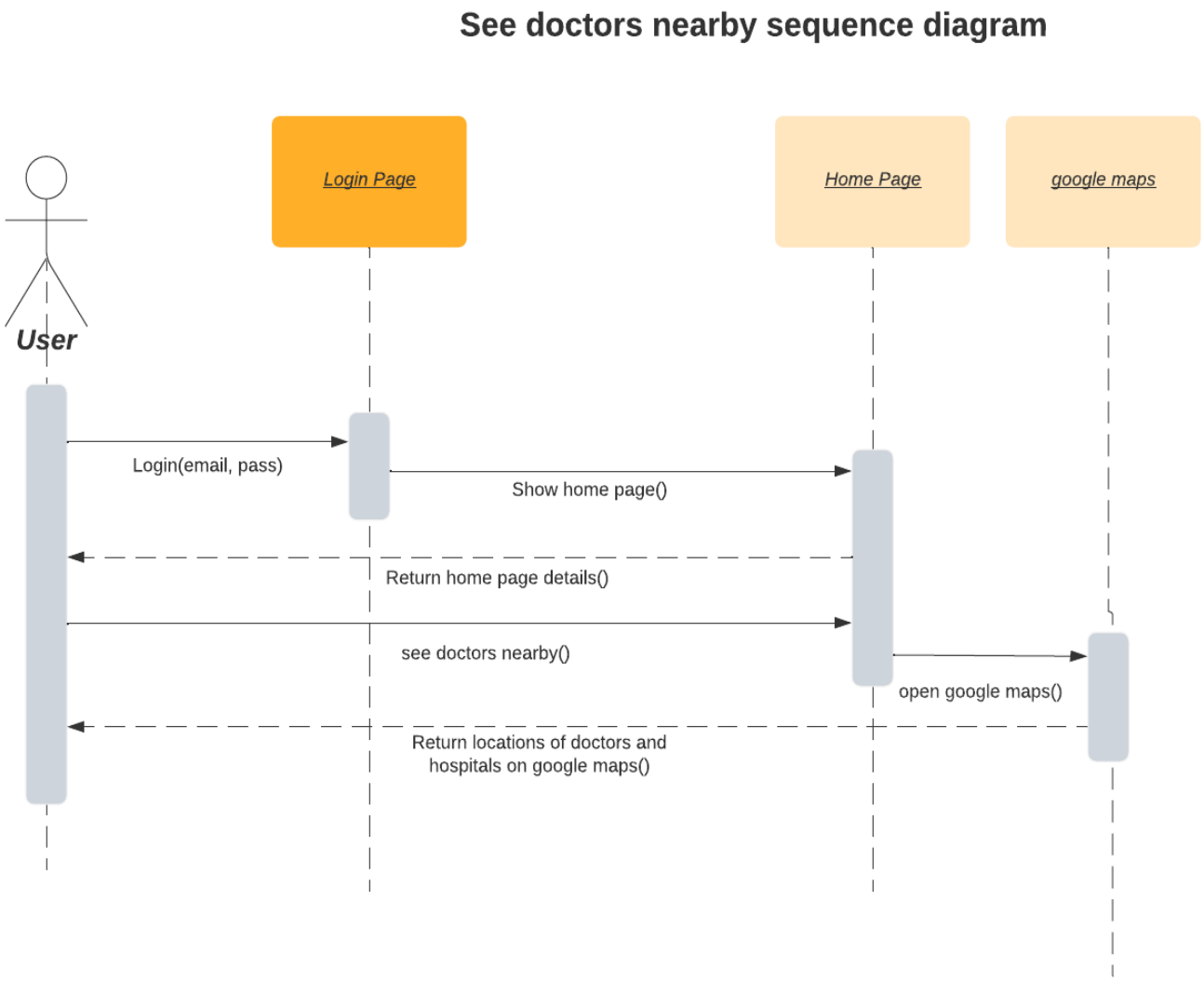


Figure 6: See doctors nearby sequence diagram

- Access patients' details by the admin sequence diagram:

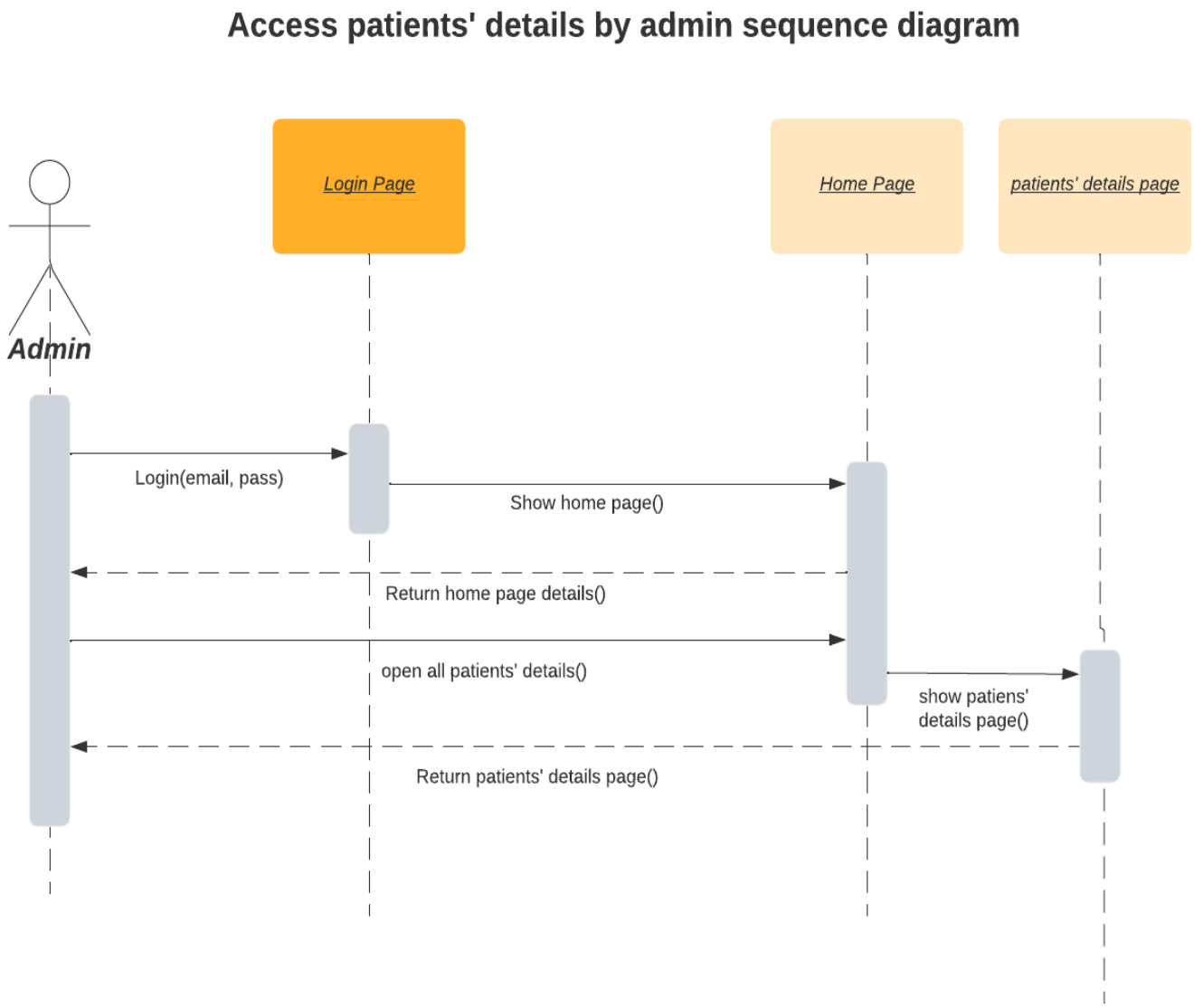


Figure 7: Access patients' details by the admin sequence diagram

Entity Relationship Diagram:

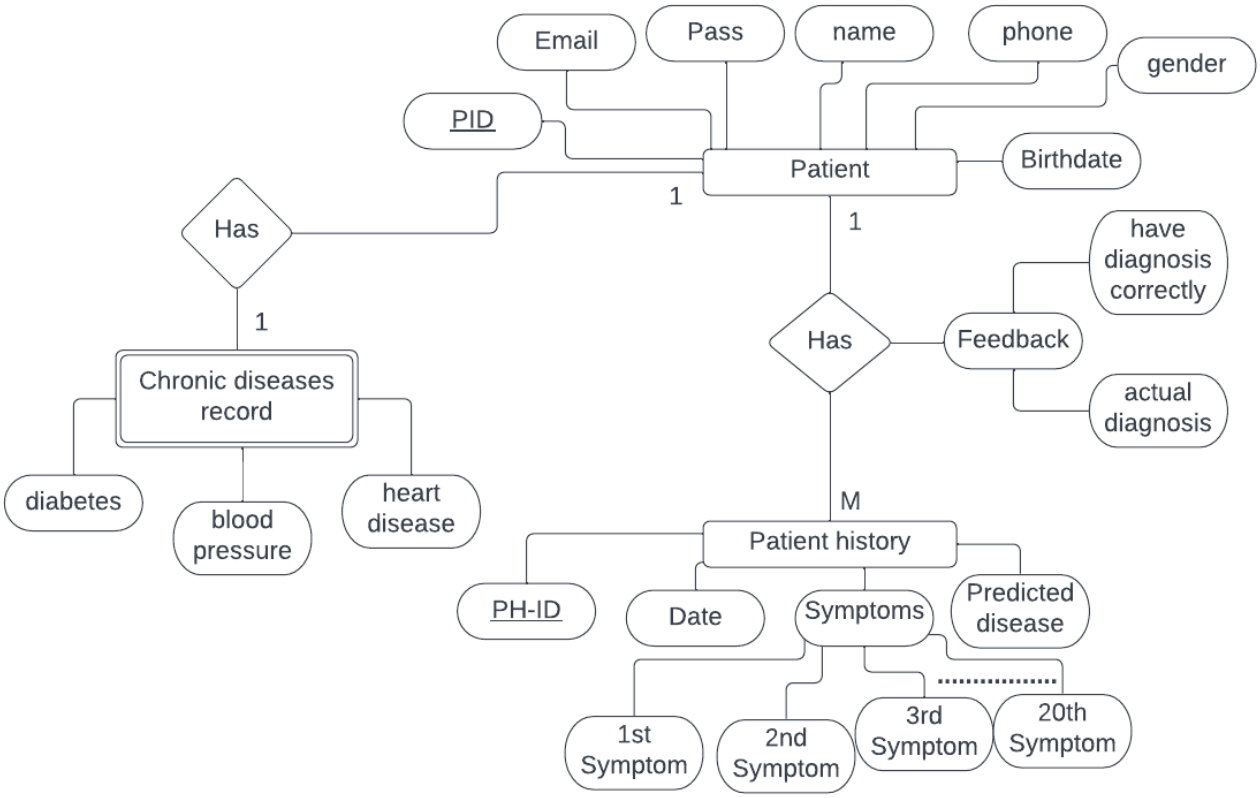
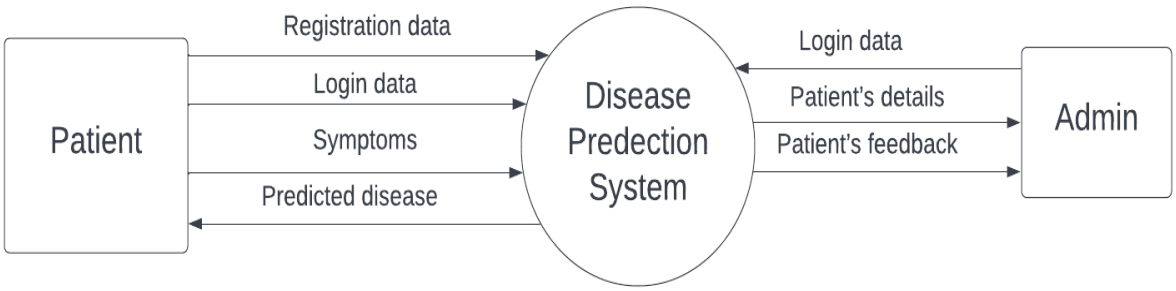


Figure 8: Entity Relationship Diagram

Data Flow Diagram:

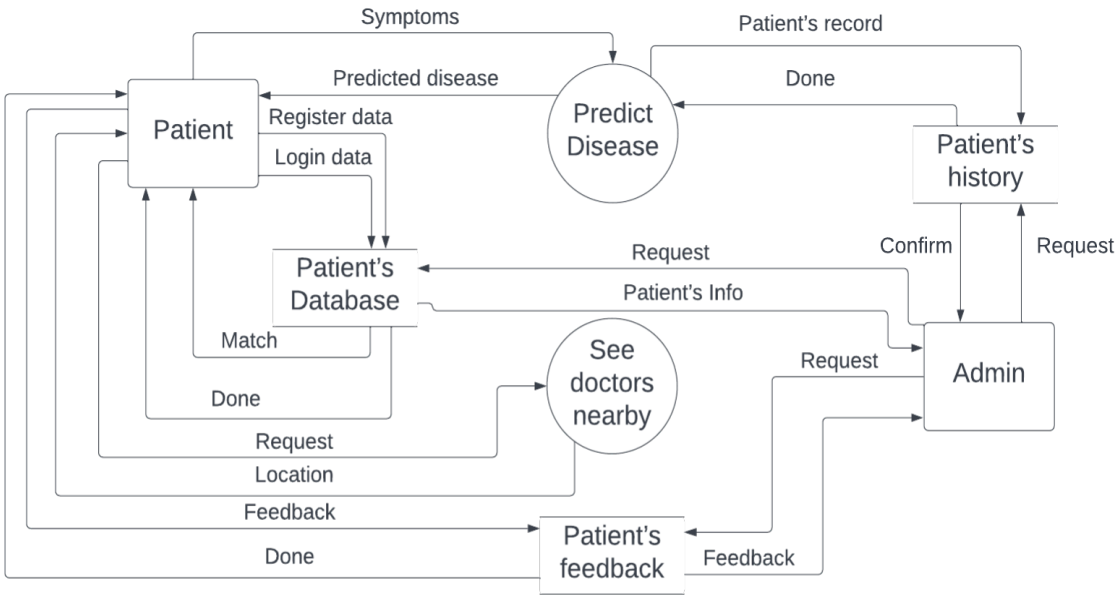
Context diagram (Level Zero):



DFD LEVEL ZERO
Context diagram

Figure 9: DFD Context diagram

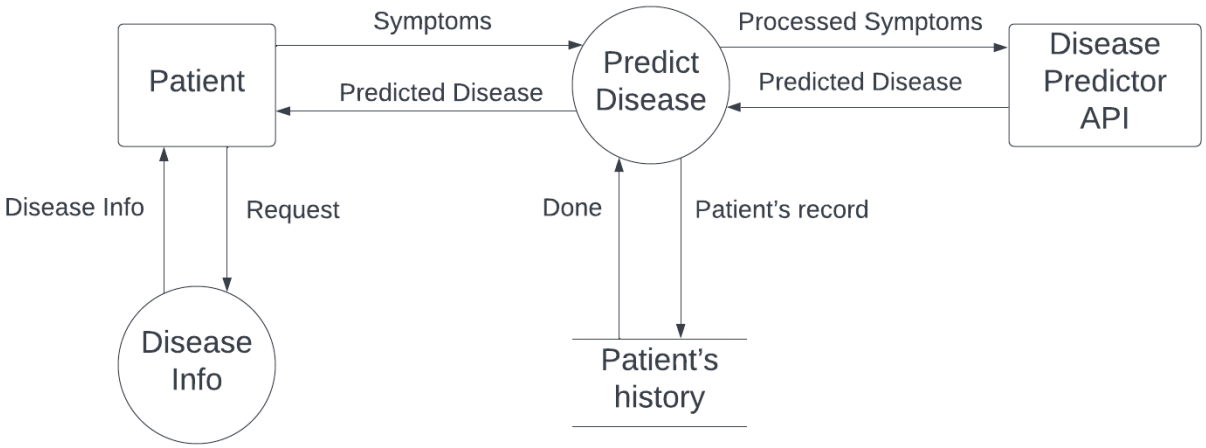
Level One Diagram:



DFD LEVEL ONE

Figure 10: DFD level one

Level Two Diagram:



DFD LEVEL TWO

Figure 11: DFD level two

Class Diagram:

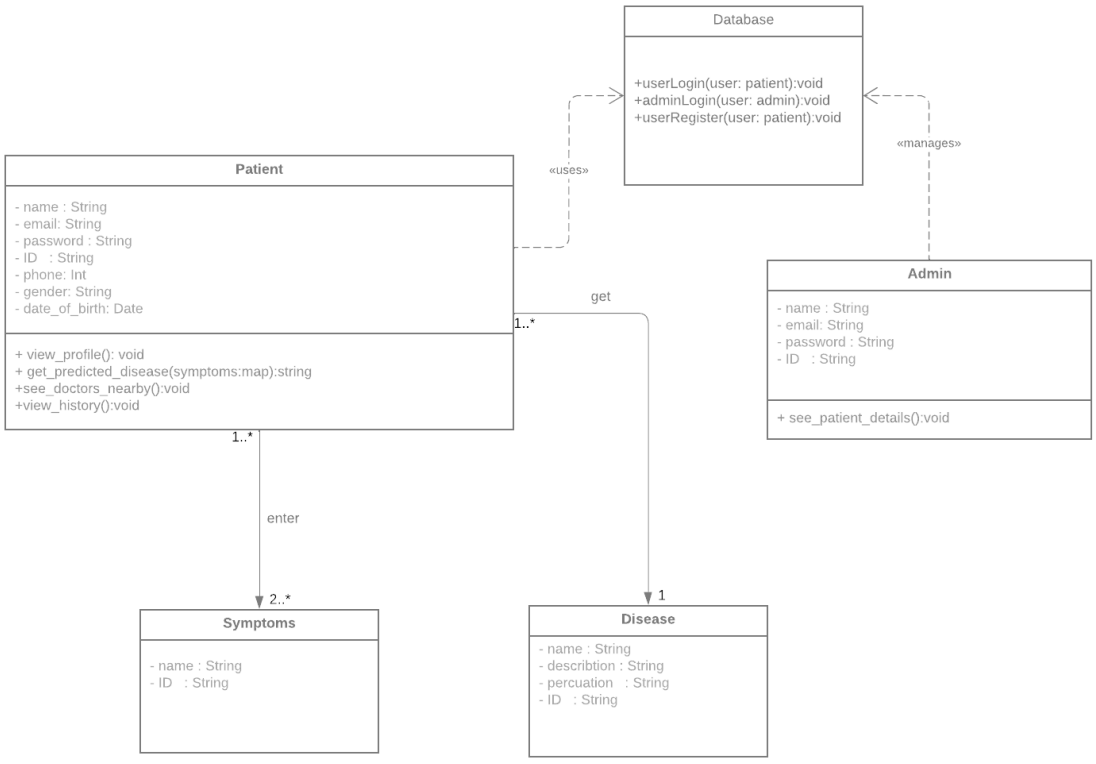


Figure 12: Class Diagram

System Architecture:

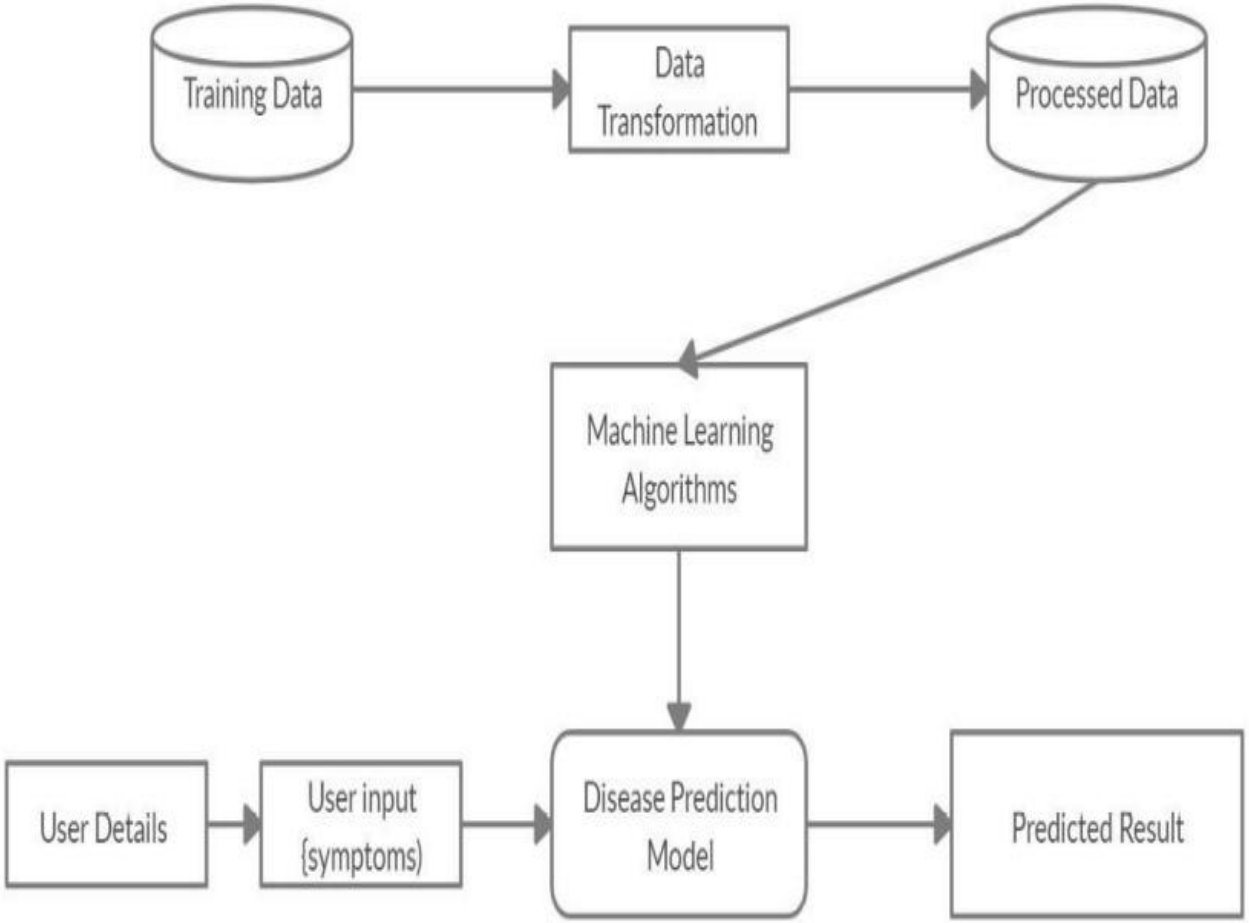


Figure 13: System block diagram

4.Tools and technologies

In this section we will show all the tools and technologies that will be used in our project.

Technologies and programming languages:

- Machine Learning: We will use several ML models like KNN and Random Forest algorithms.
- Flutter: flutter will help us to build our cross platform mobile application.
- Pandas: is a tool for data analysis, which is a Python library.
- Flask framework: will help us to build our API that will connect to the mobile app.
- Python: the programming language that will be used for data analysis, building our machine learning model, and building our API.
- Dart: the programming language that will be used for building our mobile application.
- SQL: used for building the database and making queries on it.

Tools:

- Android Studio: the software that will be used for building the mobile app.
- Anaconda: the software that will be used for making data science and machine learning works.
- MySql: the software that will be used for building our database.
- Heroku dev center: is a free server that we will use for uploading our API on it, so any application connected with the internet can connect with our API through this server.

5.Implementations

In our project the implementations will be divided into two parts:

- Data Science: in this part, we will apply the data science workflow which is the data gathering and making analysis and preprocessing on it to extract insights from it and modeling of machine learning models and at the end the deployment process of our project.
- Mobile Application: in this part, we will develop the GUI of our project that the user will interact with, and also this app will take all the data of the user and store it into the database.

5.1. Data Science

5.1.1. Introduction

Data science is the field of study that uses mathematics, programming, and domain knowledge to extract meaningful insights from data. Data scientists would apply machine learning algorithms to data such as numbers, texts, images, videos, and more to produce artificial systems that perform tasks that require human intelligence. Using these systems, we extract insights from the data and use them to make better decisions in various situations.

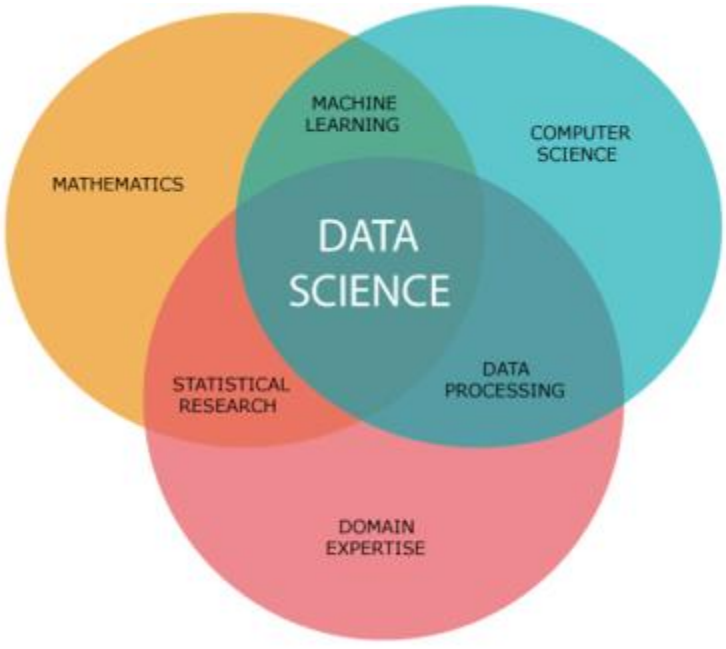


Figure 14: Data science

We live in a world that is full of data. From traffic cameras to big corporations produces tons of data every day every minute. Just one click on the Facebook post may save lots of information about that click.

Have you ever wondered how Amazon, eBay suggests items for you to buy or How Gmail filters your emails in the spam and non-spam categories? Data science is all about using data to solve these kinds of problems. The problem could be decision making, such as identifying which email is spam and which is not. Or a product recommendation such as which movie to watch? Or predicting the outcome such as who will be the next President of the USA?

This is how you get YouTube video recommendations as well. You watch a video, YouTube will automatically put a record in their database that you watched this video. Next time when you visit YouTube you will get the same types of videos you watched earlier. Which makes you happy since you get your favorites on the front page. So this decision YouTube made to show that particular type of video you like helps them to make user experience or lure you into YouTube more.

Our world now is full of data and the next plot shows how the data grow through the years. (hint: zettabyte = 10^21 bytes).

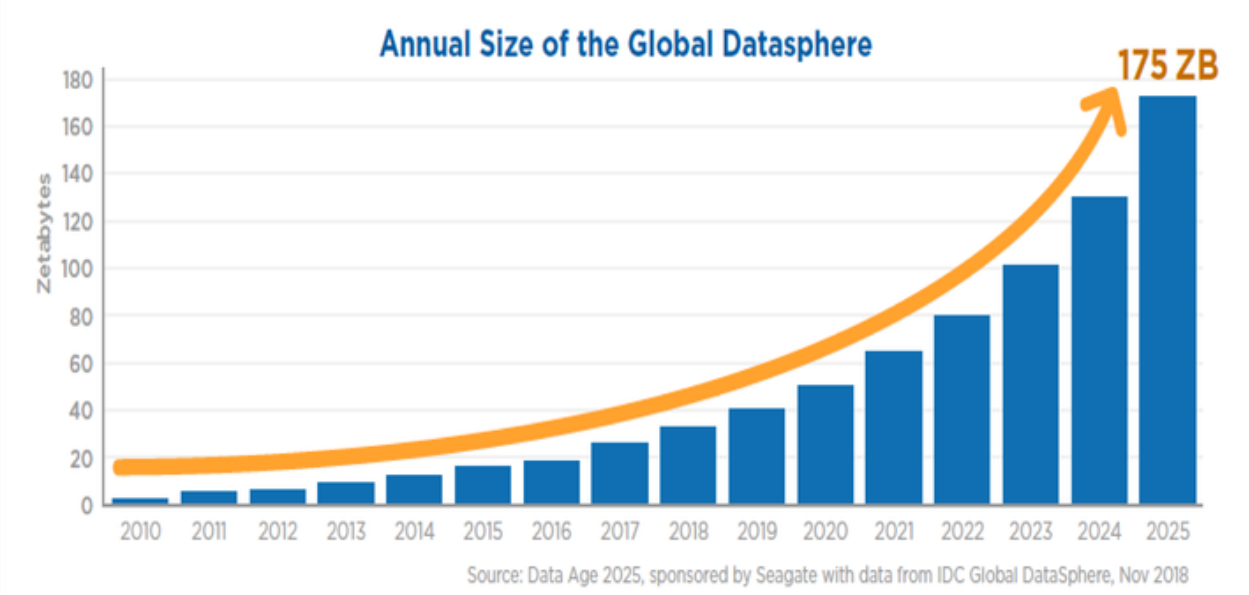


Figure 15: Annual size of global datasphere

5.1.2. Data Science Workflow

A data science workflow defines the phases (or steps) in an end-to-end data science project, and these phases are shown in the next figure.

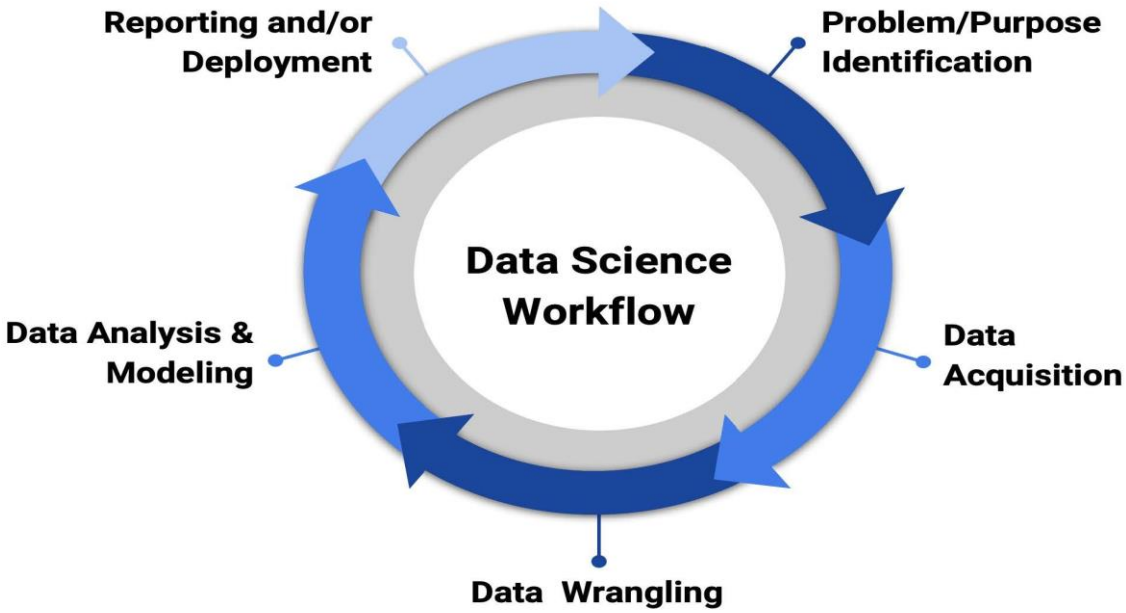


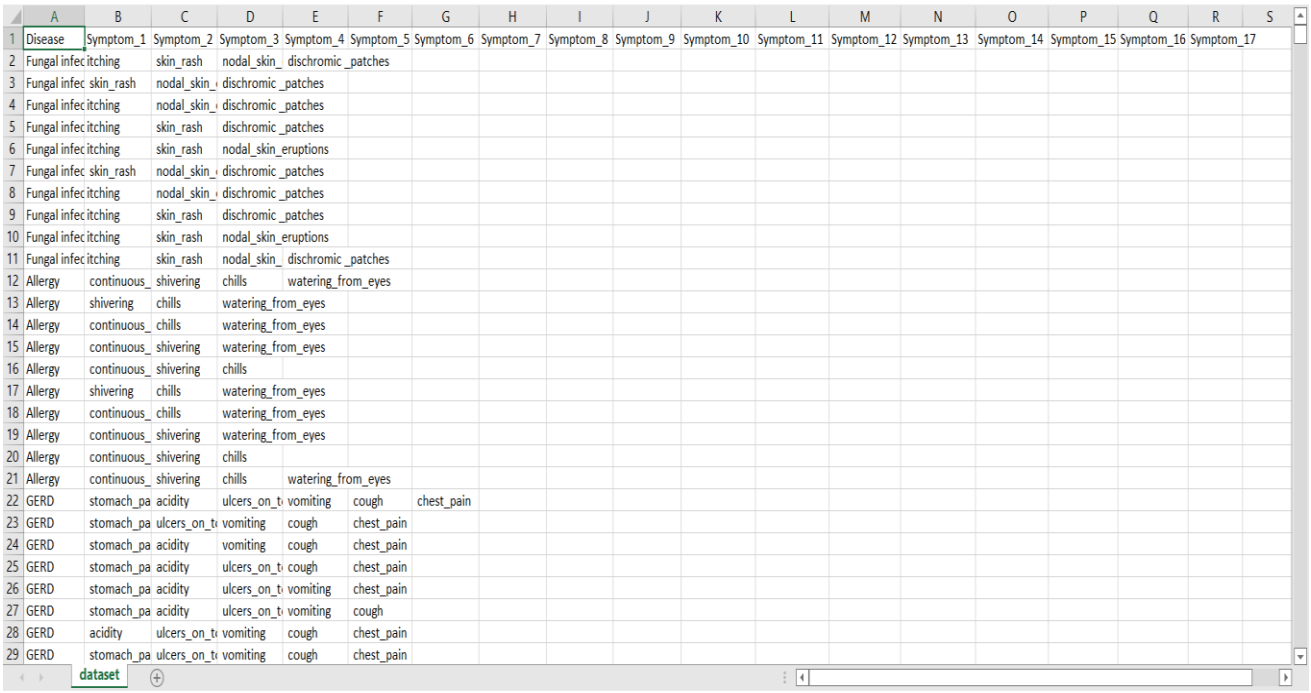
Figure 16: Data science workflow

- Problem/Purpose Identification: The first step in any data science or analytics project is identifying the problem you are trying to solve or the purpose of your data product or analysis. This step serves as a guide for the rest of your project and informs how all the other steps are going to be performed.
- Data Acquisition: Once you have a problem to solve or a purpose to fulfill, you need to acquire the data that is going to fuel the project, and this data must be real and trusted data.
- Data Wrangling: Data can come from a variety of sources, but the most predictable thing about it is that it will not come in the format you need it to be in. So once you've acquired the data, you need to wrangle it (clean it, manipulate it, transform it, etc.) into a format and structure that you are going to be able to work with.

- Data Analysis & Modeling: When your data is neatly organized, it can then be analyzed and modeled. From this step, we can obtain insights, discover the answers, and learn the patterns that will help us solve our problem or fulfill our purpose.
- Reporting and/or Deployment: The final step is to produce your reports or deploy your application. In an analytics project, the deliverable is usually a report communicating what you've learned from the analysis and how it can help inform decisions that need to be made. In a data science project, the deliverable is usually the deployment of an application so that it can start officially doing what it has been designed to do.

5.2. Data gathering

In our project, we use a real dataset from kaggel website. This dataset consist of one column for disease and 17 columns for symptoms, and for every record not all symptom columns contain data may all 17 columns contain data and may less than 17 columns, and to make it clear here's the data shape as a csv file in the following figure.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Disease	Symptom_1	Symptom_2	Symptom_3	Symptom_4	Symptom_5	Symptom_6	Symptom_7	Symptom_8	Symptom_9	Symptom_10	Symptom_11	Symptom_12	Symptom_13	Symptom_14	Symptom_15	Symptom_16	Symptom_17	
2	Fungal infec	itching	skin_rash	nodal_skin_	dischromic _patches														
3	Fungal infec	skin_rash	nodal_skin_	dischromic _patches															
4	Fungal infec	itching	nodal_skin_	dischromic _patches															
5	Fungal infec	itching	skin_rash	dischromic _patches															
6	Fungal infec	itching	skin_rash	nodal_skin_eruptions															
7	Fungal infec	skin_rash	nodal_skin_	dischromic _patches															
8	Fungal infec	itching	nodal_skin_	dischromic _patches															
9	Fungal infec	itching	skin_rash	dischromic _patches															
10	Fungal infec	itching	skin_rash	nodal_skin_eruptions															
11	Fungal infec	itching	skin_rash	nodal_skin_	dischromic _patches														
12	Allergy	continuous_	shivering	chills	watering_from_eyes														
13	Allergy	shivering	chills	watering_from_eyes															
14	Allergy	continuous_	chills	watering_from_eyes															
15	Allergy	continuous_	shivering	watering_from_eyes															
16	Allergy	continuous_	shivering	chills															
17	Allergy	shivering	chills	watering_from_eyes															
18	Allergy	continuous_	chills	watering_from_eyes															
19	Allergy	continuous_	shivering	watering_from_eyes															
20	Allergy	continuous_	shivering	chills															
21	Allergy	continuous_	shivering	chills	watering_from_eyes														
22	GERD	stomach_pa	acidity	ulcers_on_t	vomiting	cough	chest_pain												
23	GERD	stomach_pa	ulcers_on_t	vomiting	cough	chest_pain													
24	GERD	stomach_pa	acidity	vomiting	cough	chest_pain													
25	GERD	stomach_pa	acidity	ulcers_on_t	cough	chest_pain													
26	GERD	stomach_pa	acidity	ulcers_on_t	vomiting	chest_pain													
27	GERD	stomach_pa	acidity	ulcers_on_t	vomiting	cough													
28	GERD	acidity	ulcers_on_t	vomiting	cough	chest_pain													
29	GERD	stomach_pa	ulcers_on_t	vomiting	cough	chest_pain													

dataset

Figure 17: Snapshot of the dataset from excel

5.3. Data preprocessing and wrangling

First, we made some exploration to the data and we found the following:

- The data has 41 unique diseases.
- The data has 131 unique symptoms.
- All records of the data are categorical data.

As we can see, the data is not organized in the proper way that makes the data analysis and modeling steps are not easy to us, so we have to do some processing on the data before these steps.

because our data is categorical data so, the best way to convert the data to an organized and suitable shape is applying onehot encode to the data.

First, we are going to know what is onehot encode?

Onehot encode is a way to convert nominal categorical data into numerical data as shown in the figure below. So now, we can make analysis on the data and modeling, because as we know most of machine learning algorithms don't work with categorical data but numerical.

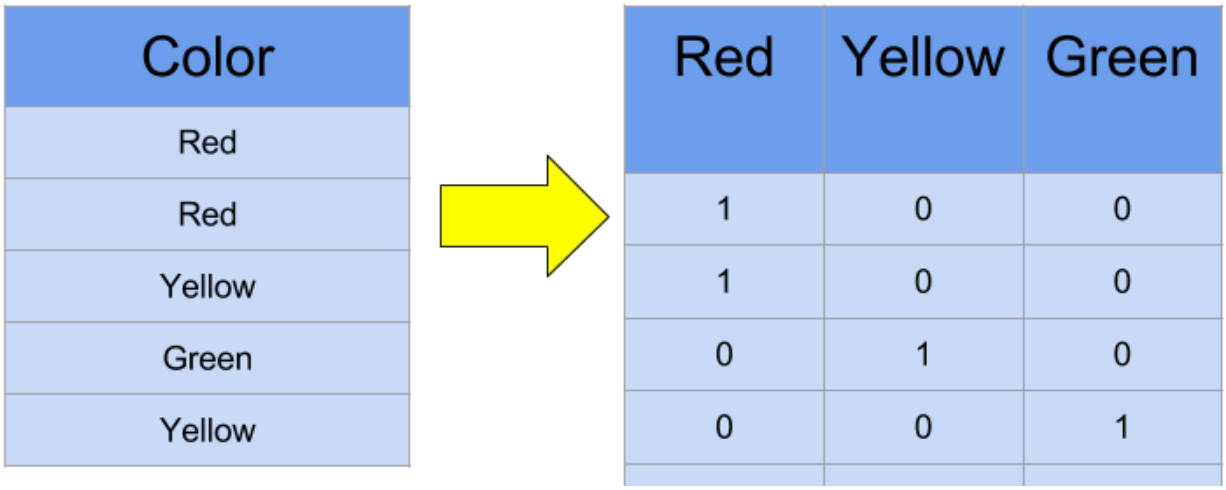


Figure 18: Example on onehot encode

The data before onehot encode:

```
In [39]: df = pd.read_csv(DATA_PATH)
         df.head()
```

Out[39]:

	Disease	Symptom_1	Symptom_2	Symptom_3	Symptom_4	Symptom_5	Symptom_6	Symptom_7	Symptom_8	Symptom_9	Symptom_10	S
0	Fungal infection	itching	skin_rash	nodal_skin_eruptions	dischromic _patches	NaN	NaN	NaN	NaN	NaN	NaN	
1	Fungal infection	skin_rash	nodal_skin_eruptions	dischromic _patches	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	Fungal infection	itching	nodal_skin_eruptions	dischromic _patches	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	Fungal infection	itching	skin_rash	dischromic _patches	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	Fungal infection	itching	skin_rash	nodal_skin_eruptions	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

Figure 19: Snapshot to the dataset from jupyter notebook before onehot encode

The data after onehot encode:

```
In [12]: # Merge onehot encoded
         new_df = pd.concat(encoded_dfs, axis=1)
         new_df.head()
```

Out[12]:

	Disease										...	Symptoms			
	(vertigo) Paroymsal Positional Vertigo	AIDS	Acne	Alcoholic hepatitis	Allergy	Arthritis	Bronchial Asthma	Cervical spondylosis	Chicken pox	Chronic cholestasis	...	vomiting	watering_from_eyes	weakness_in_limbs	we
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 172 columns

Figure 20: Snapshot to the dataset from jupyter notebook after onehot encode

5.4. Data Analysis

Data analysis is a very important part in any data science project, we extract useful and important insights and information from the data that exactly would be helpful to understand the data and the domain knowledge of the data, and from these insights we can know if the data is real data or not, and we can measure how will be the performance of our machine learning model.

Why we do data analysis?

- Understanding the data and the domain knowledge of the data by extracting insights and information from it.
- Check the data validity (check if the data is real or fake data).
- Visualize and understand the relations between the data features.
- Measure the performance of the machine learning model that will learn and train on this data, So, if the model gives you an unexpected results you can know and understand why the model gives you wrong results from the analysis and then you can go back and fix that error.

How we did data analysis in our project?

The best practice for doing good data analysis is to ask yourself questions that you need to know the answer of them from the data. And that is what we have exactly done in our project.

And also the best practice is to use visualizations and plots to understand the answers of the questions well “A wise man once said a picture is better than 1000 words”.

The first question was, how is each symptom in data frequent (symptoms frequency)? We asked this question because we need to see if the data is real data and also see the most frequent symptoms in the data to extract some insights from the data about the symptoms.

To answer this question, we used tree map visualization as shown in the following figure.

The insights we extracted from this question:

- The most frequent symptoms in the data that some of the diseases my share these symptoms are the general symptoms (like fatigue, vomiting, high fever….) which is make sense.
- Some of the symptoms are very low in frequency (like foul smell of urine), is that because these symptoms are specified for specific diseases? And from this insight we ask another question to answer the previous question which is what are the relations between the diseases and the symptoms? And this question will be the second question.

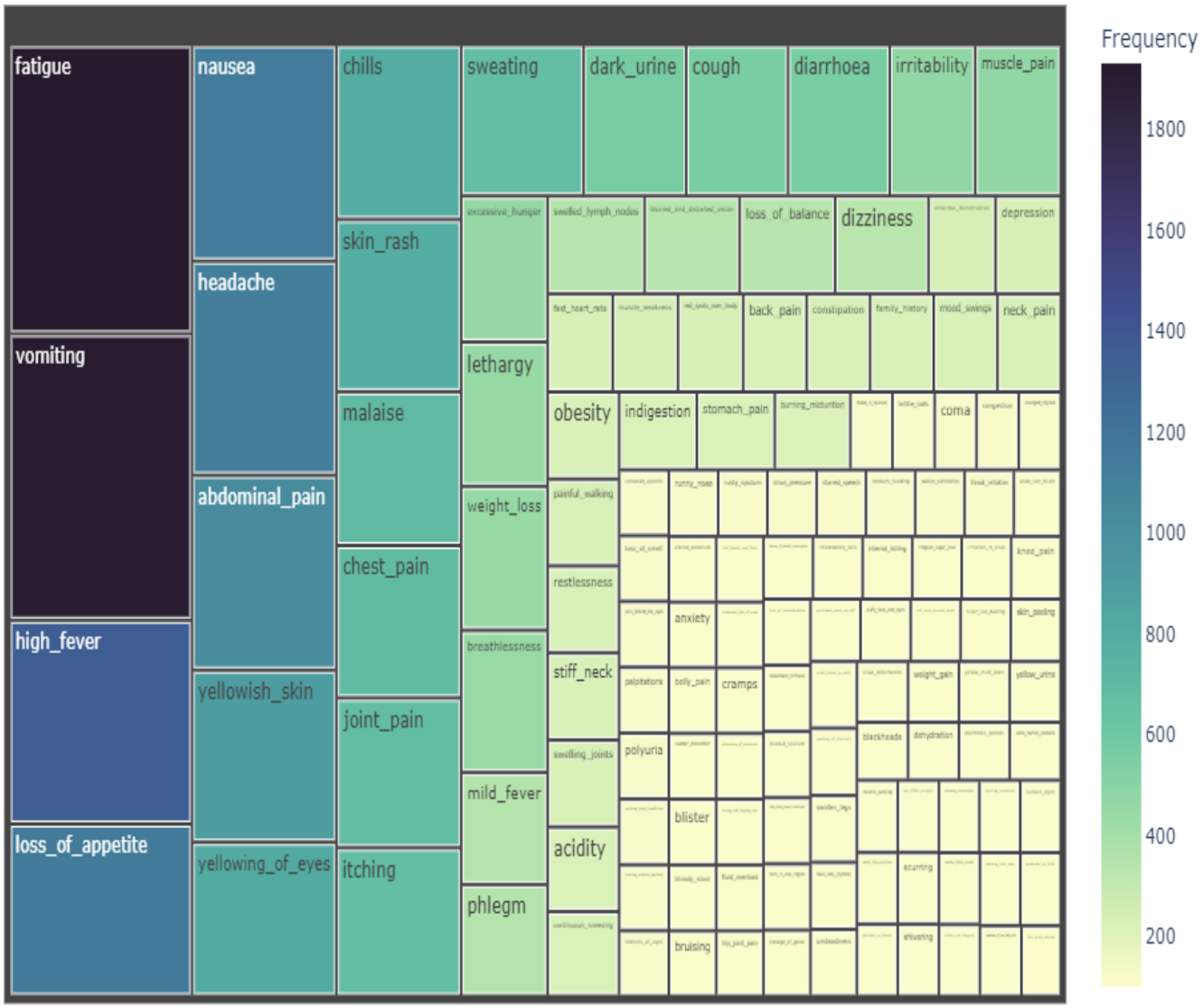


Figure 21: Symptoms frequency

The second question was the question that we have extracted from the insights of the first question, which is what is the relations between diseases and symptoms?

To answer this question, we use heatmap visualization as shown in the following figure.

The insights we extracted from this question:

- The answer of the above question which is (is the low frequency symptoms are specified for specific diseases? The answer is yes, as it clear in the heatmap, for example, we found that (foul smell of urine symptom) is a specified symptom for (urinary tract infection disease).

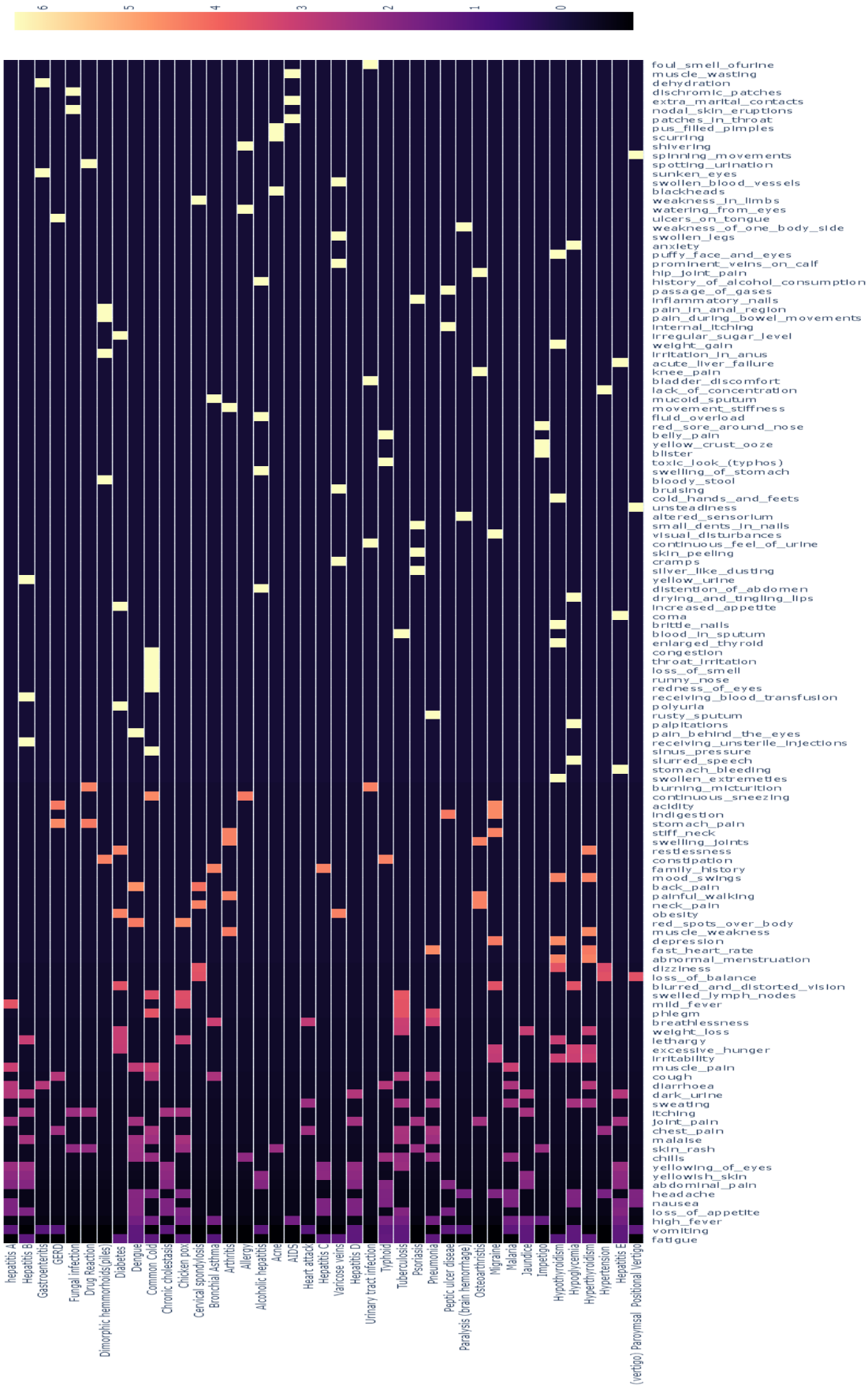


Figure 22: Relations between diseases and symptoms

For more clarity here's the Relation between sample disease and the symptoms (shown in the next figure):

In the following figure, the degree of color indicates the strength of the relationship between disease and symptom. The higher the degree of color, the stronger the relationship, and the tall of each bar indicates the frequency of each symptom in this disease.

Here, we see that symptoms like (chest pain and high fever) have higher frequencies than symptoms like (congestion, runny nose, and loss of smell), but also have less relationship with the disease than them.

From this insight we found, that it's clear that the symptoms like (chest pain and muscle pain) are general symptoms and there are more than one disease share these symptoms, and on the other hand the symptoms like (congestion, runny nose, and loss of smell) are specified symptoms for a specific disease (common cold), and we can know the disease easily from its specific symptoms that have a strong relationship with the disease.

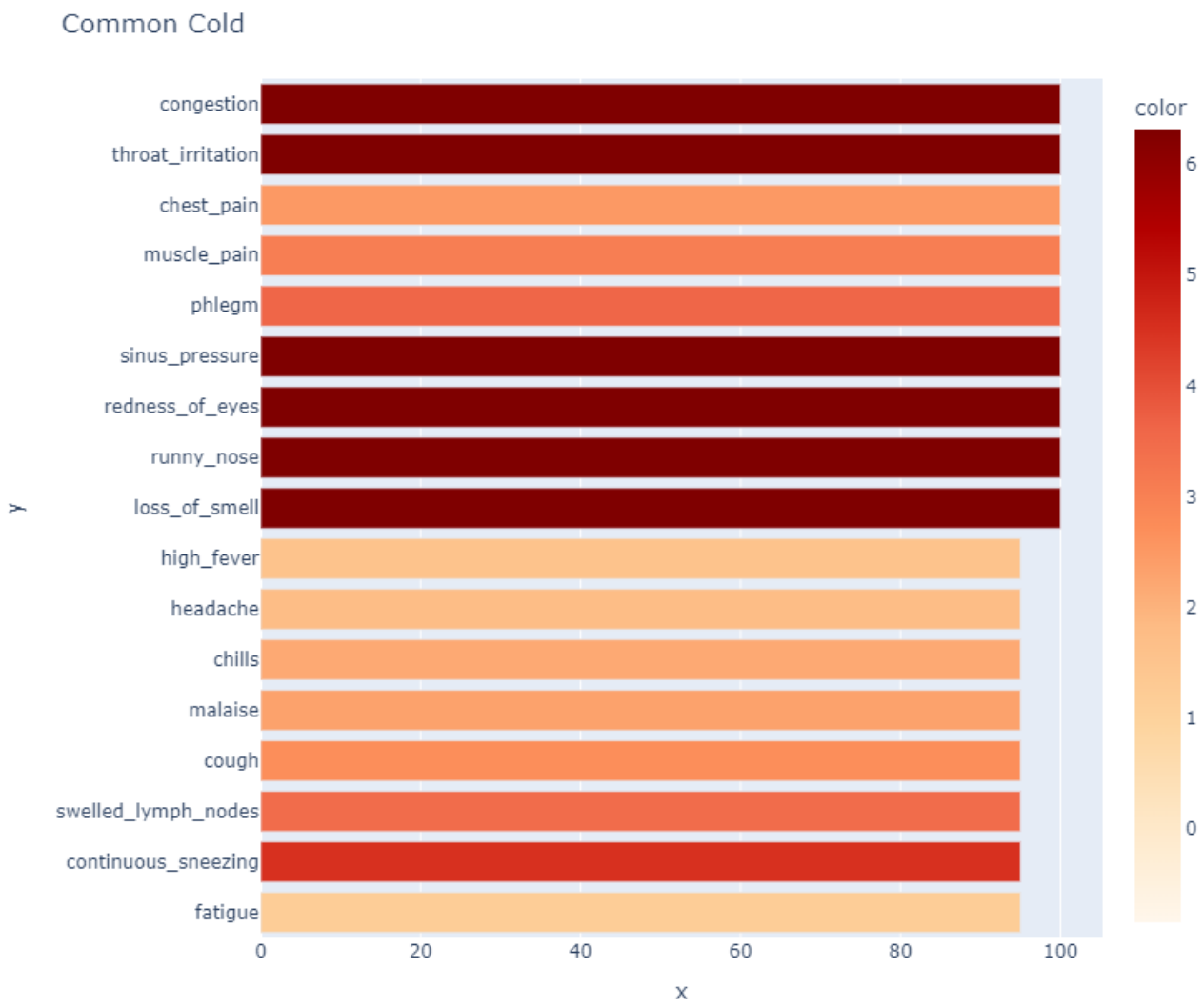


Figure 23: The relations between common cold disease and the symptoms

The third question was, what is the relations between the diseases?

To answer this question, we use dendrogram to visualize the answer as shown in the next figure.

The insights we extracted from this question:

- It seems that the relationships between the diseases is not much strong and we can differentiate between them according to the symptoms easily.
- From this insight, we can say that the performance of the machine learning model will be very good in predicting the accurate disease.

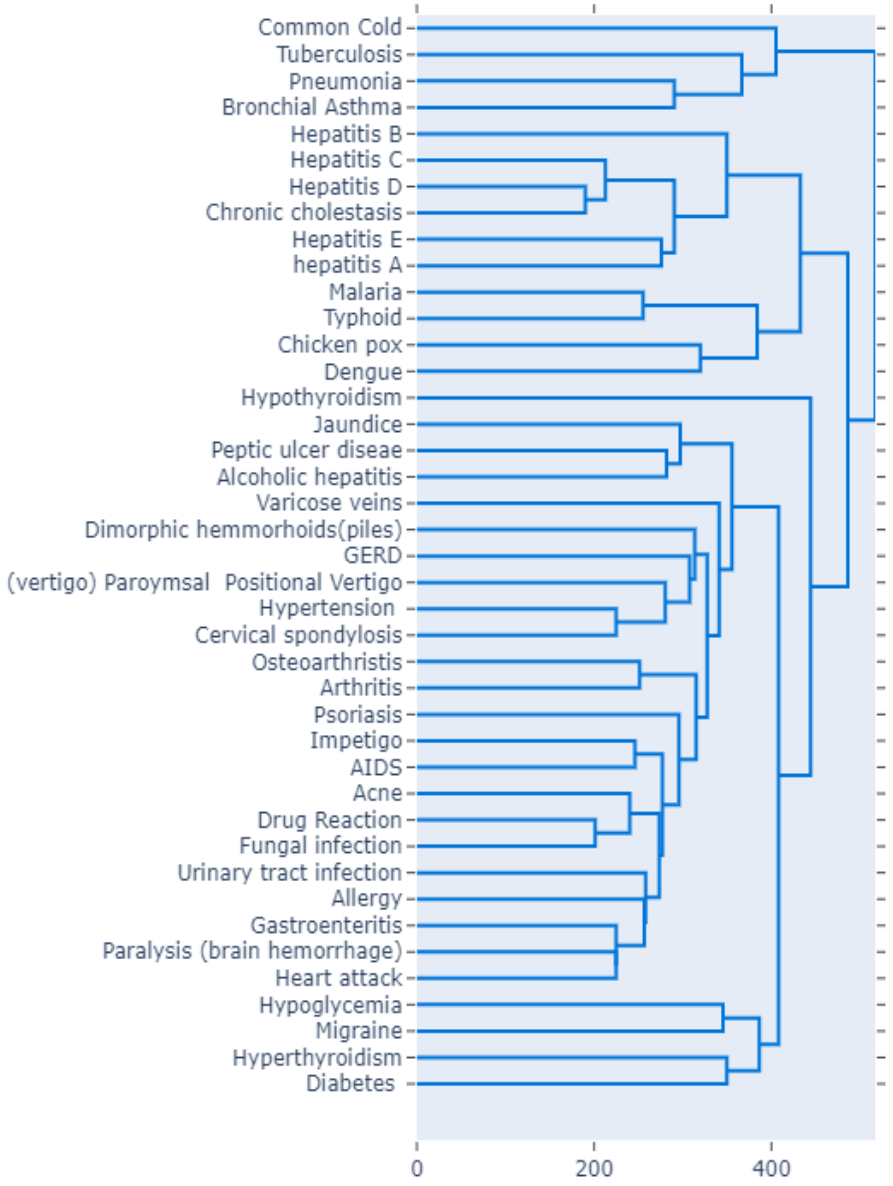


Figure 24: The relations between the diseases

The fourth question was, what is the relations between the symptoms?

To answer this question we will, use scatter plot as shown in the following figure:

Figure 25: Scatter plotting for the symptoms

To see the relations more better, we will use unsupervised learning to cluster these data and see the relations clearly as shown in the following figure.

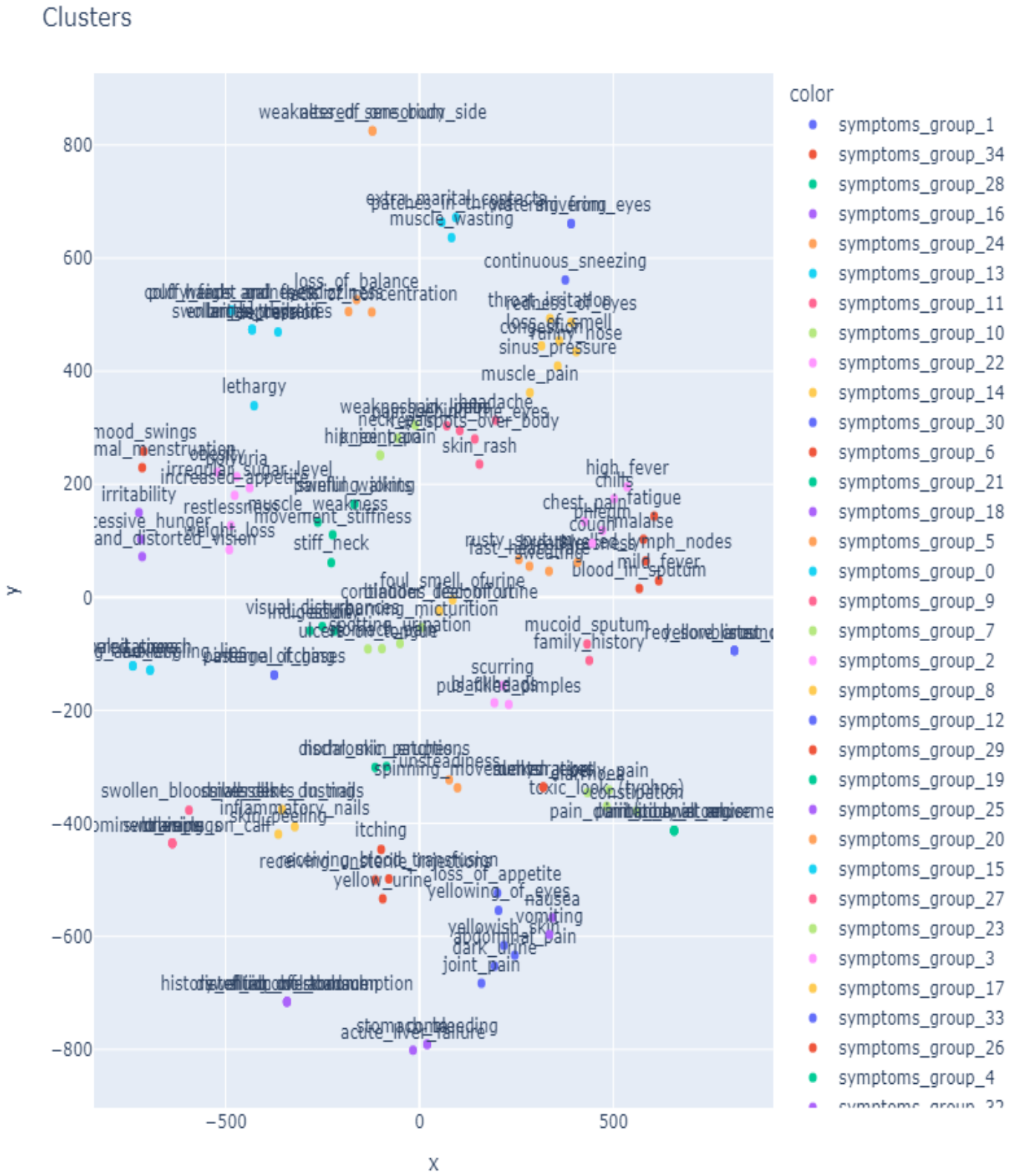


Figure 26: the symptoms clusters

5.5. Building Machine Learning Model

5.5.1. Introduction

First, we will take a look about what's machine learning and its algorithms, and then we will see how we applied machine learning in our project.

What is machine learning?

Machine learning involves computers discovering how they can perform tasks without being explicitly programmed to do so. It involves computers learning from data provided so that they carry out certain tasks. The following figure shows the difference between general programming and machine learning.

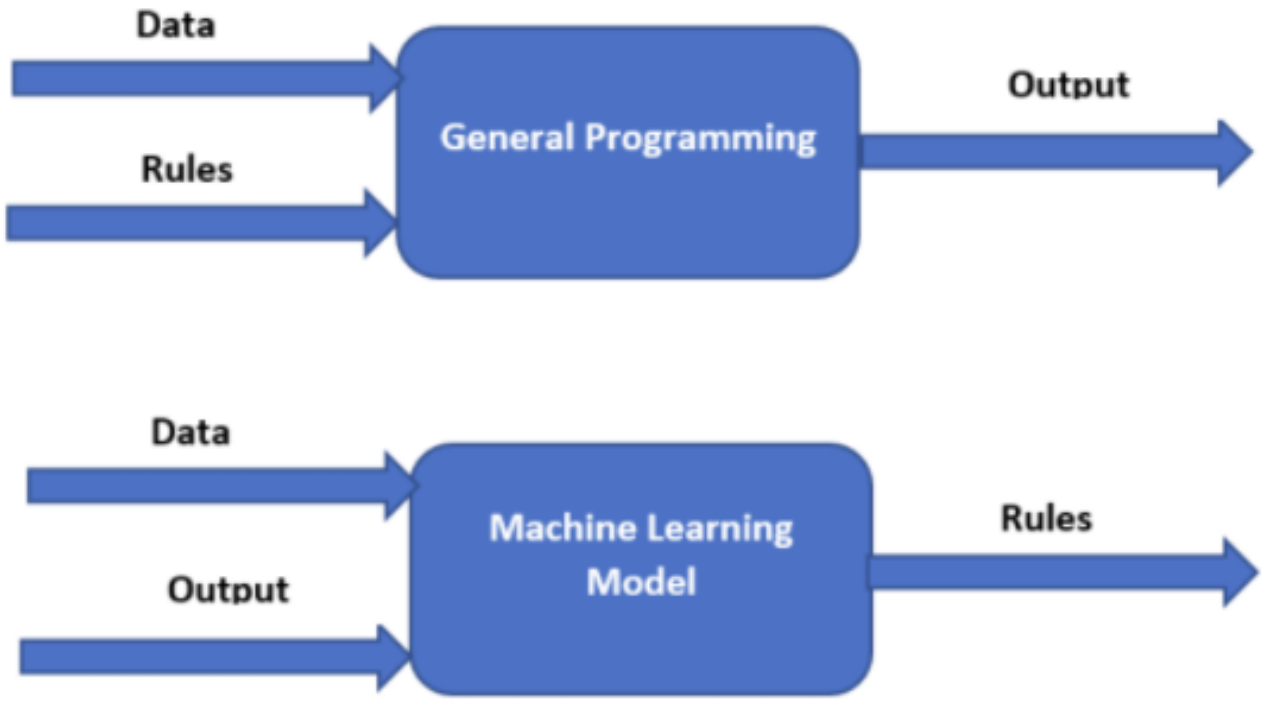


Figure 27: Difference between general programming and machine learning

The ML algorithms are broadly classified into four types: supervised, semi-supervised, unsupervised, and reinforcement Machine Learning Algorithms as shown in the following figure.

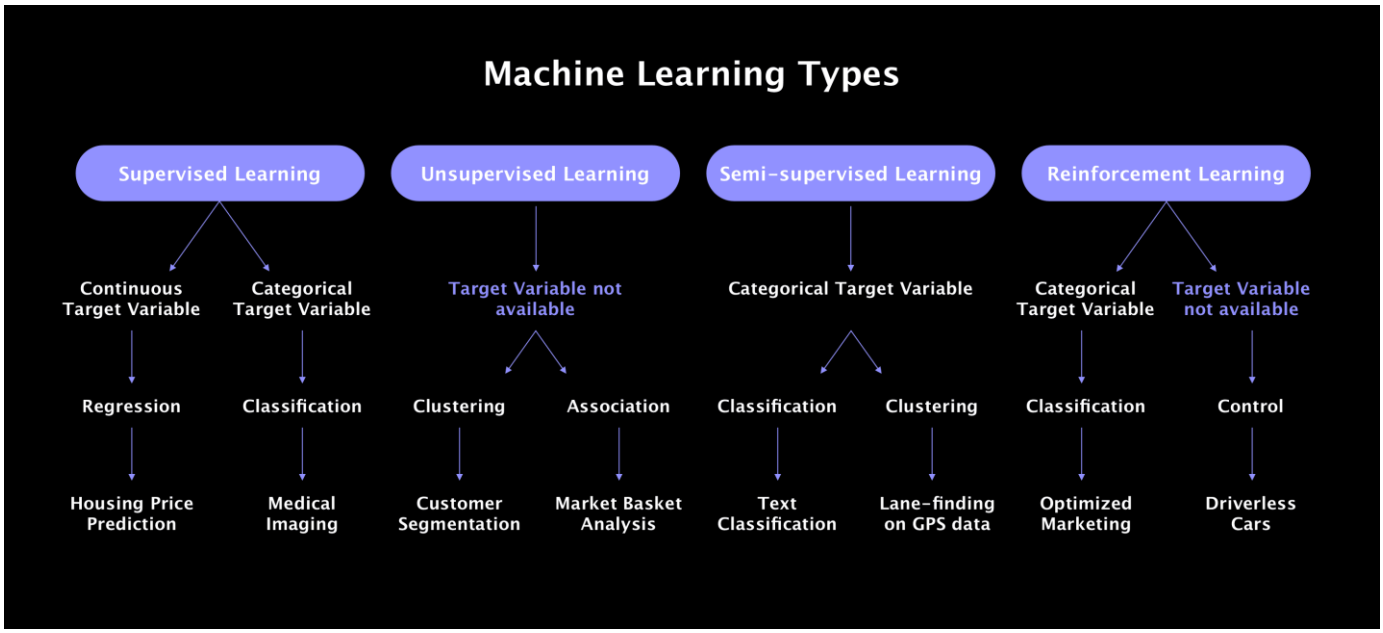


Figure 28: Machine learning algorithms types

Supervised Machine Learning Algorithms:

Supervised machine learning algorithms used with labeled data (the target output is exist), it is the basic type of Machine Learning algorithm where the programmer has greater control over the process. The engineer can decide which data he or she feeds into the system and which type of output is expected from the system. The machine must process the given data and provide solutions in the desired manner.

There are two types of supervised learning algorithms:

- Regression: Regression quantifies the relationship between one or more
- predictor variable(s) and one outcome variable.
  For example, it can be used to quantify the relative impacts of age, gender, and diet (the predictors) on weight (the output or dependent).

- Classification: Classification specifies the class to which data elements belong to and is best used when the output has finite and discrete values. It predicts a class for an input variable as well.
  For example, it can be used to classify the animal in an image, it's either a cat or a dog.

Examples of supervised machine learning algorithms:

- Linear Regression
- Logistic Regression
- Random Forest
- Decision tree
- K Nearest neighbor

Unsupervised Machine Learning Algorithms:

While in supervised ML learning, the machine is supposed to deliver one of the known results, in the unsupervised type, the result itself is not defined, and the machine has to define and deliver it. It can identify the data structure and extract useful insights from the information and detect patterns in it. The findings from the data will be used for improved efficiency in future tasks.

Examples on unsupervised machine learning algorithms:

- K-Means Clustering

Semi-supervised Machine Learning Algorithms:

Semi-supervised learning falls somewhere between the supervised and unsupervised machine learning techniques by incorporating elements of both methods. This method is used when there is only a limited set of data available to train the system, and as a result, the system is only partially trained. The information the machine generates during this partial training is called pseudo data and later on computer combines both labeled and the pseudo-data to make predictions.

Reinforcement Machine Learning Algorithms:

Reinforcement is a machine learning process closely related to Artificial Intelligence. With the aid of some available labeled and incoming data, the machine learns to reinforce and improve itself over time. It's a process which is self-sustainable and with each task completed, the system upgrades on its own. It makes use of a feedback loop called exploitation or exploration. It means that the system processes the data, analyzes the results and then improves the process for the next job. The feedback from the outcome can be both negative and positive. The system modifies itself towards the positive result and moves away from the processes, which resulted in negative feedback.

5.5.2. Model building

Now, let's back to our project implementation and see how we are going to use machine learning in our project.

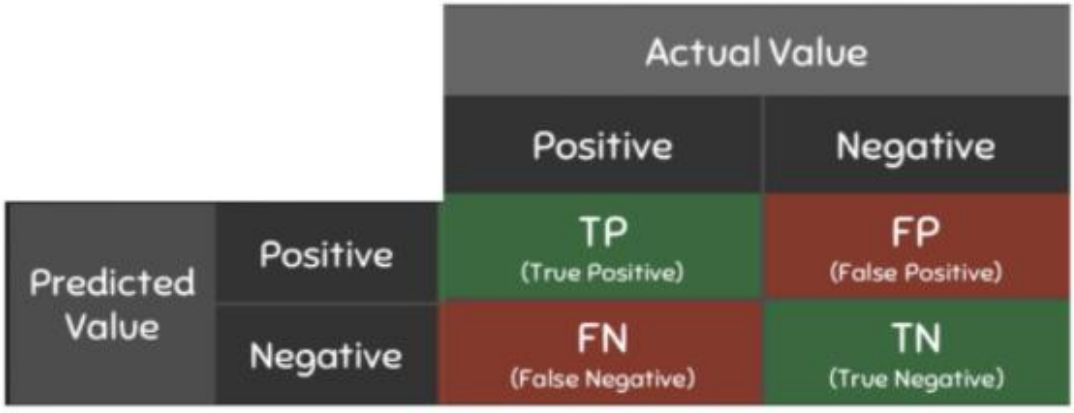
From our data, it's clear that the machine learning algorithms we will use will be supervised learning algorithms specially Classification algorithms.

The steps we are gone through to build the model are:

- We specify the input features (Symptoms), and the target output (Diseases).
- Splitting the data into train data and test data, we use the train data to train our model and the test data to evaluate the performance of the model. The training data was 75% of the total data and the test data was 25% of the total data.
- We have tried more than one model like (KNN, Decision Tree, Random Forest), and all these models gave us high performance.

5.5.3. The model evaluation

We use confusion matrix (shown in the following figure) to evaluate our models performance.



- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

Figure 29: Confusion matrix

And from this confusion matrix, we can calculate the accuracy score, precision score, recall score, F1 score (as shown in the next figure), where:

- Accuracy score: It's a measure of how good the model is. It's the ratio of the true predicted values over all the values.
- Precision score: It's a ratio of True Positives to all the positives predicted by the model. the more False positives the model predicts, the lower the precision.
- Recall score: It's a ratio of True Positives to all the positives in your data. the more False Negatives the model predicts, the lower the recall.
- F1 score: F-Score is called the Harmonic mean of precision and recall.

  F-Score provides a single score that balances both the concerns of precision and recall in one number.

  A good F1 score means that you have low false positives and low false negatives.

  An F1 score is considered perfect when it's 1, while the model is a total failure when it's 0.

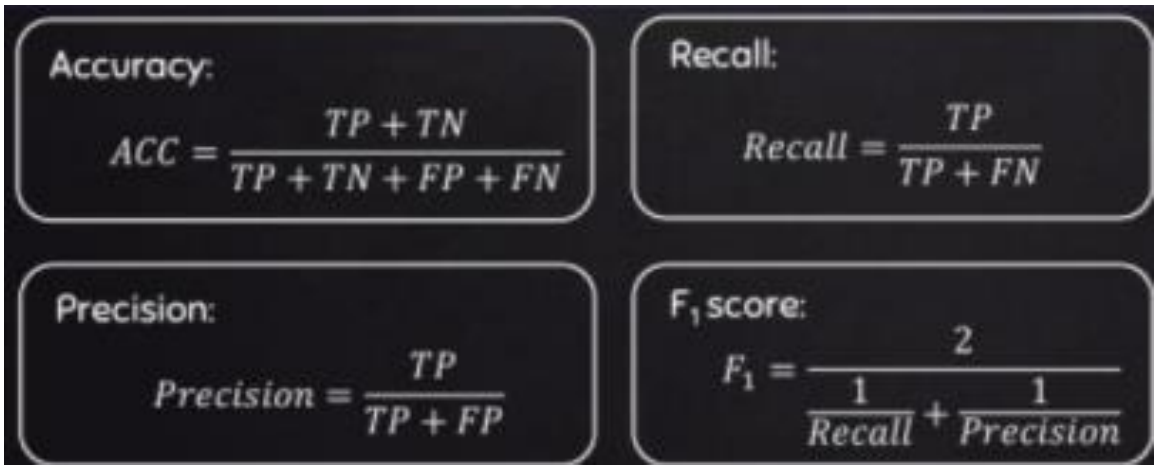


Figure 30: Evaluations scores

After we finished the evaluations on our models, we found that all models (KNN, Decision Tree, Random Forest) gives us very high performance.

The training scores was:

- Accuracy score = 99.8%
- Precision score = 99.6%
- Recall score = 99.5%
- F1 score = 99.4%

The testing scores was:

- Accuracy score = 99.7%
- Precision score = 99.6%
- Recall score = 99.6%
- F1 score = 99.6%

Why the performance of the model is very high like that, is that possible?

- The answer is simple, we have predicted this high performance before in the data analysis part above, as we saw in the analysis, we saw the relations between symptoms and diseases and also the relations between diseases, and these relations was so clear and easy to any machine learning model to train on it and gives very high performance.

We have selected the model that we will use in our project, and it will be KNN model, and here's the prediction from a sample symptom from the data as shown in the following figure.

```
In [64]: # trying our model for predicting the disease of this symptoms
         record = df["Symptoms"].loc[229].values.tolist()
         sample_syptoms = []
         for i in range(len(syptoms)):
             if record[i] != 0:
                 sample_syptoms.append(syptoms[i])
         sample_syptoms

Out[64]: ['abdominal_pain',
          'dark_urine',
          'fatigue',
          'joint_pain',
          'loss_of_appetite',
          'nausea',
          'vomiting',
          'yellowish_skin']

In [65]: prediction = model.predict(scaler.transform([record]))

In [66]: # the predicted disease
         predicted_disease = pd.DataFrame(prediction, columns=df["Disease"].columns.tolist()).T
         predicted_disease[predicted_disease[0] == 1].index.tolist()[0]

Out[66]: 'Hepatitis D'

In [67]: # the actual disease
         actual = df["Disease"].loc[229].T
         actual[actual == 1].index.tolist()[0]

Out[67]: 'Hepatitis D'
```

Figure 31: Snapshot from jupyter notebook for sample prediction

After, we have finished the modeling let's see how our model (KNN) works:

KNN algorithm:

The K-NN working can be explained on the basis of the below algorithm:

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each category.
- Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
- Step-6: Our model is ready.

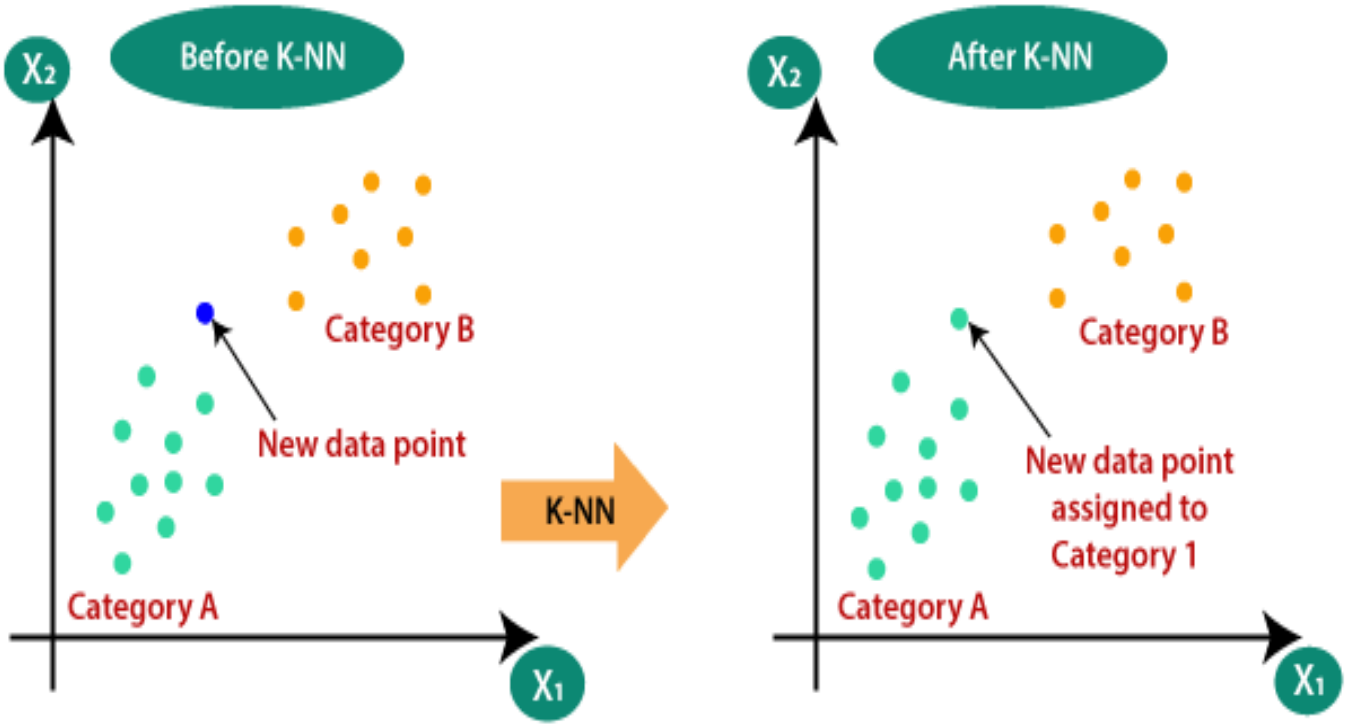


Figure 32: KNN algorithm

5.6. Deployment

Working with data is one thing, but deploying a machine learning model to production can be another.

After training, testing, and selecting the machine learning model, we are going to move to the next step which is the deployment step.

How to deploy a machine learning model in production?

Most data science projects deploy machine learning models as an on-demand prediction service or in batch prediction mode. Some modern applications deploy embedded models in edge and mobile devices.

Each model has its own merits. For example, in the batch scenario, optimizations are done to minimize model compute cost. There are fewer dependencies on external data sources and cloud services. The local processing power is sometimes sufficient for computing algorithmically complex models.

On the other hand, web services can provide cheaper and near real-time predictions. Availability of CPU power is less of an issue if the model runs on a cluster or cloud service. The model can be easily made available to other applications through API calls and so on.

What is API?

An application programming interface (API) is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software.

In contrast to a user interface, which connects a computer to a person, an application programming interface connects computers or pieces of software to each other. It is not intended to be used directly by a person (the end-user) other than a computer programmer who is incorporating it into the software.

The term API is often used to refer to web APIs, which allow communication between computers that are joined by the internet.

Web APIs:

Web APIs are a service accessed from client devices (Mobile Phones, laptops, etc.) to a web server using the Hypertext Transfer Protocol (HTTP). Client devices send a request in the form of an HTTP request, and are met with a response message usually in JavaScript Object Notation (JSON). Developers typically use Web APIs to query a server for a specific set of data from that server.

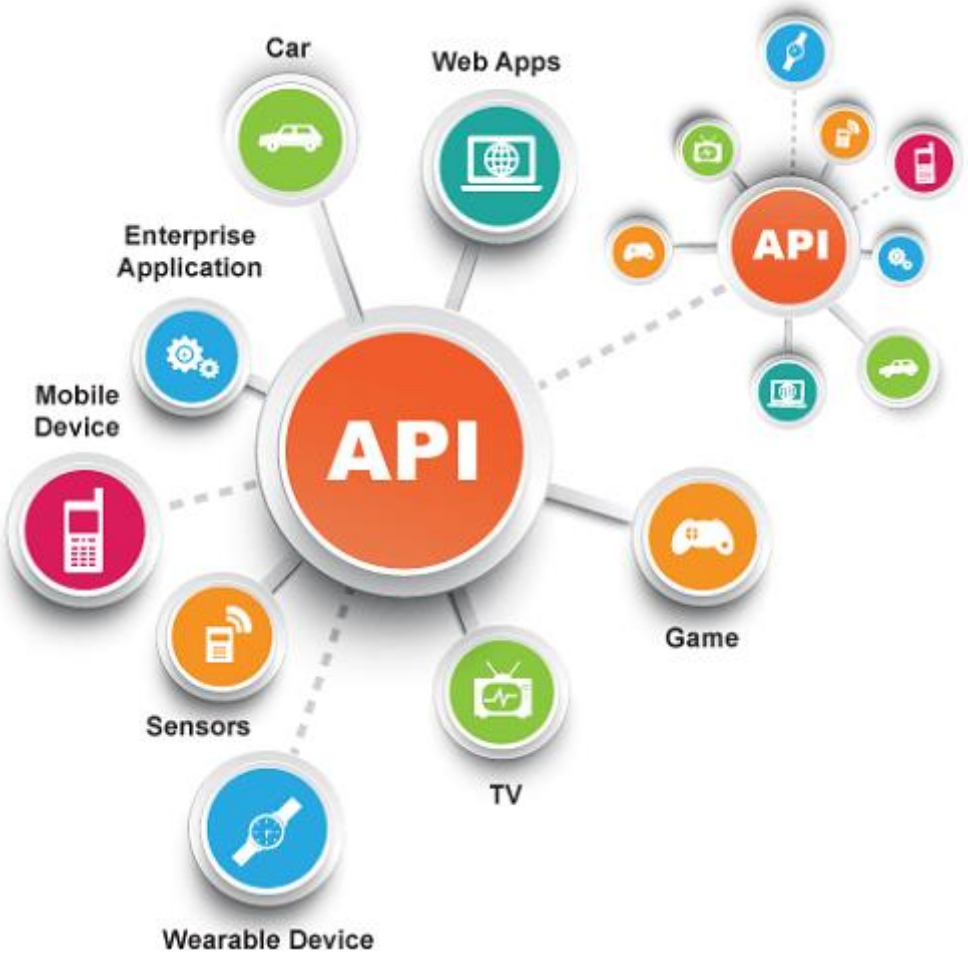


Figure 33: APIs

How we deploy our machine learning model and make it available to use? How we have built our online API?

we have deployed the machine learning model as a web API, and we have implemented our API using Flask framework.

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself.

After Building our API, we need it to be available to use. and the best way for doing that is to deploy our API on an online server.

After searching, we have found a free web server that gives us the ability to deploy our API on it and make our API online, this server is called Heroku App.

Now, anyone can access our API and send HTTP requests and receive the responses through the internet.

Here's some snapshots to our API from Heroku app:

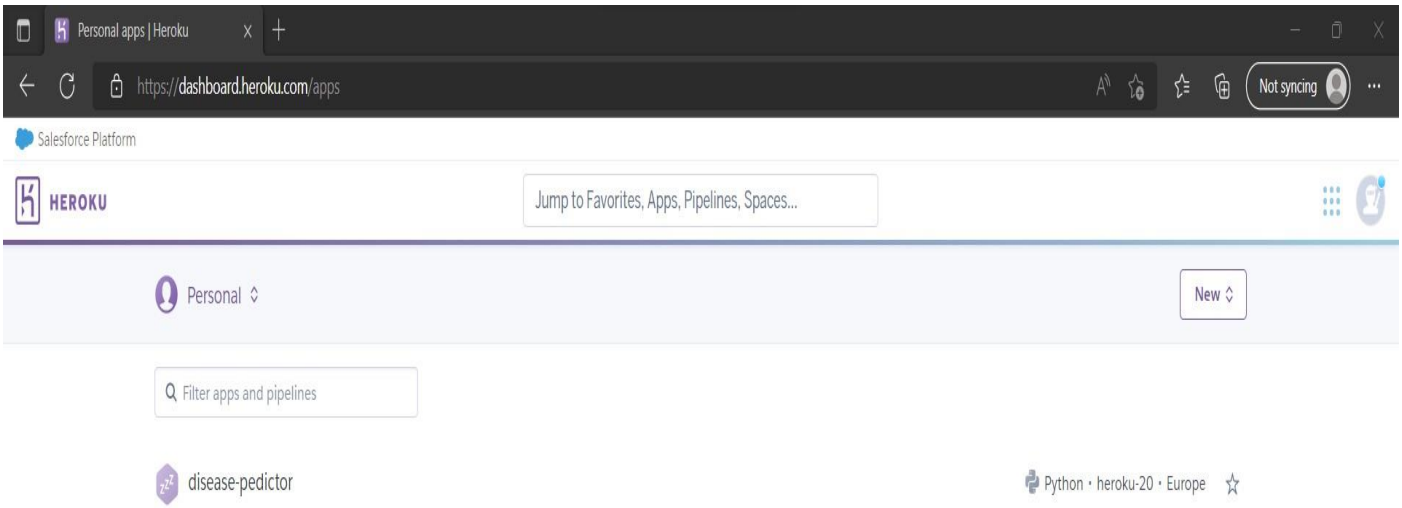


Figure 34: Heroku app dashboard

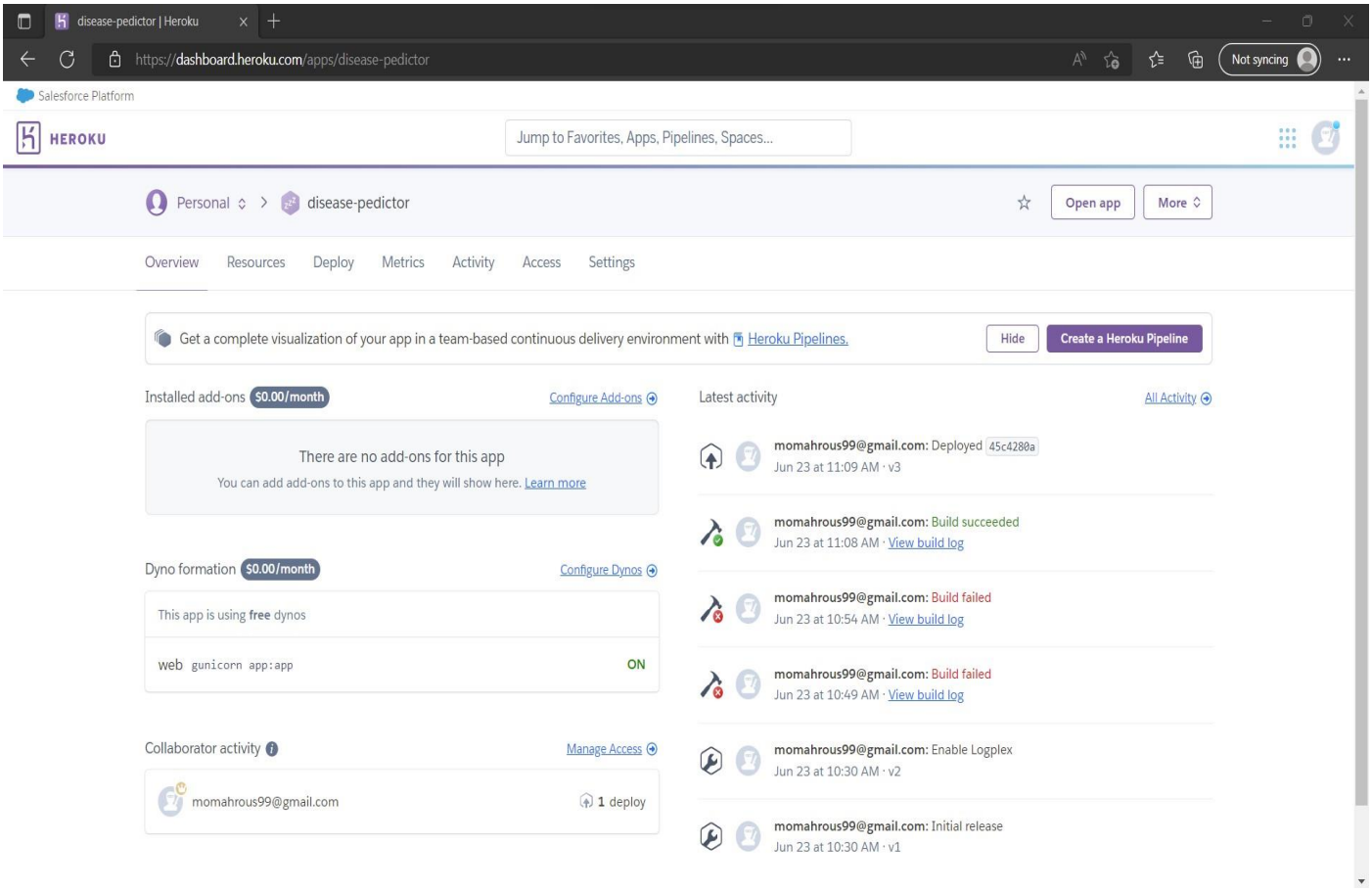


Figure 35: disease-predictor API on Heroku

Steps of our API implementation:

- Installing all the needed python packages (Flask).
- Writing the python script that will receive any request and send a response back.
- Gathering all the requirements for the deployment of the API on the Heroku web server.
- Deploy the API on the Heroku server.
- Getting the URL of our API.

How to use our API in?

It’s so simple, just send an HTTP request with your symptoms using the API’s URL, and get the response which is your predicted disease as JSON.

Here’s a sample for testing our API:

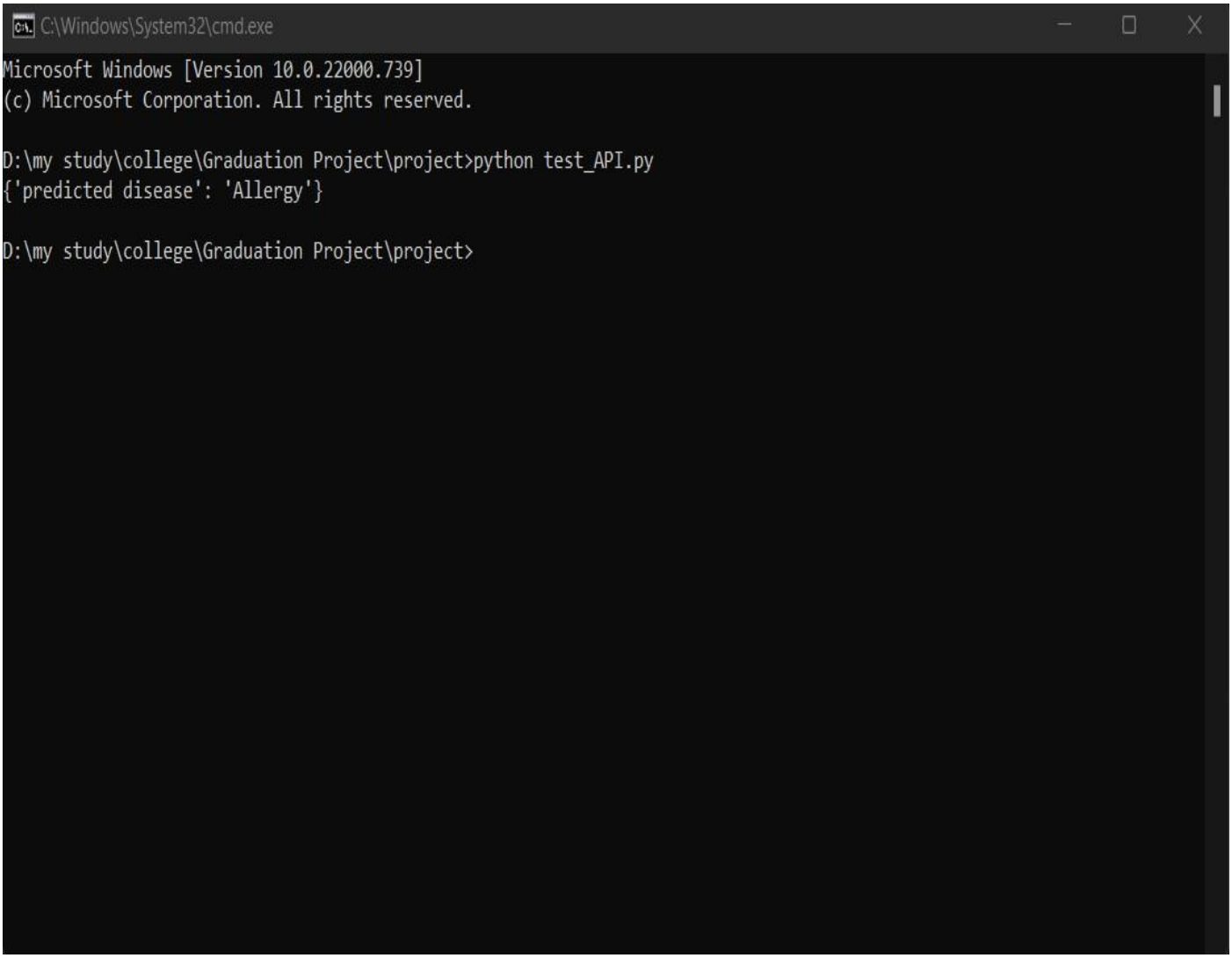


Figure 36: API response

We can also use our API through any Browser as following:

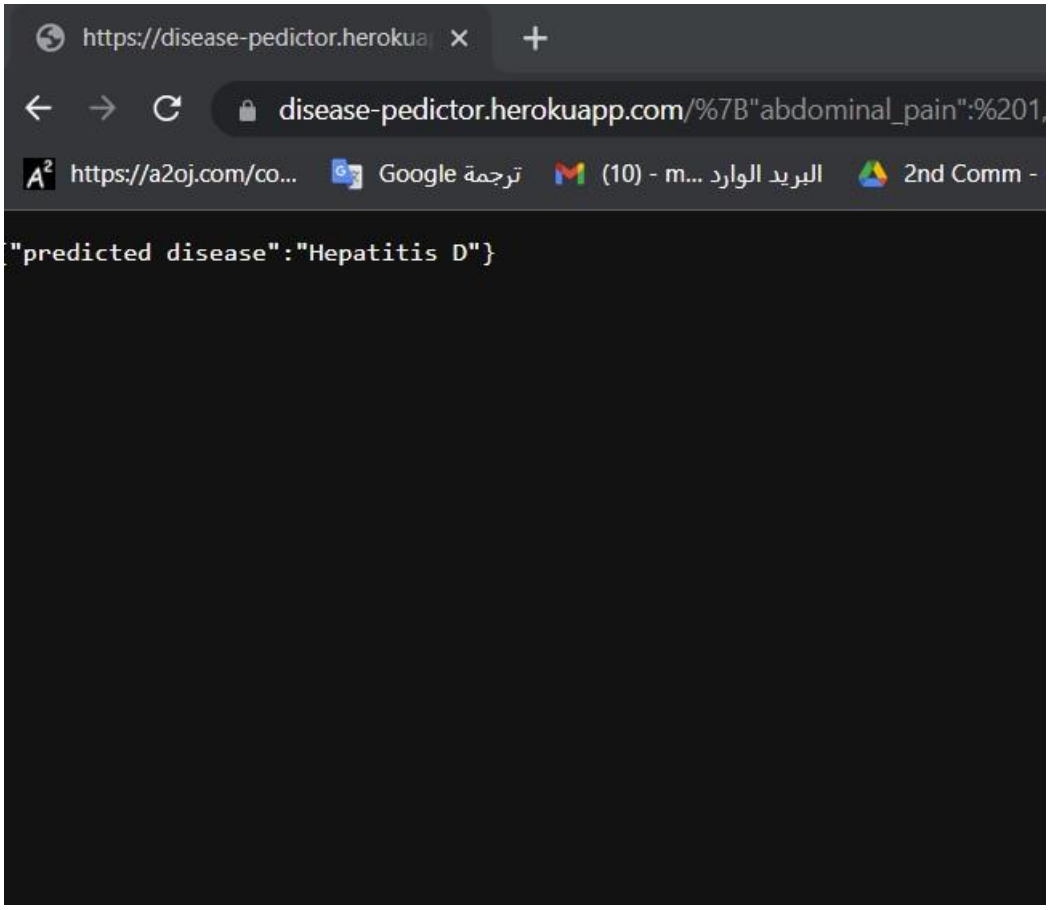


Figure 37: API response from browser

After we have seen, how we built our API, and how can we use it. The next step is to build the Mobile app that will act as the user interface and connect the app to the API. Then anyone can use our mobile app easily.

5.7. Mobile Application

5.7.1. Introducing Flutter

Flutter is Google's portable UI framework for building modern, native, and reactive applications for iOS and Android.

Google is also working on Flutter desktop embedding and Flutter for the Web (Hummingbird) and embedded devices (Raspberry Pi, home, automotive, and more).

Flutter is an open-source project hosted on GitHub with contributions from Google and the community.

Flutter uses Dart, a modern object-oriented language that compiles to native ARM code and production-ready JavaScript code.

It uses the Skia 2D rendering engine that works with different types of hardware and software platforms and is also used by Google Chrome, Chrome OS, Android, Mozilla Firefox, Firefox OS, and others.

Skia is sponsored and managed by Google and is available for anyone to use under the BSD Free Software License.

Skia uses a CPU-based path render and supports the OpenGL ES2-accelerated backend.

Dart is the language that you'll use to develop your Flutter applications.

Dart is ahead-of-time (AOT) compiled to native code, making your Flutter application fast.

It is also just-in-time (JIT) compiled, making it fast to display your code changes such as via Flutter's stateful hot reload feature

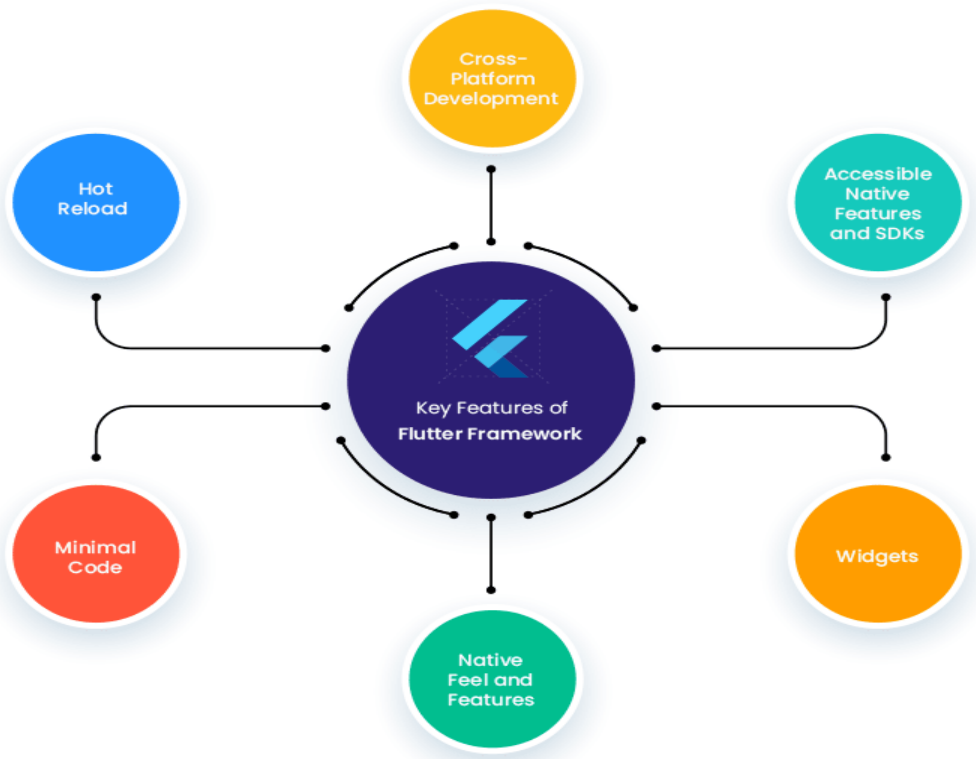


Figure 38: Key features of flutter

Defining Widgets

The Flutter UI is implemented by using widgets from a modern reactive framework.

It uses its own rendering engine to draw widgets.

You might be asking, what is a widget?

Widgets can be compared to LEGO blocks; by adding blocks together, you create an object, and by adding different kinds of blocks, you can alter the look and behavior of the object.

Widgets are the building blocks of a Flutter app, and each widget is an immutable declaration of the user interface.

In other words, widgets are configurations (instructions) for different parts of the UI.

Placing the widgets together creates the widget tree.

For example, say an architect draws a blueprint of a house; all the objects like walls, windows, and doors in the house are the widgets, and all of them work together to create the house or, in this case, the application.

Since widgets are the configuration pieces of the UI and together, they create the widget tree, how does Flutter use these configurations?

Flutter uses the widget as the configuration to build each element, which means the element is the widget that is mounted (rendered) on the screen.

The elements that are mounted on the screen create the element tree.

Here's a brief look at the wide array of widgets:

➤ Widgets with structuring elements such as a list, grid, text, and button

➤ Widgets with input elements such as a form, form fields, and keyboard listeners

➤ Widgets with styling elements such as font type, size, weight, color, border, and shadow

➤ Widgets to lay out the UI such as row, column, stack, centering, and padding

➤ Widgets with interactive elements that respond to touch, gestures, dragging, and dismissible

➤ Widgets with animation and motion elements such as hero animation, animated container, animated crossfade, fade transition, rotation, scale, size, slide, and opacity

➤ Widgets with elements like assets, images, and icons

➤ Widgets that can be nested together to create the UI needed

➤ Custom widgets you can create yourself

Figure 39: widgets samples

Understanding Widget Lifecycle Events

In programming, you have different lifecycle events that usually happen in a linear mode, one after another as each stage is completed.

In this section, you'll learn the widget lifecycle events and their purpose.

To build the UI, you use two main types of widgets, StatelessWidget and StatefulWidget.

A stateless widget is used when the values (state) do not change, and the stateful widget is used when values (state) change

A **StatelessWidget** is built based on its own configuration and does not change dynamically.

For example, the screen displays an image with a description and will not change.

A **StatefulWidget** is built based on its own configuration but can change dynamically.

For example, the screen displays an icon with a description, but values can change based on the user's interaction, like choosing a different icon or description.

This type of widget has a mutable state that can change over time.

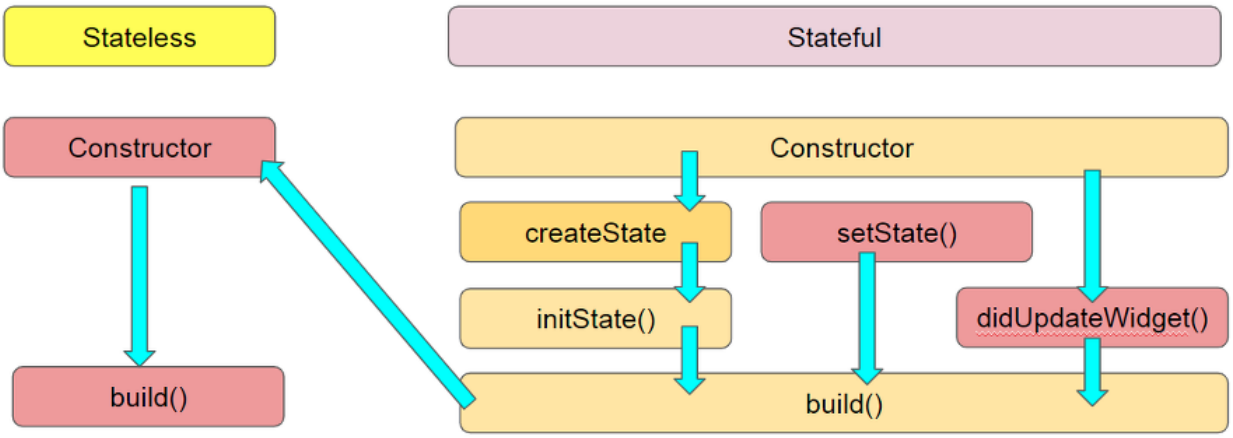


Figure 40: Widget Lifecycle Events

Understanding Themes to Style Your App

The theme widgets are a great way to style and define global colors and font styles for your app.

There are two ways to use theme widgets—to style the look and feel globally or to style just a portion of the app.

For instance, you can use themes to style the color brightness (light text on a dark background or vice versa); the primary and accent colors; the canvas color; and the color of app bars, cards, dividers, selected and unselected options, buttons, hints, errors, text, icons, and so on.

The beauty of Flutter is that most items are widgets, and just about everything is customizable. In fact, customizing the **ThemeData** class allows you to change the color and typography of widgets.

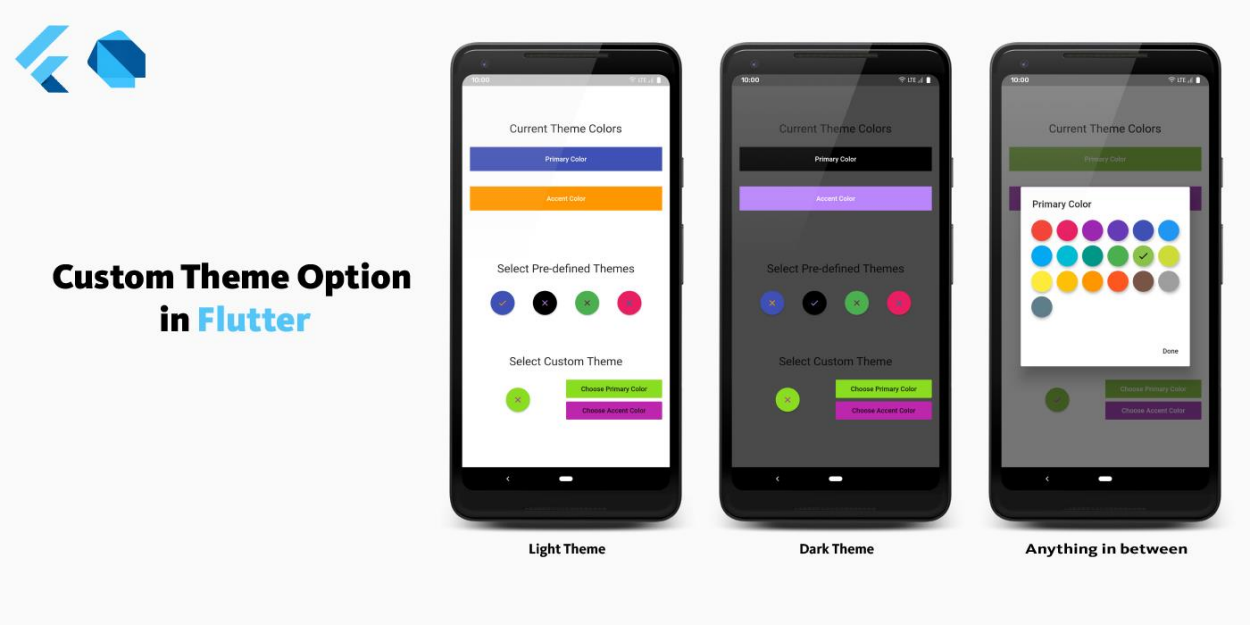


Figure 41: Themes in flutter

Why Flutter Framework?

Flutter is used in the implementation of our application as it has become one of the most hyped cross native frameworks since its stable release.

Nowadays, most of the companies are enthusiastic about flutter. It is mainly because one can develop applications for Android, iOS, Windows, Mac, Linux, and web from a single codebase. Despite fast development and flexible User Interface.

It was originally an open-source project for mobile application development. Later it was extended to support platforms like web, Windows, Google Fuchsia and Linux.

You might be already aware of Google's new operating system called Fuchsia. Here, Flutter is the primary source for developing its applications.

Recently, Flutter has become more competitive with React Native (Facebook) and Xamarin (Microsoft).

Back in the days of Objective C/Swift and Java/Kotlin as primary languages for mobile development, building apps was expensive.

You had to build two separate apps, which obviously meant doing the work twice.

To solve this problem, several frameworks have been constructed for the creation of hybrid (or cross-platform) apps in HTML5 and JavaScript.

Among the cross-platform toolkits, including Phonegap, Xamarin, React Native, and more, the Flutter framework has quickly become increasingly popular among developers, enterprises, entrepreneurs and users.

Flutter is the only framework with a mobile SDK that provides a responsive style without using a JavaScript bridge, thereby reaching a level of performance that rivals its cousin and direct competitor React Native.

It easily integrates with the different platforms such as Android, IOS and Linux, MAC, Windows, and Google Fuchsia applications.

Flutter is fast, and the rendering runs at 60 frames per second (fps) and 120fps for capable devices. The higher the fps, the smoother the animations and transitions.

Applications made in Flutter are built from a single codebase, are compiled to native ARM code, use the graphics processing unit (GPU), and can access specific iOS and Android APIs (like GPS location, image library) by communicating via platform channels.

Flutter provides the developer with tools to create beautiful and professional-looking applications and with the ability to customize any aspect of the application.

You'll be able to add smooth animations, gesture detection, and splash feedback behavior to the UI.

Flutter applications result in native performance for both iOS and Android platforms.

During development, Flutter uses hot reload to refresh the running application in milliseconds when you change the source code to add new features or modify existing ones.

Using hot reload is a great way to see the changes you make to your code on the simulator or device while keeping the application's state, the data values, on the screen.

1. Fast Development

Flutter is faster than many other application development frameworks.

With its “hot reload” feature, you can experiment, build UIs, add/remove features, test, and fix bugs faster.

Flutter’s hot reload helps you see code and user interface changes immediately while retaining state to an app running the Dart virtual machine.

In other words, every time you make code changes, you don’t need to reload the app, because the current page shows the changes immediately.

This is an incredible time-saving feature for any developer

Flutter uses the Skia Graphics Library which is a fast and mature open-source graphics library. It redraws the UI every time a view change.

The result? A quick loading and smooth app experience.

Thus, reducing the overall app development time.

2. Expressive and Flexible UI

You can really build beautiful apps in Flutter. Also, the end-user experience is similar to native apps.

Flutter has a layered architecture that lets you control every pixel on the screen.

Thus, customization is very simple in Flutter. With its powerful composting capabilities, you can overlay and animate graphics, text, video, and other controls without any limitations.

You’ll also find a set of widgets that deliver pixel-perfect experiences on Android and iOS.

It enables the ultimate realization of Material Design.

Just in case you don’t know, Material.io is Google’s initiative to build beautiful, usable products with Material Components for digital experiences.

3. Native Performance

Flutter's widgets incorporate all critical platform differences such as scrolling, navigation, icons and fonts. This provides a native performance experience on both iOS and Android.

4. Dart Language

Dart programming language is developed by Google and is meant for mobile, desktop, backend, and web applications.

It is a client-optimized language for fast performing apps on multiple platforms.

Dart is AOT (Ahead of Time) compiled to fast, predictable, native code, allowing writing almost all of Flutter code in Dart.

This makes Flutter extremely fast and customizable. Virtually, everything (including all the widgets) can be customized.

5. Important Flutter Tools

Flutter framework supports many different tools including Android Studio and Visual Studio Code.

It also provides support for building apps from the command line.

Dart DevTools, which is a new debugging tool, is more flexible and allows runtime inspection. You can also view logs, debug apps and inspect widgets for Flutter App Development.

1. Widget inspector helps to visualize and explore the tree hierarchy. Flutter uses this for UI rendering.
2. Timeline view helps you to monitor your application at a frame-by-frame level. You can also identify rendering and computational work in timeline view.
3. Source-level Debugger: It lets you step through code, set breakpoints and investigate the call stack.

4. Logging View displays events from the Dart runtime, application frameworks and app-level logging events.

Flutter vs React Native.

As you can see on Google trends Flutter has gained a lot of popularity recently and has overtaken React Native.

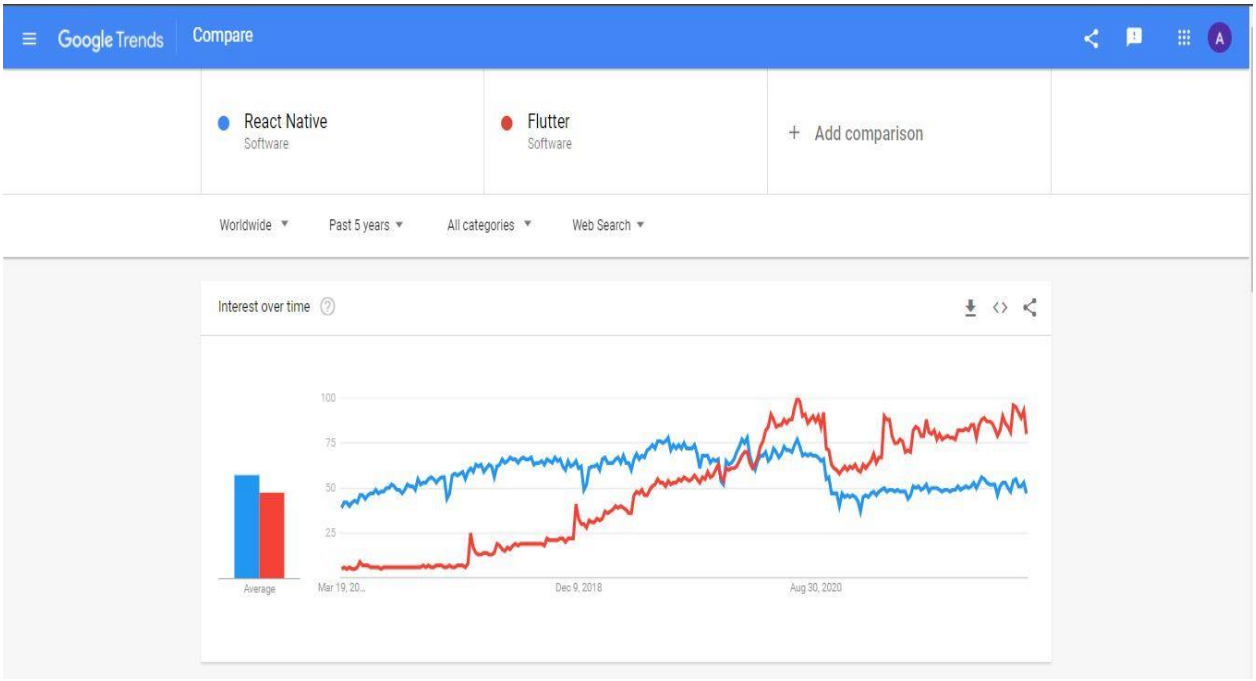


Figure 42: Difference between flutter and react native from google trends

	React Native	Flutter
backed by	Facebook	Google
launched	2015	2018
programming language	JavaScript	Dart
performance	near-native	superbly near-native
code reusability	up to 90%	up to 90%
user interface	uses native UI controllers	uses custom widgets
major use cases	Facebook Instagram Pinterest	Google Ads Alibaba Groupon

Figure 43: Comparison between flutter and react native

Apps developed on Flutter

The popularity of Flutter app development is constantly growing.

Now Flutter is widely used to create apps for the likes of Alibaba, Yandex, Airbnb, Uber, eBay and other leading companies.

Figure 44: Apps developed on flutter

5.7.2. Flutter State Management

What Is State Management?

State Management is important whether you are building a mobile app or a web application. It refers to the management of the state of one or more UI controls. UI controls can be text fields, radio buttons, checkboxes, dropdowns, toggles, form, and many more.

A website/web application consists of a number of such UI controls and the state of the one UI control depends on the state of other UI controls.

Every framework has its way to manage the state of UI controls.

Why Do We Need State Management?

State management is very important in application development. It centralizes all the states of various UI controls to handle data flow across the application.

in Flutter, everything is a widget. The widget can be classified into two categories, one is a **Stateless widget,** and another is a **Stateful widget.**

The Stateless widget does not have any internal state. It means once it is built, we cannot change or modify it until they are initialized again. On the other hand,

a Stateful widget is dynamic and has a state. It means we can modify it easily throughout its lifecycle without reinitialized it again.

What is State?

A state is information that can be **read** when the widget is built and might be changed **or modified** over a lifetime of the app.

If you want to change your widget, you need to update the state object, which can be done by using the **setState()** function available for Stateful widgets.

The **setState()** function allows us to set the properties of the state **object** that triggers a redraw of the UI.

The state management is one of the most popular and necessary processes in the lifecycle of an application.

According to official documentation, Flutter is declarative. It means Flutter builds its UI by reflecting the current state of your app.

The following figure explains it more clearly where you can build a UI from the application state.

Figure 45: Building UI from application state

Let us take a simple example to understand the concept of state management. Suppose you have created a list of customers or products in your app.

Now, assume you have added a new customer or product dynamically in that list. Then, there is a need to refresh the list to view the newly added item into the record. Thus, whenever you add a new item, you need to refresh the list.

This type of programming requires state management to handle such a situation to improve performance.

It is because every time you make a change or update the same, the state gets refreshed.

In Flutter, the state management categorizes into two conceptual types, which are given below:

1. Ephemeral State
2. App State

Ephemeral State

This state is also known as UI State or local state. It is a type of state which is related to the **specific widget,** or you can say that it is a state that contains in a single widget. In this kind of state, you do not need to use state management techniques.

The common example of this state is **Text Field**.

App State

It is different from the ephemeral state. It is a type of state that we want to **share** across various parts of our app and want to keep between user sessions. Thus, this type of state can be used globally.

Sometimes it is also known as application state or shared state. Some of the examples of this state are User preferences, Login info, notifications in a social networking app, the shopping cart in an e-commerce app, read/unread state of articles in a news app, etc

The following diagram explains the difference between the ephemeral state and the app state more appropriately.

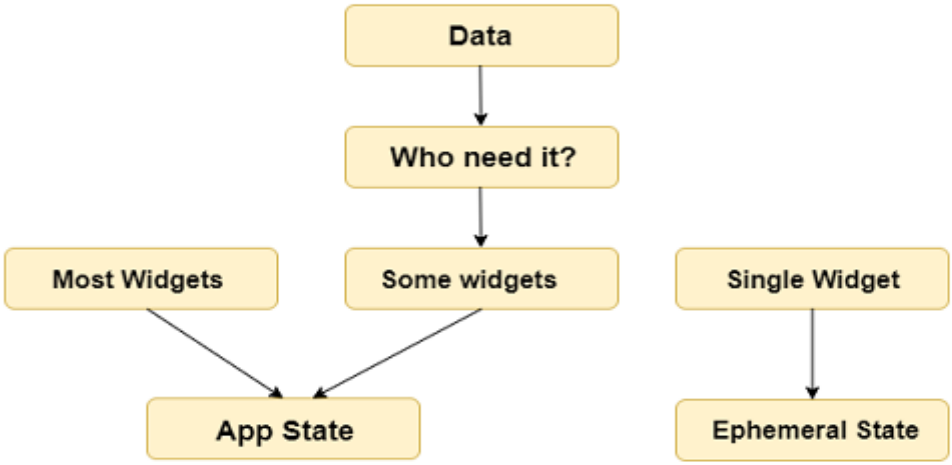


Figure 46: State management categories

List Of State Management Approaches/Techniques:

- Provider
- InheritedWidget & InheritedModel
- SetState
- Redux
- Fish-Redux
- Flutter BloC
- Flutter Commands
- GetIt
- MobX
- Riverpod
- GetX

Let’s explore each in detail.

-Provider:

Provider package is a wrapper around InheritedWidgets to make them more reusable and easier to use.

It does not require much code and it is a most basic form of provider.

It takes a value and represents it, but it does not pay attention to the changes in the value it offers.

-InheritedWidget & InheritedModel:

InheritedWidget is the base class that effectively passes all the data down the tree from top to bottom.

It simply allows any below Widget in the tree to access properties of Inherited Widget.

This approach becomes complex when app size is large as it results in much boilerplate.

InheritedModel is an inherited widget that allows user to specify which data and sections they care about and rebuilds only necessary descendants.

It ensures that inherited widget dependents should not rebuild unconditionally.

-setState:

setState is very useful for local, widget-specific state management.

-Redux:

Redux is a unidirectional data flow architecture that makes separation of concerns, i.e., business logic and presentation logic.

Those who are familiar with React Native framework, are also familiar with Redux architecture.

Because of the different separations of the application parts, it helps developers to make UI changes faster and easier and makes debugging easier.

It is unidirectional and more suitable for synchronous situations.

-Fish-Redux:

Fish-Redux is a high-level assembled Flutter application framework based on Redux architecture.

It is suitable for building medium and large applications.

It is developed by Alibaba Tech and then moved to open-source.

Redux and Fish Redux both frameworks work on different layers and simplify state management.

-Flutter Commands:

Flutter command is another way to manage state based on ValueNotifiers.

Here, the command refers to an object that wraps a function.

ValueNotifier is a special type of class that extends a ChangeNotifier. ChangeNotifier provides notifyListeners() method to inform the property changes to the listeners.

A ValueNotifier can hold a single value.

Instead of rebuilding the entire widget tree when setState() is called, ValueNotifier is better at managing state by notifying widgets whenever a value is updated/changed.

Whenever notifyListeners() is called, all its listeners' widgets(subscribed widgets) get rebuilt even the property of the widget is not changed.

-GetIt:

GetIt can be used to access objects from UI instead of InheritedWidget or Provider.

GetIt is not a state management technique, but it is a service locator for your objects.

Service locators is a concept that decouples the interface from the main implementation and allows access to concrete implementation from everywhere in the app. It is easy to use and extremely fast.

Unlike Provider or Redux, it does not require a special widget to access your data

-MobX:

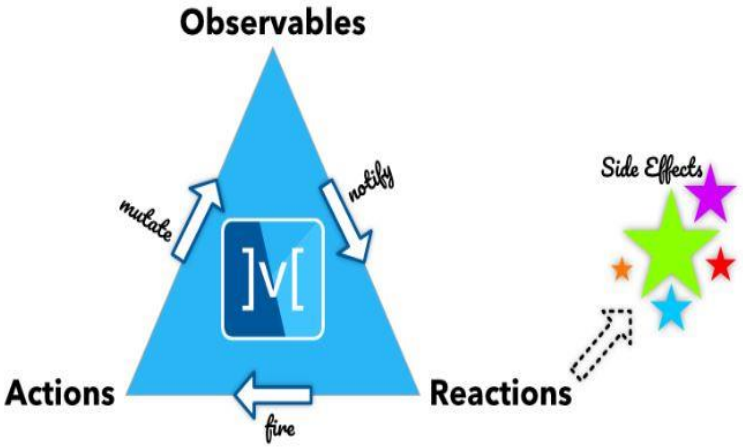


Figure 47: Mobx

Observables, Actions, Reactions – these 3 are important concepts of MobX.

It supports unidirectional data flow and focuses on this principle:

“Anything that can be derived from the application state, should be. Automatically”.

This state management library makes state management simpler and scalable.

It allows developers to manage application state outside of any UI framework.

It is designed to make state management super easy by reactively detecting all changes and propagate those to the required UI.

It automatically tracks what is being used/consumed, known as observables.

Reactions complete the MobX circle of observables, actions, and reactions.

All reactions re-run and rebuild the widgets when observables properties are changed.

Key feature: Reactions automatically track all the changes in the observables without any explicit connections.

-Riverpod:

Riverpod is similar to Provider.

A provider is an object that wraps a piece of state and allows listening to that state.

It is an improvised version of the Provider and aims to overcome the common problems that the provider has.

It ensures better performance, testability, and readability with a unidirectional data flow.

It catches programming errors at compile-time, saving developers from runtime exceptions. Also, Riverpod eliminates the dependency on BuildContext which is required in Provider.

It is more flexible, scalable, testable, and more simplified than the provider

-GetX:

GetX is an extra-light and powerful state management solution for Flutter.

It combines route management, state management, navigation, and intelligent dependency injection in a more practical and faster way.

GetX offers complete decoupling of View, presentation logic, dependency injection, and business logic.

It is the easiest, practical, secure, and scalable way to build applications with Flutter.

It offers a wide range of APIs and features that let developers build apps hassle-free and quicker.

It uses its dependency injection feature, making app development easier as you don’t require to depend on context or route or widget tree.

In our project we used Flutter BloC technique.

BLoC is a business logic component (which separates presentation from business logic).

It is a state management system and allows developers to access data from a central place.

It is recommended approach by Google Developers and the most used architecture in Flutter to manage states.

Cubit

Figure 48: Cubit

A *Cubit* is a class that extends *BlocBase* and can be extended to manage any type of state. *Cubit* requires an initial state which will be the state before *emit* has been called. The current state of a *cubit* can be accessed via the *state* getter and the state of the *cubit* can be updated by calling *emit* with a new *state*.

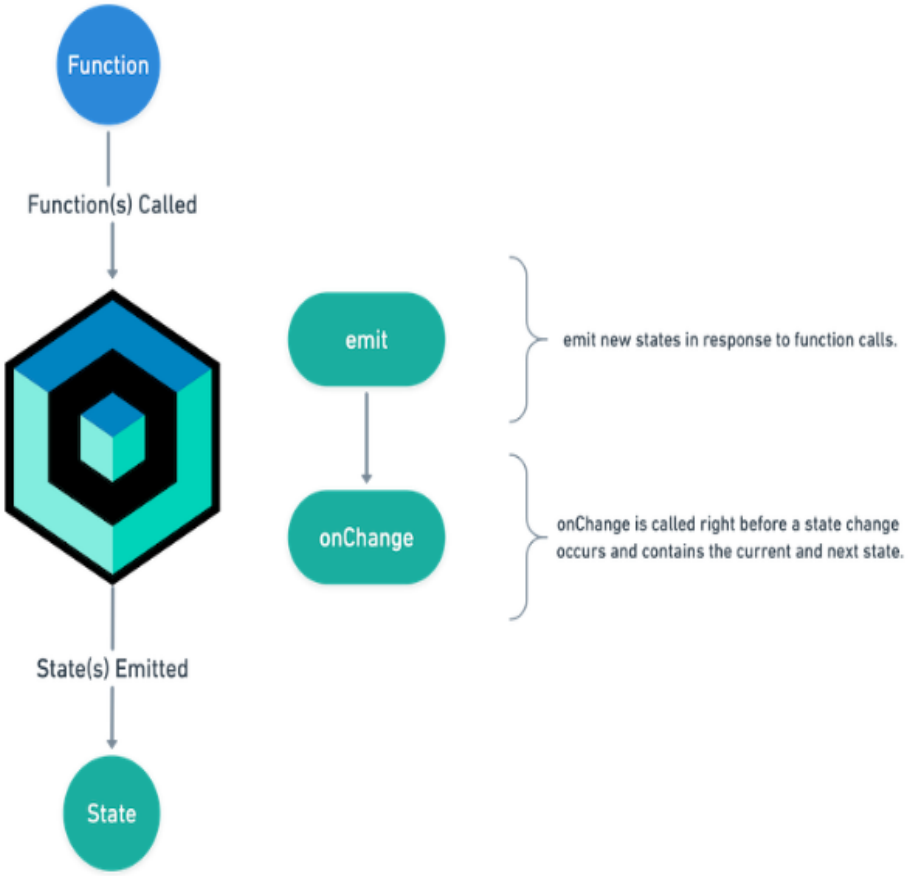


Figure 49: Cubit states

State changes in cubit begin with predefined function calls which can use the *emit* method to output new states. *onChange* is called on each state change and contains the current and next state.

Cubit is a subset of the BLoC Pattern package that does not rely on events and instead uses methods to emit new states.

Cubit reduces complexity and eliminates the event classes. It uses emit rather than yield to emit state.

Since emit works synchronously, you can ensure that the state is updated in the next line.

So, we can use Cubit for simple states, and as needed we can use the Bloc.

Bloc

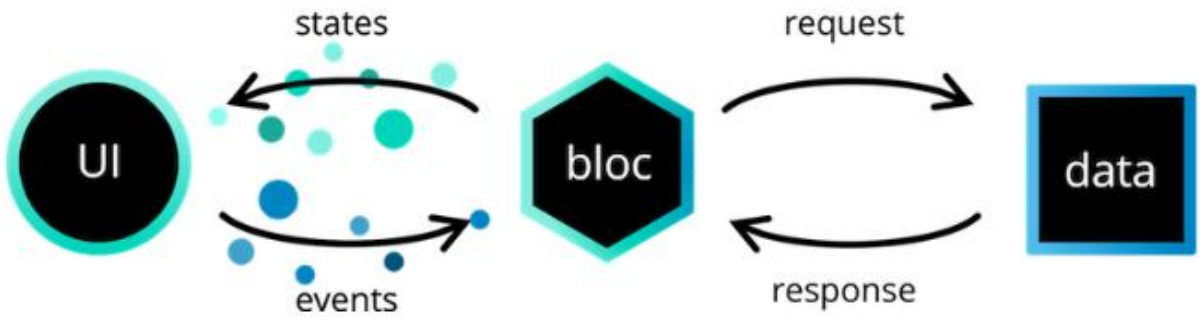


Figure 50: Bloc

A *Bloc* is a more advanced class which relies on *events* to trigger *state* changes rather than functions. *Bloc* also extends *BlocBase* which means it has a similar public API as *Cubit*. However, rather than calling a *function* on a *Bloc* and directly emitting a new *state*, *Blocs* receive *events* and convert the incoming *events* into outgoing *states*.

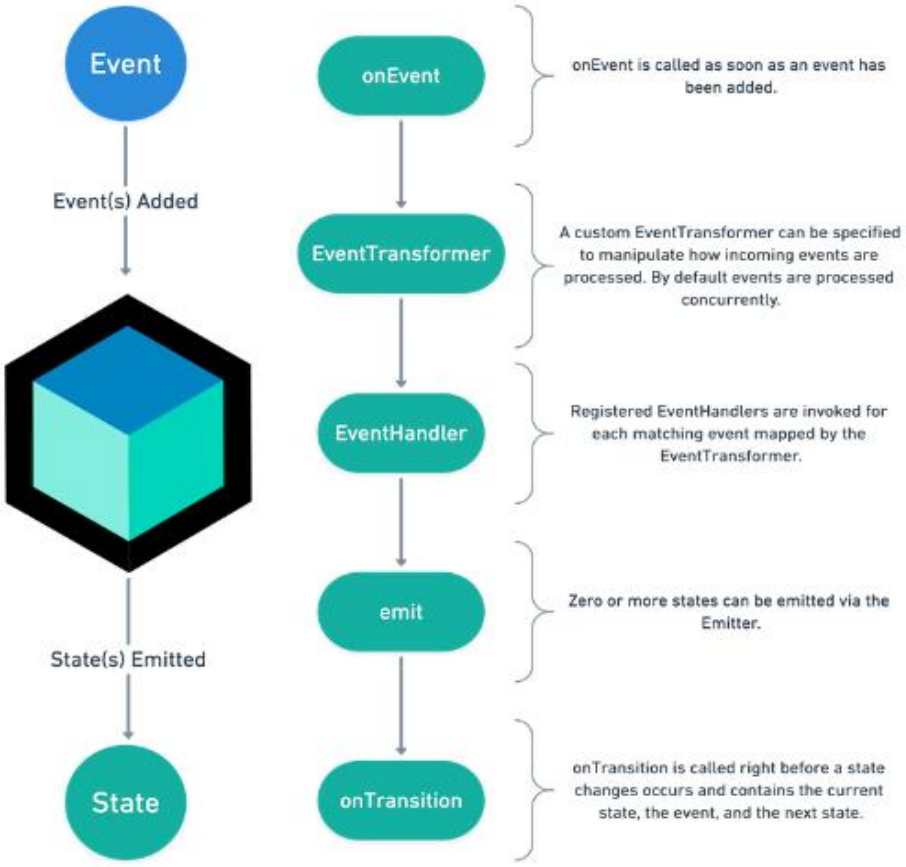


Figure 51: Bloc states

State changes in bloc begin when events are added which triggers *onEvent*. The events are then funnelled through an *EventTransformer*. By default, each event is processed concurrently but a custom *EventTransformer* can be provided to manipulate the incoming event stream. All registered *EventHandlers* for that event type are then invoked with the incoming event. Each *EventHandler* is responsible for emitting zero or more states in response to the event. Lastly, *onTransition* is called just before the state is updated and contains the current state, event, and next state.

5.7.3. What is postman?

Postman is an API (application programming interface) development tool that helps to build, test, and modify APIs.

Almost any functionality that could be needed by any developer is encapsulated in this tool.

It is used by over 5 million developers every month to make their API development easy and simple.

It has the ability to make various types of HTTP requests (GET, POST, PUT, PATCH), save environments for later use, and convert the API to code for various languages (like JavaScript, and Python).

Let’s say I wanted to make a GET. If I wanted to test a GET request against this route without using Postman instead actually writing out code in something like Flask I would have to write out a whole new route and function to perform the request, then I would have to specify with more code what I want the response to look like, and finally I would have to print out the response to the console or provide some other way of actually viewing the response.

Granted, I would probably need to write all this out anyway to make a functioning app using this API but doing all this to simply test an API's functionality is unnecessarily tedious and time consuming when something like Postman exists.

With Postman, such a test is much more streamlined.

All I have to do is plug the route into the address bar, select the GET response method on the dropdown box to its left, punch in my API key in the “Headers” section, specify that I want the response in “pretty” JSON format, and hit send.

Then, I get the response data in easy-to-read JSON with a status code of 200, confirming the GET request was successful. It’s that simple!

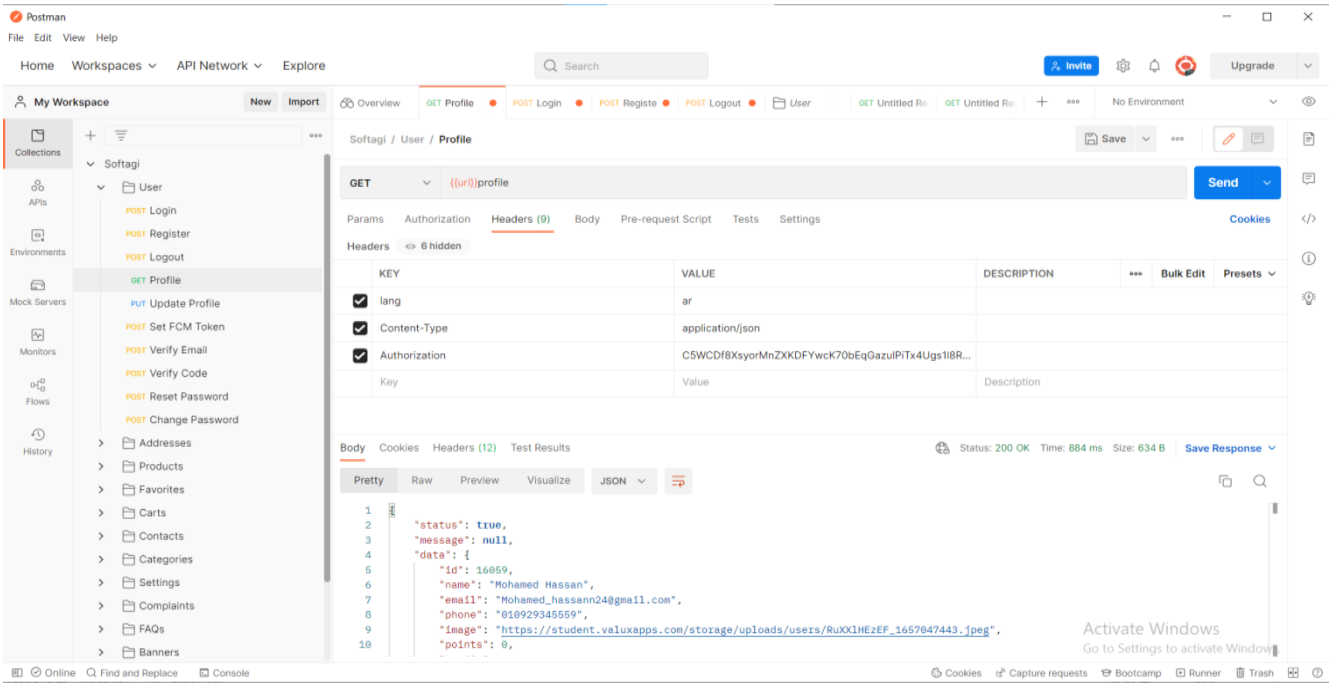


Figure 52: GET request on postman

What about making POST requests?

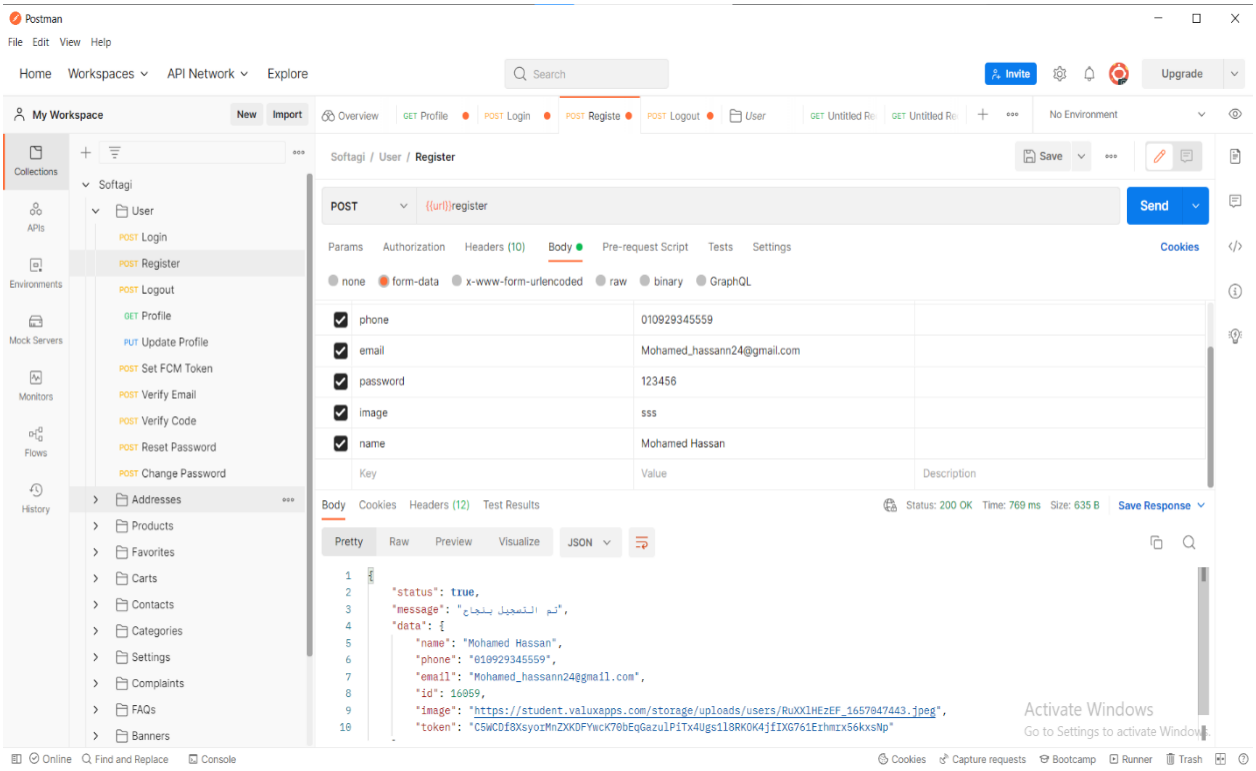


Figure 53: POST request on Postman

In the above example: we made a Post request to https://student.valuxapps.com/api/

Just like when making a GET request with Postman,

I added the route in the address bar, but instead of choosing GET in the dropdown box, I instead chose POST.

In the request body, set to “raw”, I inserted a dummy blog post in JSON format.

Because this website is made for simple testing purposes, it didn’t require an API key or any other header, but if it did, I would have put the key/value pairs in the “Headers” section just like I did with the earlier GET request example.

5.7.4. What is Dio?

Today, all the mobile and web applications depend on back-end APIs that we can consume with the help of an HTTP client.

Flutter framework offers an Http package that works great when we need to do basic stuff.

When you start working on a big application and we need to do something more advanced task. Your application will face lots of problems with network error handling.

In that case, we need something big and advanced connection library which has extra features like interceptors, logs, cache, etc, that will be helpful in many tasks like adding the token authentication for each request and logging requests.

Dio is a powerful HTTP client for Dart.

It has support for interceptors, global configuration, **FormData**, request cancellation, file downloading, and timeout, among others.

Flutter offers an HTTP package that's nice for performing basic network tasks but is daunting to use when handling some advanced features.

By comparison, Dio provides an intuitive API for performing advanced network tasks with ease.

Get Started

- Add dependency

```
dependencies:
  dio: ^4.0.6
```

- simple to use

```
import 'package:dio/dio.dart';
void getHttp() async {
  try {
    var response = await Dio().get('http://www.google.com');
    print(response);
  } catch (e) {
    print(e);
  }
}
```

Shared Preferences

What is SharedPreferences in Flutter?

Shared_preferences is a Flutter plugin that allows you to save data in a key-value format so you can easily retrieve it later.

Behind the scenes, it uses the aptly named SharedPreferences on Android and similar UserDefaults on iOS.

For our app, we used the plugin by saving the mode of our application (dark or light) and to make the user login just one time

The 'shared_preferences' package is very useful for providing a local persistent store for simple preference data.

This data is lost if the user uninstalls the app or clears the app data.

Each preference item requires its own String key to identify it.

There may come a time when you want to persist data in your Flutter app so you can reuse it later.

A common use case for this functionality is storing login credentials to be recalled the next time the user launches the app.

With SharedPreferences, you can configure your Flutter app to remember the data even after the user terminates their activity.

SharedPreferences can be used to store critical data such as passwords, tokens, and complex relational data.

SharedPreferences is what Android and iOS apps use to store simple data in an allocated space.

This data exists even when the app is shut down and starts up again; we can still retrieve the value as it was.

The data stored in SharedPreferences can be edited and deleted.

SharedPreferences stores the data in a key-value pair.

To use SharedPreferences in Flutter, a plugin called shared_preferences enable us to store data.

The plugin wraps NSUserDefaults on iOS and SharedPreferences on Android.

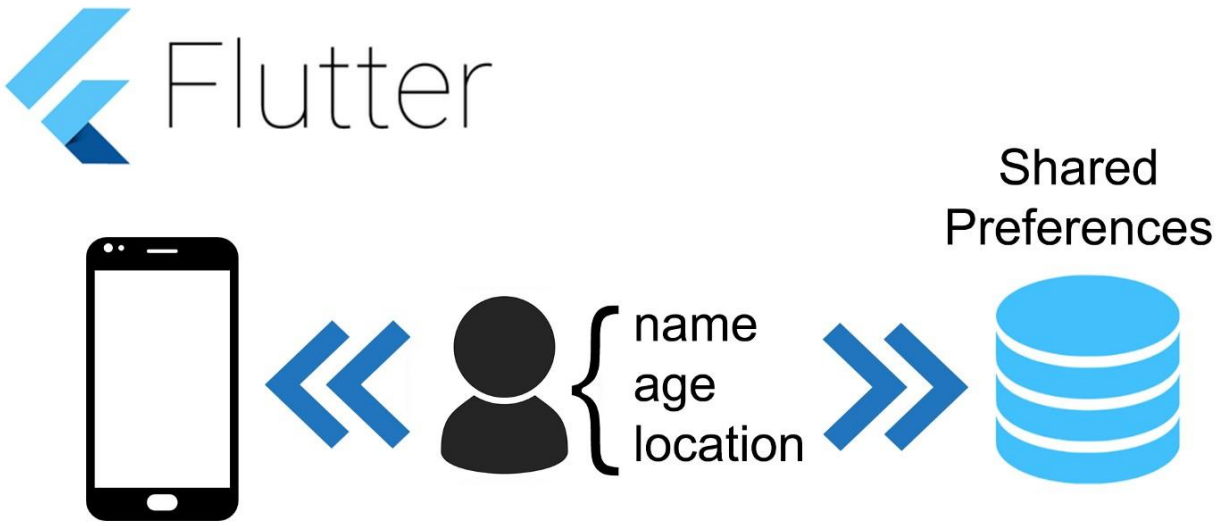


Figure 54: Shared preferences

Saving data

There are three primary ways to save data to your device:

1. Write formatted data, like JSON, to a file.

2. Use a library or plugin to write simple data to a shared location.

3. Use an SQLite database.

Writing data to a file is simple, but it requires you to handle reading and writing data in the correct format and order.

You can also use a library or plugin to write simple data to a shared location managed by the platform, like iOS and Android.

This is what we have done in our app.

For more complex data, you can save the information to a local database.

Why save small bits of data?

There are many reasons to save small bits of data.

For example, you could save the user ID when the user has logged in or if the user has logged in at all.

You could also save the onboarding state.

Note that this simple data saved to a shared location is lost when the user uninstalls the app

5.7.5. Snapshots from our UI Demo version

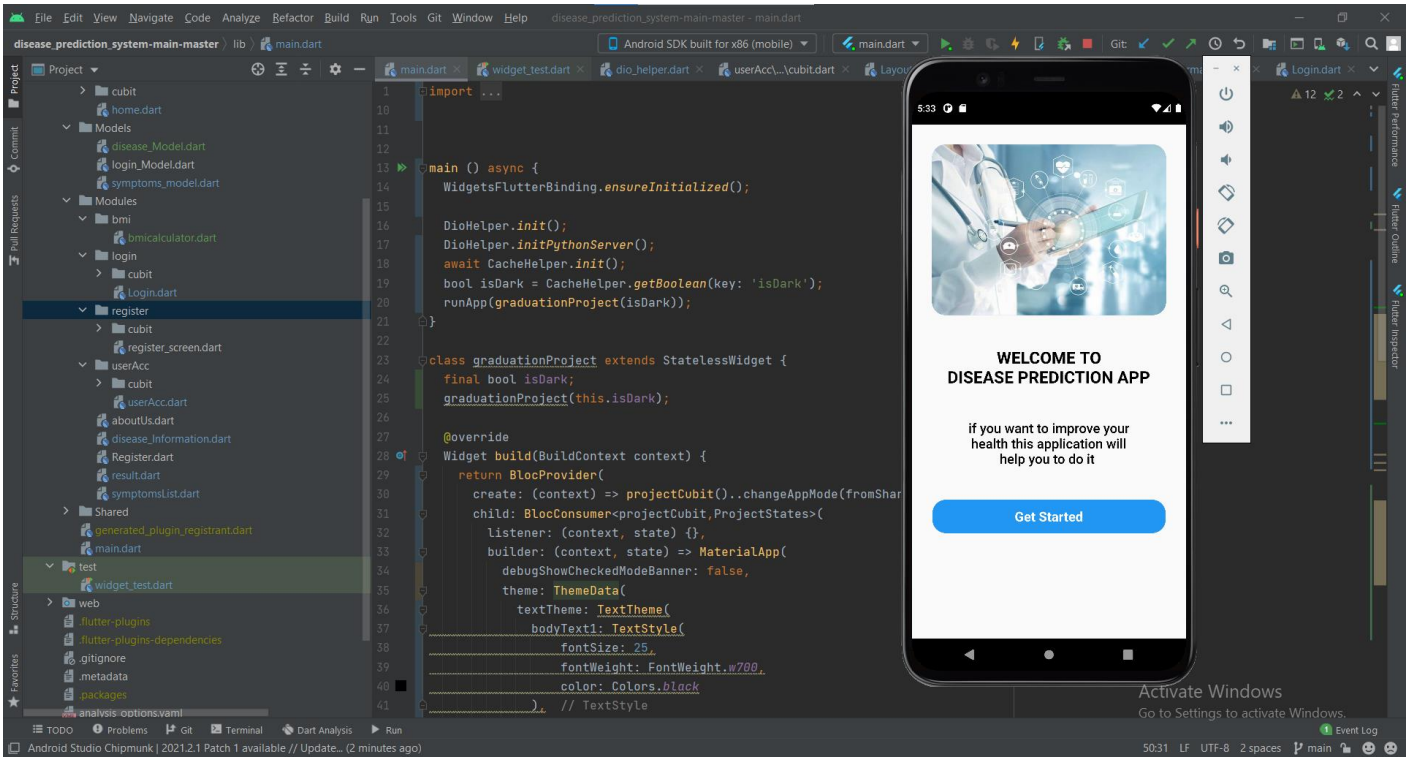


Figure 55: Welcome page

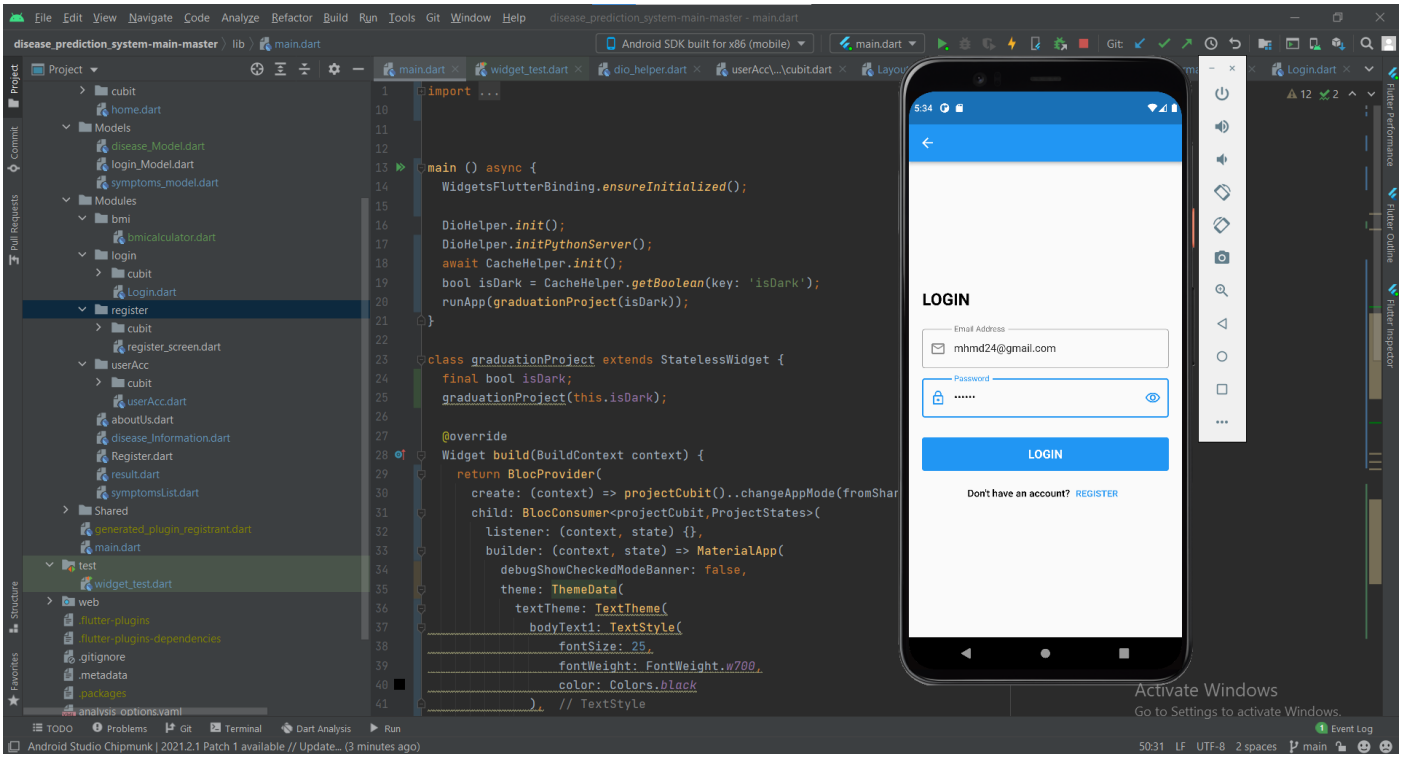


Figure 56: Login page

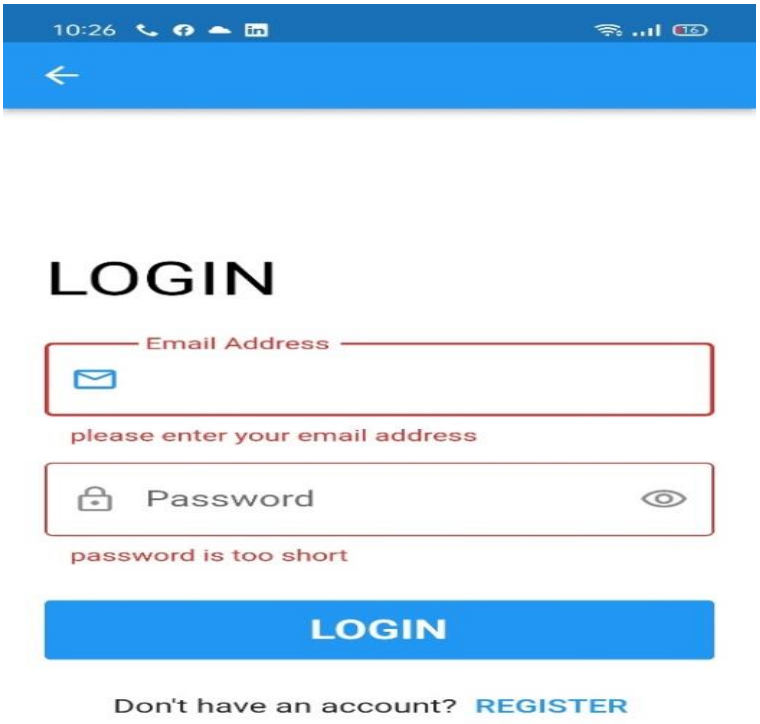


Figure 57: Login page with missing email and password

In this page the user must enter a valid email address and password to login.

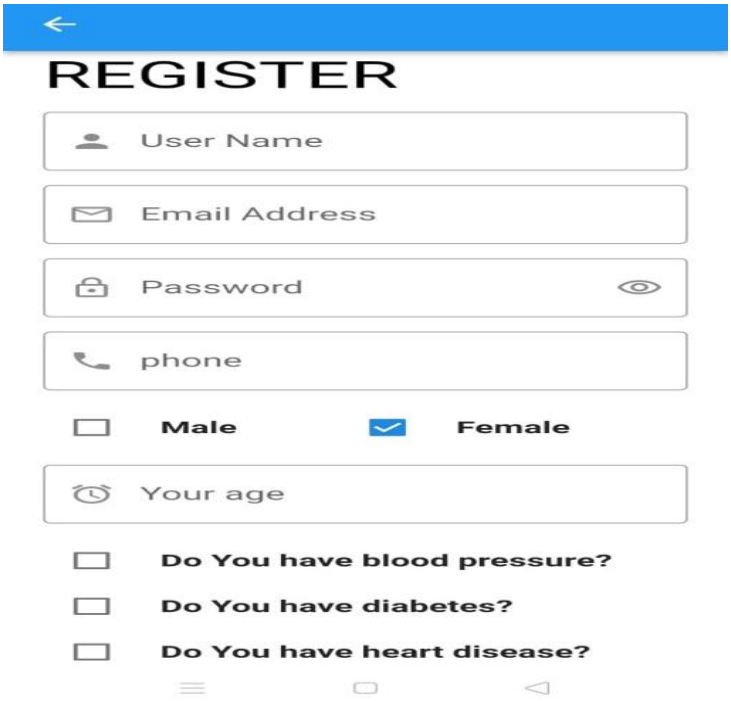


Figure 58: Register page

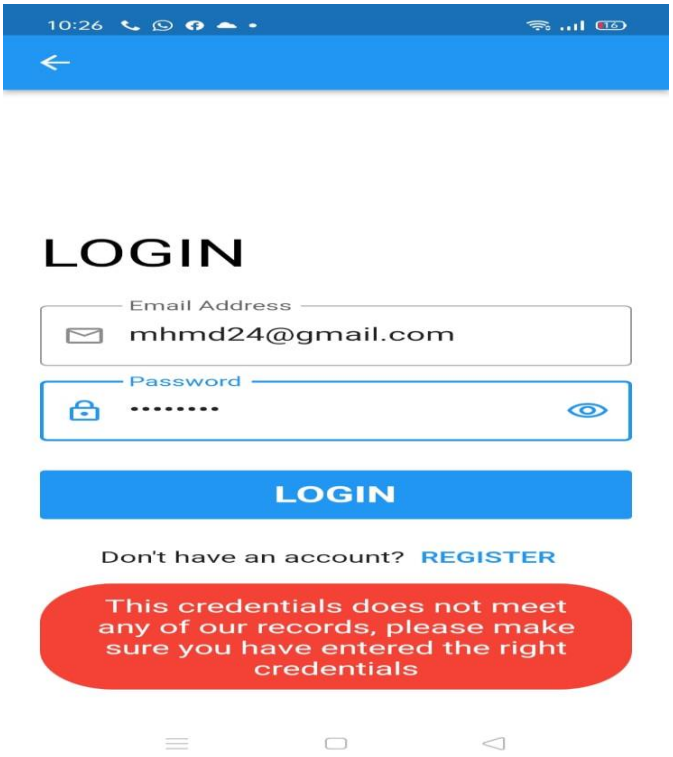


Figure 59: Login page with invalid email and password

If the user entered an unregistered credentials or used an invalid email or password an error pops up.

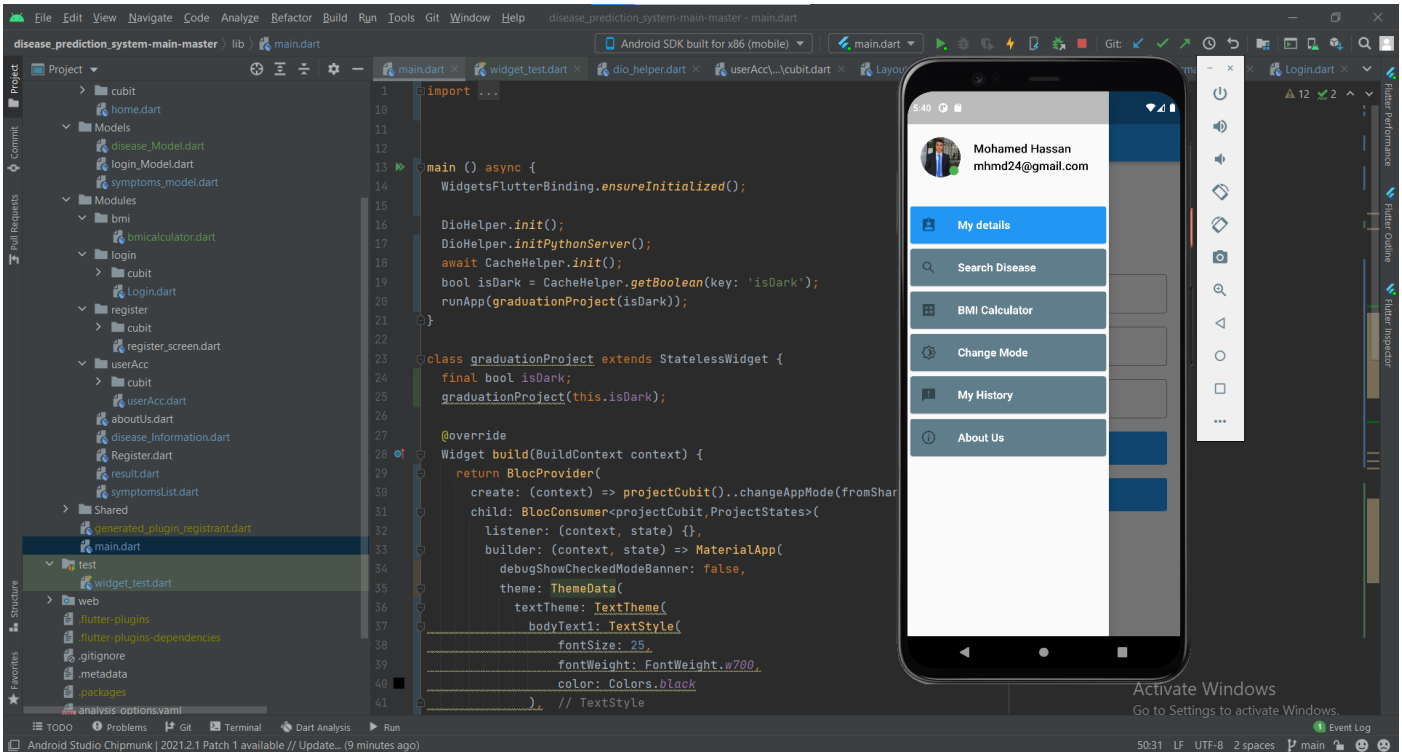


Figure 60: Home page

The first page that the user is directed to after a successful login process, it contains his profile details and all the app functional tools.

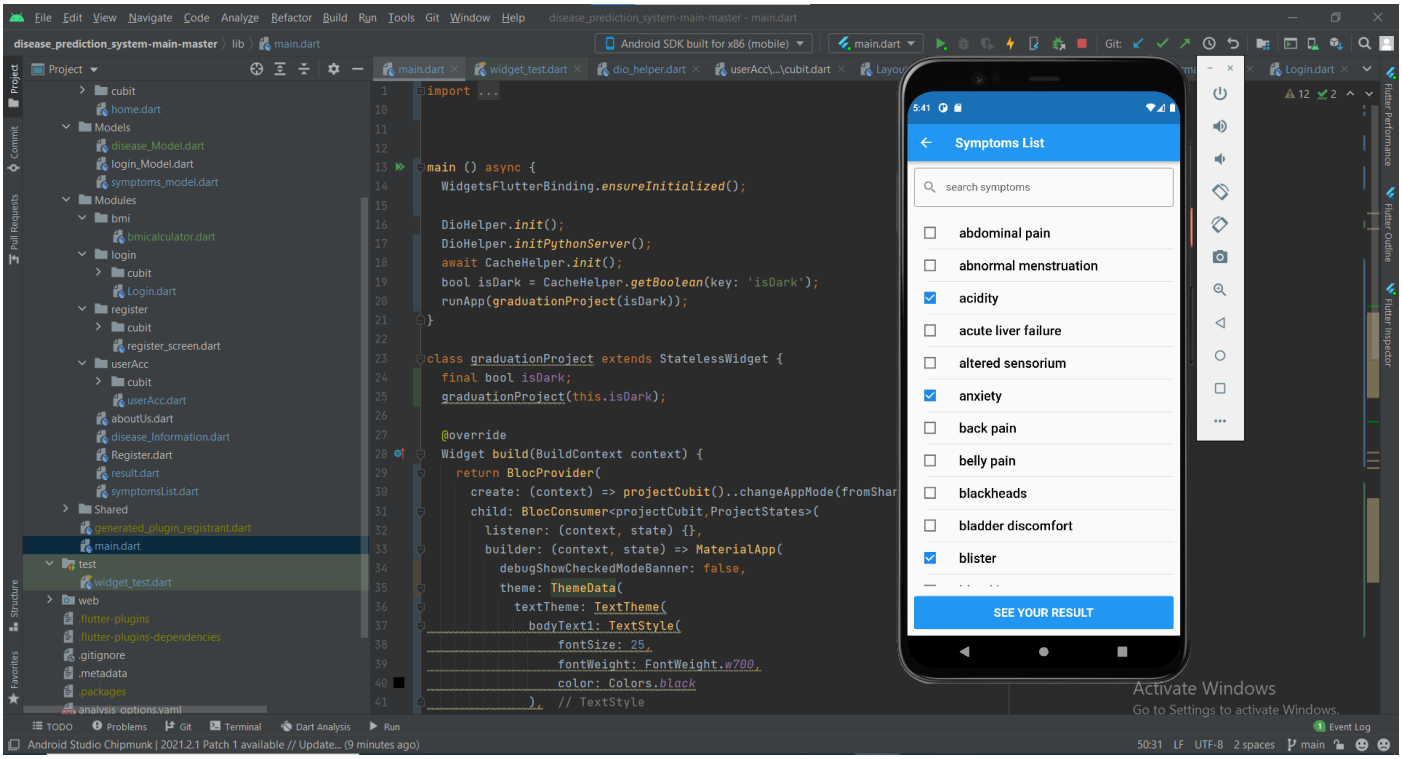


Figure 61: Entering symptoms page

The Screen Prediction Page where user chooses his symptoms to be redirected to our ML model to receive his diagnosed result.

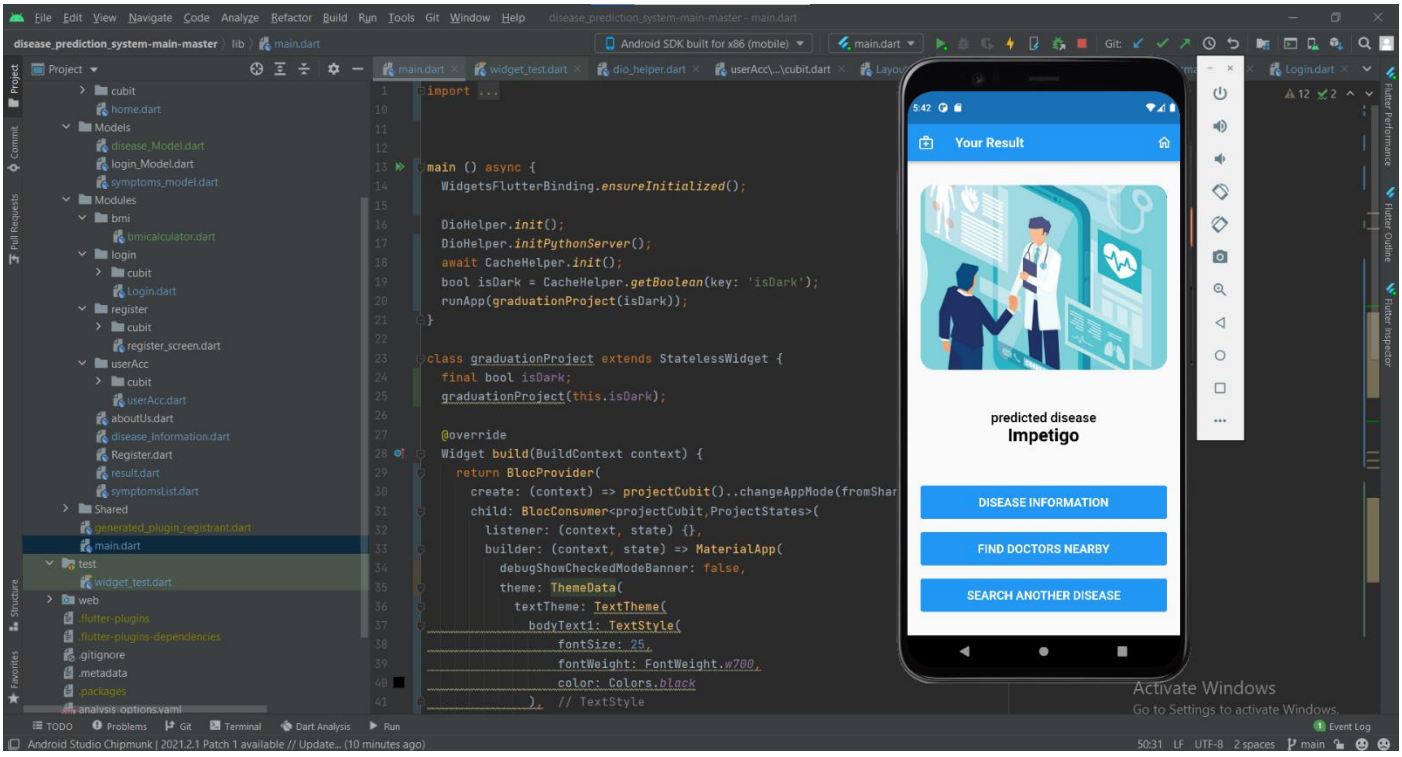


Figure 62: Results page

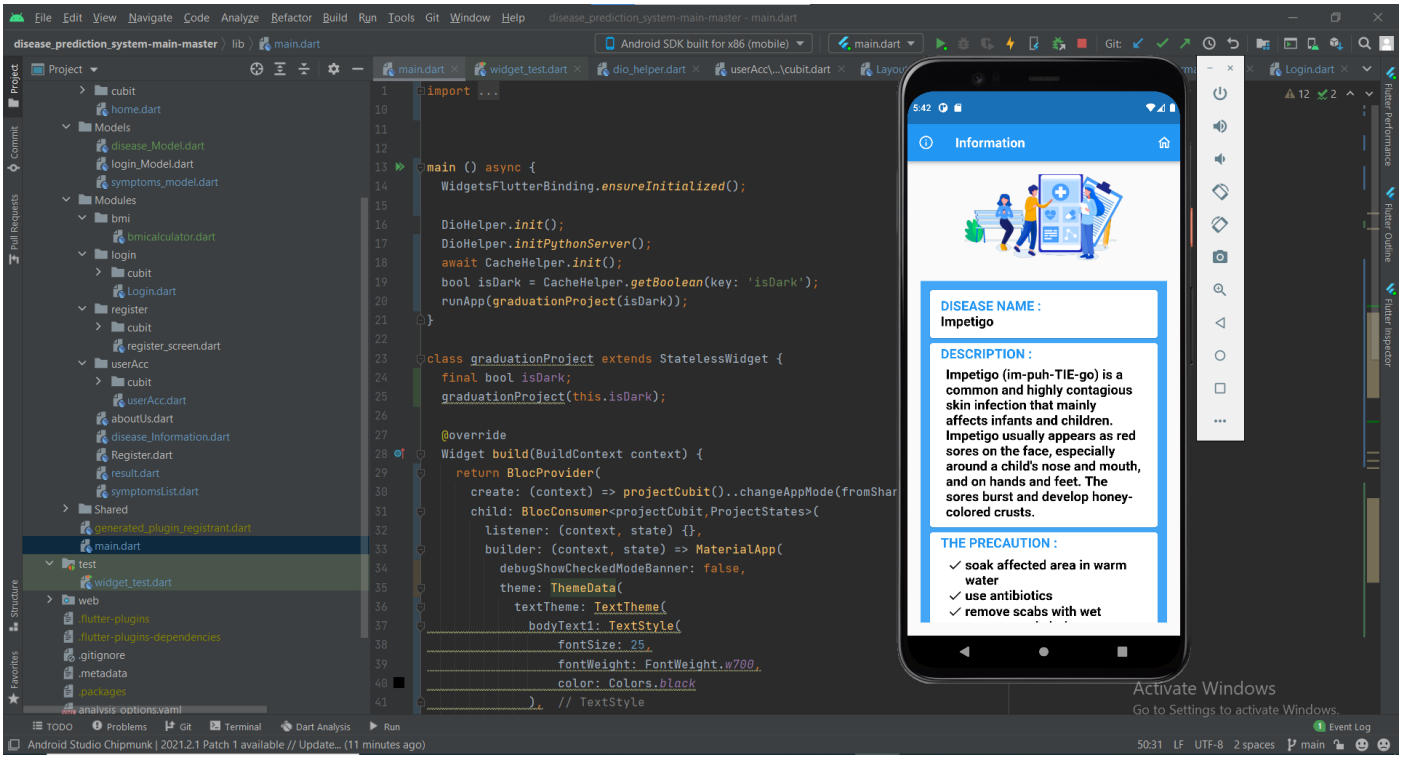


Figure 63: Disease information page

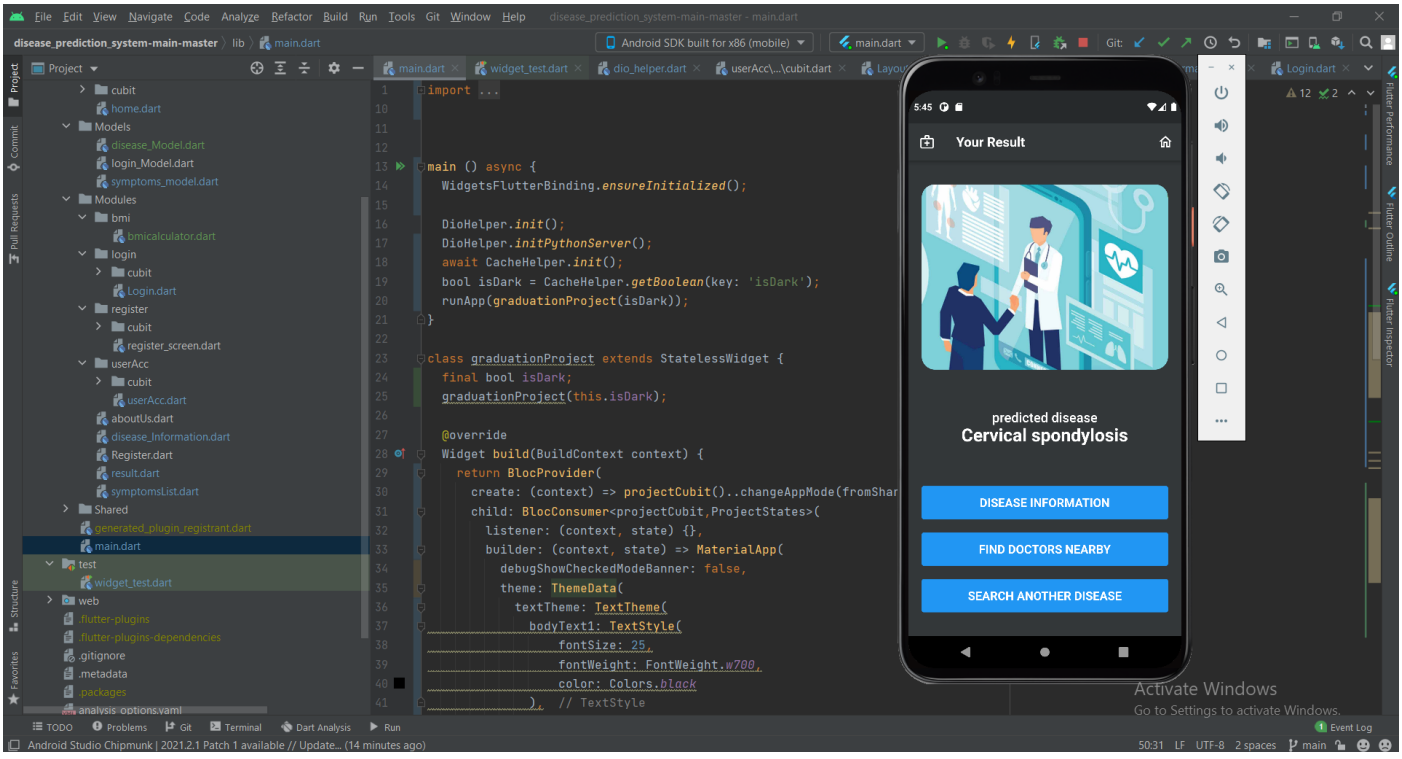


Figure 64: Dark mode

6.Testing

Here, we are going to apply black box testing to our system with various test cases for the main functions in the system.

Register:

For register an account:

Test case ID	Test scenario	Expected Results	Actual Results	Pass/Fail
TC1	All fields are blank	"fill all fields" error message should be displayed	As Expected.	**Pass**
TC2	All fields are filled with valid data	Registration done successfully	As Expected	**Pass**
TC3	Register with already existing email	"Email is already exist" error message should be displayed	As Expected	**Pass**
TC4	Wrong email format	"Incorrect email format" error message should be displayed	As Expected	**Pass**

Login:

For login to an account:

Test case ID	Test scenario	Expected Results	Actual Results	Pass/Fail
TC1	Email and Password are blank	"Enter email and password " error message should be displayed	As Expected.	**Pass**
TC2	Email is blank	"Enter email" error message should be displayed	As Expected.	**Pass**
TC3	Password is blank	"Enter password" error message should be displayed	As Expected.	**Pass**
TC4	Wrong email format	"Incorrect email format" error message should be displayed	As Expected.	**Pass**
TC5	Wrong email	"This email is invalid" error message should be displayed	As Expected.	**Pass**
TC6	Wrong password	"Password is incorrect" error message should be displayed	As Expected.	**Pass**
TC7	Valid email and password	Login successfully	As Expected.	**Pass**

disease diagnosis:

Entering symptoms:

Test case ID	Test scenario	Expected Results	Actual Results	Pass/Fail
TC1	No symptoms are selected	"choose symptoms " error message should be displayed	As Expected.	**Pass**
TC2	Symptoms are entered	User gets his disease	As Expected	**Pass**
TC3	Ambiguous symptoms are entered	"Please enter your actual symptoms" error message should be displayed	As Expected	**Pass**

7.References

- Aggarwal, Charu et al. 2019. "How Can AI Automate End-to-End Data Science?" http://arxiv.org/abs/1910.14436.
- Chen, Min et al. 2017. "Disease Prediction by Machine Learning over Big Data from Healthcare Communities." *IEEE Access* 5: 8869–79.
- Farooqui, Md. Ehtisham, and Dr. Jameel Ahmad. 2020a. "A DETAILED REVIEW ON DISEASE PREDICTION MODELS THAT USES MACHINE LEARNING." *International Journal of Innovative Research in Computer Science & Technology* 8(4).
- ———. 2020b. "DISEASE PREDICTION SYSTEM USING SUPPORT VECTOR MACHINE AND MULTILINEAR REGRESSION." *International Journal of Innovative Research in Computer Science & Technology* 8(4).
- Keniya, Rinkal et al. *Disease Prediction from Various Symptoms Using Machine Learning*. https://ssrn.com/abstract=3661426.
- (n.d.). Retrieved from https://medium.com/analytics-vidhya/introduction-to-data-science-28deb32878e7
- (n.d.). Retrieved from https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
- (n.d.). Retrieved from https://pub.dev/
- (n.d.). Retrieved from https://en.wikipedia.org/wiki/API
- (n.d.). Retrieved from https://towardsdatascience.com/3-ways-to-deploy-machine-learning-models-in-production-cdba15b00e
- (n.d.). Retrieved from https://flask.palletsprojects.com/en/2.1.x/
- (n.d.). Retrieved from https://id.heroku.com/login
- (n.d.). Retrieved from https://www.anaconda.com/