



Hoang Nguyen

# DEVELOPING A MACHINE LEARNING- POWERED WEB APPLICATION TO ENHANCE HEART DISEASE PREDICTION AND DIAGNOSIS

Heart Disease Prediction Web Application

Technology and Communication

2023

## VAASAN AMMATTIKORKEAKOULU

## Tietotekniikka

**TIIVISTELMÄ**

Tekijä	Hoang Nguyen
Opinnäytetyön nimi	Koneoppimukseen Perustuvan Verkkosovelluksen Kehittäminen Sairauksien Ennustamisen ja Diagnosoinnin Parantamiseksi
Vuosi	2023
Kieli	Englanti
Sivumäärä	45
Ohjaaja	Johan Dams

---

Nykyään ihmiset elävät kiireisiä elämiä eivätkä ehdi tarkistaa terveyttään. Siksi vakavien sairauksien määrä on kasvanut viime aikoina, ja sydänsairaus on yksi niistä. Tutkielman tavoitteena on luoda koneoppimisen avulla toimiva verkkosovellus, joka ennustaa nopeasti sydänsairauden käyttäjän tiedon perusteella ja antaa asiakkaiden päättää, onko syytä mennä sairaalaan.

Sovelluksessa on ensinnäkin ydinkoneoppimismalli, joka rakennetaan, koulutetaan ja testataan annetuilla aineistoilla useiden kirjastojen, kuten pandas tai Tensorflow, avulla sydämen tilan diagnosoimiseksi. Toiseksi käytetään Django-Python-kehystä kehittämään palvelin, joka vastaanottaa pyynnöt, generoi ennusteen ja lähettää sitten tuloksen asiakkaille. Lopuksi MERN-pinon verkkosovellus on tärkeä osa, joka kerää käyttäjätiedot, pyytää diagnoosia Django-palvelimelta ja näyttää lopullisen tuloksen käyttäjän näytöllä.

Tutkimus paljastaa datan tärkeyden, mikä tarkoittaa, että paremmilla aineistoilla koulutus tuottaa tehokkaamman tulostusmallin ja päinvastoin. Lisäksi tutkielma osoittaa REST API:ien joustavuuden, jotka siirtävät dataa asiakkaiden ja

palvelinten välillä. Viimeisenä mutta ei vähäisimpänä, projekti myös demonstroi Redux-virtausten käsitteitä tilanhallinnassa front-end-puolella.

VAASA UNIVERSITY OF APPLIED SCIENCES

Information Technology

## SUMMARY

Author	Hoang Nguyen
Title	Developing a Machine Learning-powered Web Application to Enhance Heart Disease Prediction and Diagnosis
Year	2023
Language	English
Number of pages	45
Supervisor	Johan Dams

---

Modern people tend to live busy lives and do not have time to check their health. That is why the number of serious diseases has risen recently, and heart disease is one of them. The thesis aims to create a web application driven by machine learning to quickly predict heart disease based on user information and allow customers to decide whether to go to the hospital.

First, the application has a core machine-learning model, which was built, trained and tested with given datasets using several libraries, such as pandas or Tensorflow, to diagnose the status of the heart. Secondly, utilizing the Django - Python framework a server was developed to get requests, generate the prediction, and then send the result to clients. Lastly, a MERN stack web application was a crucial part that will collect user information, request the diagnosis from the Django server, and display the final result on the user's screen.

The research discovers the importance of data, which means training with a better dataset will produce a more efficient output model and vice versa. Moreover, the thesis proves the flexibility of REST APIs that transfer data

between clients and servers. Finally, the project also demonstrates the concepts of Redux flow on state management on the front-end side.

## CONTENTS

### ABSTRACT

1	INTRODUCTION .....	10
2	BACKGROUND AND PURPOSE OF THE PROJECT .....	11
2.1	Background – Cardiovascular Disease Facts .....	11
2.2	Purpose – Planning of Web Application Development .....	11
3	THEORICAL BACKGROUND .....	12
3.1	Framing Problem.....	12
3.2	Types of Machine Learning .....	12
3.3	Key Machine Learning Terminology .....	12
3.4	Linear Regression .....	13
4	APPROACH AND IMPLEMENTATION .....	14
4.1	Machine Learning Process .....	14
4.1.1	Defining a Problem.....	14
4.1.2	Constructing a Dataset.....	14
4.1.3	Transform Dataset .....	19
4.1.4	Train a Model .....	20
4.2	Django Server.....	24
4.2.1	Set up .....	25
4.2.2	Load The Saved Model.....	25
4.2.3	Handle Requests .....	26
4.2.4	Response .....	26
4.3	Node Server .....	26
4.3.1	Libraries Installation.....	27
4.3.2	Set Up.....	28
4.3.3	Front End Communication .....	28
4.3.4	Django Server Communication .....	30
4.4	Client Side .....	31

4.4.1	Libraries Installation .....	32
4.4.2	Set Up .....	33
4.4.3	Intro Page .....	35
4.4.4	Login Page .....	36
4.4.5	Dashboard .....	38
4.4.6	User Profile .....	41
5	OUTCOME OF THE PROJECT .....	43
5.1	Overall of The Project .....	43
5.2	Outcome .....	44
6	CONCLUSIONS AND DISCUSSION .....	46
6.1	Thesis Assessment .....	46
6.2	Continuation .....	46
6.3	Conclusion .....	47
	REFERENCES .....	

## LIST OF FIGURES AND TABLES

<b>Figure 1.</b> Data Frame Shape.	p.15
<b>Figure 2.</b> Data Frame Describe.	p.16
<b>Figure 3.</b> Correlation Heat Map.	p.16
<b>Figure 4.</b> Number of heart disease cases.	p.16
<b>Figure 5.</b> Number of heart disease cases of male and female.	p.17
<b>Figure 6.</b> Folder Structure.	p.17
<b>Figure 7.</b> Needed Libraries.	p.18
<b>Figure 8.</b> Load dataset, define type of features, and shuffle.	p.18
<b>Figure 9.</b> Construct dataset.	p.18
<b>Figure 10.</b> Construct dataset details.	p.18
<b>Figure 11.</b> Transform Features.	p.19
<b>Figure 12.</b> Transform Feature File.	p.19
<b>Figure 13.</b> Transform Data Frame.	p.20
<b>Figure 14.</b> Transform Data Frame Function.	p.20
<b>Figure 15.</b> Building a model.	p.20
<b>Figure 16.</b> Build Model Function.	p.21
<b>Figure 17.</b> Train and Save Model.	p.21
<b>Figure 18.</b> Train Model Function.	p.21
<b>Figure 19.</b> Testing Model.	p.22
<b>Figure 20.</b> Metrics Charts.	p.23
<b>Figure 21.</b> Saving Model.	p.24
<b>Figure 22.</b> Django Server System.	p.24
<b>Figure 23.</b> Django Folder Structure.	p.25
<b>Figure 24.</b> Reload Model.	p.25
<b>Figure 25.</b> Main Prediction Function.	p.26
<b>Figure 26.</b> Node Server System.	p.27
<b>Figure 27.</b> Node Libraries Overview.	p.27
<b>Figure 28.</b> Node Folder Structure.	p.28
<b>Figure 29.</b> Node Routes.	p.28



<b>Figure 30.</b> User Route.	p.29
<b>Figure 31.</b> Prediction Route.	p.29
<b>Figure 32.</b> Node Axios Config.	p.30
<b>Figure 33.</b> Prediction Service.	p.31
<b>Figure 34.</b> Prediction API.	p.31
<b>Figure 35.</b> Front End System.	p.32
<b>Figure 36.</b> Front End Libraries.	p.32
<b>Figure 37.</b> Folder Structure.	p.33
<b>Figure 38.</b> App Providers.	p.33
<b>Figure 39.</b> Firebase Config.	p.34
<b>Figure 40.</b> Axios Config.	p.35
<b>Figure 41.</b> Intro Page 1.	p.35
<b>Figure 42.</b> Intro Page 2.	p.36
<b>Figure 43.</b> Login Page.	p.36
<b>Figure 44.</b> Email Modal.	p.37
<b>Figure 45.</b> Sign In Workflow.	p.37
<b>Figure 46.</b> Dashboard 1.	p.38
<b>Figure 47.</b> Prediction Details.	p.38
<b>Figure 48.</b> Dashboard 2.	p.39
<b>Figure 49.</b> Dashboard Calling API.	p.39
<b>Figure 50.</b> Input Form.	p.40
<b>Figure 51.</b> Input Form Submit.	p.40
<b>Figure 52.</b> Results of The Prediction.	p.41
<b>Figure 53.</b> Profile Options.	p.41
<b>Figure 54.</b> User Profile.	p.42
<b>Figure 55.</b> Overall Project.	p.43
<b>Table 1.</b> Dataset Description.	p.15

## 1 INTRODUCTION

Cardiovascular disease is one of the leading causes of death worldwide, and disastrous results can often be avoided with early identification. Accurate diagnosis and prediction of heart disease could be enhanced by machine learning.

Scientists have recently created predicted models for cardiac disease using machine learning algorithms. However, because these models are difficult for patients and healthcare professionals to access, they are not yet commonly used in clinical settings.

By creating a web application that makes better use of machine learning methods to improve heart disease detection and prediction, we want to close this gap in our thesis. Users of the program will have access to a user-friendly interface where they can input their information, which a machine learning algorithm will evaluate to determine their risk of getting heart disease.

This thesis aims to extend the accessibility of predictive models to healthcare, especially heart disease, which means that the targeted people are everyone. It does not matter who users are; they can access the product to get benefits. However, patients and doctors are good examples for the thesis because they often demand more to predict heart disease.

## **2 BACKGROUND AND PURPOSE OF THE PROJECT**

### **2.1 Background – Cardiovascular Disease Facts**

Heart disease and other cardiovascular disorders are among the most common causes of morbidity and mortality globally.

- Cardiovascular diseases (CVDs) are the leading cause of death globally.
- An estimated 17.9 million people died from CVDs in 2019, representing 32% of all global deaths. Of these deaths, 85% were due to heart attack and stroke.
- Of the 17 million premature deaths (under 70) due to noncommunicable diseases in 2019, 38% were caused by CVDs.
- Most cardiovascular diseases can be prevented by addressing behavioral risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity, and harmful use of alcohol.

(WHO, 2021, online)

### **2.2 Purpose – Planning of Web Application Development**

Creating a web application powered by machine learning has the potential to dramatically enhance the detection and diagnosis of cardiac disease, improving patient outcomes and contributing to a healthier population.

The application utilizes MERN (MongoDB, Express, React, Node.js) as a core stack and TensorFlow. Moreover, the trained machine learning model will be embedded in the Django server, which means that this Python server will handle requests as well as responses from the Node server.

### **3 THEORICAL BACKGROUND**

#### **3.1 Framing Problem**

Problem framing is the process of analyzing a problem to isolate the individual elements that need to be addressed to solve it. Problem framing helps determine the technical feasibility of a project and provides a clear set of goals and success criteria. When considering an ML solution, effective problem framing can determine whether your product ultimately succeeds (Google, nd., online).

Calculating the risk of heart disease based on the user's information is the ideal income of the heart disease prediction app. Moreover, the model's goal is to predict the potential of heart disease from input information.

#### **3.2 Types of Machine Learning**

Because the used dataset contains lots of data with correct answers, supervised learning models are the most suitable type to discover the relationships between the data elements that result in accurate responses, instead of unsupervised learning and reinforcement learning.

Furthermore, the heart disease prediction application has only two possible results: True or False. That is why classification models are a perfect option for this use case, especially binary classification.

#### **3.3 Key Machine Learning Terminology**

Supervised machine learning is a system that learns how to combine input to produce useful predictions on never-before-seen data (Google, nd., Online).

A typical dataset contains many instances, which are called examples in machine learning language. Examples break into two main categories: unlabeled and labeled ones.

### 3.4 Linear Regression

$$y' = b + w_1x_1 + w_2x_2 + w_3x_3 \quad (1)$$

where:

$y'$ : is the predicted label (the desired output)

$x_1, x_2, x_3$ : are features (or inputs)

$w_1, w_2, w_3$ : are weights of features (or slope)

$b$ : is the bias, sometimes referred to as  $w_0$

A model defines the relationship between features and label.

## 4 APPROACH AND IMPLEMENTATION

### 4.1 Machine Learning Process

Train a model with a large dataset to predict cardiovascular disease potentials.

#### 4.1.1 Defining a Problem

Based on pieces of information from users, the web application needs an algorithm to calculate or predict the risk of heart disease with high precision and less errors or mistakes.

The web application helps people check their health status at home using a laptop or perhaps even a smartphone without going to the hospital frequently. So, users will decide whether they need a doctor or not.

#### 4.1.2 Constructing a Dataset

Data is a critical point in every machine learning model, which means that a model is trained with better or higher-quality data will predict more precisely.

Firstly, collecting right data with these criteria:

- Size of data: large.
- Quality of data: reliability.

Because the web application needs basic information from users, such as age or total sleep time per day, the information is easy to answer at home without going to the hospital.

The project's dataset has 18 categories and about 320,000 examples. (Elsayed, 2022, Online)

Category	Data Type
Heart Disease	Yes/No

BMI (Body Mass Index)	Number e.g.: 12.2...
Smoking	Yes/No
Alcohol Drinking	Yes/No
Stroke	Yes/No
Physical Health	Number e.g.: 20
Mental Health	Number e.g.: 30
Difficult Walking	Yes/No
Sex	Male/Female
Age Category	Ranges e.g.: 70-74
Race	
Diabetic	Yes/No
Physical Activity	Yes/No
General Health	
Sleep Time (per day)	Number e.g.: 12
Asthma	Yes/No
Kidney Disease	Yes/No
Skin Cancer	Yes/No

**Table 1.** Dataset Description.

More details about the dataset description are given in Figures 1-5:

```

24 # ===== || DATASET - DATAFRAME || =====
25 heart_csv_path = './data/heart_2020_cleaned.csv'
26 df = pd.read_csv(heart_csv_path)
27 # Shuffle the examples
28 # df = df.reindex(np.random.permutation(df.index))
29 print(df.shape)
30
31
32 # ===== || DATA PREPARATION || =====

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** COMMENTS

PS C:\Users\HOANG\Downloads\CS\Projects\Heart\_Disease> python index.py  
(319795, 18)

PS C:\Users\HOANG\Downloads\CS\Projects\Heart\_Disease>

**Figure 1.** Data Frame Shape.

```

24 # =====|| DATASET - DATAFRAME || =====
25 heart_csv_path = './data/heart_2020_cleaned.csv'
26 df = pd.read_csv(heart_csv_path)
27 # Shuffle the examples
28 # df = df.reindex(np.random.permutation(df.index))
29 print(df.describe())
30
31
32 # =====|| DATA PREPARATION || =====

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** COMMENTS

PS C:\Users\HOANG\Downloads\CS\Projects\Heart\_Disease> python index.py

	BMI	PhysicalHealth	MentalHealth	SleepTime
count	319795.0	319795.0	319795.0	319795.0
mean	28.3	3.4	3.9	7.1
std	6.4	8.0	8.0	1.4
min	12.0	0.0	0.0	1.0
25%	24.0	0.0	0.0	6.0
50%	27.3	0.0	0.0	7.0
75%	31.4	2.0	3.0	8.0
max	94.8	30.0	30.0	24.0

PS C:\Users\HOANG\Downloads\CS\Projects\Heart\_Disease>

Figure 2. Data Frame Describe.

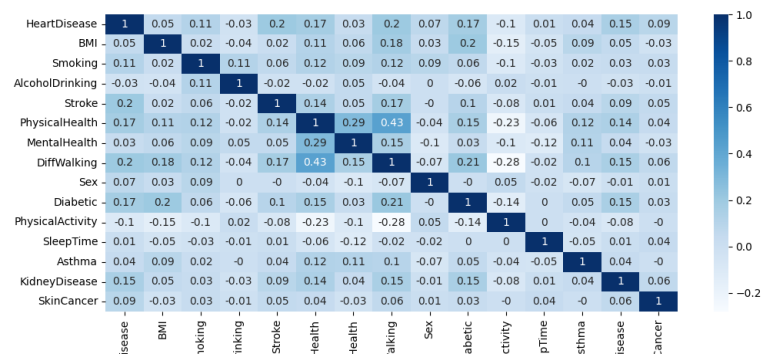


Figure 3. Correlation Heat Map.

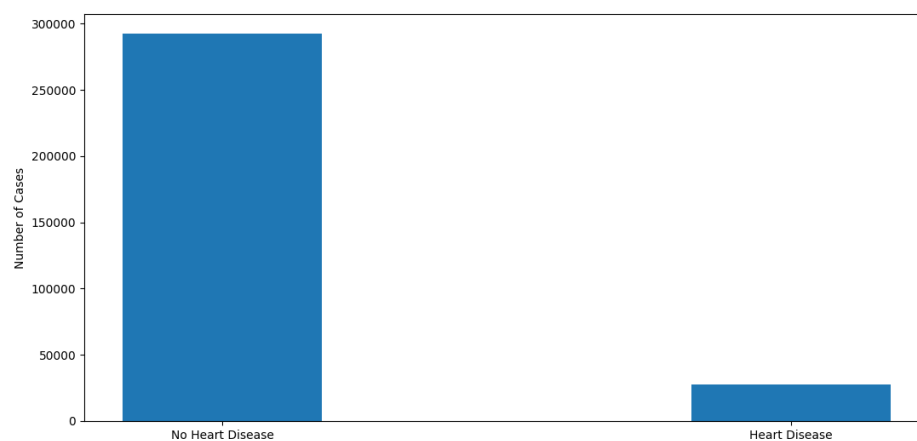
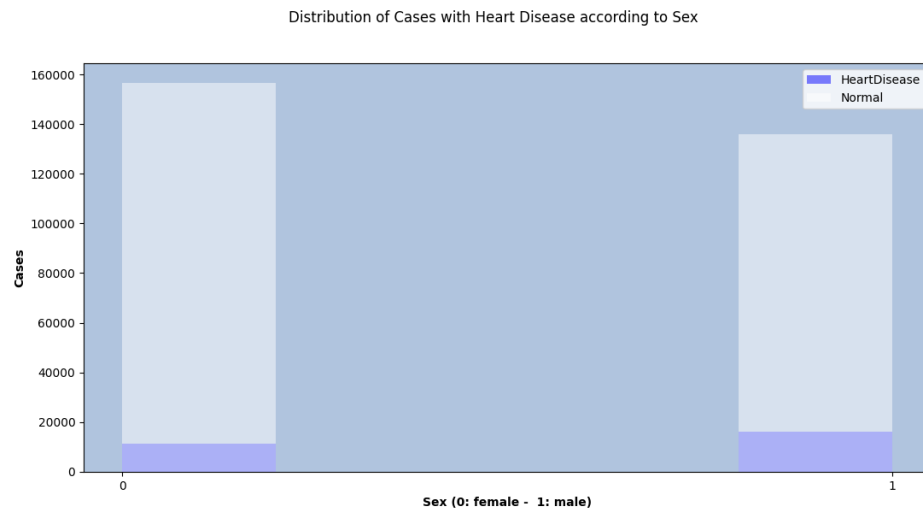


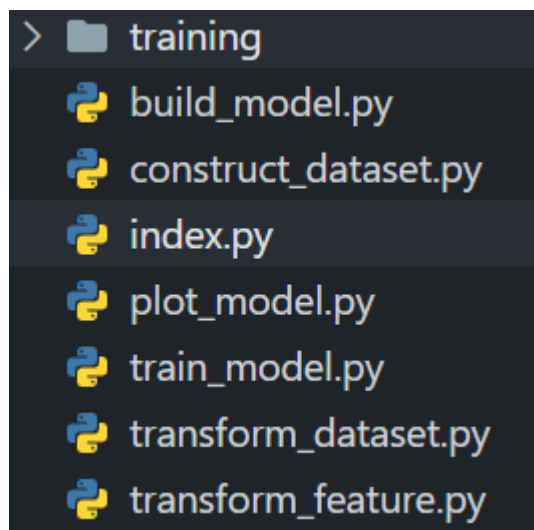
Figure 4. Number of heart disease cases.





**Figure 5.** Number of heart disease cases of male and female.

Moreover, the training process has a simple structure where index.js file is the main file and uses several support functions, such as build\_model.js or transform\_dataset.js files, to build, transform and train a model.



**Figure 6.** Folder Structure.

Secondly, load and transform dataset.

```

1  import numpy as np
2  import pandas as pd
3  import tensorflow as tf
4  import os

```

**Figure 7.** Needed Libraries.

```

57 # =====|| DATASET - DATAFRAME || =====
58
59 heart_csv_path = './data/heart_2020_cleaned.csv'
60 df = pd.read_csv(heart_csv_path)
61
62 # Shuffle the examples
63 df['HeartDisease'] = df['HeartDisease'].replace({'Yes': 1, 'No': 0})
64 df['HeartDisease'] = df['HeartDisease'].astype(float)
65 df = df.reindex(np.random.permutation(df.index))

```

**Figure 8.** Load dataset, define type of features, and shuffle.

According to Figure 8 above, the dataset is loaded using the pandas library, then changed the type of “Heart Disease” category to Float (1 or 0), and shuffle the whole dataset.

Finally, the loaded data frame is constructed.

```

73 # 2) Construct dataset
74 train_df, val_df, test_df = construct_dataset(df, RANDOM_SEED)

```

**Figure 9.** Construct dataset.

Passing the data frame as an argument (df) into the construct\_dataset function.

```

ML > ML_model > construct_dataset.py > ...
1  from sklearn.model_selection import train_test_split
2
3
4  def construct_dataset(df, RANDOM_SEED):
5      # Split into training and testing sets
6      train_df, test_df = train_test_split(
7          df,
8          test_size=0.15,
9          random_state=RANDOM_SEED
10     )
11
12     # Split the training set into training and validation sets
13     train_df, val_df = train_test_split(
14         train_df, test_size=0.18, random_state=RANDOM_SEED)
15
16     return train_df, val_df, test_df

```

**Figure 10.** Construct dataset details.

The `train_test_split` function from the `sklearn` library is used to split a monolith dataset into smaller pieces of data: train, validation, and test. By training with many different datasets, a model will avoid overfitting in the future. That is the critical reason why data construction is crucial in machine learning.

### 4.1.3 Transform Dataset

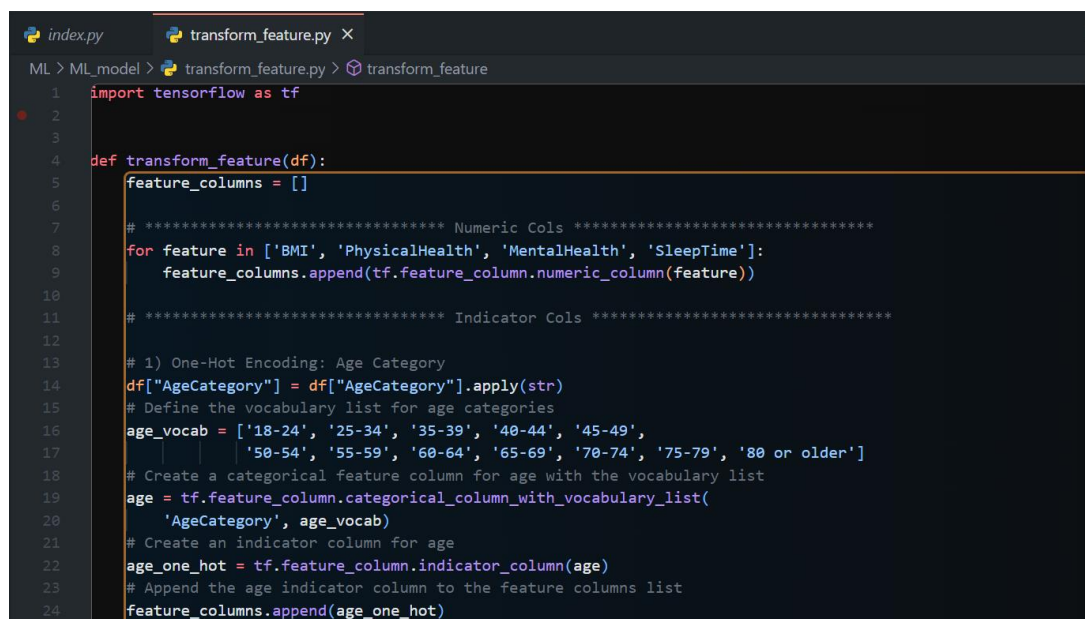
Firstly, feature engineering the dataset, which means the raw data will be selected, manipulated, and transformed into new features before training a model.

```

65 df = df.reindex(np.random.permutation(df.index))
66
67
68 # ===== || DATA PREPARATION || =====
69
70 # 1) Transform dataset
71 feature_columns = transform_feature(df)

```

Figure 11. Transform Features.



```

index.py  transform_feature.py X
ML > ML_model > transform_feature.py > transform_feature
1 import tensorflow as tf
2
3
4 def transform_feature(df):
5     feature_columns = []
6
7     # ***** Numeric Cols *****
8     for feature in ['BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime']:
9         feature_columns.append(tf.feature_column.numeric_column(feature))
10
11    # ***** Indicator Cols *****
12
13    # 1) One-Hot Encoding: Age Category
14    df["AgeCategory"] = df["AgeCategory"].apply(str)
15    # Define the vocabulary list for age categories
16    age_vocab = ['18-24', '25-34', '35-39', '40-44', '45-49',
17                '50-54', '55-59', '60-64', '65-69', '70-74', '75-79', '80 or older']
18    # Create a categorical feature column for age with the vocabulary list
19    age = tf.feature_column.categorical_column_with_vocabulary_list(
20        'AgeCategory', age_vocab)
21    # Create an indicator column for age
22    age_one_hot = tf.feature_column.indicator_column(age)
23    # Append the age indicator column to the feature columns list
24    feature_columns.append(age_one_hot)

```

Figure 12. Transform Feature File.

There are two types of feature transformation: converting the type of columns directly and one-hot encoding. After the transformation process is done, the new features will be appended and returned.

The next step is transforming those constructed data frames (train, validation, and test) into Tensor type because their current data type is pandas.

```
76 # Dataset: Pandas to Tensorflow
77 train_ds = transform_dataset(train_df)
78 val_ds = transform_dataset(val_df)
79 test_ds = transform_dataset(test_df)
```

**Figure 13.** Transform Data Frame.

```
1 import tensorflow as tf
2
3
4 def transform_dataset(dataframe, batch_size=32):
5     dataframe = dataframe.copy()
6     labels = dataframe.pop('HeartDisease')
7     return tf.data.Dataset.from_tensor_slices((dict(dataframe), labels)) \
8         .shuffle(buffer_size=len(dataframe)) \
9         .batch(batch_size)
```

**Figure 14.** Transform Data Frame Function.

#### 4.1.4 Train a Model

Firstly, building a model is a must before the training process starts.

```
82 # ===== || MODEL || =====
83
84 # 1) Build model
85 # model = build_model(learning_rate, feature_columns, METRICS)
86 model = build_model_without_learning_rate(feature_columns, METRICS)
87
88 # 1.1) Create a callback that saves the model's weights
89 cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
90     save_weights_only=True,
91     verbose=1)
92
```

**Figure 15.** Building a model.

The new transformed features will be passed into the building function together with the METRICS variable. After that, a callback will be generated to save the model weights.

```

30 def build_model_without_learning_rate(feature_columns, metrics):
31     """Create and compile a simple classification model."""
32     # Most simple tf.keras models are sequential.
33     model = tf.keras.models.Sequential()
34
35     # Add the feature layer (the list of features and how they are represented)
36     # to the model.
37     model.add(tf.keras.layers.DenseFeatures(feature_columns))
38
39     # Funnel the regression value through a sigmoid function.
40     model.add(tf.keras.layers.Dense(units=128, activation='relu', input_shape=(17,)))
41     model.add(tf.keras.layers.Dropout(rate=0.2))
42     model.add(tf.keras.layers.Dense(units=128, activation='relu'))
43     model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
44
45
46     model.compile(optimizer='adam',
47                   loss='mean_squared_error', metrics=metrics)
48
49     return model

```

Figure 16. Build Model Function.

Secondly, the built model is trained with the transformed dataset. After the training process is done, the model checkpoint will be saved into a new file that will be used to test without rebuild or retrain model.

```

93 # 2) Train model
94 # epochs, hist = train_model(model, train_ds, val_ds, epochs, batch_size)
95 epochs, hist = train_model_without_learning_rate(
96     model, train_ds, val_ds, epochs, cp_callback)
97
98 # 2.1) Loads the weights/ trained info
99 model.load_weights(checkpoint_path)

```

Figure 17. Train and Save Model.

```

22 def train_model_without_learning_rate(model, train_ds, val_ds, epochs, checkpoint_callback):
23     history = model.fit(train_ds, validation_data=val_ds,
24                         epochs=epochs, use_multiprocessing=True, callbacks=[checkpoint_callback])
25
26     # The list of epochs is stored separately from the rest of history.
27     epochs = history.epoch
28
29     # Isolate the classification metric for each epoch.
30     hist = pd.DataFrame(history.history)
31
32     return epochs, hist

```

Figure 18. Train Model Function.

Thirdly, after training with the train and validation dataset, the model must be tested with the test dataset to ensure that it is not overfitting. This process will return a set of metrics, such as loss, precision, recall, and accuracy, which will demonstrate how the model prediction is accurate.

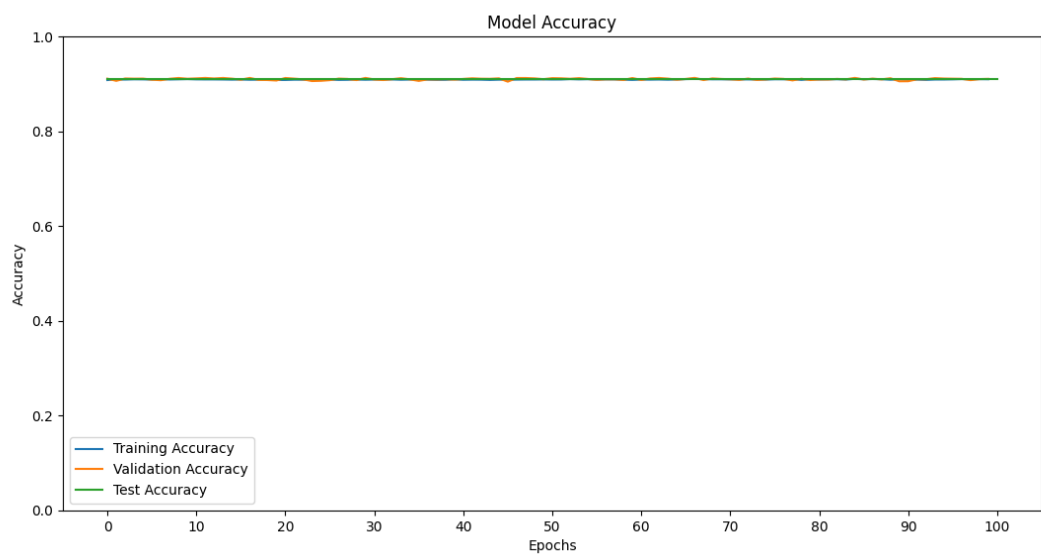
```

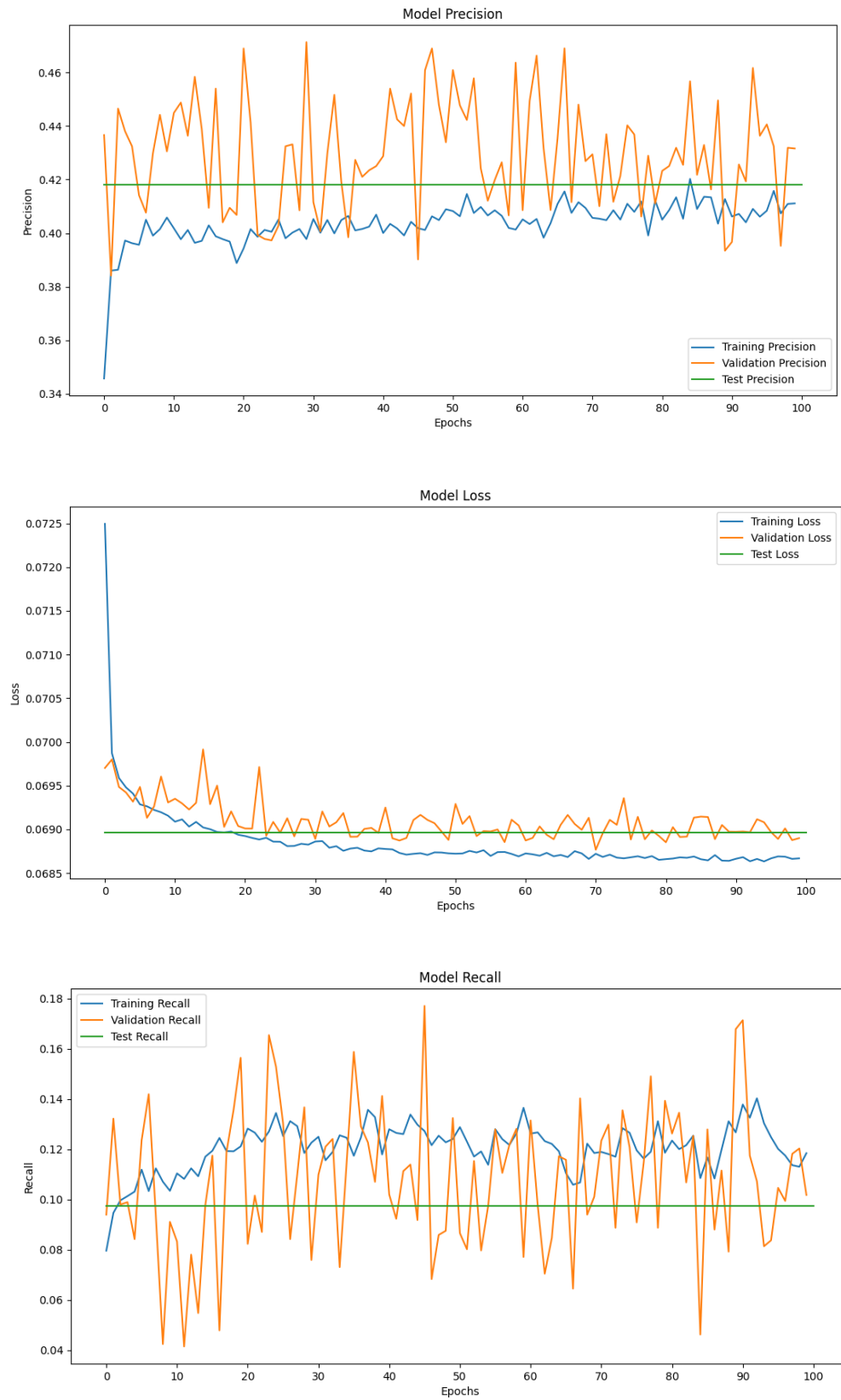
101 # 3) Test model
102 test_loss, test_acc, test_prec, test_rec = model.evaluate(test_ds)
103
104 # ===== || PLOT || =====
105
106 # Model Accuracy
107 108 plot_model_accuracy(epochs, hist, test_acc)
109
110 # Model Precision
111 111 plot_model_precision(epochs, hist, test_prec)
112
113 # Model Recall
114 114 plot_model_recall(epochs, hist, test_rec)
115
116 # Model Loss
117 117 plot_model_loss(epochs, hist, test_loss)

```

**Figure 19.** Testing Model.

Several charts are drawn by utilizing the returned metrics after evaluation process, which compares the predictions of 3 different processes: train, validation, and test.



**Figure 20. Metrics Charts.**

Finally, the tested model will be saved into a new folder that will be imported later in the Django server.

```

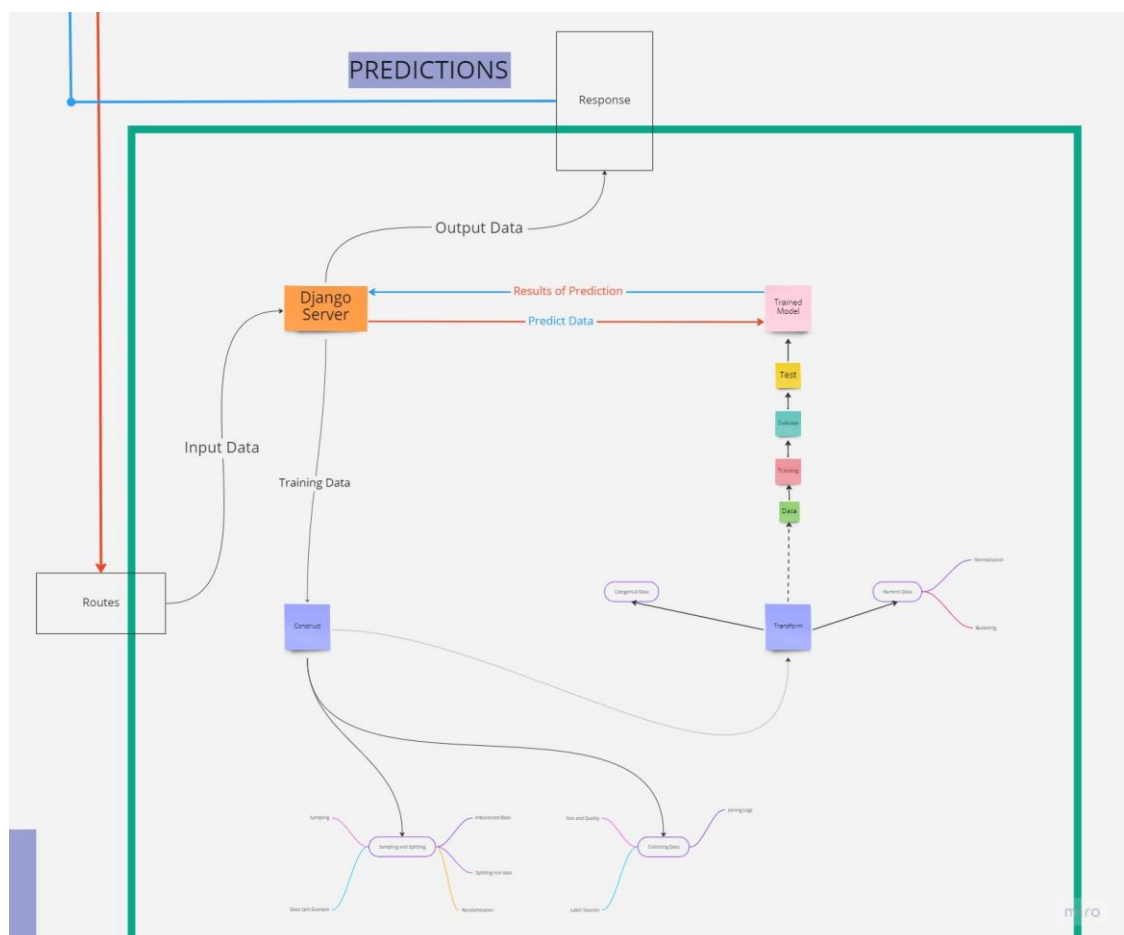
120 # ===== || SAVE || =====
121 model.save('model/hd_model')
122

```

**Figure 21.** Saving Model.

## 4.2 Django Server

Django is a high-level Python-based framework that follows MTV (Model Template Views) architecture pattern. This server is where the trained model will be used to predict heart disease based on data from the requests.

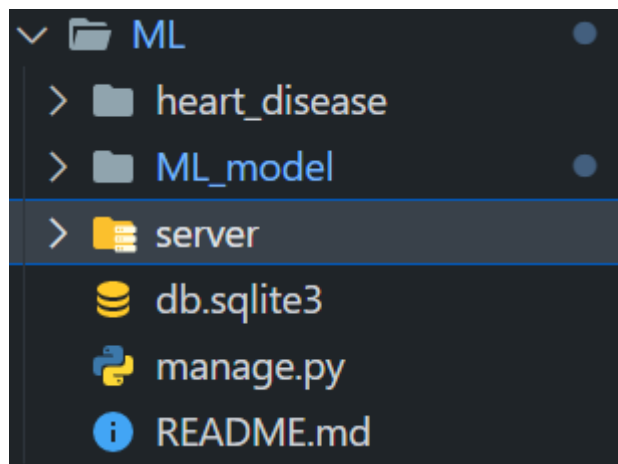


**Figure 22.** Django Server System.



### 4.2.1 Set up

Folder structure includes 3 main ones: heart\_disease (branch server where handles requests/responses related to heart disease prediction), ML\_model (contains code of training model), and server (the main server).



**Figure 23.** Django Folder Structure.

The manage.py file is used to run the whole server by the command.

```
py manage.py runserver
```

### 4.2.2 Load The Saved Model

Basically, the /ML\_model folder is the place that stores the trained model, so it just simply import/reload the saved model using Tensorflow library's function.

```
# Reload model
model = tf.keras.models.load_model('ML_model/model/hd_model')
```

**Figure 24.** Reload Model.

The main function will handle requests and responses called heartdisease\_predict which receives an argument as the coming request information.

```

12 def heartdisease_predict(request):
13     # Reload model
14     model = tf.keras.models.load_model('ML_model/model/hd_model')
15
16     if request.method == 'POST':
17         body_unicode = request.body.decode('utf-8')
18         body = json.loads(body_unicode)
19
20         # Define the column names
21         data = [{'BMI': body['BMI'], 'Smoking': body['Smoking'], 'AlcoholDrinking': body['AlcoholDrinking'],
22
23
24         # Create a Pandas DataFrame from the values and column names
25         df = pd.DataFrame(data)
26         # pandas to Tensor
27         input_tensor = tf.data.Dataset.from_tensor_slices((dict(df))) \...
28
29         # apply the trained machine learning model to the input data
30         predicted_output = model.predict(input_tensor)
31         flatten_output = tf.round(predicted_output).numpy().flatten()
32
33         # create a dictionary with the predicted output
34         # Convert number type Numpy - float32 to Python float
35         prediction_dict = {'predicted_output': float(flatten_output[0])}
36
37         # convert the predicted data to a JSON object
38         json_data = json.dumps(prediction_dict)
39
40         # send the predicted output to the Node.js server
41         response = Response(json_data)
42
43         return response

```

Figure 25. Main Prediction Function.

### 4.2.3 Handle Requests

Firstly, the function will check whether the request type is POST, then execute the rest of the main code, including transforming the data type of the body request to JSON and defining the input data type.

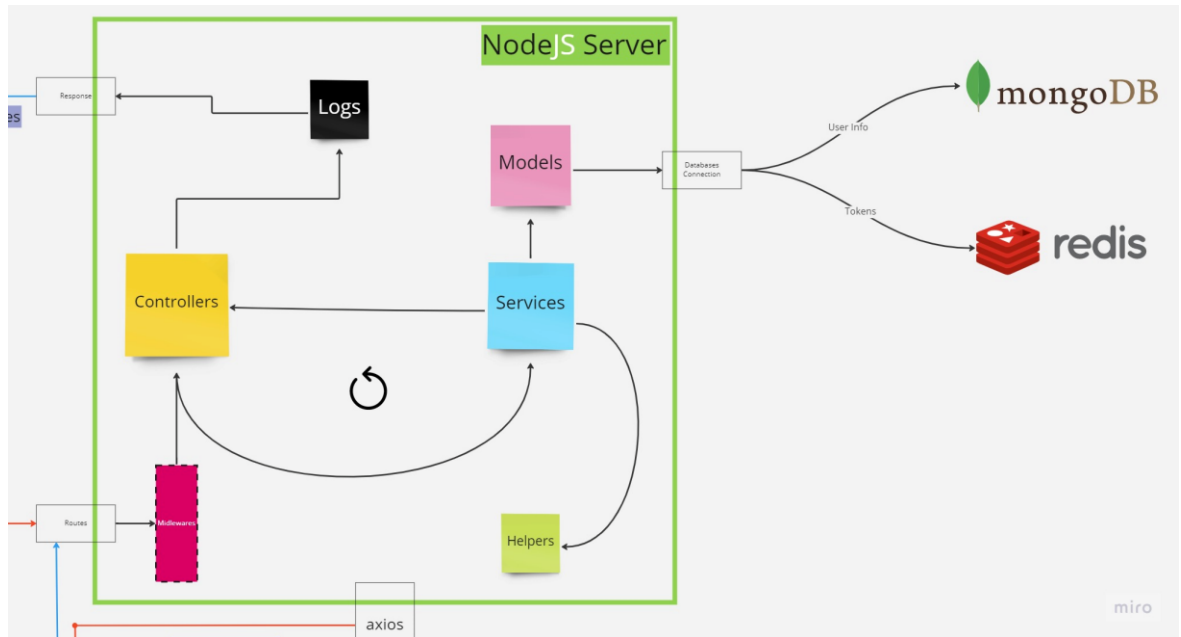
Secondly, according to Figure 24, the raw data will be transformed into pandas type, then tensor type because the model only accepts the tensor type as inputs.

### 4.2.4 Response

Finally, the model will predict and return the result: predicted\_ouput, which contains a string (e.g. 0.5899) that will be converted back to JSON type for responding.

## 4.3 Node Server

The server will communicate with both the Django Sever and Client Side. Receiving the user's data and request for the prediction from the machine learning server.



**Figure 26.** Node Server System.

The server system follows the MVC (Model Views Controller) architecture pattern, using axios to request to the Django Serve and writing several APIs (Application Programming Interface) for Client-Side communication. Moreover, the server will connect with the MongoDB database to manage user information, such as authentication and prediction history.

### 4.3.1 Libraries Installation

Install command: `yarn add + /library name/`

```
"express-generator": to generate an application and to get started with express
"axios": manage HTTP requests.
"nodemon": automatically restarting the node application when file changes are detected.
"mongoose": to work with MongoDB.
"cors": to enable CORS with varios options.
"dotenv": to load environment variables from a ".env" file into process.env.
"helmet": to secure the app by setting several HTTP headers.
"http-errors": to create HTTP errors for each request (if any).
```

**Figure 27.** Node Libraries Overview.

### 4.3.2 Set Up

Folder structure workflow

1. Folder explanation:

- bin: where we create server and listen to a port.
- src/api/v1 (v1 means version 1):
  - controllers: receive requests from app, handle requests and response back.
  - databases: initialize databases, such as MongoDB and Redis.
  - helpers: small functions - used to handle a couple of small tasks like: validation, error handler...
  - logs: contain event logs
  - middlewares: in this project, it will verify access/refresh token (not truly needed now)
  - models: define schema and generate form of data in MongoDB
  - routes: define routes of a whole app
  - services: is called by controllers to find out data from databases.
- app.js: main file - initialize project.

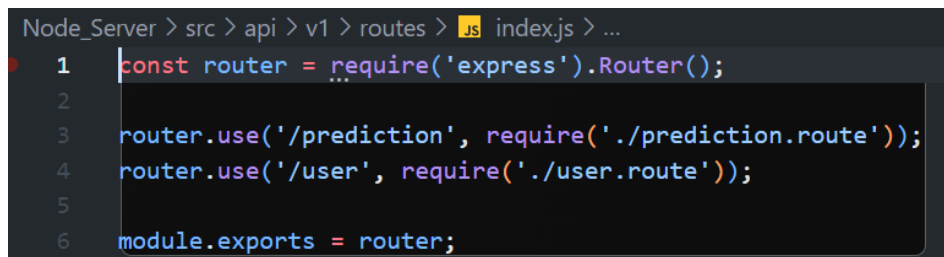
**Figure 28.** Node Folder Structure.

The version (v1) is needed because it will clarify the different versions if the application continues to grow up in the future.

The typical folder workflow follows the order: routes, middleware, controllers, services & helpers.

### 4.3.3 Front End Communication

The Front End will be able to request two different types of APIs, user APIs and prediction APIs.



```
Node_Server > src > api > v1 > routes > .js index.js > ...
1  const router = require('express').Router();
2
3  router.use('/prediction', require('./prediction.route'));
4  router.use('/user', require('./user.route'));
5
6  module.exports = router;
```

**Figure 29.** Node Routes.

User APIs will allow clients to register, sign in, sign out, and get information about their accounts.

```

Node_Server > src > api > v1 > routes > JS user.route.js > ...
1  const router = require('express').Router();
2
3  const { userController } = require('../controllers');
4  const { verifyToken } = require('../middlewares/tokenHandler');
5
6  //
7  router.post('/', userController.postUserInfo);
8  router.post('/checkToken', verifyToken, (req, res) => {
9      res.status(200).json('Authorized');
10 });
11 router.post('/signOut', verifyToken, userController.signOut);
12
13 router.get('/:uid', verifyToken, userController.getUser);
14 module.exports = router;

```

**Figure 30.** User Route.

Moreover, prediction APIs are entrances to get predictions and retrieve prediction history.

```

Node_Server > src > api > v1 > routes > JS prediction.route.js > ...
1  const router = require('express').Router();
2
3  const { predictionController } = require('../controllers');
4
5  router.post('/', predictionController.postPrediction);
6  router.get('/:uid', predictionController.getUserPrediction);
7
8  module.exports = router;

```

**Figure 31.** Prediction Route.

Each API will call a corresponding controller to process the request; then, the controller will use several functions in the services and helpers folder. If errors occur, the logs folder will write down these errors, which will help developers find out what happened.

Finally, the Node server will respond to the Client Side with corresponding data in the JSON type.

#### 4.3.4 Django Server Communication

Basically, the Node server uses Axios library to communicate with the Django Sever, which means that they are sending HTTP requests to each other.

Firstly, the server needs to customize how it will send and receive messages.

```
Node_Server > src > api > v1 > api > .js axiosClient.js > ...
1  const axios = require('axios'); 53.6k (gzipped: 19.6k)
2  const { BASE_URL } = require('./constants.axios');
3
4  // ===== AXIOS SETTING =====
5  const axiosClient = axios.create({
6    baseURL: BASE_URL,
7    headers: {
8      'Content-Type': 'application/json',
9    },
10 });
11
12 axiosClient.interceptors.response.use(
13   (response) => {
14     if (response && response.data) {
15       return response.data;
16     }
17     return response;
18   },
19   (error) => {
20     return error;
21   }
22 );
23
24
25 module.exports = axiosClient;
```

**Figure 32.** Node Axios Config.

This config allows the server to manage the data and errors by utilizing the interceptors of the Axios library.

Secondly, whenever the client-side requests to post a new prediction through a prediction API, the server will call the prediction controller, which allows the prediction service request to the Django Server.

```

7 // ===== PREDICTION SERVICE =====
8 const getPrediction = async (data) => {
9     const DATA_LENGTH = Number(process.env.DATA_LENGTH);
10    try {
11        if (Object.keys(data).length !== DATA_LENGTH) {
12            return false;
13        }
14        const res = await predictionApi.predict(data);
15        return res;
16    } catch (error) {
17        console.log(error);
18        throw new Error('Predict failed!');
19    }
20 };

```

**Figure 33.** Prediction Service.

```

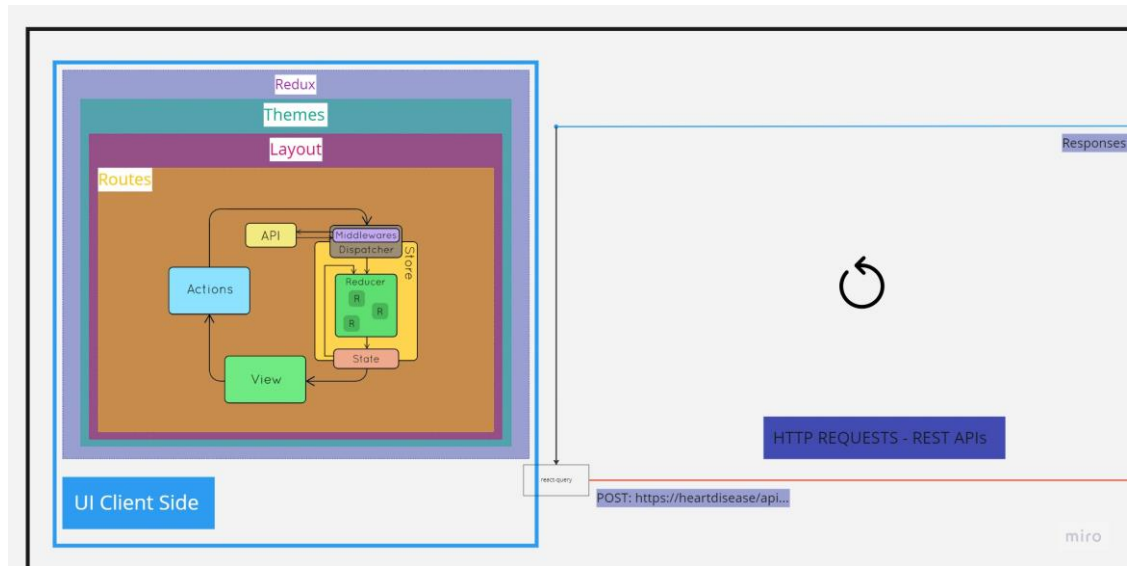
Node_Server > src > api > v1 > api > js prediction axios.js > ...
1 // Project import
2 const axiosClient = require('./axiosClient');
3 const { BASE_URL } = require('./constants.axios');
4
5 // ===== AXIOS SETTING =====
6 const predictionApi = {
7     predict: (data) => axiosClient.post(`${BASE_URL}/heart_disease`, data),
8 };
9
10 module.exports = predictionApi;

```

**Figure 34.** Prediction API.

In the service function above, it uses an Axios API to request to the Django Server with the user information and returned the response to the controller. This service also follows up the potential errors using try/catch method.

#### 4.4 Client Side



**Figure 35.** Front End System.

The Front-End system will manage the state following the redux/flux concepts and communicate with the Node Server using the Axios library.

#### 4.4.1 Libraries Installation

Installation command: `yarn add + /library name/`, and `yarn add -D + /library name/` for the dev dependencies.

```
"@mui/material": is a library of React UI components that implements Google's Material Design.
"axios": using to manage HTTP requests/responses.
"prop-types": type checking for React props.
"react": is a JavaScript library for creating user interfaces.
"react-router-dom": to manage router in a whole of app.
"yup": validate app's forms.
"react-hook-form": form management.
"styled-components": to customize components from MUI.

***** Dev Dependencies *****
"tailwindcss": styling mini route/components.
```

**Figure 36.** Front End Libraries.



## 4.4.2 Set Up

### 1. Folder Structure (./src/...):

- api: where we create axios client and apis.
- components: contains several dumb components.
- firebase: initialize firebase and authorization's config.
- layout: defines main/mini layout of this application.
- model: defines type of data.
- pages: contains pages of this webapp including: auth, dashboard and profile page.
- providers: state providers.
- routes: separate main/mini route of the web application.
- sections: form structures.
- store: where we initialize reducer store, actions, and constants.
- utils: store a couple of small functions like: authHandler, cut time string, etc.

**index.ts && App.ts: initialize client side.**

**Figure 37.** Folder Structure.

The web application will be divided into two main layouts, main and mini. Each page will follow its layout. For example: the main layout will be bond together with dashboard or check page.



```

Webapp > src > providers > Provider.tsx > ...
1  import { ReactElement } from 'react';  4.1k (gzipped: 1.8k)
2
3  // project import
4  import ScrollTop from './ScrollTop';
5  import UserProvider from './UserProvider';
6
7  // ===== || PROVIDER || =====
8
9  const Provider = ({ children }: { children: ReactElement }) => {
10     return (
11         <ScrollTop>
12             <UserProvider>{children}</UserProvider>
13         </ScrollTop>
14     );
15 };
16
17 export default Provider;
  
```

**Figure 38.** App Providers.

The providers will apply a couple of main properties, such as Scroll Top, and share the global state to the entire application.

Moreover, the project will use firebase authentication to manage the user sign in and sign out, so it needs to be config also.

```
Webapp > src > firebase > ts config.ts > ...
1 // Import the functions you need from the SDKs you need
2 import { initializeApp } from 'firebase/app'; 22.7k (gzipped: 7.3k)
3 import { getAnalytics } from 'firebase/analytics'; 40.1k (gzipped: 12.7k)
4 // TODO: Add SDKs for Firebase products that you want to use
5 // https://firebase.google.com/docs/web/setup#available-libraries
6
7 // Your web app's Firebase configuration
8 // For Firebase JS SDK v7.20.0 and later, measurementId is optional
9 const firebaseConfig = {
10   apiKey: 'AIzaSyA88CrIXph1JBLfHo2Ox3sOKz43l6sDoNY',
11   authDomain: 'heart-disease-prediction-543cc.firebaseio.com',
12   projectId: 'heart-disease-prediction-543cc',
13   storageBucket: 'heart-disease-prediction-543cc.appspot.com',
14   messagingSenderId: '642158468992',
15   appId: '1:642158468992:web:a7df200aba41892a20ab65',
16   measurementId: 'G-1CSV6SPNQ7',
17 };
18
19 // Initialize Firebase
20 const app = initializeApp(firebaseConfig);
21 const analytics = getAnalytics(app);
22
23 export { app };
```

**Figure 39.** Firebase Config.

Finally, the web application communicates with servers by HTTP requests, so Axios is a crucial part of the system. At this point, the Axios will embed the access token into the request headers, verifying the user identity in the back end. The setting is quite identical to the server one utilizing the interceptors. That is the reason why this configuration will be able to handle the errors and inform to the user also.

```

Webapp > src > api > axiosClient.ts > ...
1  import axios, { AxiosRequestConfig, AxiosResponse } from 'axios'; 55.1k (gzipped: 20.1k)
2
3  // Project import
4  import { BASE_URL } from '../constants/axios';
5
6  // helpers
7  const getToken = () => {
8    const accessToken = localStorage.getItem('accessToken');
9    return accessToken;
10 };
11
12 // =====|| MAIN AXIOS - CONFIG AXIOS ||===== //
13
14 const axiosClient = axios.create({
15   baseURL: BASE_URL,
16   headers: {
17     'Content-Type': 'application/json',
18   },
19 });
20
21 axiosClient.interceptors.request.use((config: AxiosRequestConfig | any) => {
22   return {
23     ...config,
24     headers: {
25       'Content-Type': 'application/json',
26       Authorization: `Bearer ${getToken()}`,
27     },
28   };
29 });

```

Figure 40. Axios Config.

#### 4.4.3 Intro Page



Figure 41. Intro Page 1.

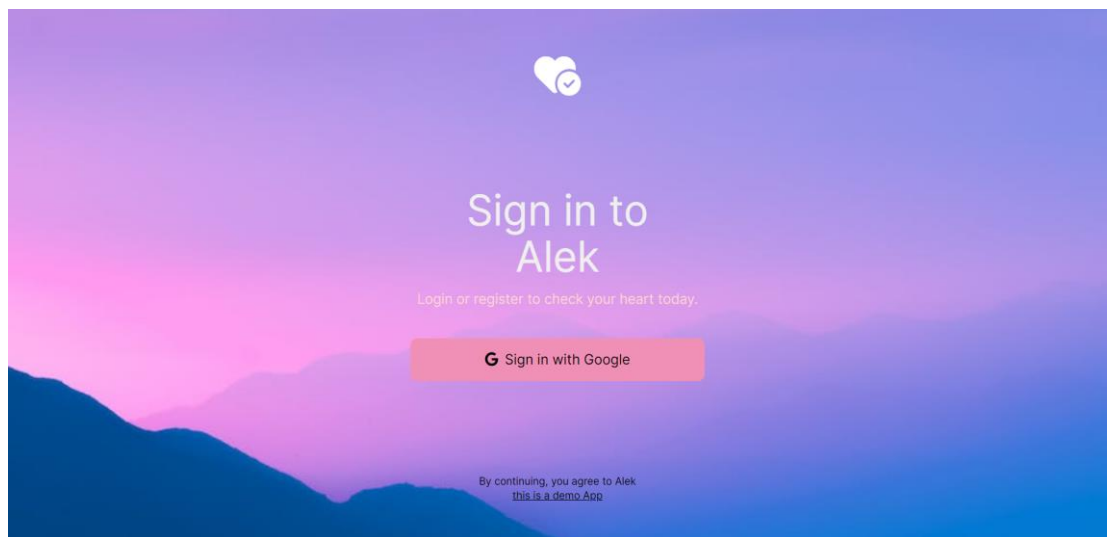


**Figure 42.** Intro Page 2.

The intro page (Codepen, nd. , Online) applies the parallax concept which is a beautiful scroll and it has two buttons:

- Sign In: will redirect to the auth page.
- Go Fullscreen: allows user to open the app in full screen mode.

#### 4.4.4 Login Page



**Figure 43.** Login Page.

By clicking on the “Sign in with Google” button, the page will open a modal or a dialog that allows the sign-in or registration with email credentials.

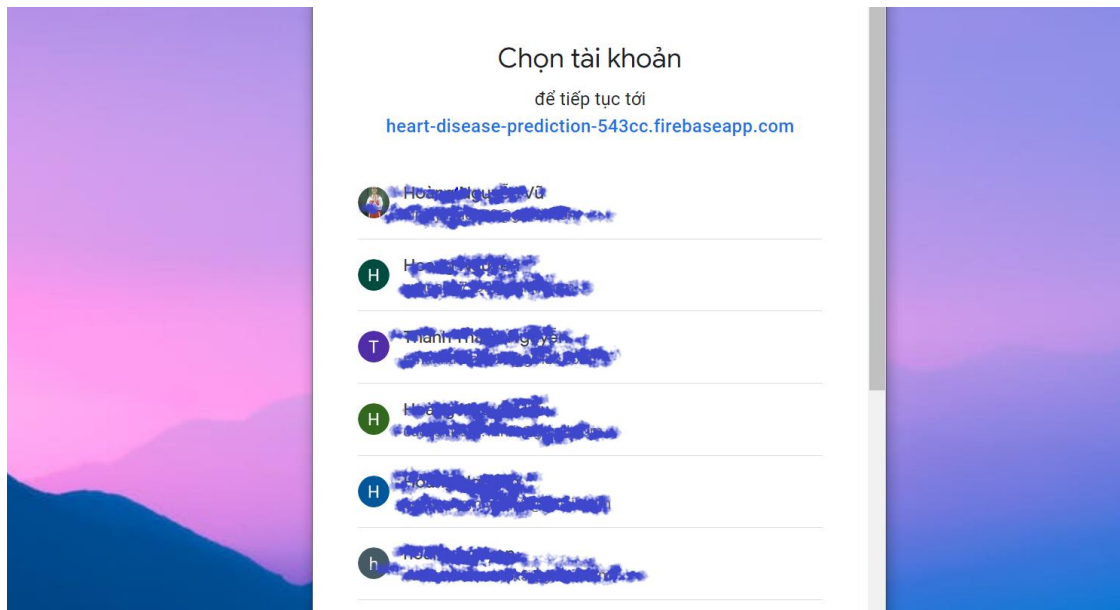


Figure 44. Email Modal.

```
Webapp > src > pages > auth > SignIn.tsx > ...
21 const SignIn = () => {
22   const navigate = useNavigate();
23
24   const handleGoogleSignIn = async () => {
25     const results: any = await signInGoogle();
26     if (!results.user) {
27       toast.error('Log in failed');
28     }
29     const userInfo: UserInfo = {
30       displayName: results.user.displayName,
31       email: results.user.email,
32       photoURL: results.user.photoURL,
33       uid: results.user.uid,
34       accessToken: results.user.stsTokenManager.accessToken,
35     };
36     await authApi.saveInfo(userInfo);
37     signInHandler(results.user.stsTokenManager.accessToken, results.user.uid, navigate);
38   };

```

Figure 45. Sign In Workflow.

Firstly, the application will call the Google Firebase authentication to verify that email. If it succeeds, the results will contain the user information, including name, photo URL, or tokens.

Secondly, the information is sent to the Node server through an API and the server will save it in a particular database.

Finally, the web application is navigated to the main layout which contains the dashboard or user information.

#### 4.4.5 Dashboard

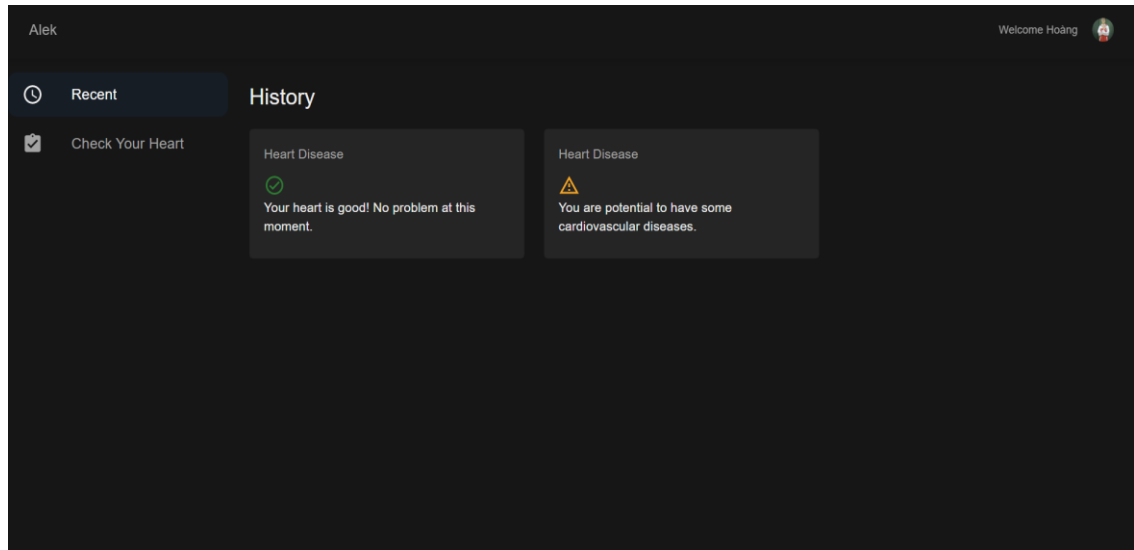


Figure 46. Dashboard 1.

The dashboard automatically opens the history of the user's predictions, if there are not any of them, the different view will be rendered. In addition, if the user clicks on each item of history list, it will display all related details of the prediction.

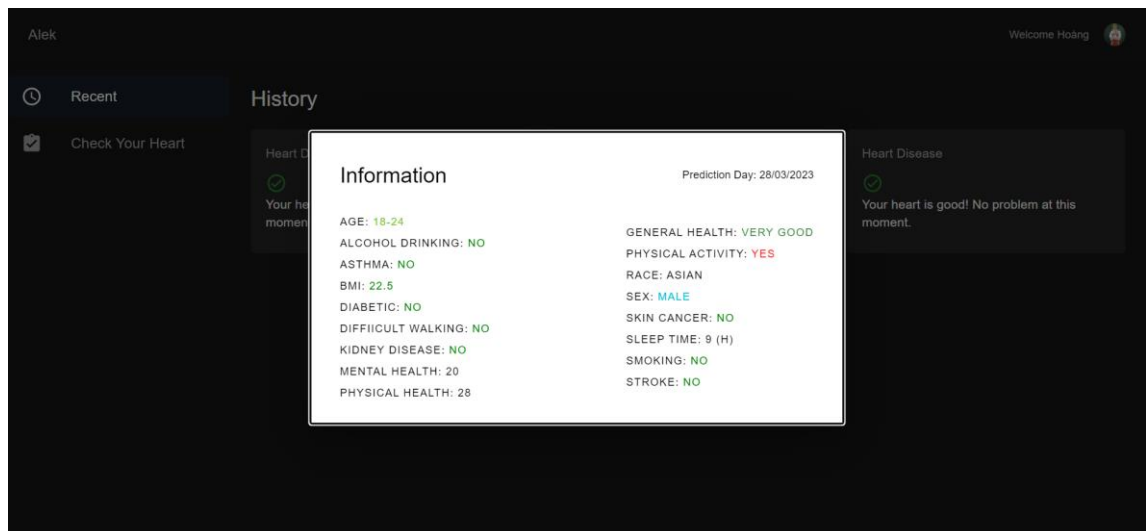
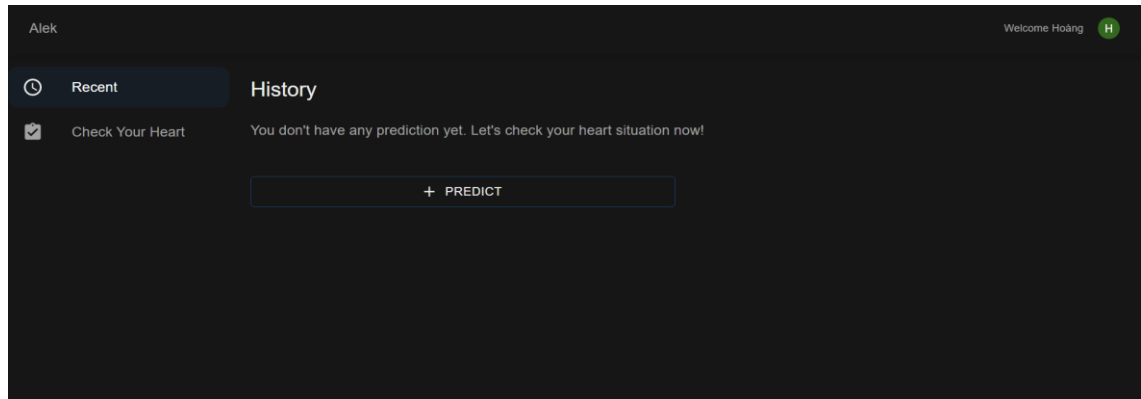


Figure 47. Prediction Details.



**Figure 48.** Dashboard 2.

In the empty history case, the application will inform the user and provide a button to navigate to the prediction page.

```

11 // ===== || DASHBOARD PAGE || ===== //
12 const DashBoard = () => {
13   const [predictionList, setPredictionList] = useState<PredictionType[]>();
14   const navigate = useNavigate();
15   useEffect(() => {
16     (async () => {
17       const uid = localStorage.getItem('uid');
18       const userInfo = await predictionApi.getPredictionList(uid as string);
19       setPredictionList(userInfo);
20     })();
21   }, []);

```

**Figure 49.** Dashboard Calling API.

To retrieve the history data, whenever a user opens the dashboard page, the application will call an API (`predictionApi.getPrediction`) to get these predictions and render to the screen.

The screenshot shows a web application interface for a heart prediction form. The form is titled "Prediction Form" and is located within a "Check Your Heart" section. It contains 17 input fields arranged in a grid. The inputs are: BMI (33.15), DiffWalking (No), GenHealth (Excellent), Smoking (No), Sex (Male), Sleep Time (9), AlcoholDrinking (No), AgeCategory (18-24), Asthma (No), Stroke (No), Race (Asian), KidneyDisease (No), PhysicalHealth (0-100) (28), Diabetic (No), SkinCancer (No), MentalHealth (0-100) (50), and PhysicalActivity (Yes). A "PREDICT" button is located at the bottom right of the form.

**Figure 50.** Input Form.

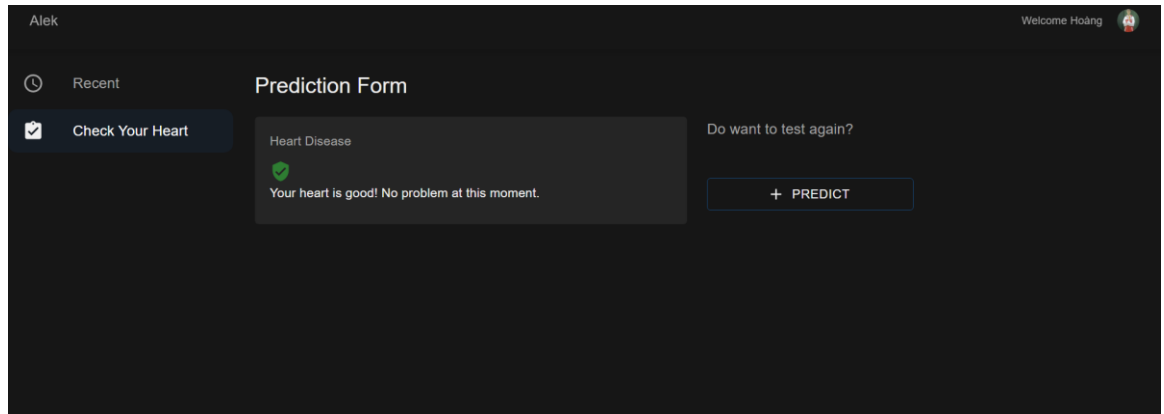
The Check page is where users will fill their information related to the heart prediction, which are 17 different inputs. The clients will get the result after a couple of seconds after clicking on the predict button.

```
const onSubmit = async (data: any) => {
  try {
    // submit data to backendd
    dispatch(actions.userPredict());
    const uid = localStorage.getItem('uid');
    const result = await predictionApi.getPrediction({ uid, data });
    dispatch(actions.userPredictDone(result));
  }
}
```

**Figure 51.** Input Form Submit.

The form will dispatch an action to notify that the process is loading and request to the server with user ID and data. After all, the response from the server will be shared to the entire application, then the prediction message will be displayed on the screen by using another component.



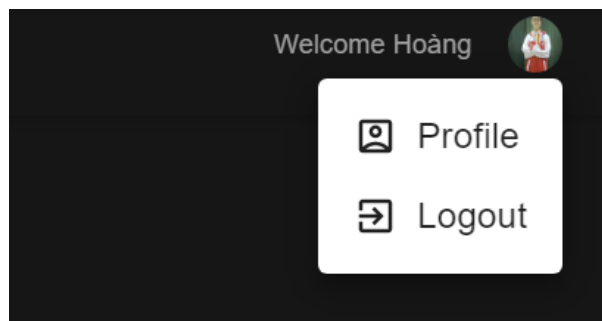


**Figure 52.** Results of The Prediction.

#### 4.4.6 User Profile

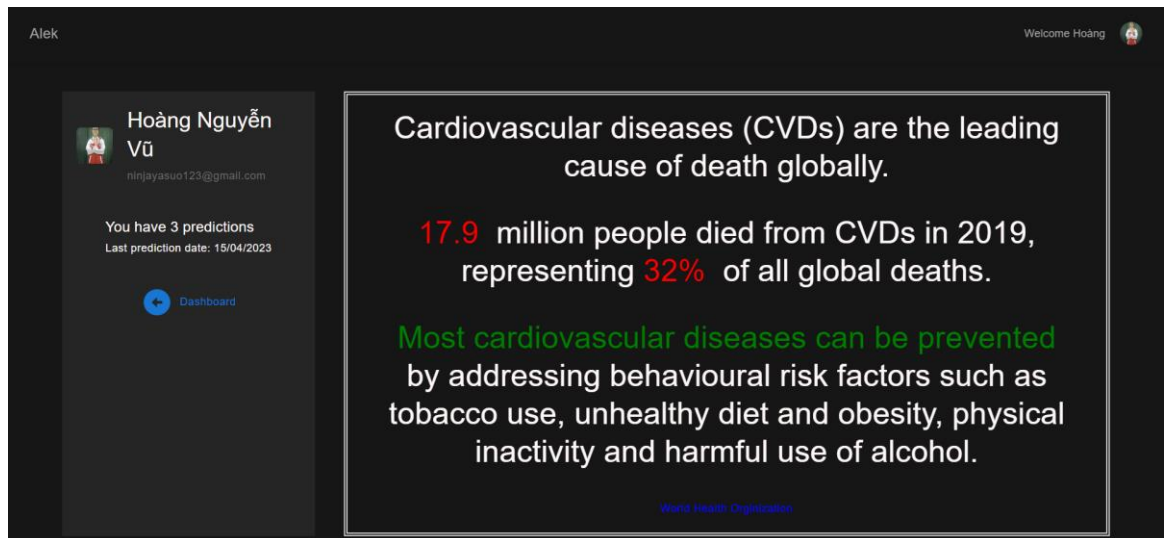
There are two options when a user clicks on profile image in the top right corner.

- Sign out: will clear the local storage variables and redirect user to the sign in page.
- Profile: opens the user profile where contains all user information.



**Figure 53.** Profile Options.

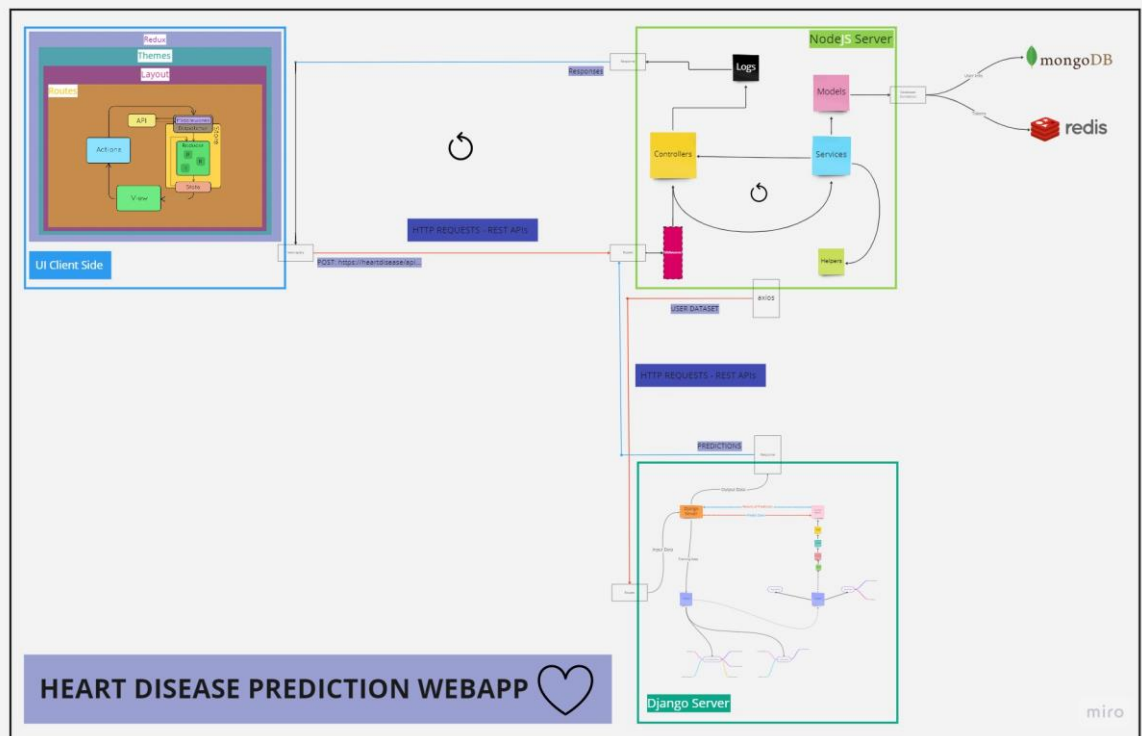
All information which are saved in the database will be retrieved and rendered on the screen, including name, email, number of predictions, etc.



**Figure 54.** User Profile.

## 5 OUTCOME OF THE PROJECT

### 5.1 Overall of The Project



**Figure 55.** Overall Project.

There are three main parts in the developed application:

- Django Server: contains training model process and running the Django server to predict the heart disease based on requested information.
- Node Server: works as a middleware which communicates with both Client and the Django server and manages user information.
- Client Side: gathers user data and requests to the Node server.

The main workflow will run through these steps in order:

1. Client side/UI Web application: collects user data both authentication and prediction information, then requests to the Node server in the JSON type.
2. Depending on the type of information, the Node server will save or modify the databases.

3. If there is a prediction request, the Node server will save this information and send a request to the Django server to get a new prediction.
4. The Django server loads the model, transforms the raw data to Tensor type, gets output from the model, converts it to JSON type, and responds to the Node server.
5. The Node server saves the respond from the Django server and sends the result to the Client side.
6. Finally, the Web Application will transform the received data from 1 or 0 to Yes or No which will be rendered on the screen.

There are two main output information packages:

1. User Basic Data: includes name, email address, avatar, and the total number of predictions.
2. User's Predictions Data: a list of history related to previous predictions and a result after filling in input forms.

A user account registered with an email will be able to predict several times, and the prediction results will vary with different health situations so that the user can compare their heart status each period.

## **5.2 Outcome**

Finally, the expected outcomes of this thesis are as follows:

1. The training process for a machine model that accurately predicts the heart status.
2. Comparing metrics, such as recall, accuracy, precision, and loss, after testing how a model reacts to the new dataset.
3. The development of the Django framework utilizing a machine learning model that gets data from requests, predicts, and responds with the prediction output.

4. The configuration and setting of a Node server follow the MVC architecture pattern, communicate with the Client Side, as well as another server, and manage user information.
5. Configuring the connection between a Node server and many databases.
6. The development of the User Interface of a React web application incorporates the Tailwind CSS and Material UI library and manages the state applying the Redux concepts.
7. Utilizing the Firebase authentication to allow people to sign in and sign out of the web application.
8. Prove how the REST APIs are flexible and how to convert various data types from JSON to tensor and vice versa.

## 6 CONCLUSIONS AND DISCUSSION

### 6.1 Thesis Assessment

The consistency and accuracy of the web application in predicting the possibility of heart disease are referred to as reliability. The evaluation metrics listed below can prove this assessment:

- Accuracy: more than 95%, which means the web application is considered acceptable to predict heart disease.
- Loss: lower than 7%, which indicates the web application has a minimal chance of predicting inaccurate results.
- Precision: about 45%, which means the web application is not reliable yet in identifying the potential of the disease.
- Recall: less than 20% and the low recall score demonstrated that the web application is poor at predicting most patients at risk of heart disease.

However, the project has not yet undergone a usability assessment; therefore, it still does not clarify the ease of use, learnability, and user satisfaction of the web application.

### 6.2 Continuation

Firstly, the current model does not have a high recall and precision score, so it causes a lower quality of prediction. That is why the training process needs to be updated with different methods, including changing the current hyperparameters, such as epochs, and learning rate.

Secondly, because user information is crucial, the future enhancement of security is also improving, for example, by applying two-factor authentication for the login page.

Finally, the user will be able to delete their history and have a chart to compare how their heart status is going.

### 6.3 Conclusion

The development of a web application that predicts cardiovascular diseases using a machine learning model can be a helpful tool for the early identification and prevention of heart disease at home.

The project processes follow a typical workflow to train a model using the TensorFlow library, from data collection to how to retrieve the saved model to predict the disease. Furthermore, the thesis discovers how to embed a model to a Django server to get requests and respond to another server.

The application also proves the excellent User Interface and User Experience of React framework, which allows using the website in the same way as the mobile application. Moreover, the Node server is very powerful in the MVC architecture pattern, which quickly accesses the MongoDB databases and responds to the others platform.

The thesis discovers the importance of data, which means training with a better dataset will produce a more efficient output model and vice versa. Moreover, the thesis proves the flexibility of REST APIs that transfer data between clients and servers. Finally, the project also demonstrates the concepts of Redux flow on state management on the front-end side.

## REFERENCES

World Health Organization (2021). "Cardiovascular diseases (CVDs)". Accessed 29.3.2023. [https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-\(cvds\)](https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds))

Google (2022). Problem Framing. Accessed 6.3.2023. <https://developers.google.com/machine-learning/problem-framing/problem-framing>

Google (2022). Introduction to Machine Learning. Accessed 6.3.2023. <https://developers.google.com/machine-learning/intro-to-ml/what-is-ml>

Elsayed, Mohamed (2022). Heart Disease Prediction. Accessed 11.3.2023. <https://www.kaggle.com/code/andls555/heart-disease-prediction/notebook>

Codepen. (2020). Parallax Scroll Animation. Accessed 14.3.2023. <https://codepen.io/isladjan/pen/abdyPBw>



**Thesis passport** (to be given to the supervisor for filing after the thesis is finished).

Name of student \_\_\_\_\_ Nguyen Hoang \_\_\_\_\_

Group \_\_\_\_\_ I-IT-19 \_\_\_\_\_

Contact information \_\_\_\_\_

Topic of thesis \_\_\_\_\_

Client \_\_\_\_\_

	At latest	Date	Signature
1. Introduction to thesis and related assignments completed	_____	_____	_____
2. Approval of topic	_____	_____	_____
3. Thesis plan approved	_____	_____	_____
4. Interim seminar presentation held	_____	_____	_____
5. Contents of thesis approved	_____	_____	_____
6. Layout and language approved	_____	_____	_____
7. Abstract in foreign language accepted	_____	_____	_____
8. Thesis submitted to the opponent	_____	_____	_____
9. Participation in presentation seminars			
- excluding acting as an opponent and own presentation			
1. _____	_____	_____	_____
2. _____	_____	_____	_____
3. _____	_____	_____	_____
4. _____	_____	_____	_____
5. _____	_____	_____	_____
6. _____	_____	_____	_____
10. Presentation of own thesis held		_____	_____
11. Acting as an opponent (title/student)			

- \_\_\_\_\_
12. Maturity test accepted \_\_\_\_\_
13. Written version of the thesis submitted to the supervisor \_\_\_\_\_
14. I hereby assure that I saved my thesis in electronic form into the Theseus at the address \_\_\_\_\_

\_\_\_\_\_

Address

Signature of the student