**1. Introduction**

**1.1 Project Overview:**

The Calorie Burn Prediction System is a web application that provides accurate estimates of calorie consumption based on the user's physical activity. Users enter data manually, including personal data such as occupation, duration, age, weight and height. The system uses machine learning algorithms to analyze these inputs and predict energy consumption.The user interface includes a simple dashboard that displays historical and real-time calorie consumption data.

It also includes personalized recommendations to optimize your lifestyle goals. In fact, Burn Prediction System is a great tool for people who want to understand their fitness performance with the simplicity of a web app.

**1.2 Purpose**

The calorie prediction feature is designed to help users understand their calorie intake while working out. This feature calculates calories burned during exercise taking into account individual factors such as gender, age, height, exercise time, heart rate and body temperature. This information not only helps users track their progress, but also allows them to improve their physical performance.

The main advantages are:

* Fitness Guidance: By predicting how many calories will be burned during different activities, machine learning models can provide fitness recommendations. Users can get training plans and exercise suggestions based on their goals, fitness level and needs.
* Exercise Optimization: Calorie forecasts can help you optimize your activity by choosing activities that match your desired calorie burn. You can exercise more efficiently and effectively and avoid wasting time on activities that are not aligned with your goals.

**2. LITERATURE SURVEY**

**2.1 Existing Problem:**

In the domain of fitness and calorie management, a notable challenge exists due to the absence of precise and personalized methods for anticipating caloric expenditure during diverse activities. Conventional approaches often rely on general estimations, neglecting individual differences in metabolism, fitness levels, and activity preferences. Additionally, the manual tracking of burned calories proves burdensome and susceptible to inaccuracies. This gap impedes the delivery of tailored fitness advice, efficient workout optimization, and effective weight management based on real-time and personalized data.

**2.2 References:**

* [Challenges faced by the modern fitness industry in India](https://timesofindia.indiatimes.com/blogs/voices/the-challenges-faced-by-the-modern-fitness-industry-in-india-and-the-side-effects/) - TOI
* [Exercise, metabolism, and weight: New research from The Biggest Loser](https://www.health.harvard.edu/blog/exercise-metabolism-and-weight-new-research-from-the-biggest-loser-202201272676) - Harvard Health

**2.3 Problem Statement Definition:**

The task at hand involves crafting a comprehensive machine learning solution capable of predicting the calories burnt during various activities. This necessitates the extraction of pertinent features from input data, encompassing factors such as input relevant features here. The goal is to construct a robust model that accommodates the nuances of individual metabolism and fitness levels.

The problem statement encompasses:

- Designing and implementing machine learning algorithms to extract relevant features from input data.

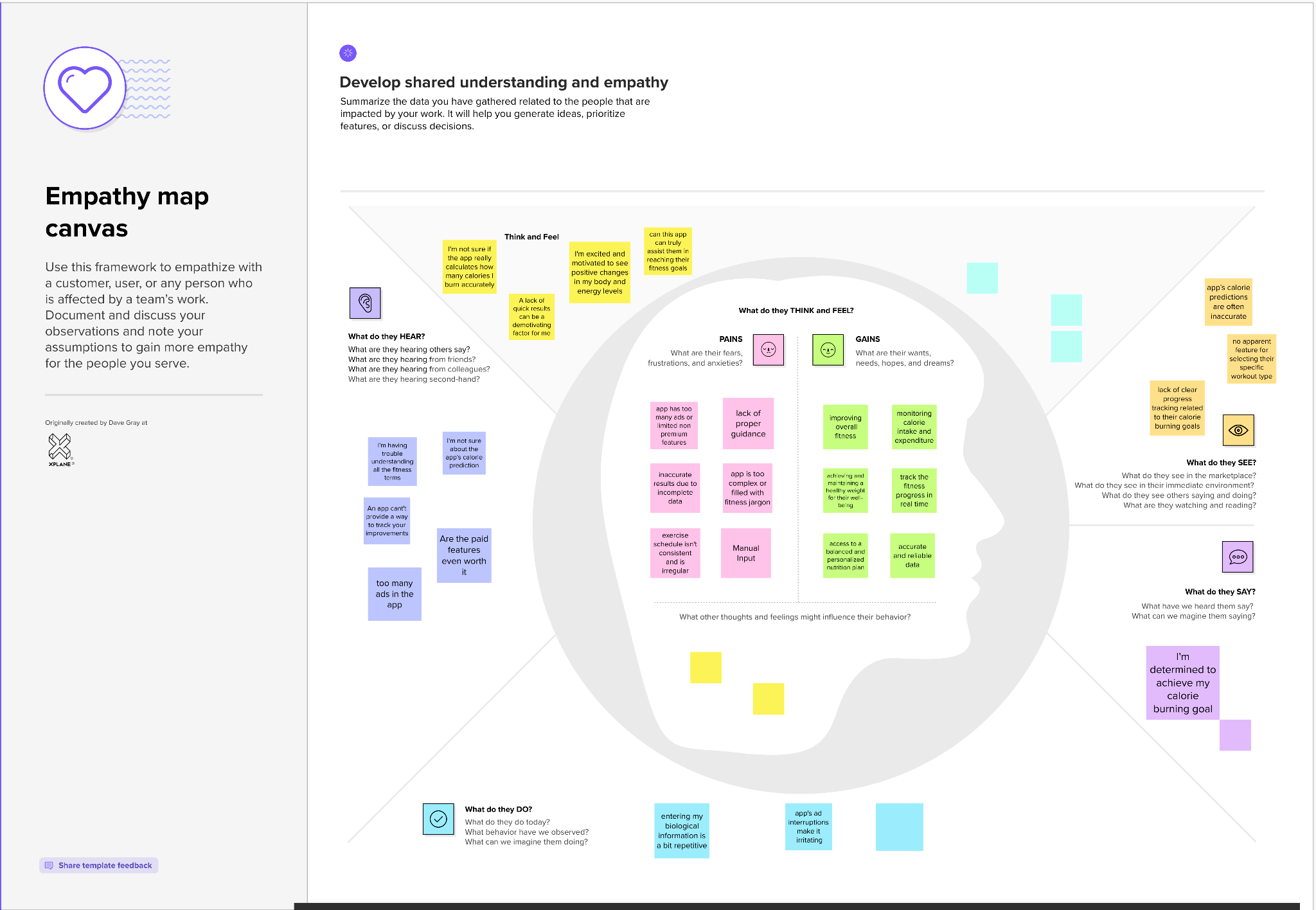
- Developing a model capable of accurately predicting calories burned based on the extracted features.

- Establishing an intuitive web application that not only showcases these predictions to the user but also seamlessly saves static user details in the database for effortless data input. Moreover, it records each user's activities for future reference.

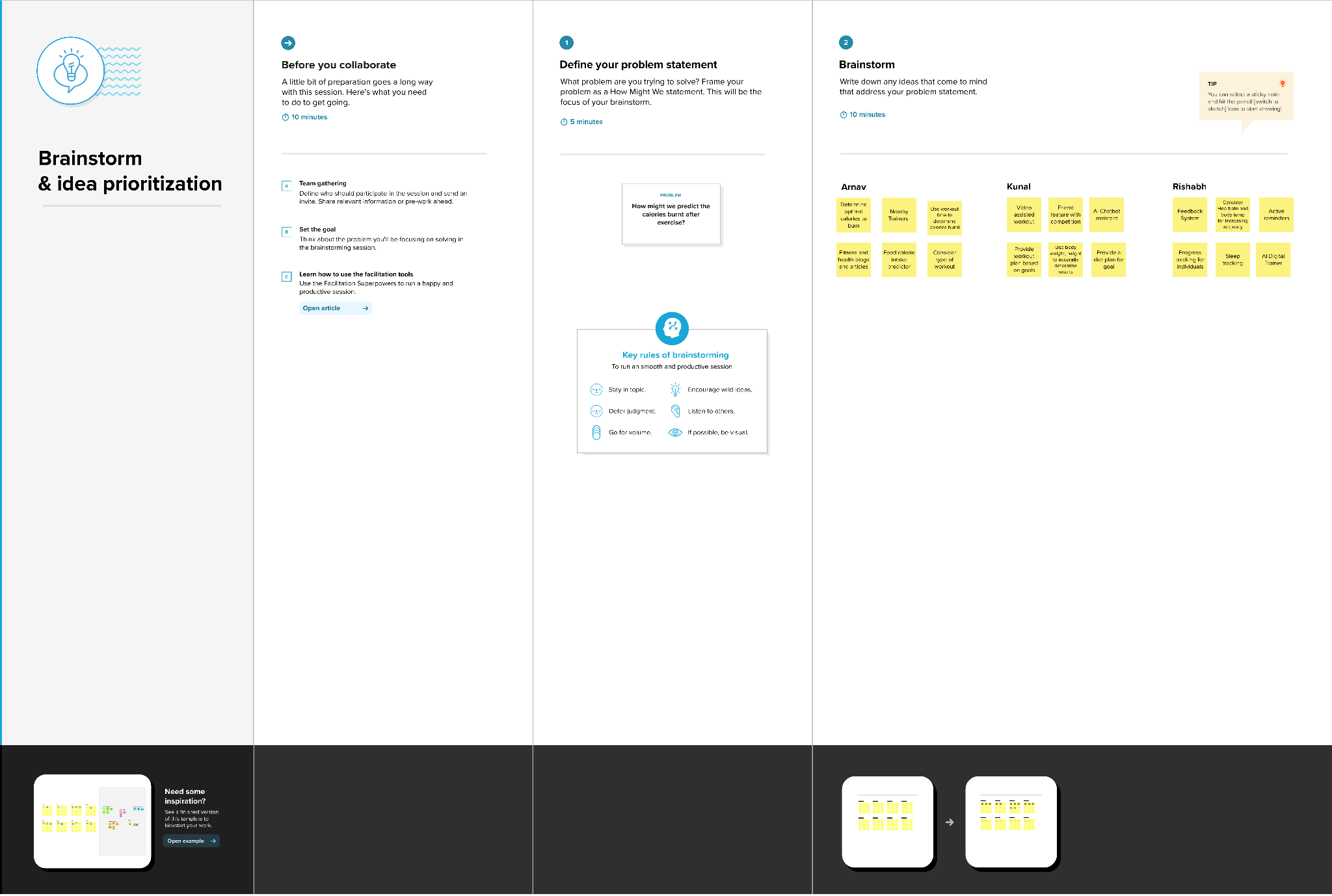
By attaining these objectives, the project aspires to modernize individuals' approach to fitness, offering a holistic and personalized method for calorie management through machine learning.

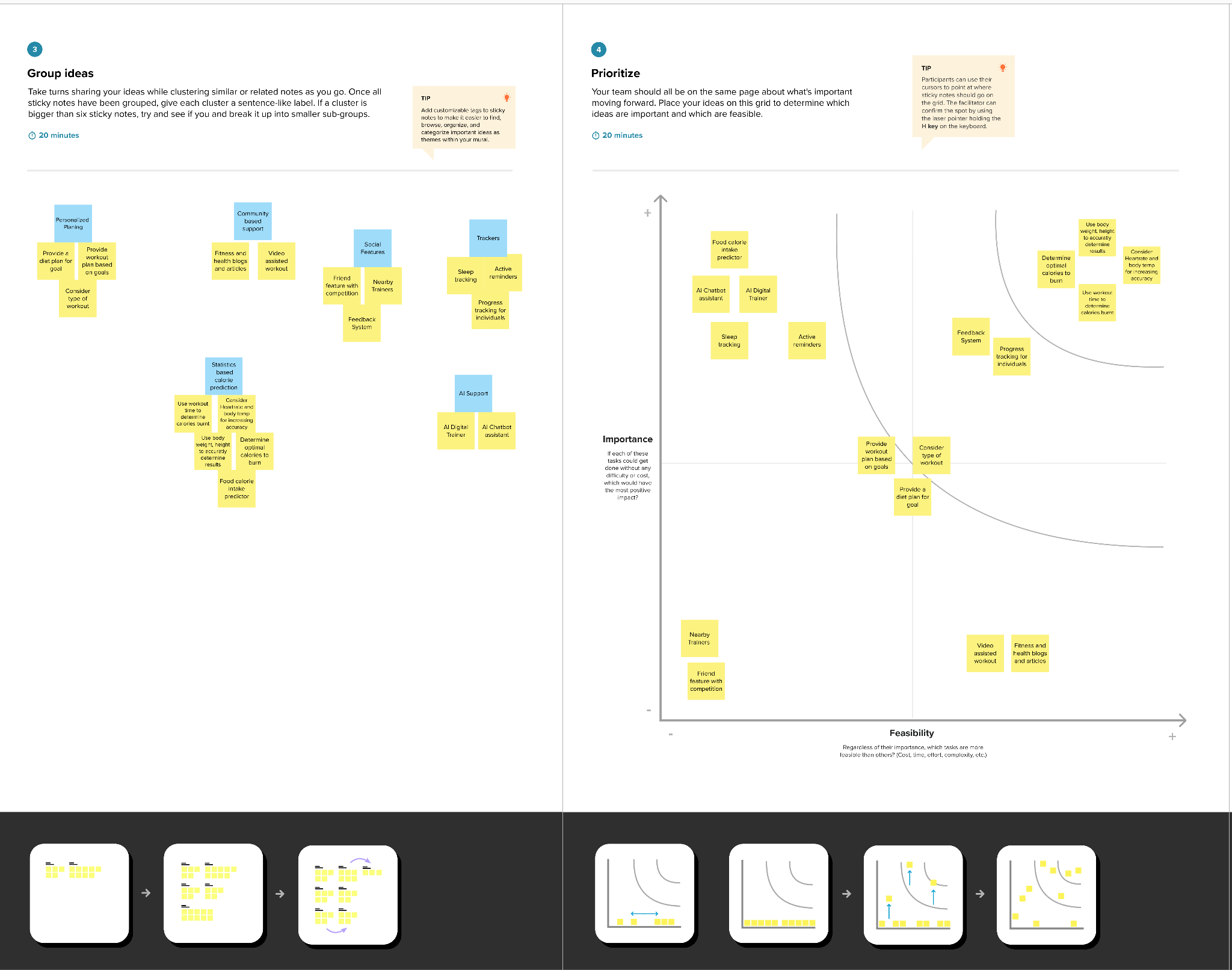
**3. IDEATION AND PROPOSED SOLUTION**

**3.1 Empathy Map Canvas**

****

**3.2 Ideation and Brainstorming**

****

****

**4. REQUIREMENT ANALYSIS**

**4.1 Functional Requirements**

1. Calorie Prediction Algorithm:

=> Develop and deploy a precise calorie prediction algorithm based on user-provided information.

=> Ensure the algorithm accounts for user-specific factors such as height, weight, and BMI.

1. Input Data Handling:

=> Facilitate manual data input for users, including metrics like height, BMI, and weight.

=> Accept and store user-provided data related to personal metrics.

1. User Authentication and Authorization:

=> Enable secure user account creation and login.

=>Incorporate different user roles (e.g., guest user) with corresponding access levels.

1. Integration with Wearable Devices:

=> Integrate with fitness trackers or wearable devices if applicable to enhance prediction accuracy.

1. Activity Tracking Database:

=> Maintain a database to track user workouts, calories burned, and other relevant metrics.

=> Regularly update the database to include new user workouts and current metrics.

**4.2 Non-Functional Requirements**

1. Reliability:

=> Ensure the system's consistent availability and reliability.

=> Implement backup and recovery mechanisms to protect against data loss.

1. Security:

=> Implement robust user authentication and data encryption protocols.

=> Safeguard user data against unauthorized access through stringent security measures.

1. Performance:

=> Ensure prompt system responses to user inputs.

=> Design the system to effectively handle a reasonable number of simultaneous users.

1. Scalability:

=> Design the system to be scalable, accommodating an increasing number of users and data.

1. Accuracy:

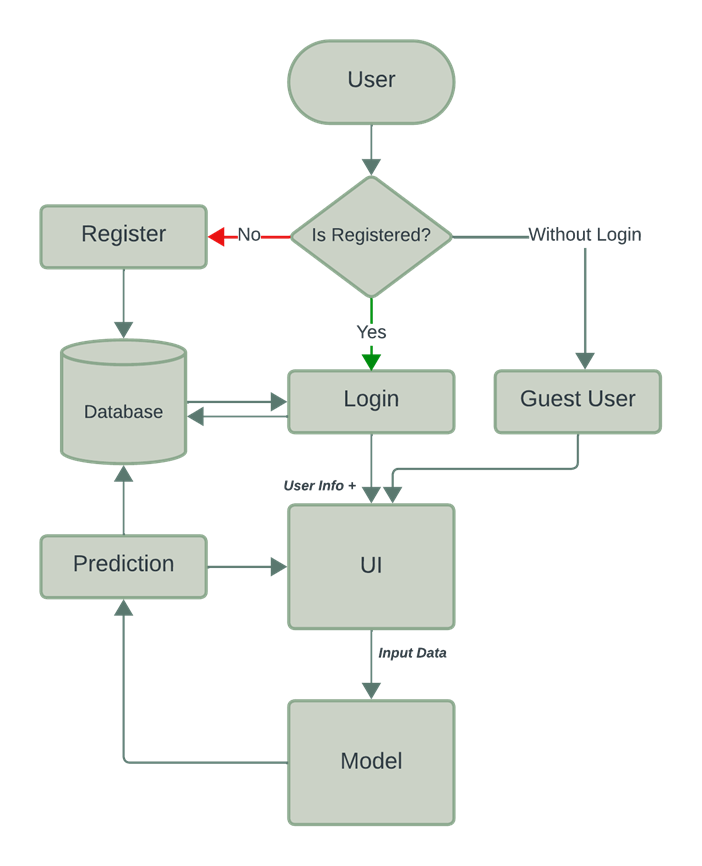
=> Implement a highly accurate calorie prediction algorithm.

=> Minimize errors in database entries to enhance overall system precision.

**5. PROJECT DESIGN**

**5.1 Data Flow Diagrams and User Stories**

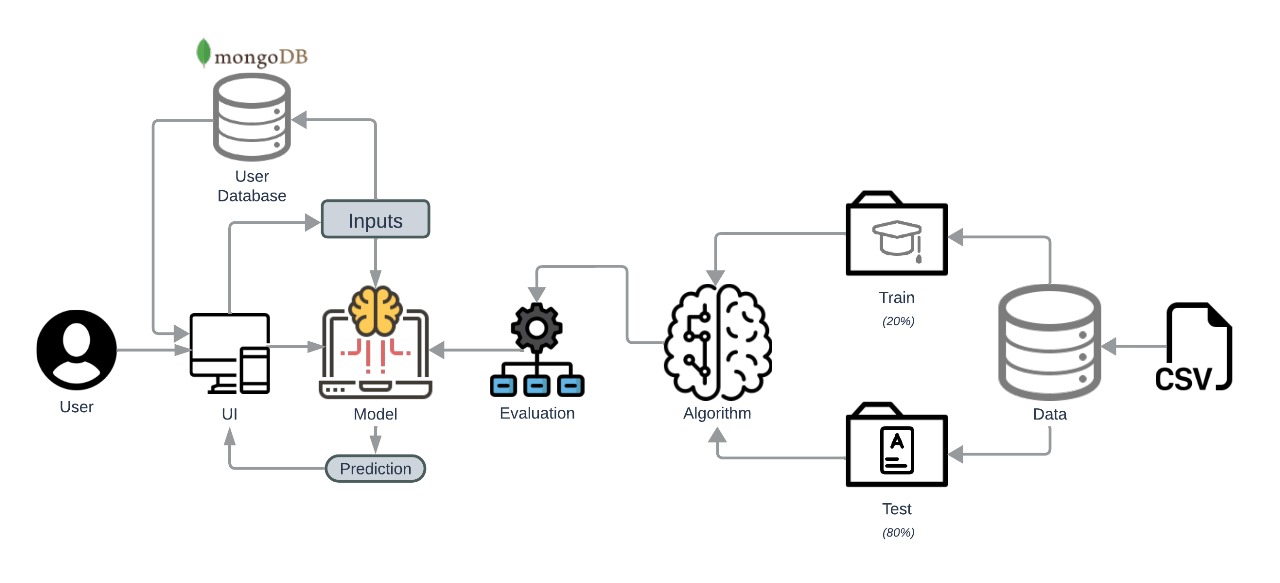
=> A data flow diagram (DFD) is a traditional way of visually representing the flow of information in a system. A simple and clear DFD can represent many system requirements. It describes how data enters and leaves the system, what information changes, and where the data is stored.



**User Stories**

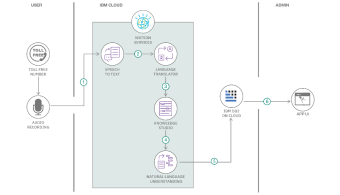
| **User Type** | **Functional Requirements** | **User  Story Number** | **User Stor / Task** | **Acceptance criteria** | **Priority** | **Release** |
| --- | --- | --- | --- | --- | --- | --- |
| Customer | Registration | USN-1 | As a user, I can register for the website by entering my email-id, password, and confirming my password. | I can access my account / dashboard | High | Sprint-1 |
|  | Login | USN-2 | As a user, I can log into the website by entering my email-id and password |  | High | Sprint-1 |
|  | Dashboard | USN 3 | As a user, I can track my goal progress | I can see my previous workout activities | Medium | Sprint-2 |
|  |  | USN-4 | As a user, I can input and update my health data | I can update my user data anytime | High | Sprint-2 |
|  | Calculator | USN-5 | As a user, I can enter my body/exercise details | I can get the calories burnt | High | Sprint-3 |

**5.2 Solution Architecture**



**6. PROJECT PLANNING AND SCHEDULING**

**6.1 Technical Architecture**



**Table-1 : Components & Technologies:**

| **S.No** | **Component** | **Description** | **Technology** |
| --- | --- | --- | --- |
| 1. | User Interface | How user interacts with application | HTML, CSS, JavaScript |
| 2. | Application Logic-1 | Logic for a process in the application - Login/Register | Python(Flask) |
| 3. | Database | User info, Calorie info etc. | SQL, CSV |
| 4. | Machine Learning Model | Prediction of the calories burnt | XGB Regressor etc. |
| 5. | Infrastructure (Server / Cloud) | Application Deployment on Cloud Server /  Cloud Server Configuration : |  |

**6.2 Sprint Planning & Estimation**

| **Sprint** | **Functional**  **Requirement (Epic)** | **User Story**  **Number** | **User Story / Task** | **Story Points** | **Priority** | **Team**  **Members** |
| --- | --- | --- | --- | --- | --- | --- |
| Sprint-1 | Registration | USN-1 | As a user, I can register for the application by entering my email, password, and confirming my password. | 2 | High | 3 |
| Sprint-1 | Login | USN-2 | As a user, I can log into the application by entering email & password | 1 | High | 3 |
| Sprint-2 | Dashboard | USN-3 | As a user, I can track my progress | 2 | Medium | 3 |
| Sprint-2 |  | USN-4 | As a user, I can input and update my health data | 2 | High | 3 |
| Sprint-3 | Calculator | USN-5 | As a user, I can enter the body/exercise details | 1 | High | 3 |

**6.3 Sprint Delivery Schedule**

| **Sprint** | **Total Story**  **Points** | **Duration** | **Sprint Start Date** | **Sprint End Date**  **(Planned)** | **Story Points**  **Completed (as on Planned End Date)** | **Sprint Release Date**  **(Actual)** |
| --- | --- | --- | --- | --- | --- | --- |
| Sprint-1 | 20 | 6 Days | 1 Nov 2023 | 6 Nov 2023 | 3 | 6 Nov 2023 |
| Sprint-2 | 20 | 6 Days | 7 Nov 2023 | 12 Nov 2023 | 7 | 12 Nov 2023 |
| Sprint-3 | 20 | 6  Days | 13 Nov 2023 | 18 Nov 2023 | 8 | 16 Nov 2023 |

**7. CODING AND SOLUTIONING**

**7.1 & 7.2 Aspects**

**Introduction**

The incorporation of User Registration and Personalized Dashboard features, along with the innovative Calorie Prediction functionality, enhances the application's user experience. These features collectively deliver a smooth and personalized journey, promoting engagement and motivation in users' fitness endeavors. The registration process offers an enriched, customized experience aligned with their fitness goals.

**Key Features**

User Registration and Profile Management

**1.1 Registration Process**

Users can efficiently sign up by providing necessary details, ensuring a quick and trouble-free onboarding experience.

**1.2 Managing Profiles**

Effortless profile editing enables users to update information like weight, height, and other relevant details seamlessly.

Changes in the profile settings promptly reflect in the personalized dashboard.

**1.3 Engagement and Commitment**

Personalized dashboards foster consistent engagement, instilling a sense of ownership in users regarding their fitness journey.

**2. Personalized Dashboard**

**2.1 Overview**

After registration and login, users are directed to a dedicated dashboard, serving as a central hub for tracking and visualizing workout-related data.

**2.2 Workout Statistics**

Insights into workout history, including exercise duration, total calories burned, average BPM, and more, empower users.

Graphical representations offer an intuitive understanding of progress over time.

**3. Calorie Prediction Feature**

**3.1 User Input Parameters**

Users input fundamental details such as gender, age, height, exercise duration, heart rate, and body temperature for accurate calorie predictions.

**3.2 Calorie Prediction Algorithm**

An advanced algorithm processes user-input data, considering physiological factors and exercise intensity for precise calorie estimations.

**3.3 Personalized Calorie Insights**

Users receive a detailed breakdown of estimated calories burnt, with graphical representations showcasing trends over time.

Goal alignment highlights whether users are meeting, exceeding, or falling short of targeted calorie burn.

**3.4 Benefits**

Precision in Fitness Planning: Users can plan workouts effectively by understanding the calorie impact of different exercises and durations.

Motivation Through Data: Real-time feedback on calorie expenditure motivates users to push themselves during workouts.

Personalized Optimization: Users can fine-tune workouts based on personalized data, optimizing calorie-burning potential.

**Conclusion**

The combined synergy of User Registration, Personalized Dashboards, and the innovative Calorie Prediction feature amplifies the user experience. By providing tailored insights, robust tracking capabilities, and personalized predictions, these features empower users on their fitness journey and foster sustained engagement with the application.

Code:

Login and Register Routes:

| **Login** | **Register** |
| --- | --- |
| @app.route('/login.html',methods=("GET","POST"))  def login():  if request.method=="POST":  login\_id=request.form['userid']  passwd = request.form['password']  login\_conn = sqlite3.connect(app.config['DATABASE'])  cur=login\_conn.cursor()  cur.execute('SELECT name,age,height,gender FROM users WHERE userid = ? AND password = ?', (login\_id,passwd))  result = cur.fetchone()  login\_conn.close()  if result:  print(result)  session['userid'] = request.form['userid']  session['name'] = result[0]  session['age'] = result[1]  session['height'] = result[2]  session['gender'] = result[3]  return redirect(url\_for('home'))  else:  print("Wrong")  flash("Wrong Id or Password")  return render\_template('login.html')  return render\_template('login.html') | @app.route('/register.html',methods=("GET","POST"))  def register():  if request.method=="POST":  user\_id=request.form['userid']  passwd = request.form['password']  name = request.form['name']  age = request.form['age']  height = request.form['height']  gender = request.form['gender']  register\_conn = sqlite3.connect(app.config['DATABASE'])  register\_cur=register\_conn.cursor()  register\_cur.execute('SELECT userid FROM users WHERE userid = ?', (user\_id,))  result = register\_cur.fetchone()  if result:  register\_conn.close()  flash("User already exists")  return render\_template('login.html')  else:  register\_cur.execute('insert into users values(?,?,?,?,?,?)',(user\_id,passwd,name,age,height,gender))  register\_conn.commit()  register\_conn.close()  session['userid'] = request.form['userid']  session['name'] = request.form['name']  session['age'] = request.form['age']  session['height'] = request.form['height']  session['gender'] = request.form['gender']  return redirect(url\_for('home'))  return render\_template('login.html') |

**Explanation:**  
  
Login Route:

The login code is a Python Flask web application route that handles user authentication. When a user submits the login form, the server checks if the request method is POST. If it is, the code retrieves the entered username and password from the form, connects to a SQLite database, and executes a SELECT query to fetch user data (name, age, height, gender) corresponding to the provided login credentials. If a matching user is found, their information is stored in the session, and the user is redirected to the home page. If the login credentials are incorrect, an error message is displayed, and the user stays on the login page. The login function also handles GET requests, rendering the login page initially.

Register Route:

The register code is another Flask route for user registration. When a user submits the registration form, the server checks if the request method is POST. If it is, the code retrieves the user details from the form, connects to the SQLite database, and checks if the entered username already exists. If the username is taken, an error message is displayed, and the user is redirected to the login page. If the username is available, a new user entry is added to the database, and the user is registered. Their information is stored in the session, and the user is redirected to the home page. Similar to the login code, the register function also handles GET requests, rendering the login page initially.

Dashboard:

@app.route('/dashboard.html',methods=("GET","POST"))

def dashboard():

user\_name = session.get('name')

userid = session.get('userid')

if user\_name:

exercise\_data, time\_data, calories\_data, heart\_data = fetch\_data(userid)

bar\_chart = create\_bar\_chart(exercise\_data)

line\_chart = create\_line\_chart(time\_data)

pie\_chart = create\_pie\_chart(calories\_data, title='Exercise Distribution and Calories Burned')

scatter\_plot = create\_heart\_rate\_scatter\_plot(heart\_data)

return render\_template('user\_dashboard.html',

username=user\_name,

bar\_chart=bar\_chart,

line\_chart=line\_chart,

pie\_chart=pie\_chart,

scatter\_plot=scatter\_plot

)

import plotly

import plotly.graph\_objs as go

import plotly.offline as opy

import sqlite3

import plotly.express as px

import json

def fetch\_data(userid):

conn = sqlite3.connect('Calorie.db')

cursor = conn.cursor()

# Example query: Retrieve total quantity sold for each product

query = "SELECT exercise\_name, duration FROM exercise WHERE userid = ?"

cursor.execute(query, (userid,))

data = cursor.fetchall()

# Example query: Retrieve total quantity sold over time

time\_query = "SELECT date, calories FROM exercise WHERE userid = ?"

cursor.execute(time\_query, (userid,))

time\_data = cursor.fetchall()

calories\_query = "SELECT exercise\_name, calories FROM exercise WHERE userid = ?"

cursor.execute(calories\_query, (userid,))

calories\_data = cursor.fetchall()

heart\_query = "SELECT bpm, calories FROM exercise WHERE userid = ?"

cursor.execute(heart\_query, (userid,))

heart\_data = cursor.fetchall()

cursor.close()

conn.close()

return data, time\_data, calories\_data, heart\_data

def create\_bar\_chart(data):

exercise, duration = zip(\*data)

trace = go.Bar(x=exercise, y=duration)

layout = go.Layout(title='Exercise vs duration', xaxis=dict(title='Exercise'), yaxis=dict(title='Duration'))

fig = go.Figure(data=[trace], layout=layout)

return opy.plot(fig, auto\_open=False, output\_type='div')

def create\_line\_chart(time\_data):

date, calories = zip(\*time\_data)

trace = go.Scatter(x=date, y=calories, mode='lines+markers', marker=dict(size=10), line=dict(width=2))

layout = go.Layout(title='Total Calories Over Time', xaxis=dict(title='Date'), yaxis=dict(title='Total Calories'))

fig = go.Figure(data=[trace], layout=layout)

return opy.plot(fig, auto\_open=False, output\_type='div')

def create\_pie\_chart(data, title):

exercise\_names, calories = zip(\*data)

trace = go.Pie(labels=exercise\_names, values=calories)

layout = go.Layout(title=title)

fig = go.Figure(data=[trace], layout=layout)

return opy.plot(fig, auto\_open=False, output\_type='div')

def create\_heart\_rate\_scatter\_plot(data):

bpm, calories = zip(\*data)

trace = go.Scatter(x=bpm, y=calories, mode='markers', marker=dict(size=12))

layout = go.Layout(title='Heart Rate vs. Calories Burned', xaxis=dict(title='Heart Rate (BPM)'), yaxis=dict(title='Calories Burned'))

fig = go.Figure(data=[trace], layout=layout)

return opy.plot(fig, auto\_open=False, output\_type='div')

Code Explanation:

The dashboard code defines a Flask route for displaying a user dashboard. It retrieves the user's name and ID from the session and, if the user is logged in, fetches exercise data, time data, calories data, and heart rate data using the `fetch\_data` function. It then creates four different types of charts (bar chart, line chart, pie chart, and scatter plot) using the Plotly library based on the fetched data. These charts are then passed to the 'user\_dashboard.html' template along with the username for rendering. The charts are displayed to the user, providing visual insights into their exercise and fitness data.

The `fetch\_data` function connects to a SQLite database and executes several queries to retrieve exercise-related data, including exercise names, durations, calories, dates, and heart rates. The data is then returned for use in creating the charts.

The four chart creation functions (`create\_bar\_chart`, `create\_line\_chart`, `create\_pie\_chart`, `create\_heart\_rate\_scatter\_plot`) utilize Plotly to generate the charts. The charts are converted to JSON format for easy rendering in the HTML template.

This code essentially creates a dynamic and visually appealing dashboard for users to track and analyze their exercise data.

Calorie Prediction:

def prediction(req):

print(req)

Gender=int(req['gender'])

Age = float(req['age'])

Height = float(req['height'])

Duration=float(req['duration'])

Heart\_Rate = float(req['heart\_rate'])

Body\_Temp = float(req['temperature'])

model=pickle.load(open('final\_model.pkl','rb'))

data=[[Gender,Age,Height,Duration,Heart\_Rate,Body\_Temp]]

print(data)

result = model.predict(data)

result = round(float(result),2)

return result

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from xgboost import XGBRegressor

# NOTE: Make sure that the outcome column is labeled 'target' in the data file

tpot\_data = pd.read\_csv('PATH/TO/DATA/FILE', sep='COLUMN\_SEPARATOR', dtype=np.float64)

features = tpot\_data.drop('target', axis=1)

training\_features, testing\_features, training\_target, testing\_target = \

train\_test\_split(features, tpot\_data['target'], random\_state=None)

# Average CV score on the training set was: -1.3279745078720981

exported\_pipeline = XGBRegressor(learning\_rate=0.1, max\_depth=10, min\_child\_weight=10, n\_estimators=100, n\_jobs=1, objective="reg:squarederror", subsample=0.7500000000000001, verbosity=0)

exported\_pipeline.fit(training\_features, training\_target)

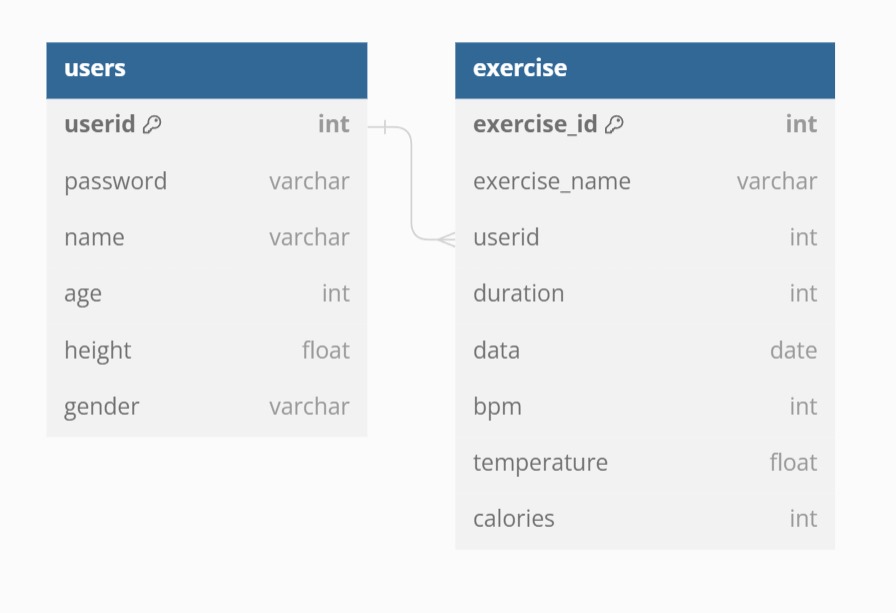
results = exported\_pipeline.predict(testing\_features)

Code Explanation:

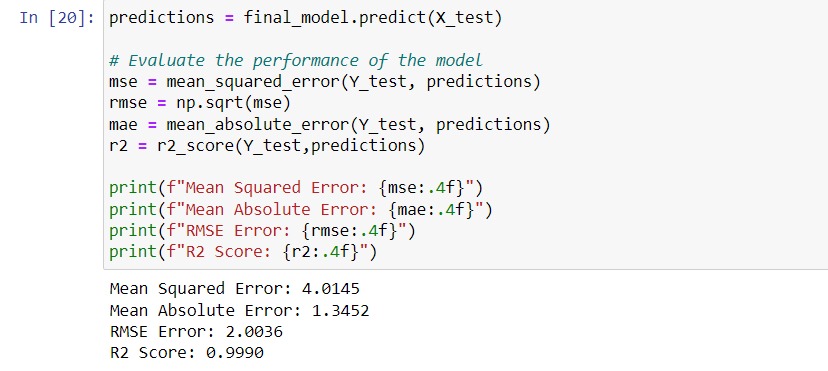
The provided code defines a function `prediction` that takes a dictionary `req` as input, extracts relevant numerical values (gender, age, height, duration, heart rate, body temperature) from the dictionary, loads a pre-trained XGBoost regression model from a pickled file ('final\_model.pkl'), and uses this model to predict an outcome based on the input data. The input data is formatted into a list of lists (`data`), and the `predict` method is applied to obtain the result. The result is rounded to two decimal places and returned.

The code also includes additional machine learning-related imports and code that seem to be part of the model training process. It imports necessary libraries (numpy, pandas, train\_test\_split, XGBRegressor), reads a CSV file ('PATH/TO/DATA/FILE') containing training data, performs a train-test split, and trains an XGBoost regression model (`exported\_pipeline`). The model is then used to predict outcomes on the testing features.

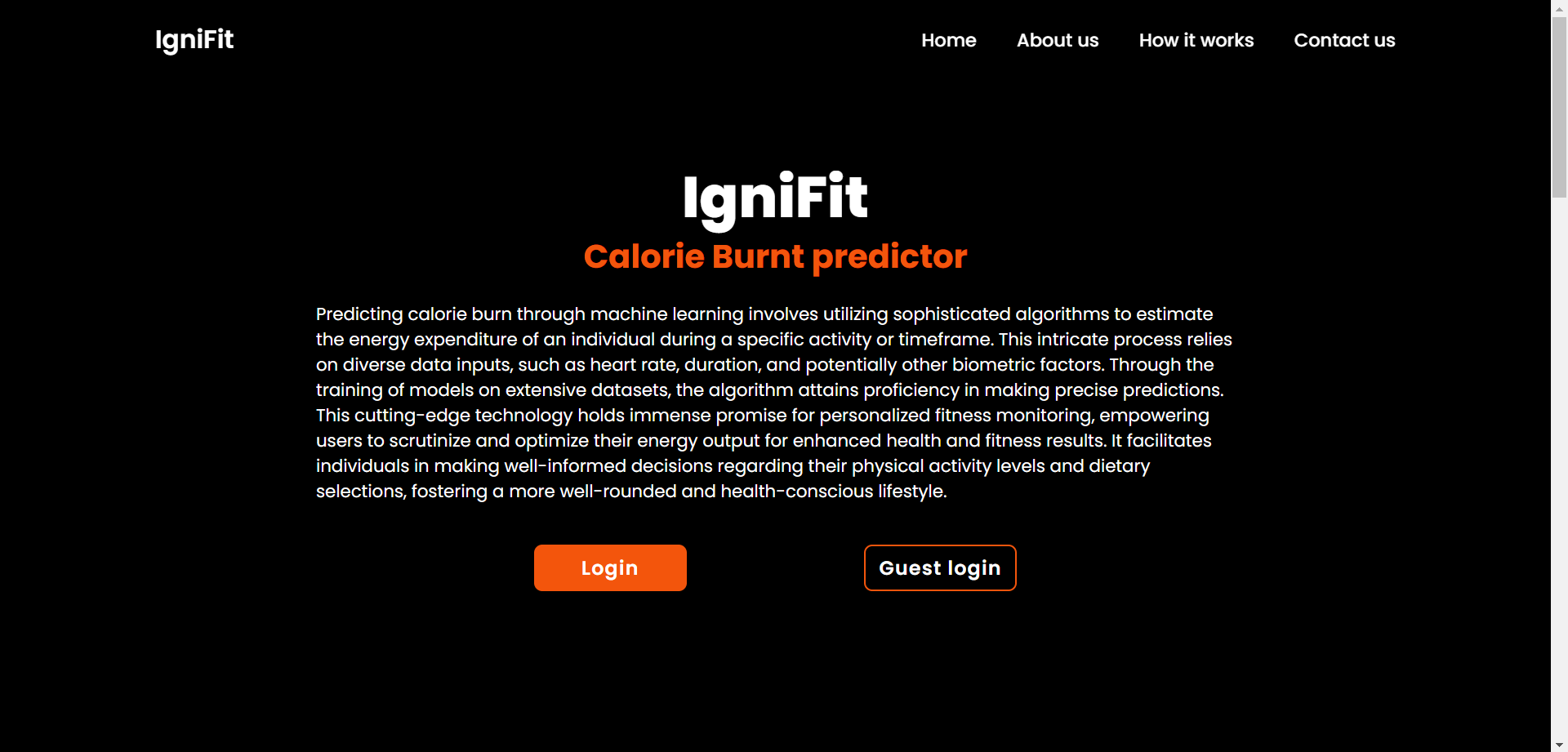
7.3 Database Schema

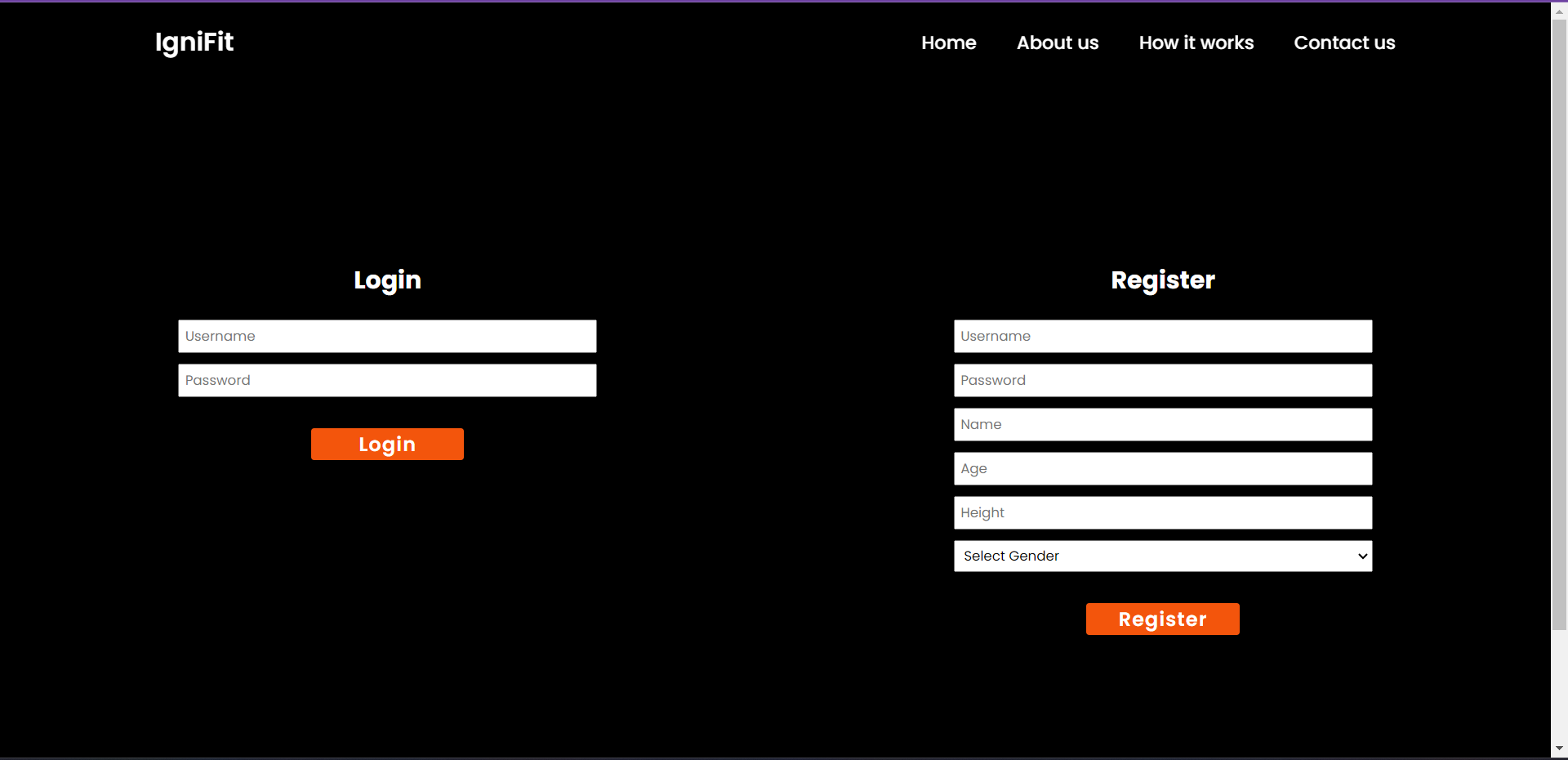


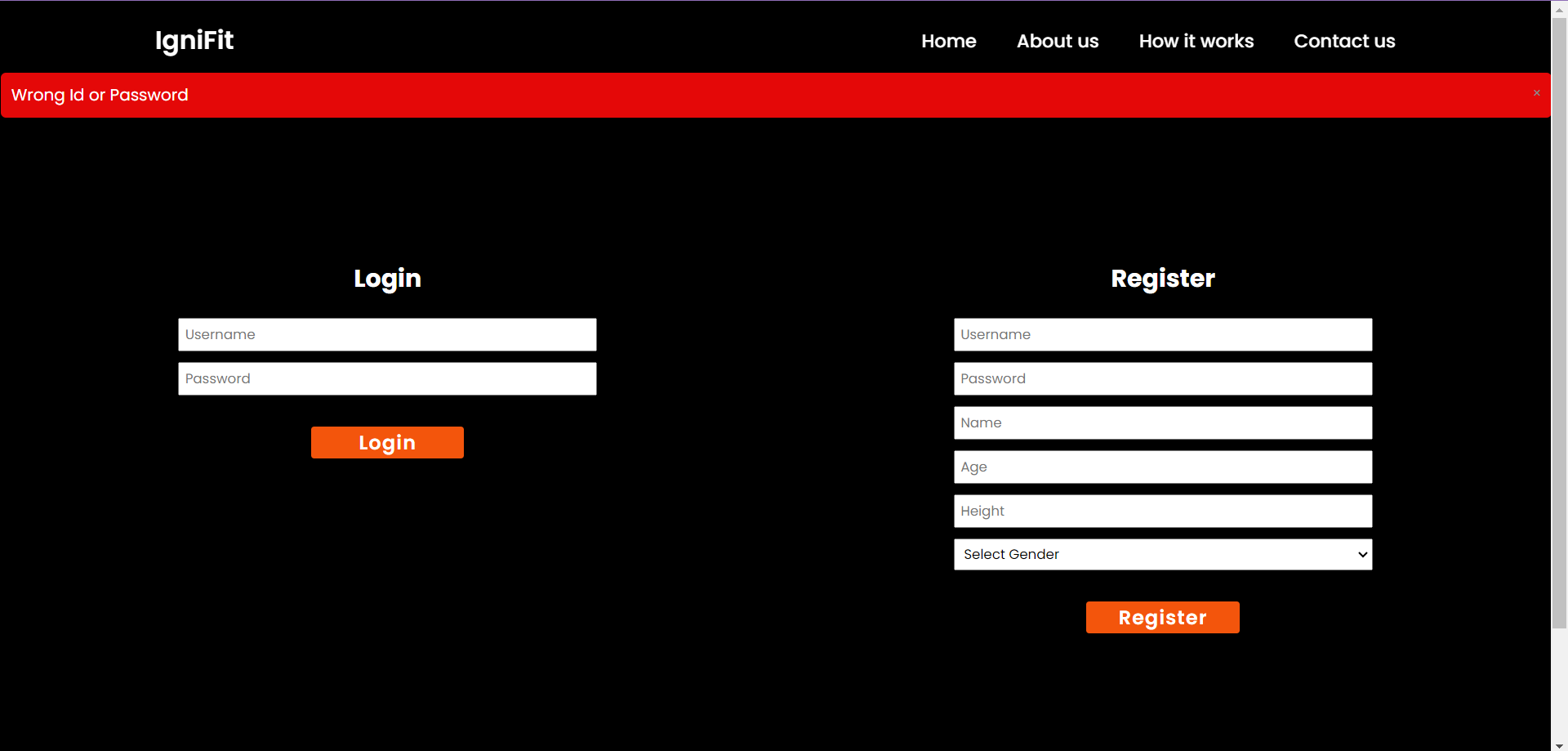
**8. PERFORMANCE TESTING**

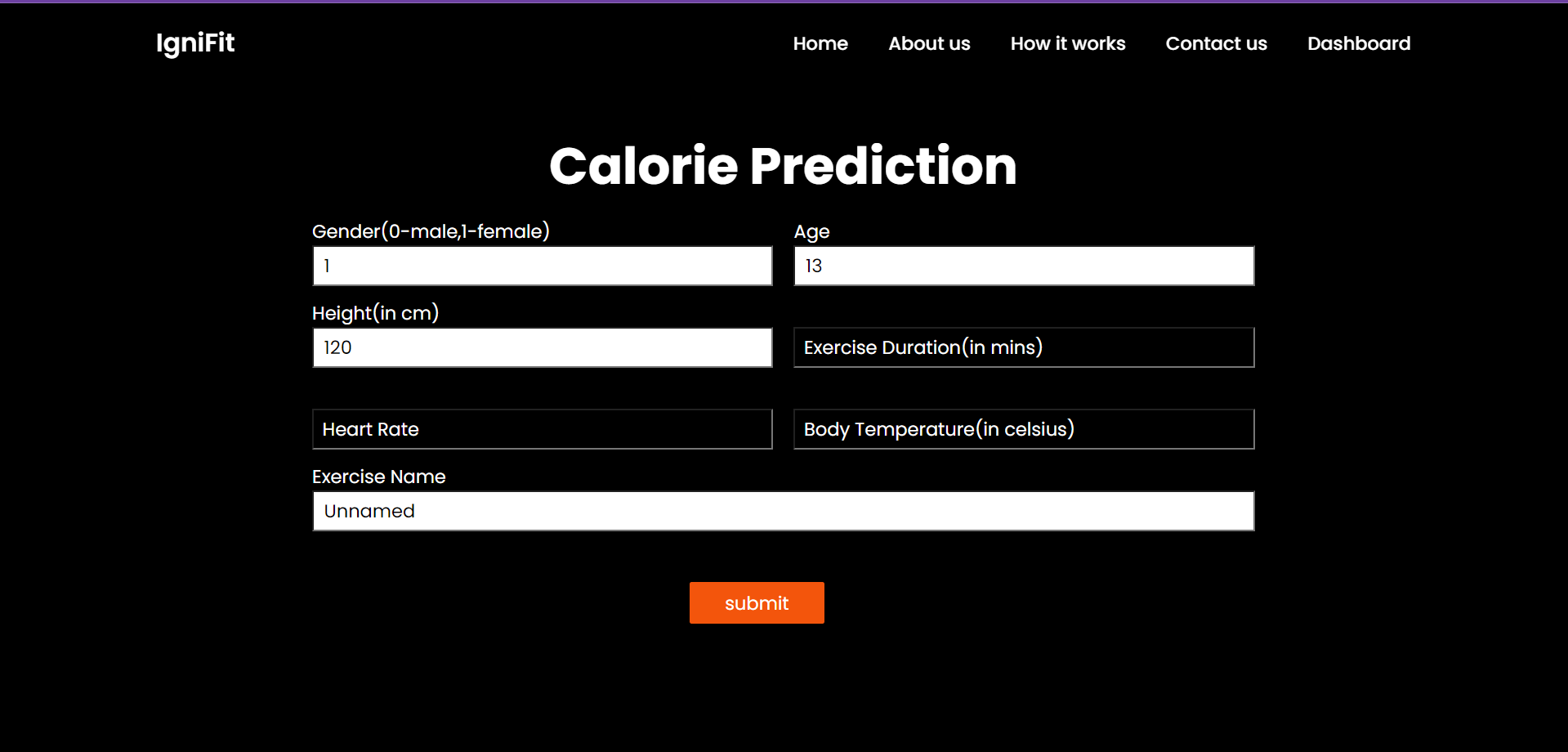


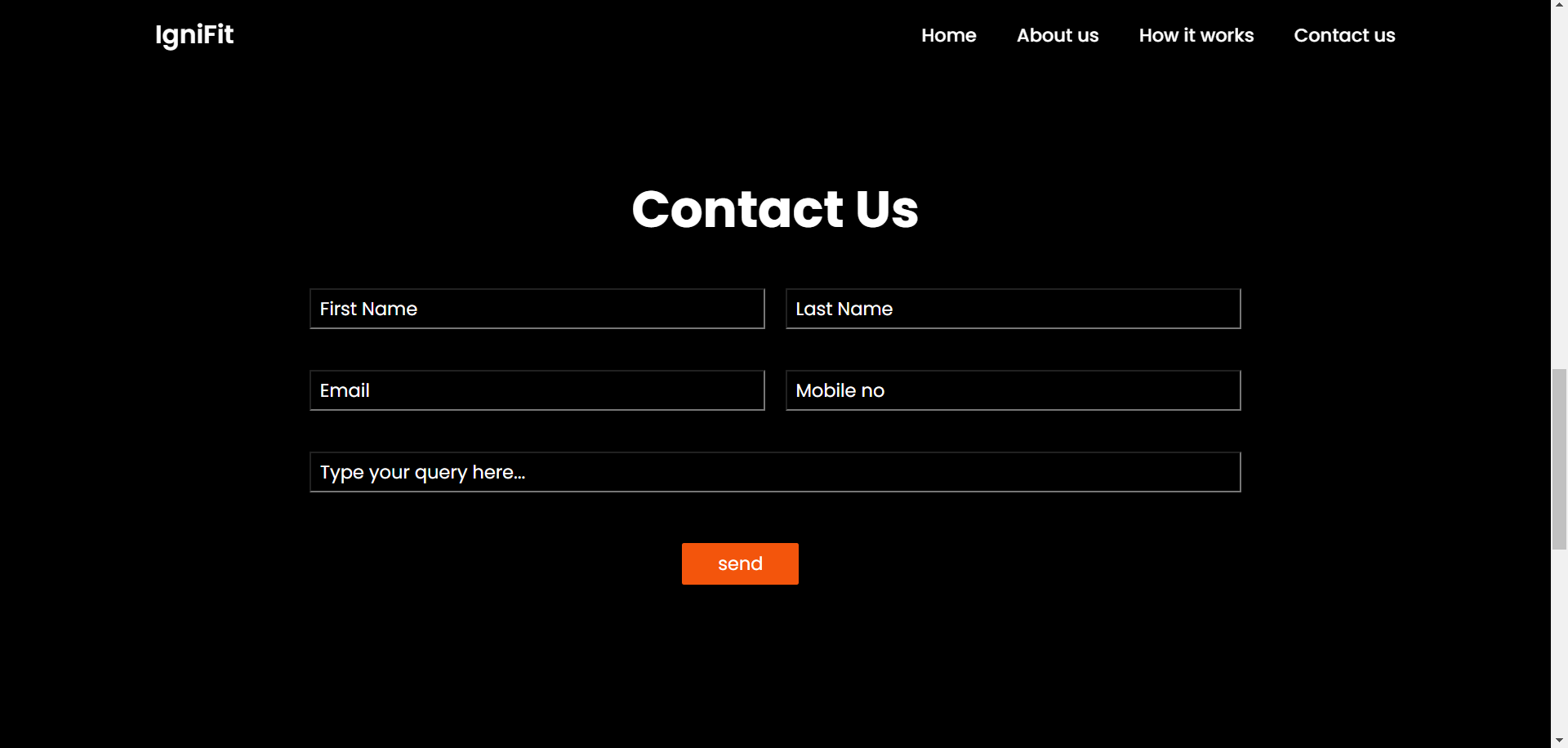
**9. RESULTS**

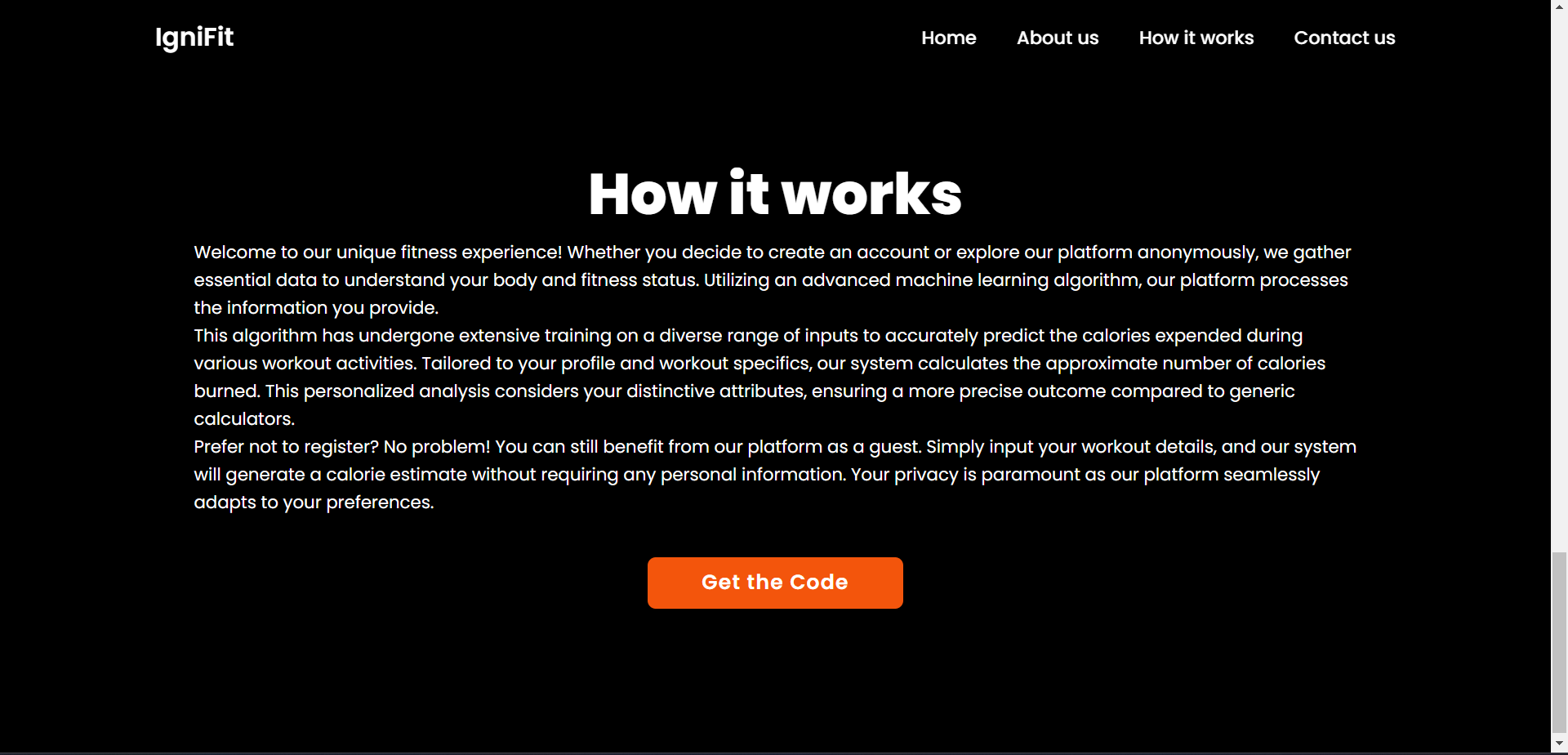


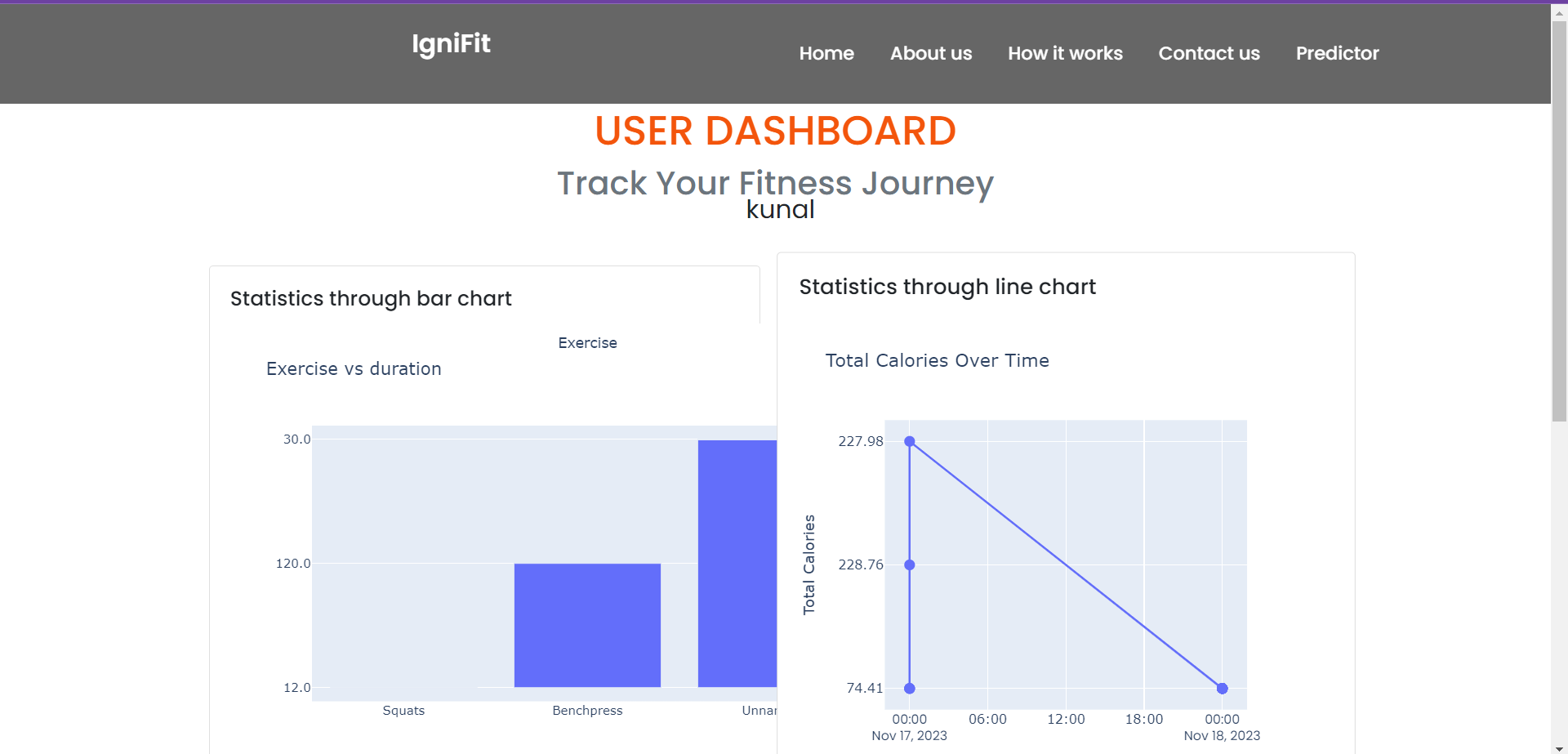


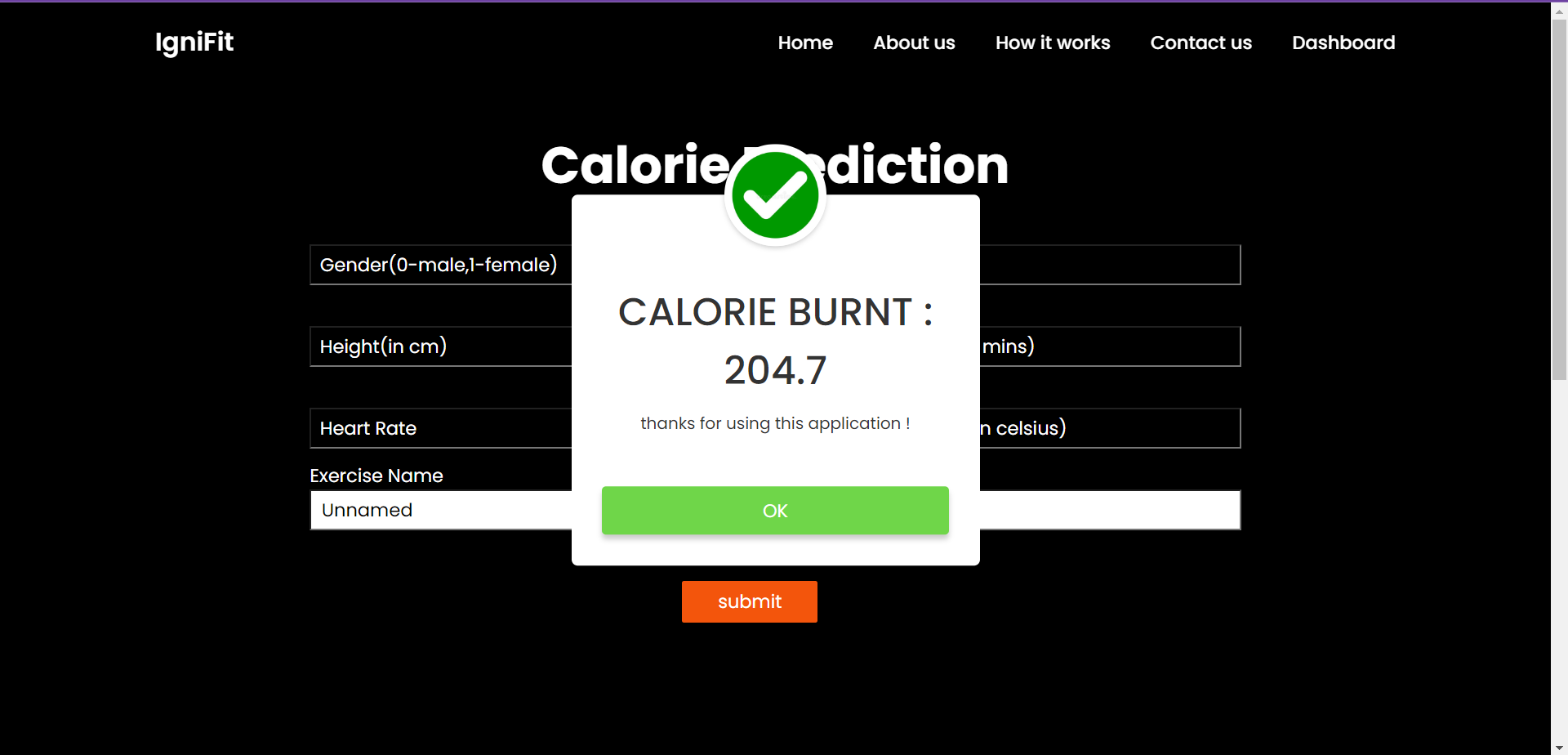












**10. ADVANTAGES & DISADVANTAGES**

* **Advantages:**

1. High Predictive Accuracy: The XGBoost Regressor model is renowned for its ability to provide highly accurate predictions, surpassing traditional regression models.
2. Non-Linear Relationship Handling: Excelling at capturing non-linear relationships, XGBoost is well-suited for modeling the intricate connections between input features and caloric expenditure.
3. Feature Importance Insight: XGBoost furnishes a feature importance score, facilitating the identification of key features influencing caloric expenditure predictions.

* **Disadvantages:**

1. Complex Parameter Tuning: Optimizing the many hyperparameters of XGBoost can be intricate, demanding meticulous tuning efforts.
2. Resource Intensiveness: Training XGBoost models, particularly with extensive datasets or complex configurations, may necessitate substantial computational resources and time.
3. Overfitting Risk: Without proper hyperparameter tuning and regularization, XGBoost models may succumb to overfitting, particularly in smaller datasets.

**11. CONCLUSION**

The purpose of this study was to determine how many calories our bodies burn, which is dependent on a number of variables like age, gender, height, weight, duration, and heart rate. To maintain our health and fitness, it is critical to comprehend how many calories we consume. Regression techniques like Random Forest, XGBoost regression, and Linear Regression can all be used to forecast the number of calories burned. Extreme Gradient Boosting, or XG Boost, regression produces the most accurate result out of any of these algorithms. The XG Boost's Mean Absolute Error (MAE) number is 1.68, which implies there were very few errors. Thus, the best method for predicting calories burnt at present is the XGBRegressor algorithm. The flexibility of the suggested technique was improved with variations. With the help of this, we are able to develop a fully functional website with a compatible user interface (UI), which is necessary to allow users to enter their values and receive results that display the number of calories they have burned in addition to other features like progress tracking.

**12. FUTURE SCOPE**

Using the XGBRegressor Model, the key features derived from the feature selection and evaluation process were also related to the problem domain, providing insights and implications for potential applications. However, the study also had some limitations, such as the limited size of the dataset and the possibility of overfitting. Future research could address these limitations and further improve the performance of the models and feature selection approaches.

**13. APPENDIX**

Source Code:  
  
app.py:

from flask import Flask, render\_template, request, redirect, url\_for, flash, session

import sqlite3

import os

from dashboard import fetch\_data, create\_bar\_chart, create\_line\_chart, create\_pie\_chart, create\_heart\_rate\_scatter\_plot

import pickle

from datetime import datetime

app = Flask(\_\_name\_\_)

app.config['DATABASE'] = 'Calorie.db'

app.secret\_key = 'TH15\_1S\_@\_S3CR3T\_K3Y'

def prediction(req):

print(req)

Gender=int(req['gender'])

Age = float(req['age'])

Height = float(req['height'])

Duration=float(req['duration'])

Heart\_Rate = float(req['heart\_rate'])

Body\_Temp = float(req['temperature'])

model=pickle.load(open('final\_model.pkl','rb'))

data=[[Gender,Age,Height,Duration,Heart\_Rate,Body\_Temp]]

print(data)

result = model.predict(data)

result = round(float(result),2)

return result

@app.route('/',methods=("GET","POST"))

def main():

if request.method=="POST":

result=prediction(request.form)

return render\_template('index.html',calories = result)

return render\_template('index.html')

@app.route('/login.html',methods=("GET","POST"))

def login():

if request.method=="POST":

login\_id=request.form['userid']

passwd = request.form['password']

login\_conn = sqlite3.connect(app.config['DATABASE'])

cur=login\_conn.cursor()

cur.execute('SELECT name,age,height,gender FROM users WHERE userid = ? AND password = ?', (login\_id,passwd))

result = cur.fetchone()

login\_conn.close()

if result:

print(result)

session['userid'] = request.form['userid']

session['name'] = result[0]

session['age'] = result[1]

session['height'] = result[2]

session['gender'] = result[3]

return redirect(url\_for('home'))

else:

print("Wrong")

flash("Wrong Id or Password")

return render\_template('login.html')

return render\_template('login.html')

@app.route('/register.html',methods=("GET","POST"))

def register():

if request.method=="POST":

user\_id=request.form['userid']

passwd = request.form['password']

name = request.form['name']

age = request.form['age']

height = request.form['height']

gender = request.form['gender']

register\_conn = sqlite3.connect(app.config['DATABASE'])

register\_cur=register\_conn.cursor()

register\_cur.execute('SELECT userid FROM users WHERE userid = ?', (user\_id,))

result = register\_cur.fetchone()

if result:

register\_conn.close()

flash("User already exists")

return render\_template('login.html')

else:

register\_cur.execute('insert into users values(?,?,?,?,?,?)',(user\_id,passwd,name,age,height,gender))

register\_conn.commit()

register\_conn.close()

session['userid'] = request.form['userid']

session['name'] = request.form['name']

session['age'] = request.form['age']

session['height'] = request.form['height']

session['gender'] = request.form['gender']

return redirect(url\_for('home'))

return render\_template('login.html')

@app.route('/home.html',methods=("GET","POST"))

def home():

if request.method=="POST":

result = prediction(request.form)

session['calories'] = result

exercise\_conn = sqlite3.connect(app.config['DATABASE'])

exercise\_cur=exercise\_conn.cursor()

exercise\_cur.execute('insert into exercise(exercise\_name,userid,duration,date,bpm,temperature,calories) values(?,?,?,?,?,?,?)',

(request.form['exercise\_name'],

session['userid'],

float(request.form['duration']),

datetime.today().date(),

float(request.form['heart\_rate']),

float(request.form['temperature']),

result))

exercise\_conn.commit()

exercise\_conn.close()

return render\_template('home.html',calories = result)

return render\_template('home.html',name = session['name'], age = session['age'], height = session['height'], gender = 0 if session['gender']=='Male' else 1)

@app.route('/dashboard.html',methods=("GET","POST"))

def dashboard():

user\_name = session.get('name')

userid = session.get('userid')

if user\_name:

exercise\_data, time\_data, calories\_data, heart\_data = fetch\_data(userid)

bar\_chart = create\_bar\_chart(exercise\_data)

line\_chart = create\_line\_chart(time\_data)

pie\_chart = create\_pie\_chart(calories\_data, title='Exercise Distribution and Calories Burned')

scatter\_plot = create\_heart\_rate\_scatter\_plot(heart\_data)

return render\_template('user\_dashboard.html',

username=user\_name,

bar\_chart=bar\_chart,

line\_chart=line\_chart,

pie\_chart=pie\_chart,

scatter\_plot=scatter\_plot

)

# Afterwards can just redirect it to the login page directly, or can display a

# Homepage with dev info/project info and login button.

@app.route('/')

def method\_name():

return "HELLO WORLD!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

dashboard.py

import plotly

import plotly.graph\_objs as go

import plotly.offline as opy

import sqlite3

import plotly.express as px

import json

def fetch\_data(userid):

conn = sqlite3.connect('Calorie.db')

cursor = conn.cursor()

# Example query: Retrieve total quantity sold for each product

query = "SELECT exercise\_name, duration FROM exercise WHERE userid = ?"

cursor.execute(query, (userid,))

data = cursor.fetchall()

# Example query: Retrieve total quantity sold over time

time\_query = "SELECT date, calories FROM exercise WHERE userid = ?"

cursor.execute(time\_query, (userid,))

time\_data = cursor.fetchall()

calories\_query = "SELECT exercise\_name, calories FROM exercise WHERE userid = ?"

cursor.execute(calories\_query, (userid,))

calories\_data = cursor.fetchall()

heart\_query = "SELECT bpm, calories FROM exercise WHERE userid = ?"

cursor.execute(heart\_query, (userid,))

heart\_data = cursor.fetchall()

cursor.close()

conn.close()

return data, time\_data, calories\_data, heart\_data

"""

def create\_bar\_chart(data):

exercise, duration = zip(\*data)

trace = go.Bar(x=exercise, y=duration)

layout = go.Layout(title='Exercise vs duration', xaxis=dict(title='Exercise'), yaxis=dict(title='Duration'))

fig = go.Figure(data=[trace], layout=layout)

return json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)

def create\_line\_chart(time\_data):

date, calories = zip(\*time\_data)

trace = go.Scatter(x=date, y=calories, mode='lines+markers', marker=dict(size=10), line=dict(width=2))

layout = go.Layout(title='Total Calories Over Time', xaxis=dict(title='Date'), yaxis=dict(title='Total Calories'))

fig = go.Figure(data=[trace], layout=layout)

return json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)

def create\_pie\_chart(data, title):

exercise\_names, calories = zip(\*data)

trace = go.Pie(labels=exercise\_names, values=calories)

layout = go.Layout(title=title)

fig = go.Figure(data=[trace], layout=layout)

return json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)

def create\_heart\_rate\_scatter\_plot(data):

bpm, calories = zip(\*data)

trace = go.Scatter(x=bpm, y=calories, mode='markers', marker=dict(size=12))

layout = go.Layout(title='Heart Rate vs. Calories Burned', xaxis=dict(title='Heart Rate (BPM)'), yaxis=dict(title='Calories Burned'))

fig = go.Figure(data=[trace], layout=layout)

return json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)

"""

def create\_bar\_chart(data):

exercise, duration = zip(\*data)

trace = go.Bar(x=exercise, y=duration)

layout = go.Layout(title='Exercise vs duration', xaxis=dict(title='Exercise'), yaxis=dict(title='Duration'))

fig = go.Figure(data=[trace], layout=layout)

return opy.plot(fig, auto\_open=False, output\_type='div')

def create\_line\_chart(time\_data):

date, calories = zip(\*time\_data)

trace = go.Scatter(x=date, y=calories, mode='lines+markers', marker=dict(size=10), line=dict(width=2))

layout = go.Layout(title='Total Calories Over Time', xaxis=dict(title='Date'), yaxis=dict(title='Total Calories'))

fig = go.Figure(data=[trace], layout=layout)

return opy.plot(fig, auto\_open=False, output\_type='div')

def create\_pie\_chart(data, title):

exercise\_names, calories = zip(\*data)

trace = go.Pie(labels=exercise\_names, values=calories)

layout = go.Layout(title=title)

fig = go.Figure(data=[trace], layout=layout)

return opy.plot(fig, auto\_open=False, output\_type='div')

def create\_heart\_rate\_scatter\_plot(data):

bpm, calories = zip(\*data)

trace = go.Scatter(x=bpm, y=calories, mode='markers', marker=dict(size=12))

layout = go.Layout(title='Heart Rate vs. Calories Burned', xaxis=dict(title='Heart Rate (BPM)'), yaxis=dict(title='Calories Burned'))

fig = go.Figure(data=[trace], layout=layout)

return opy.plot(fig, auto\_open=False, output\_type='div')

tpot\_cal.py:

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from xgboost import XGBRegressor

# NOTE: Make sure that the outcome column is labeled 'target' in the data file

tpot\_data = pd.read\_csv('PATH/TO/DATA/FILE', sep='COLUMN\_SEPARATOR', dtype=np.float64)

features = tpot\_data.drop('target', axis=1)

training\_features, testing\_features, training\_target, testing\_target = \

train\_test\_split(features, tpot\_data['target'], random\_state=None)

# Average CV score on the training set was: -1.3279745078720981

exported\_pipeline = XGBRegressor(learning\_rate=0.1, max\_depth=10, min\_child\_weight=10, n\_estimators=100, n\_jobs=1, objective="reg:squarederror", subsample=0.7500000000000001, verbosity=0)

exported\_pipeline.fit(training\_features, training\_target)

results = exported\_pipeline.predict(testing\_features)

Github and project demo link:

<https://github.com/smartinternz02/SI-GuidedProject-612456-1699198599>

<https://drive.google.com/file/d/1H8RXQrYtPv1RyqEEHisP6kG8ERAYZgo5/view?usp=sharing>