AI BASED EXERCISE RECOMMENDATION SYSTEM

UIT2511 – SOFTWARE DEVELOPMENT PROJECT – II

A PROJECT REPORT

Submitted by

HARISH K	**3122215002034**
HARISH RM	**3122215002035**

SSN COLLEGE OF ENGINEERING,

KALAVAKKAM

NOVEMBER 2023

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

BONAFIDE CERTIFICATE

Certified that this project titled "AI Based Exercise Recommendation System" is the bonafide work of " HARISH K 3122215002034 HARISH RM 312215002035 ", and is submitted for project viva-voce examination held on

Signature of examiner(s)

Submitted on

Internal Examiner **External Examiner**

AI BASED EXERCISE RECOMMENDATION SYSTEM

1.ABSTRACT

In today's era of rapid technological advancement and increasingly sedentary lifestyles, the importance of regular physical exercise for maintaining health and well-being cannot be overstated. However, the path to achieving optimal fitness is often hindered by the daunting complexity of exercise choices, misinformation, and a lack of personalized guidance. This project presents a comprehensive solution in the form of a web-based exercise recommendation system that integrates artificial intelligence (AI) algorithms to provide intelligent, personalized, and adaptable exercise recommendations to users.

At its core, the project addresses a multifaceted problem. The first challenge is the lack of personalization in exercise recommendations. Traditional, one-size-fits-all fitness programs overlook the unique characteristics of individuals, such as age, gender, current fitness levels, medical conditions, and specific health goals. This lack of personalization often results in exercises that may not be suitable, effective, or safe for individual users.

The second challenge is the information overload in the digital age. An abundance of fitness information, much of it contradictory, inundates individuals through online sources and social media. This information overload can create confusion and hinder effective decision-making, making it challenging for individuals to discern accurate, evidence-based advice from fads and misinformation.

The third challenge lies in the complexity of exercise choices. The diversity of exercise types, each with its own benefits and considerations, poses a complex challenge for individuals seeking an exercise routine that aligns with their specific needs and goals. Selecting the right exercises becomes a time-consuming and uncertain process.

Ultimately, the project aims to make a profound impact on the health and well-being of individuals. By simplifying exercise choices and making them more accessible, it seeks to motivate people to embrace physical activity, contributing to a society that is not only more informed about fitness but also more committed to a healthier future. The project embodies the fusion of technology and wellness, designed to inspire and empower individuals to make informed and sustainable exercise choices, aligning with the growing trend of AI-based health applications and promising to revolutionize the way individuals approach fitness and exercise routines in the digital age.

2.INTRODUCTION

In an age characterized by technology-driven convenience, we often find ourselves grappling with the paradox of modern life: while our daily routines become increasingly sedentary, the importance of regular physical exercise for our health and well-being remains as paramount as ever. For many individuals, the prospect of embarking on a fitness journey can be a daunting and perplexing endeavor. The abundance of exercise options, coupled with varying fitness goals and unique medical conditions, presents a complex puzzle that can be challenging to solve.

The answer to this challenge lies at the intersection of artificial intelligence (AI) and knowledge representation systems, offering a web-based exercise recommendation system that is both personalized and intelligent. This project seeks to create a dynamic and adaptable platform that leverages cutting-edge AI algorithms and a comprehensive knowledge graph, empowering individuals to embark on fitness journeys tailored to their specific needs and preferences.

Vision and Impact

The vision for this project is rooted in the belief that technology can be harnessed to promote healthier, more active lives. By democratizing access to personalized exercise recommendations, we aspire to motivate individuals to embrace physical activity, thereby contributing to a society that is not only more informed about fitness but also more committed to a healthier future.
As the world continues to recognize the importance of health and well-being, the development of this web-based exercise recommendation system represents a significant step towards ensuring that individuals are equipped with the knowledge and guidance to take charge of their fitness journeys. It's a convergence of technology and wellness, designed to inspire and empower, and it promises to revolutionize the way we approach exercise and well-being in the digital age.

MOTIVATION

In an age where convenience and technology have become integral parts of our lives, the paradox of modern living is strikingly evident. On one hand, we are empowered by the limitless possibilities of digital innovation, making our lives more comfortable and efficient. Yet, on the other hand, our routines have become increasingly sedentary, leading to a concerning rise in health issues, most notably obesity and its related complications. It is no secret that regular physical exercise remains one of the most effective and proven methods to combat these health challenges. However, a stark dilemma persists - as the importance of exercise has grown, so too has the complexity of embarking on a fitness journey.

For many, the task of selecting an appropriate exercise routine can be a bewildering journey into the unknown. The sheer number of exercise options, the myriad of fitness goals, and the diverse medical conditions that individuals may face present a challenging puzzle to solve. Moreover, as the world grows more interconnected and digitized, a deluge of information on health and fitness inundated us daily, often causing more confusion than clarity. This conundrum begs the question: How can we harness the very technology that both simplifies and complicates our lives to make informed choices about our physical well-being?

The motivation behind this project is deeply rooted in the recognition of this pressing need for personalized and intelligent exercise recommendations. It's about breaking through the complexity and confusion surrounding fitness choices, empowering individuals to embark on tailored, sustainable, and safe fitness journeys. This project seeks to provide a holistic solution that bridges the gap between our innate desire for a healthier lifestyle and the vast landscape of exercise possibilities.

In conclusion, the motivation behind this project is deeply rooted in the desire to address a pressing need for personalized, intelligent, and data-driven exercise recommendations in the digital age. It's about enabling individuals to embrace physical activity with confidence and commitment. It's about making health choices less daunting and more empowering. It's about leveraging technology to steer us towards a future where health and fitness are not burdens but joys, where each step taken is a step toward a healthier, happier life.

The importance of this initiative cannot be overstated. It's a response to the challenges of our times, a bridge between the complexities of fitness and the simplicity of technology, and a call to empower individuals to take control of their health and well-being in a world that is increasingly characterized by both opportunities and challenges.

PROBLEM STATEMENT

The challenge we confront is the absence of a personalized and data-driven solution for exercise recommendations in a world inundated with fitness information. With a lack of tailored guidance, individuals struggle to make informed exercise choices, leading to suboptimal fitness routines and missed health benefits. The goal of this project aims to create a web-based exercise recommendation system that employs AI algorithms to provide individualized, adaptive, and evidence-based exercise recommendations, empowering users to achieve their health and fitness goals with confidence and efficiency.

PROJECT OBJECTIVES

1. Develop an Exercise database: Create a database that encompasses a wide range of exercise types, their benefits, and their suitability for various fitness goals, age groups, and health conditions. Populate this database with trustworthy and up-to-date information.

2. User Profiling and Personalization: Implement a user profiling system where users can input information such as age, gender, fitness level, health conditions, and exercise preferences. Utilize this data to provide personalized exercise recommendations.

3. AI-Powered Recommendation Engine: Develop AI algorithms that analyze user profiles and exercise database data to generate intelligent exercise recommendations. These recommendations should consider the individual's goals, limitations, and preferences.

4. Dynamic Adjustment of Recommendations: Create a system that adapts and updates exercise recommendations as users' fitness levels, goals, or health conditions change over time, ensuring that the exercise routines remain effective and safe.

5. User-Friendly Web Interface: Design and implement an intuitive web interface that allows users to access their exercise recommendations easily, view detailed instructions, and track their progress.

6. Educational Resources: Provide supplementary educational content to help users understand the benefits of recommended exercises and their impact on health and fitness.

7. Promote Health and Well-Being: Ultimately, the primary objective of the project is to promote health and well-being by empowering users to make informed and sustainable exercise choices, thereby contributing to a healthier and more active society.

3.REQUIREMENTS ENGINEERING

Client details

Name : Dr. K.S. Gayathri
B.E., M.E., Ph.D.

Profession : Assistant Professor, Department of Information Technology

List of all functional modules:

Sprint #	Epic	Essential or Desirable	Description of the Requirement	Remarks
1	Customer signup/login	Essential	Customer needs to sign up/login to access exercises	Login credentials are stored and accessed from database 'Customer.db'
1	App Data Integration Genetic Algorithm	Essential	Exercise related data has to be taken from live apps and public dataset	Exercises for workout plan have been taken from existing apps in the playstore

1	App Data Integration CSP	Essential	Exercise related data has to be taken from live apps and public dataset	Exercise data has been taken by referring websites
2	Genetic algorithm	Essential	This algorithm is used to generate a 5 day workout plan for the user.	Genetic algorithm helps in creating a plan and generates the top most plan
2	CSP algorithm	Essential	This algorithm is used to generate exercises for specific body parts as requested by the user.	This take into considerations of various constraints given by the user and generate the best exercises for the user
3	User Forms	Essential	These forms are used to take input from the user for both genetic and csp generations of algorithms	User details are taken from user and used for exercise generation.

3	Database integration	Essential	Customers can login/signup through email id and password.	Credentials are validated through data in the database.
4	Home, support, explore page	Essential	Webpages to get to know about our project	Making the user comfortable to use our website application.
4	Final recommendation pages-submit ages	Essential	These are the pages where the user gets his/her recommended exercises.	These are the final webages of our application.
4	Graph Generation	Desirable	Diagrammatic charts to explain the results	User to get more insights about how the system devises a solution

Test Log Report

S.No	**Test case**	**Description**	**Expected Result**	**Outcome**
1	Login Check	The username and password must be of correct form	Error if the password is not correct	Passed
2	Registration check	The mail ,password,name must be of correct of form	Error if the input details is not of correct form	passed
3	The number of input fields	Checks if the user has entered a required number of input field for unique solution	Raise error if inadequate input filler	Passed
4	Solve csp	Solves any valid scenario with the	Displays valid solution	Passed

		given user's input		
5	solve genetic	generates a 5 day workout plan for the user's input	recommends valid solution	Passed
6	Graph generation	Various graphs are generated for the given user details and results	Graph image gets loaded on the web application	Passed

4.Implementation and Risk Management

Name : Harish K

Register number : 3122215002034

Role: Developer

A. Implementation

Sprint#	Epic	User story	Requirement	Remarks
1	Exercise recommendation using CSP	For the given inputs from the user the algorithm should recommend the exercise which satisfies the most constraints	A dataset of exercises and variables,domains and constraints for the algorithm to recommend the best exercise	Applied CSP for recommendation of exercise
2	Designing of pages	A user friendly interface with minimal design	The user should easily navigate through pages and should be able to enter the inputs	Used HTML,CSS for designing the pages
3	Integration	Integrate the frontend with the backend	The inputs from the user should be fed to the algorithm to recommend the best exercise	Used Flask for the integration of the web pages

B. Risk Management

Risk #	Risk Description	Probability	Impact	Mitigation Plan
1	Defining the constraints in order of the priority	High	Wrong inputs leading to inaccurate output	The constraints have been ordered from the highest to lowest priority

Name : Harish Rm

Register number : 3122215002035

Role: Developer

A. Implementation

Sprint#	Epic	User story	Requirement	Remarks
1	Database Initialization and integration.	Users have to login/signup for getting their set of recommended exercises	Customer should be able to sign up/login by giving details and can change his/her details.	Learnt about database creation and utilization.
2	Genetic algorithm	users can get a personalized full body workout plan. This plan is generated using the genetic algorithm	Genetic algorithm devises a set of 50 full body exercises and the best 5 are displayed.	Learnt about genetic algorithms and the various steps involved in it.
4	Home page	A user friendly interface to access our application	Customer should be able comfortable in accessing the website	Learnt about html and css styling

4	Graph Generation	User to get more insights about how the system devises a solution	Diagrammatic charts to explain the results	Graph image gets loaded on the web application

B. Risk Management

Risk #	Risk Description	Probability	Impact	Mitigation Plan
1	Same set of exercises getting repeated	High	Users get the same set set of exercises for majority of the days	Various conditions are implemented and set data structure has been used.

Software Requirements

1. Python: Implementing AI algorithms
2. HTML, CSS and JavaScript: For building the front-end user interface of the system.
3. Flask: Choose a web framework for the back-end development to handle HTTP requests, route traffic, and interact with the database.
4. Sqlite3: A database to store the user mail id and password
5. Jira : For project tracking and management.

5. Project Management

Agile methodology is an iterative and collaborative approach to project management and software development. It emphasizes flexibility, adaptability, and customer collaboration throughout the development process.

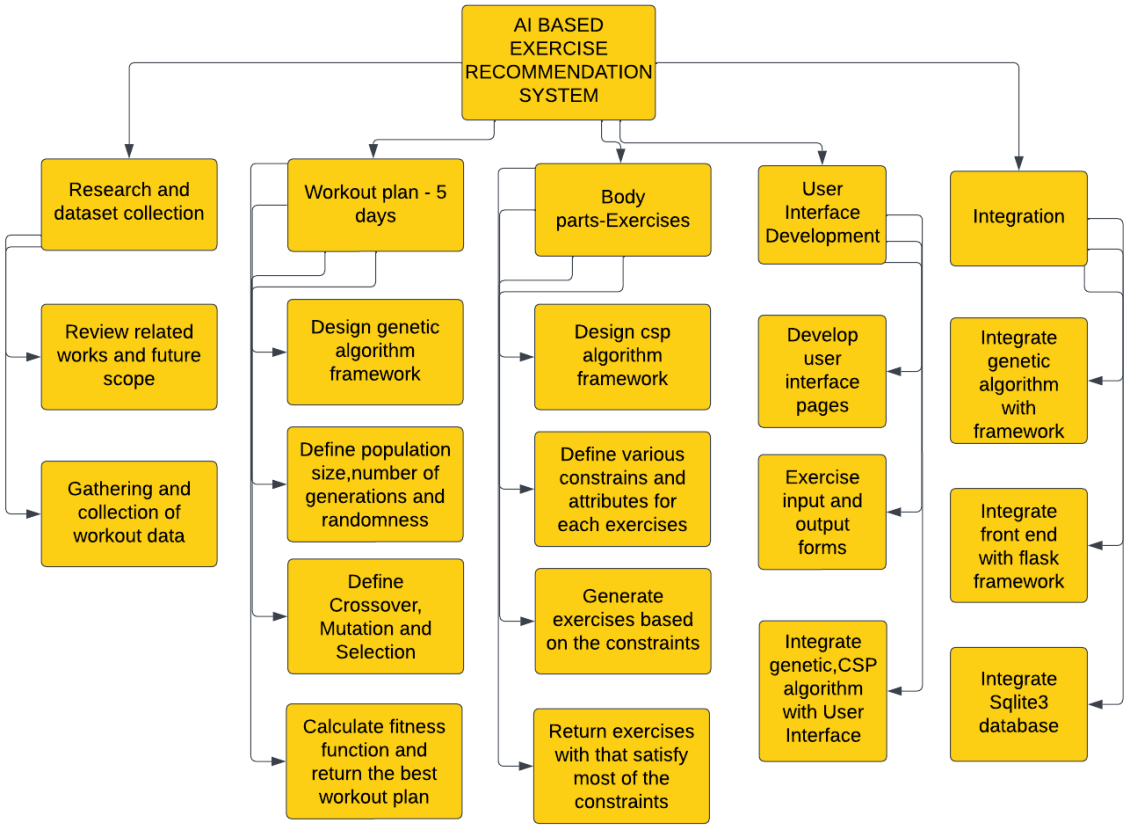
The core principles of Agile revolve around delivering value in short iterations called sprints, embracing change, and fostering continuous improvement. Agile teams work in a highly

collaborative and cross-functional manner, with regular feedback loops and frequent communication. This methodology promotes transparency, empowers team members to make decisions, and enables rapid response to changing requirements. By embracing Agile, organizations can enhance productivity, quality, and customer satisfaction by delivering incremental and iterative solutions.

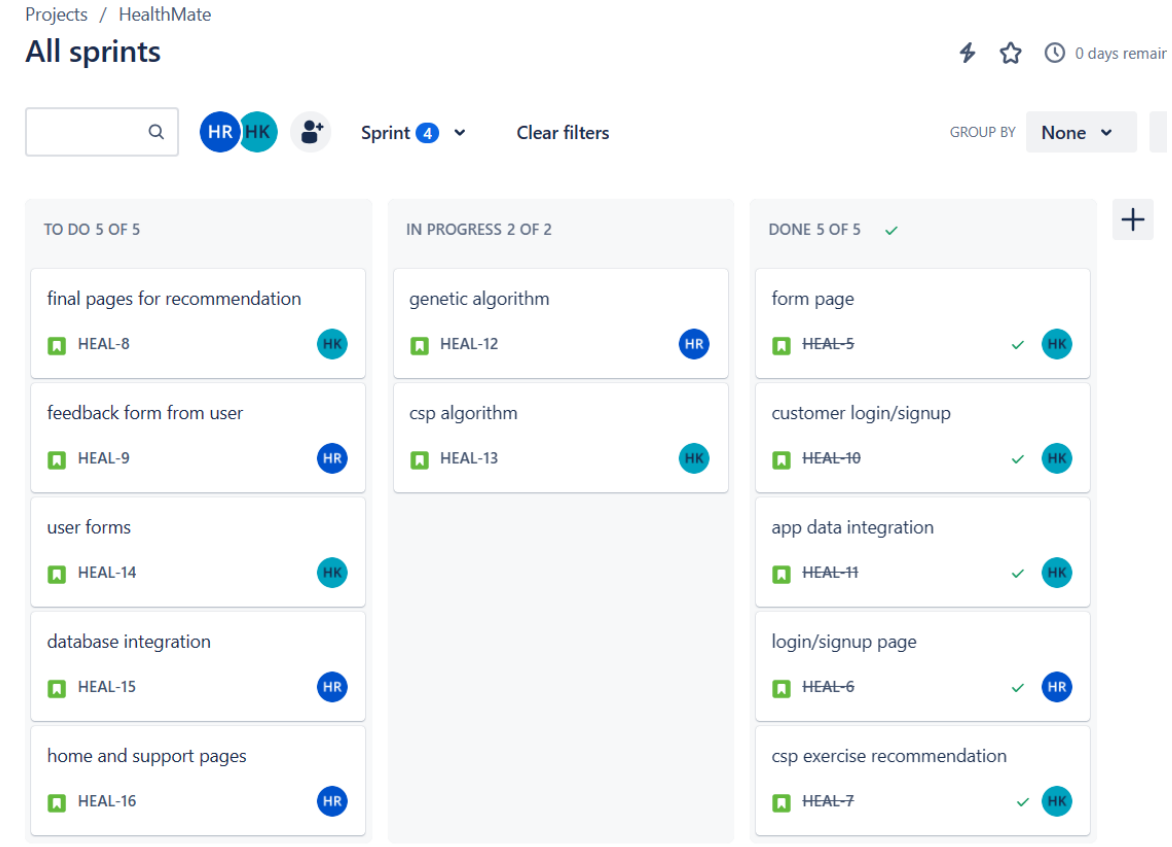
Scrum is an Agile framework that focuses on iterative and incremental product development. It involves self-organizing, cross-functional teams that work in short iterations called sprints. Daily stand-up meetings, product backlogs, and sprint reviews are key components of Scrum, promoting transparency, collaboration, and rapid delivery of value.

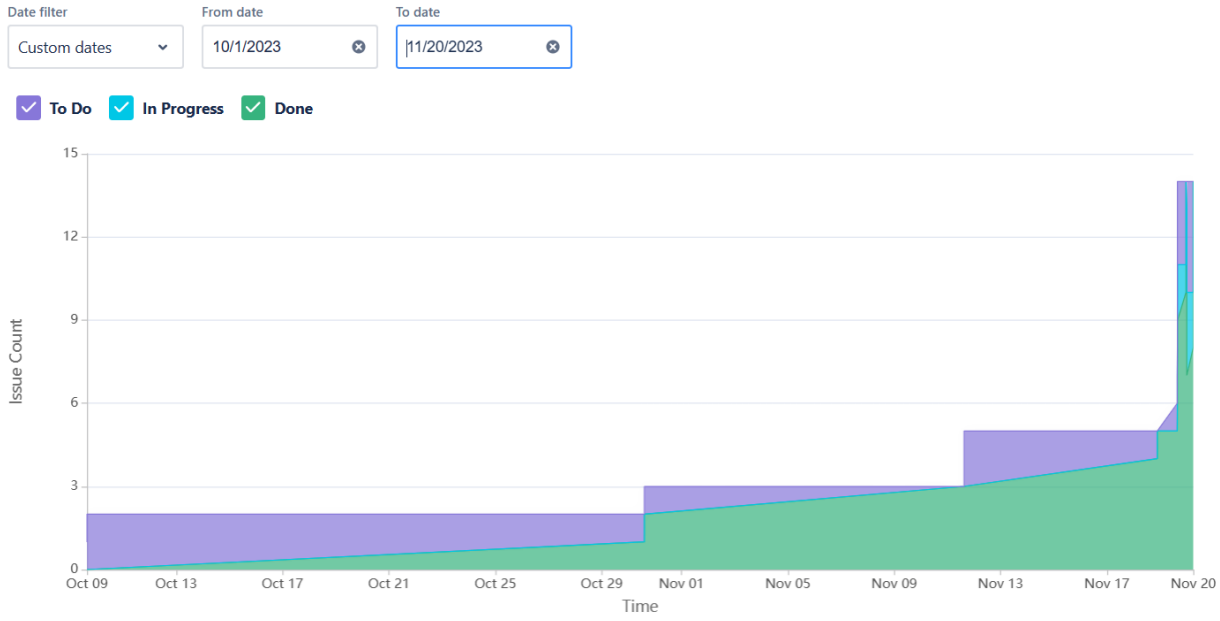
Through the course of this project, we have adopted Scrum Methodology of working by incorporating our project into various sprints and organizing our requirement lists into user stories and assigning them to each team member. We have implemented Scrum methodology using JIRA software.

WORK BREAKDOWN CHART

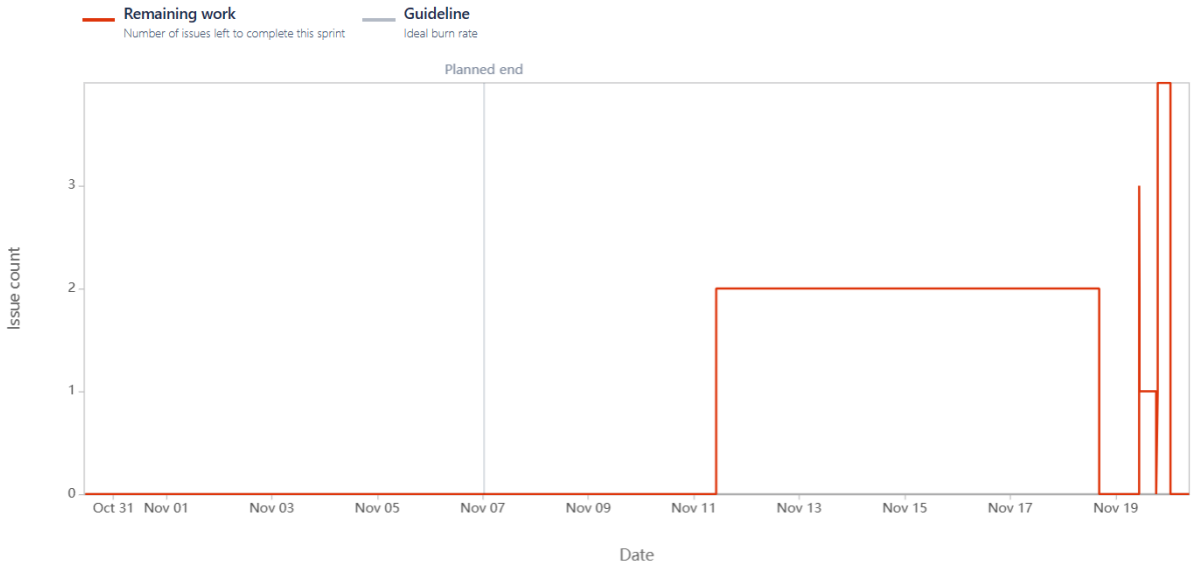


JIRA SCREENSHOTS

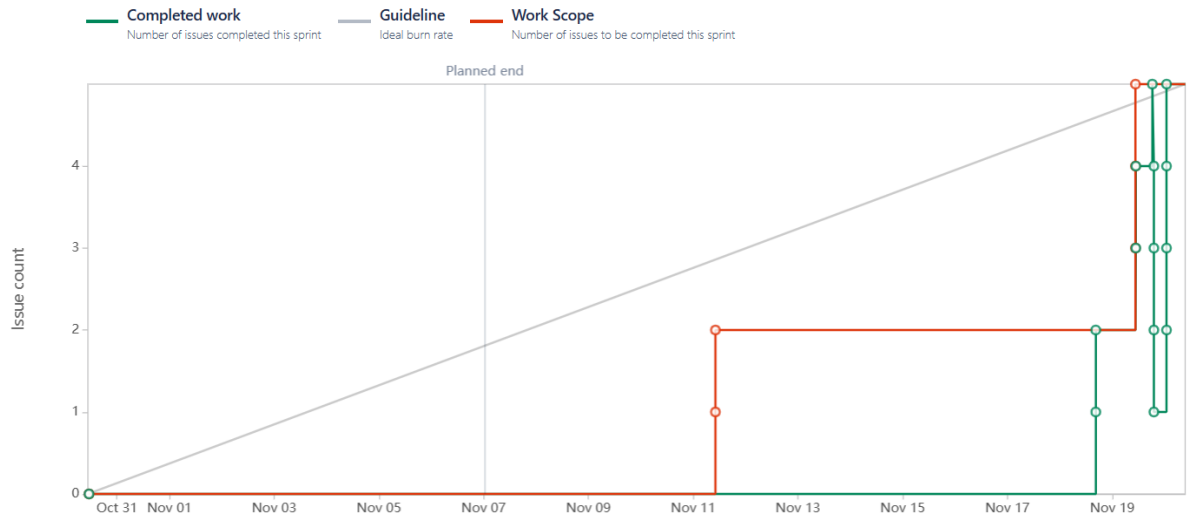




Date - October 30th, 2023 - November 7th, 2023



Date - October 30th, 2023 - November 7th, 2023



6. Project Outcomes

CODE SNIPPETS

csp.py

app.py csp.py submit.html genetic_final.html

csp.py > calculate_best_exercise

```python
def calculate_best_exercise(equipment, duration, intensity, calories, target_muscle, i, variables):
    c = 0
    if i < len(domain):
        exercise = domain[i]
        if equipment.lower() in dataset[exercise][4]:
            c += 1
            if duration >= dataset[exercise][0] and duration <= dataset[exercise][1]:
                c += 1
                if calories >= dataset[exercise][5] and calories <= dataset[exercise][6]:
                    c += 1
                    if intensity >= dataset[exercise][2] and intensity <= dataset[exercise][3]:
                        c += 1
                        if target_muscle.lower() in dataset[exercise][-1]:
                            variables.append([c, exercise])
                            calculate_best_exercise(
                                equipment, duration, intensity, calories, target_muscle, i + 1, variables
                            )
                            return variables
                        else:
                            variables.append([c, exercise])
                            calculate_best_exercise(
                                equipment, duration, intensity, calories, target_muscle, i + 1, variables
                            )
                            return variables
                    else:
                        variables.append([c, exercise])
                        calculate_best_exercise(
                            equipment, duration, intensity, calories, target_muscle, i + 1, variables
                        )
                        return variables
                else:
                    variables.append([c, exercise])
                    calculate_best_exercise(
                        equipment, duration, intensity, calories, target_muscle, i + 1, variables
                    )
                    return variables
            else:
                variables.append([c, exercise])
                calculate_best_exercise(equipment, duration, intensity, calories, target_muscle, i + 1, variables)
```

genetic.py

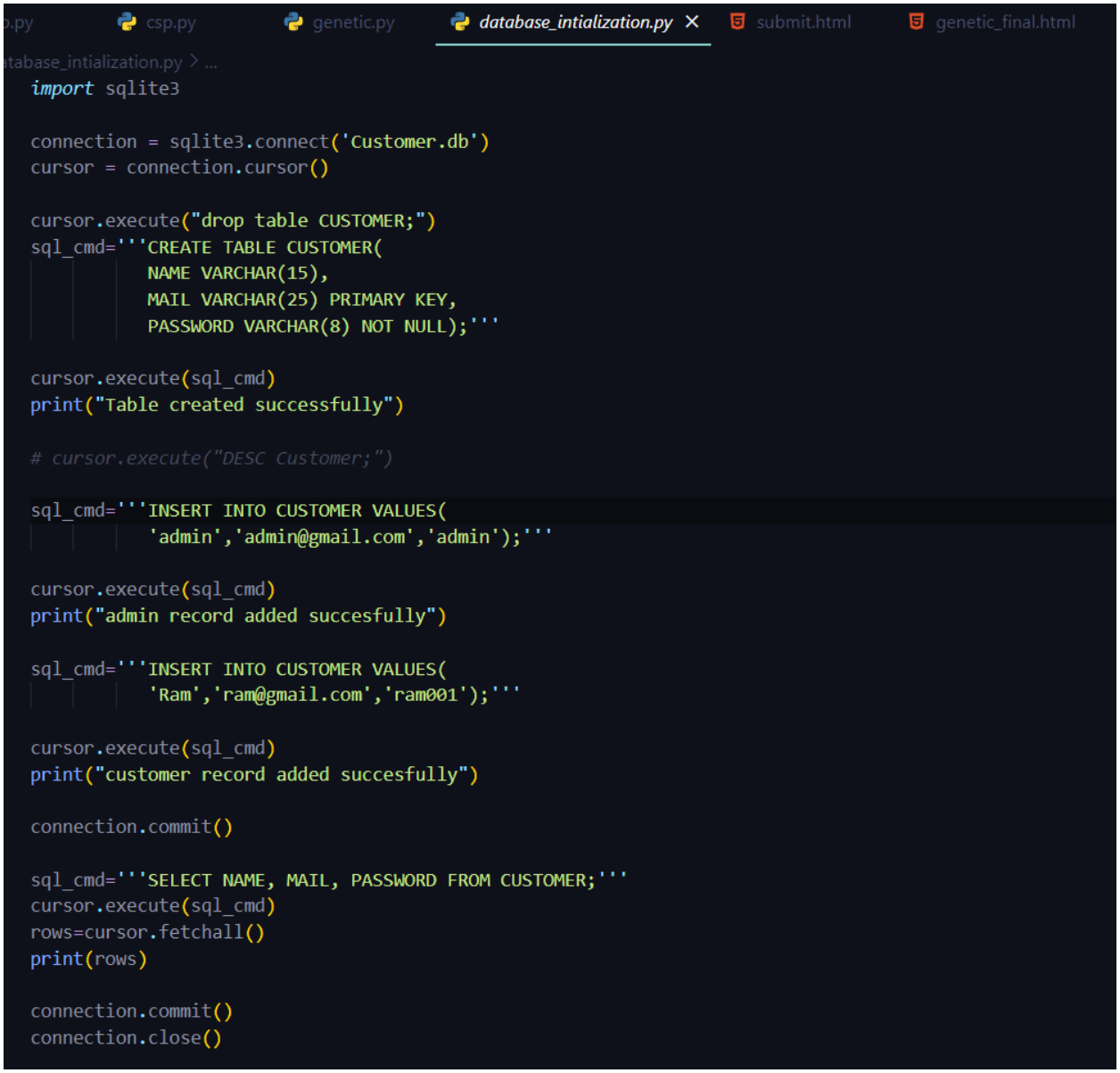
app.py csp.py genetic.py submit.html genetic_final.html

genetic.py > ...

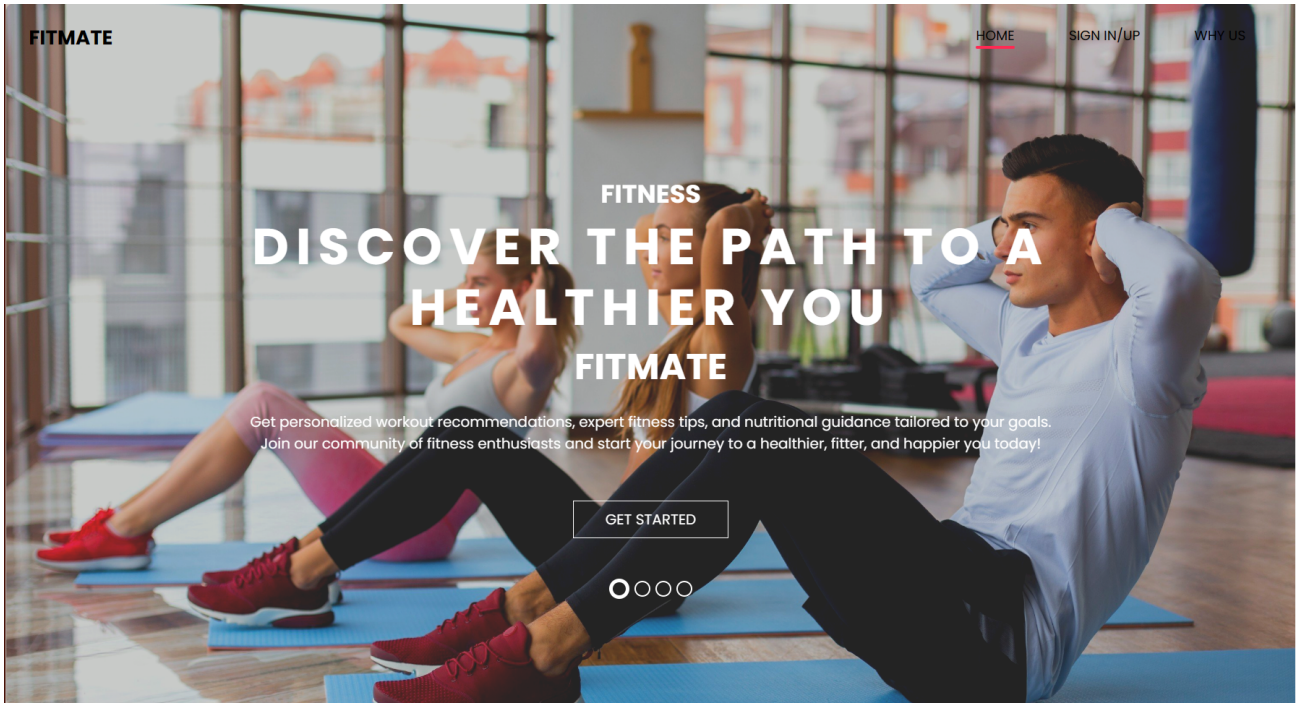
```python
def genetic_algorithm(level,pushup):
    global a,n
    if level=='Beginner':
        a=0
        n=3
    elif level=='Intermediate':
        a=3
        n=7
    else:
        a=5
        n=9
    global pushups
    pushups=pushup
    population = [generate_individual(level) for _ in range(population_size)]
    for generation in range(num_generations):
        fitness_scores = [calculate_fitness(individual) for individual in population]
        parents=[]
        for i in range(population_size):
            if random.random() < (fitness_scores[i]/sum(fitness_scores)):
                parents.append(population[i])

        if not parents:
            best_individual_index = fitness_scores.index(max(fitness_scores))
            parents.append(population[best_individual_index])
        children = []
        while len(children) < population_size - len(parents):
            parent1 = random.choice(parents)
            parent2 = random.choice(parents)
            child = crossover(parent1, parent2)
            children.append(child)
        for i in range(len(children)):
            if random.random() < 0.1:
                children[i] = mutate(children[i])
        population = parents + children
    best_plan=max(population, key=calculate_fitness)

    return best_plan,calculate_fitness(best_plan)
```

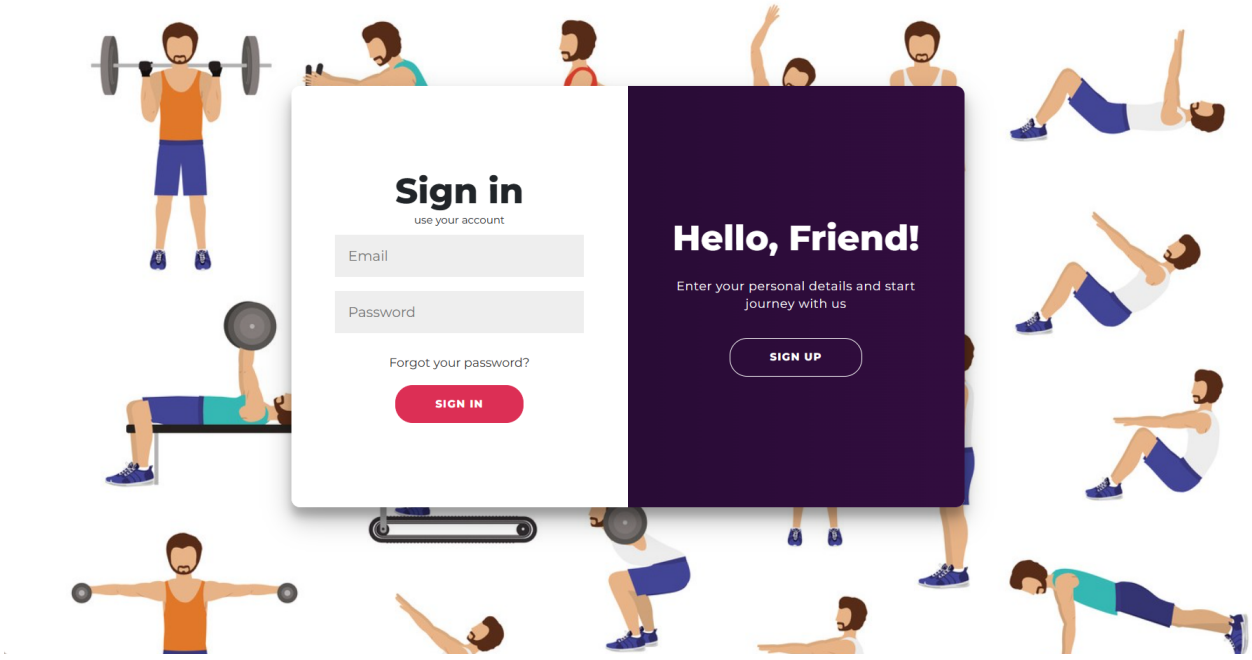
```python
@app.route('/gettoknow', methods=['GET','POST'])
def gettoknow():
    msg=''
    global age,skill,push
    global total_plan
    if request.method == 'POST':
        age=int(str(request.form['age']))
        skill=str(request.form['skill-level']).capitalize()
        push=int(str(request.form['pushups']))
        if (15<age<25) and skill=='Beginner':
            skill='Intermediate'
            push+=3
        if (15<age<25) and skill=='intermediate':
            skill='Advanced'
            push+=2
        num_days=5
        total_plan=[]
        fit_score=[]
        day=0
        while day < num_days:
            best_plan,fitness = genetic.genetic_algorithm(skill,push)
            day_plan=[]
            for category, exercise in best_plan.items():
                day_plan.append((category,exercise,fitness))
            if fitness not in fit_score:
                total_plan.append(day_plan)
                fit_score+=[fitness]
                day+=1
        return render_template('genetic_exer.html')
    else:
        return render_template('p_details.html',msg=msg)
```

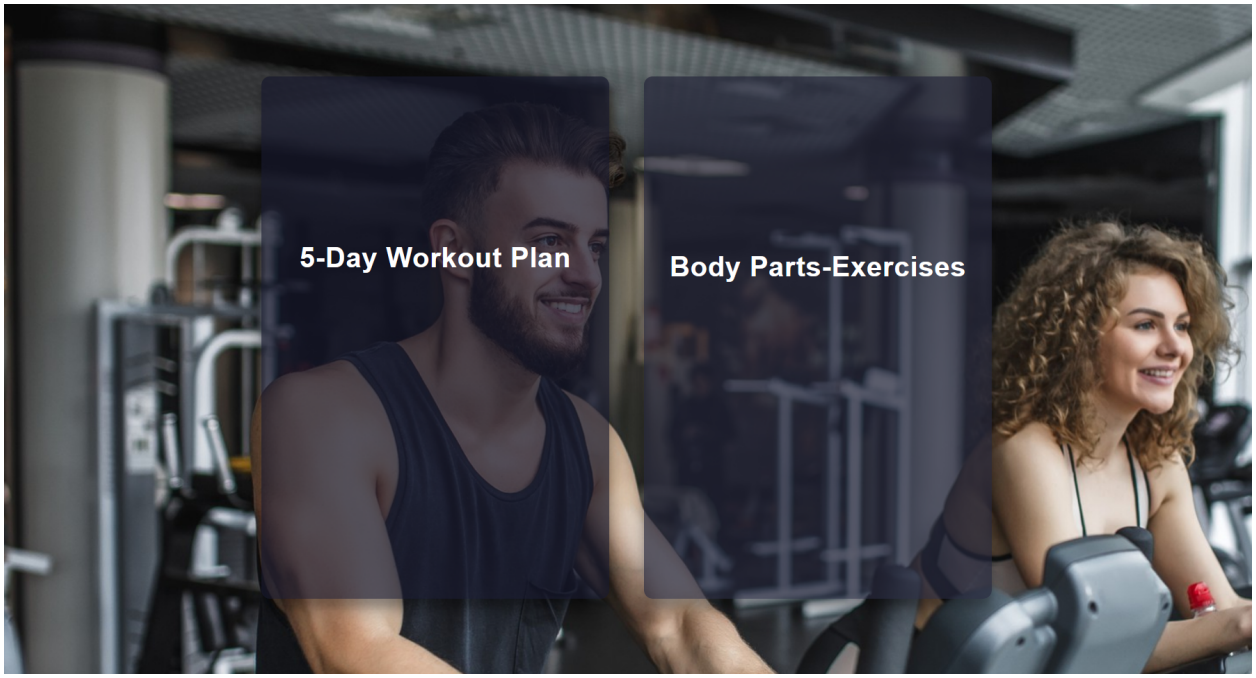
database creation and initialization

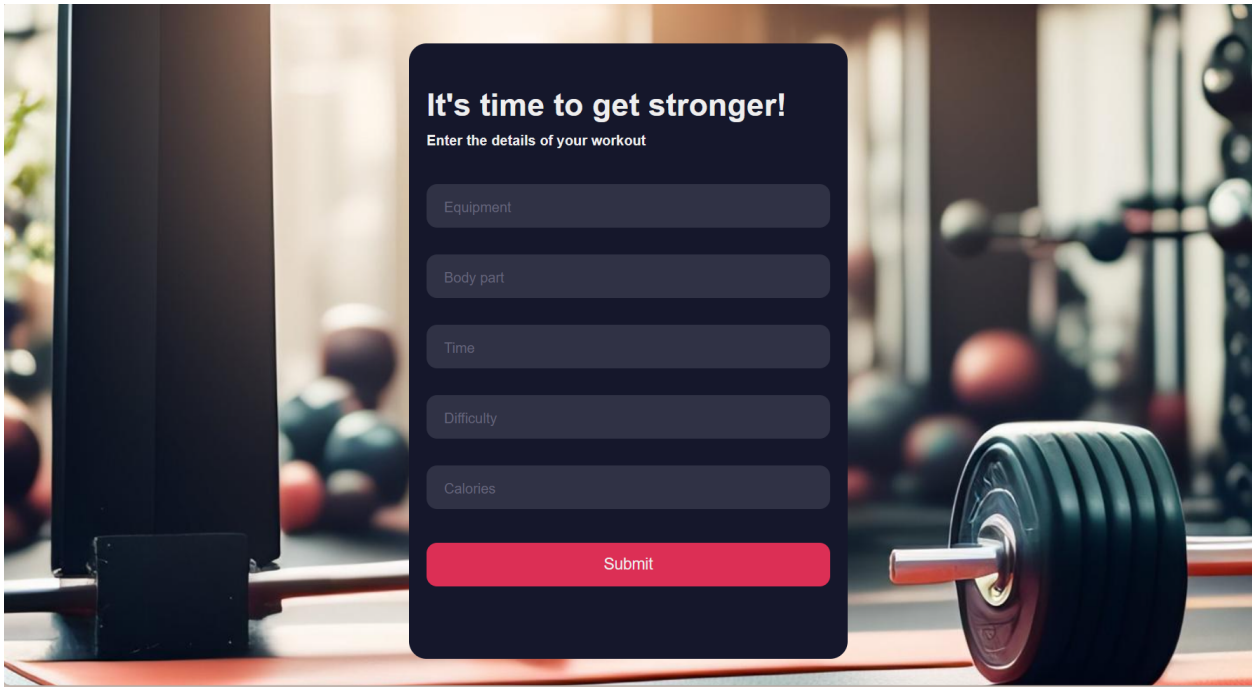


OUTPUT SCREENSHOTS

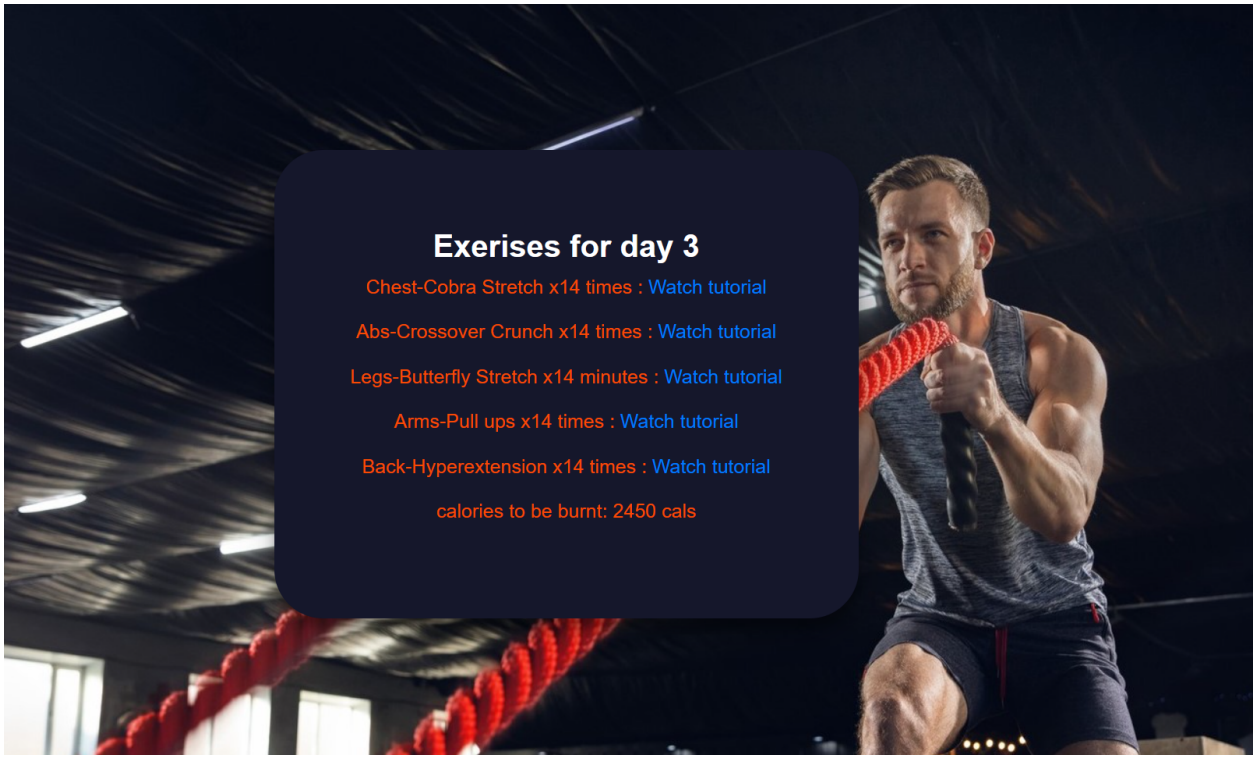


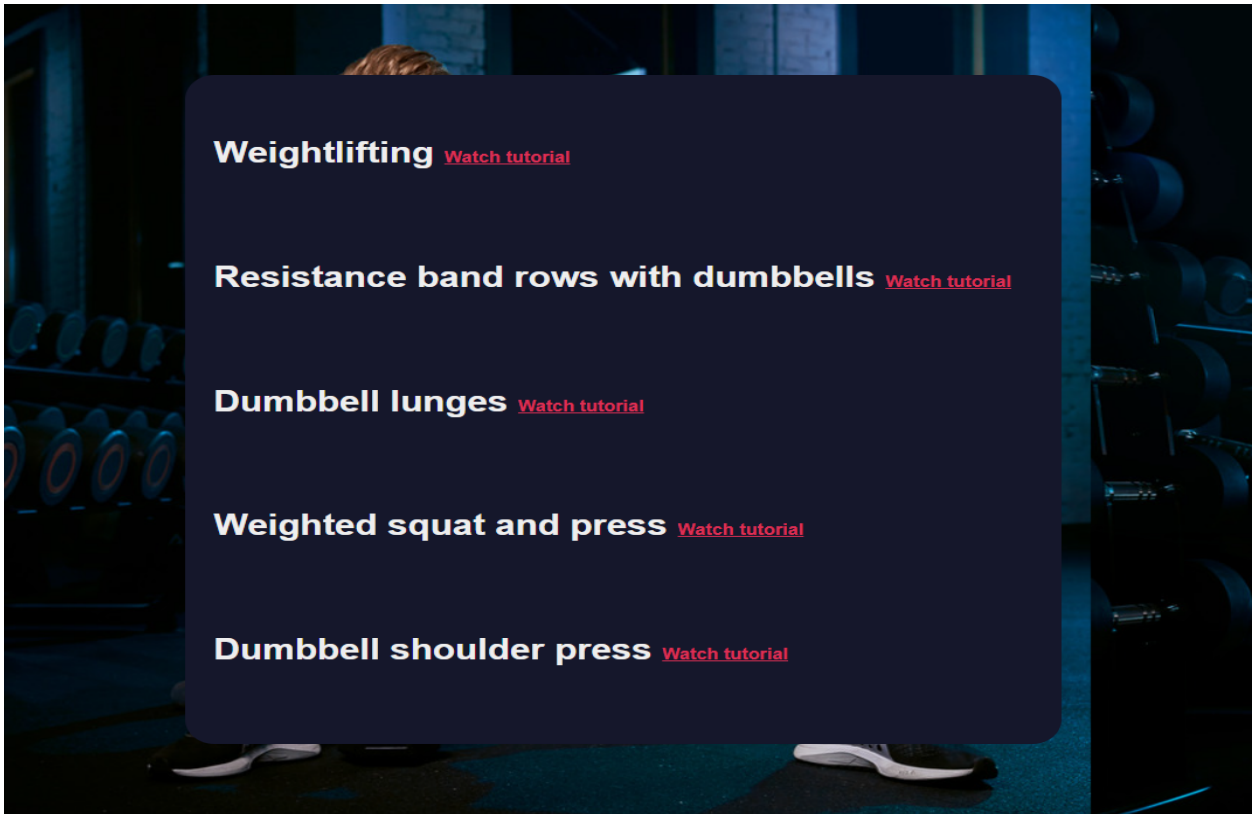



5-Day Workout Plan
Body Parts-Exercises

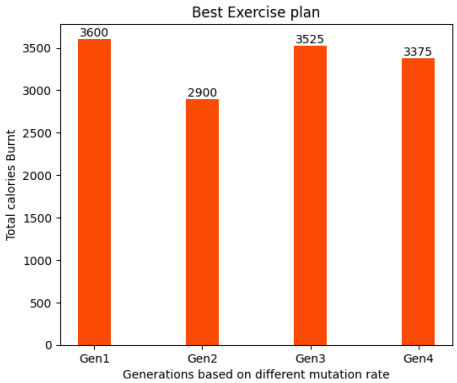

It's time to get stronger!
Enter the details of your workout
Equipment
Body part
Time
Difficulty
Calories
Submit


Workout plan Generated !
Get your Exercises
Day 1
Day 2
Day 3
Day 4
Day 5


Exerises for day 3
Chest-Cobra Stretch x14 times : Watch tutorial
Abs-Crossover Crunch x14 times : Watch tutorial
Legs-Butterfly Stretch x14 minutes : Watch tutorial
Arms-Pull ups x14 times : Watch tutorial
Back-Hyperextension x14 times : Watch tutorial
calories to be burnt: 2450 cals



barplot and pieplot screenshots



7. Conclusion and Future Directions

In addressing the dearth of personalized and data-driven exercise recommendations, this project has successfully deployed a web-based system integrating genetic algorithms and Constraint Satisfaction Problem (CSP) algorithms. The resulting platform delivers individualized, adaptive, and evidence-based exercise recommendations, providing users with tailored guidance for optimal fitness routines. By leveraging artificial intelligence, the system dynamically evolves exercise plans through genetic algorithms while refining recommendations with CSP algorithms, offering users a comprehensive and personalized fitness experience. As the project concludes, it represents a pivotal achievement in empowering individuals to make informed exercise choices, mitigating the challenges of navigating the flood of fitness information. The success of this endeavor underscores the potential of AI-driven solutions in revolutionizing personalized health and fitness guidance. Moving forward, the system's continual improvement, user feedback integration, and exploration of real-time data and advanced AI techniques promise a future where technology becomes an indispensable ally in achieving health and fitness goals.

Future Directions:

While the current project has laid a solid foundation, future development and enhancement involves:

1. Expanding the Exercise Database with many new workouts.

2. User Feedback and Iterative Improvement:

    - Implement mechanisms to collect user feedback on recommended exercises.
    - Incorporate a feedback loop to iteratively improve the recommendation system based on user preferences and outcomes.

3. Real-time Data Integration:

    - Consider integrating real-time data, such as heart rate, fitness tracking, or biometric data, to provide even more personalized recommendations.
    - Leverage IoT devices for seamless data collection and analysis.

4. Diversification of Recommendation Criteria:

    - Expand recommendation criteria to include factors like user mood, stress levels, and external factors (e.g., weather) to offer holistic and adaptive recommendations.

5. Mobile Application Development:

    - Extend the system to a mobile application to reach a broader audience and provide on-the-go recommendations.

8. References

1. https://www.fitnessai.com/
2. A.I. Workout Generator: ChatGPT Gives Training Advice - The Barbell
3. Wikipedia, Genetic Algorithm. [Online] Genetic algorithm - Wikipedia
4. Banos, O., et al. (2014). "mDurance: A Novel Mobile Health System to Support Trainers and Managers of Team Sports."
5. Chen, Z., et al. (2017). "An Adaptive Personalized Fitness Plan Recommendation System."
6. Yang, C., et al. (2017). "A Survey of Big Data Architectures and Machine Learning Algorithms in Healthcare."

EXERCISE – 1

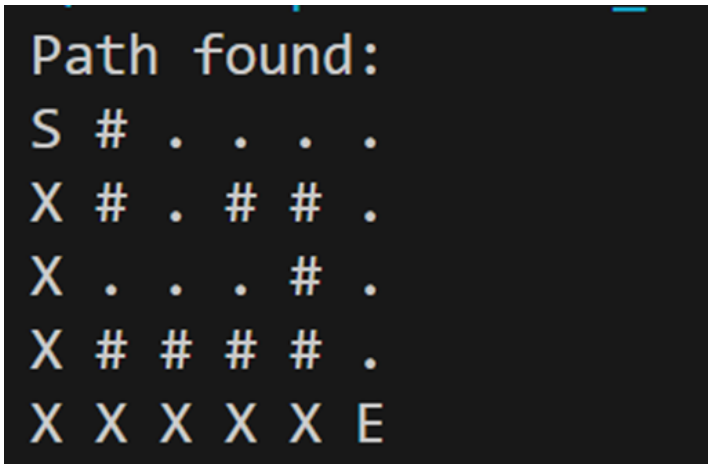
1) ROBOT – MAZE PROBLEM

CODE:

```
from collections import deque

def bfs_maze_solver(maze, start, end):
    def is_valid_move(x, y):
        return 0 <= x < len(maze) and 0 <= y < len(maze[0]) and
maze[x][y] == 0

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    queue = deque([(start[0], start[1], [])])
    visited = set()

    while queue:
        x, y, path = queue.popleft()
        if (x, y) == end:
            return path + [(x, y)]
        if (x, y) in visited:
            continue
        visited.add((x, y))
        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if is_valid_move(new_x, new_y):
                new_path = path + [(x, y)]
                queue.append((new_x, new_y, new_path))
    return None

def print_maze_with_path(maze, path):
    for i in range(len(maze)):
        for j in range(len(maze[0])):
            if (i, j) == path[0]:
                print("S", end=" ")
            elif (i, j) == path[-1]:
                print("E", end=" ")
            elif (i, j) in path:
                print("X", end=" ")
            elif maze[i][j] == 0:
                print(".", end=" ")
            else:
                print("#", end=" ")
        print()
```

```
if __name__ == "__main__":

    maze = [
        [0, 1, 0, 0, 0, 0],
        [0, 1, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 0],
        [0, 1, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 0],
    ]

    start = (0, 0)
    end = (4, 5)
    path = bfs_maze_solver(maze, start, end)

    if path:
        print("Path found:")
        print_maze_with_path(maze, path)
    else:
        print("No path found.")
```

OUTPUT:



2) TIC TAC TOE PROBLEM

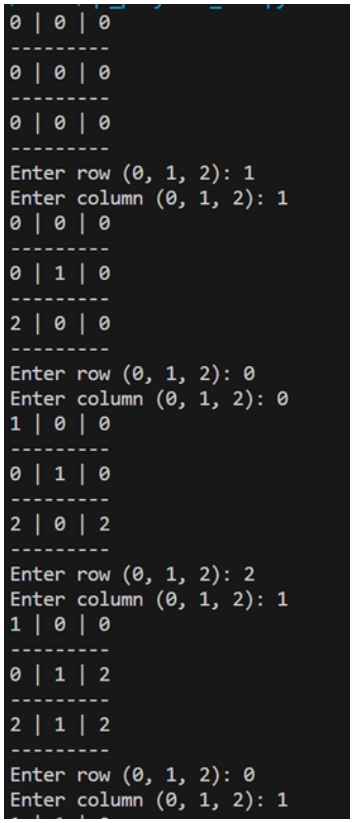
CODE:

```
import random
from collections import deque

class Game:
    def __init__(self):
        self.board = [[0, 0, 0],
                      [0, 0, 0],
                      [0, 0, 0]]

    def display_board(self):
        for row in self.board:
            print(" | ".join(map(str, row)))
            print("-" * 9)

    def is_valid_move(self, row, col):
        return 0 <= row < 3 and 0 <= col < 3 and self.board[row][col]
== 0

    def make_move(self, player, row, col):
        if self.is_valid_move(row, col):
            self.board[row][col] = player
            return True
        return False

    def machine_move(self, player):
        # Implement the machine's move here using DFS or BFS.
        # For example, you can use random moves for demonstration
purposes.
        available_moves = [(i, j) for i in range(3) for j in range(3)
if self.board[i][j] == 0]
        if available_moves:
            return random.choice(available_moves)
        else:
            return None

def check_winner(board):
```

```
    for row in board:
        if row[0] == row[1] == row[2] != 0:
            return row[0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != 0:
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] != 0 or board[0][2]
== board[1][1] == board[2][0] != 0:
        return board[1][1]
    return 0

def is_board_full(board):
    return all(all(cell != 0 for cell in row) for row in board)

def play_game():
    game = Game()
    human_player = 1
    machine_player = 2
    while True:
        game.display_board()
        # Human's move
        while True:
            try:
                row = int(input("Enter row (0, 1, 2): "))
                col = int(input("Enter column (0, 1, 2): "))
                if game.make_move(human_player, row, col):
                    break
                else:
                    print("Invalid move. Try again.")
            except ValueError:
                print("Invalid input. Enter a number between 0 and
2.")
        winner = check_winner(game.board)
        if winner:
            game.display_board()
            print(f"Human wins! Player {winner}")
            break

        if is_board_full(game.board):
            game.display_board()
```

```
            print("It's a draw!")
            break
        # Machine's move
        machine_move = game.machine_move(machine_player)
        if machine_move:
            game.make_move(machine_player, *machine_move)
        winner = check_winner(game.board)
        if winner:
            game.display_board()
            print(f"Machine wins! Player {winner}")
            break
        if is_board_full(game.board):
            game.display_board()
            print("It's a draw!")
            break
if __name__ == "__main__":
    play_game()
```

OUTPUT:



EXERCISE – 2

1) GRID COLORING PROBLEM

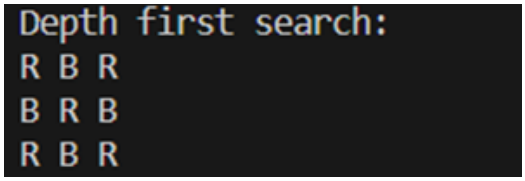
CODE:

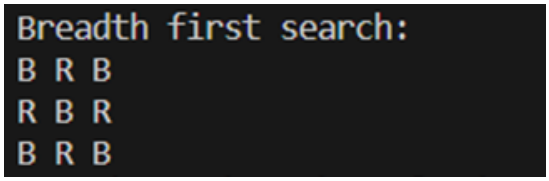
```
class Grid:
    def __init__(self, n):
      self.grid = [[None for _ in range(n // 3)] for _ in range(n //
3)]
      self.goal_patterns = [[[1, 0, 1], [0, 1, 0], [1, 0, 1]], [[0, 1,
0], [1, 0, 1], [0, 1, 0]]]
      self.stack = None
      self.size = 3

    def is_valid(self, row, col, color):
      if row - 1 >= 0 and self.grid[row - 1][col] == color:
            return False
      if col - 1 >= 0 and self.grid[row][col - 1] == color:
            return False
      if row + 1 < len(self.grid) and col<self.size:
            if self.grid[row + 1][col] == color:
                  return False
      if col + 1 < len(self.grid) and self.grid[row][col + 1] == color:
            return False
      return True

    def dfs(self, depth):
      if self.grid in self.goal_patterns:
            return self.grid
      for i in range(self.size):
            for j in range(self.size):
                  if self.grid[i][j] is None:
                  for col in [0, 1]:
                        if self.is_valid(i, j, col):
                              self.grid[i][j] = col
                              if self.dfs(depth-1):
                                    return self.grid
                              self.grid[i][j] = None
                  return None

    def bfs(self):
      queue=[(0,0)]
```

```
  while queue:
        # print(self.grid)
        if self.grid in self.goal_patterns:
              for row in self.grid:
              # print(row)
              for i in row:
                    if i==1:
                          print("B",end=" ")
                    else:
                          print("R",end=" ")
              print()
              return self.grid
        current=queue.pop(0)
        if 0<=current[0]<self.size and 0<=current[1]+1<self.size:
             queue.append((current[0],current[1]+1))
        if 0<=current[0]<self.size and 0<=current[1]-1<self.size:
             queue.append((current[0],current[1]-1))
        if 0<=current[0]+1<self.size and 0<=current[1]<self.size:
             queue.append((current[0]+1,current[1]))
        if 0<=current[0]-1<self.size and 0<=current[1]<self.size:
             queue.append((current[0]-1,current[1]))

        for color in 0,1:
              if self.is_valid(current[0],current[1],color):
              self.grid[current[0]][current[1]]=color

def solve_colouring(self):
  if not self.stack:
        self.stack = [self.grid]
  for i in range(1, (self.size * self.size) + 1):
        colored_grid = self.dfs(i)
        if colored_grid:
              for row in colored_grid:
              # print(row)
              for i in row:
                    if i==1:
                          print("B",end=" ")
                    else:
                          print("R",end=" ")
              print()
              return
  return False
```

```
if __name__ == "__main__":
    grid_instance = Grid(9)
    print("Depth first search:")
    grid_instance.solve_colouring()
    print("Breadth first search:")
    grid_instance.bfs()
```

OUTPUT:





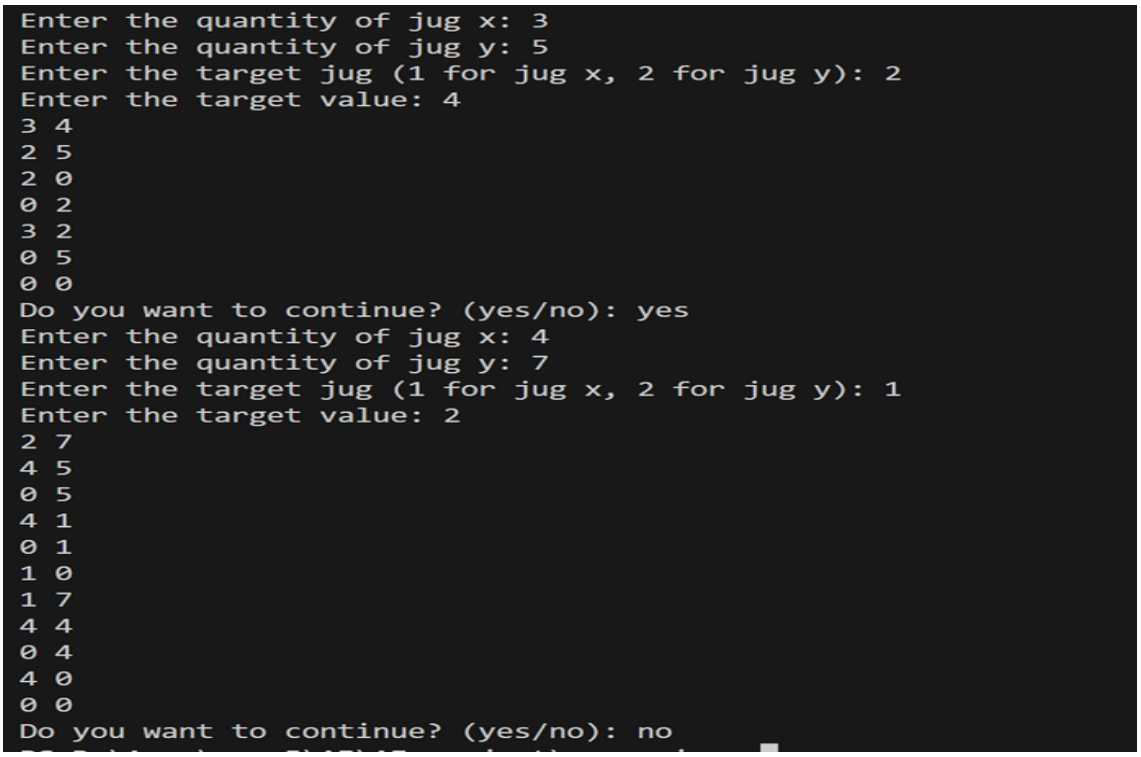
2) WATER JUG PROBLEM:

CODE:

```
class node:
    def __init__(self, x, y, prev):
        self.x = x
        self.y = y
        self.prev = prev


def conditions(x, y, cur):
    prev = cur

    # filling jug1
    if x < jug_x:
        queue.append(node(jug_x, y, prev))

    # filling jug2
    if y < jug_y:
        queue.append(node(x, jug_y, prev))

    # transferring contents of jug2 to jug1
    if x < jug_x:
        if x + y >= jug_x:
            d = jug_x - x
            queue.append(node(jug_x, y - d, prev))
        else:
            if x + y != 0:
                queue.append(node(x + y, 0, prev))

    # transferring contents of jug1 to jug2
    if y < jug_y:
        if x + y >= jug_y:
            d = jug_y - y
            queue.append(node(x - d, jug_y, prev))
        else:
            if x + y != 0:
                queue.append(node(0, x + y, prev))

    # emptying jug1
```

```
    if x > 1:
        queue.append(node(0, y, prev))
    # emptying jug2
    if y > 1:
        queue.append(node(x, 0, prev))

def solve_jug_problem():
    queue = [node(0, 0, None)]

    while queue:
        cur = queue.pop(0)
        x = cur.x
        y = cur.y

        if y == target and choice == 2:
            return cur
        elif x == target and choice == 1:
            return cur
        else:
            conditions(x, y, cur)


if __name__ == "__main__":
    while True:
        jug_x = int(input("Enter the quantity of jug x: "))
        jug_y = int(input("Enter the quantity of jug y: "))
        choice = int(input("Enter the target jug (1 for jug x, 2 for
jug y): "))
        target = int(input("Enter the target value: "))

        ansnode = solve_jug_problem()

        while ansnode is not None:
            print(ansnode.x, ansnode.y)
            ansnode = ansnode.prev

        user_input = input("Do you want to continue? (yes/no): ")
        if user_input.lower() != "yes":
            break
```

OUTPUT:


Enter the quantity of jug x: 3
Enter the quantity of jug y: 5
Enter the target jug (1 for jug x, 2 for jug y): 2
Enter the target value: 4
3 4
2 5
2 0
0 2
3 2
0 5
0 0
Do you want to continue? (yes/no): yes
Enter the quantity of jug x: 4
Enter the quantity of jug y: 7
Enter the target jug (1 for jug x, 2 for jug y): 1
Enter the target value: 2
2 7
4 5
0 5
4 1
0 1
1 0
1 7
4 4
0 4
4 0
0 0
Do you want to continue? (yes/no): no

EXERCISE – 3

1) HEURISTIC SEARCH

CODE:

```
from heapq import heappop, heappush


initial_state = [[], [], [1,2,3,4,5]]
goal_state = [[1,2,3,4,5], [], []]


def heuristic(state):
    max_height = max(len(stack) for stack in state)
    return sum(len(stack) != max_height for stack in state)

actions = [('move', 0, 1), ('move', 0, 2), ('move', 1, 0), ('move', 1,
2), ('move', 2, 0), ('move', 2, 1)]

def apply_action(state, action):
    new_state = [list(stack) for stack in state]
    move_type, source, target = action

    if move_type == 'move':
        if len(new_state[source]) > 0:
            block = new_state[source].pop()
            new_state[target].append(block)

    return new_state

def a_star_search(initial_state, goal_state, heuristic):
    frontier = [(0 + heuristic(initial_state), 0, initial_state, [])]
    explored = set()

    while frontier:
        _, path_cost, current_state, path = heappop(frontier)

        if current_state == goal_state:
            return path
```

```
        explored.add(tuple(map(tuple, current_state)))

        for action in actions:
            new_state = apply_action(current_state, action)

            if tuple(map(tuple, new_state)) not in explored:
                new_path = path + [action]
                g = path_cost + 1  # Uniform cost

                # f(n) = g(n) + h(n)
                f = g + heuristic(new_state)
                heappush(frontier, (f, g, new_state, new_path))

    return None

# Perform A* search
solution_path = a_star_search(initial_state, goal_state, heuristic)

# Print the first four expanded nodes
for i in range(4):
    action = solution_path[i] if i < len(solution_path) else None
    state = initial_state if i == 0 else apply_action(initial_state,
action)
    path_cost = i
    heuristic_value = heuristic(state)
    print(f"Node {i + 1}: Action={action}, State={state}, Path
Cost={path_cost}, Heuristic Value={heuristic_value}")


def heuristic_sum_of_distances(state, goal_state):
    total_distance = 0

    for block in set(block for stack in goal_state for block in
stack):
        current_position = find_block_position(state, block)
        goal_position = find_block_position(goal_state, block)
        total_distance += manhattan_distance(current_position,
goal_position)
```

```
    return total_distance

def find_block_position(state, block):
    for i, stack in enumerate(state):
        if block in stack:
            return i, stack.index(block)

def manhattan_distance(position1, position2):
    return abs(position1[0] - position2[0]) + abs(position1[1] -
position2[1])
```

OUTPUT:

```
Solution found:
Stack 0: []
Stack 1: []
Stack 2: [1, 2, 3]
Node 1: Action=('move', 2, 1), State=[[], [], [1, 2, 3, 4, 5]], Path Cost=0, Heuristic Value=2
Node 2: Action=('move', 2, 1), State=[[], [5], [1, 2, 3, 4]], Path Cost=1, Heuristic Value=2
Node 3: Action=('move', 2, 1), State=[[], [5], [1, 2, 3, 4]], Path Cost=2, Heuristic Value=2
Node 4: Action=('move', 2, 1), State=[[], [5], [1, 2, 3, 4]], Path Cost=3, Heuristic Value=2
```