NUTRI FIT: DIET RECOMMENDATION SYSTEM USING AI

UIT2511 Software Development Project – II

A PROJECT REPORT

Submitted by

Kanitha S A	3122 21 5002 044
Karthick V	3122 21 5002 045
Karunagaran R	3122 21 5002 046

SSN COLLEGE OF ENGINEERING,

KALAVAKKAM

DECEMBER 2023

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

BONAFIDE CERTIFICATE

Certified that this project titled "Nutri Fit: Diet Recommendation system using AI" is the bonafide work of Kanitha SA-3122215002044 and Karthick V-3122215002045 and Karunagaran R – 3122215002046 and is submitted for project viva-voce examination held in Nov 2023

Signature of examiner(s)

Submitted on ----------

Internal Examiner **External Examiner**

ABSTRACT

In the midst of the modern lifestyle, health often is giving rise to challenges like poor nutrition and a lack of consistent workout routines. Recognizing the paramount importance of health and well-being in today's fast-paced world, there is a growing demand for a comprehensive application offering personalized diet and workout recommendations. This software harnesses the power of a knowledge-based system fortified with fuzzy-logic algorithms, aiming to deliver tailored strategies for weight loss. By seamlessly incorporating user input and applying fuzzy logic processing, the application meticulously crafts diet and workout plans, providing a precise evaluation of the user's current health status. At its nucleus, the software focuses on addressing the weight-loss requirements of individuals classified as obese, presenting a dynamic diet regimen with three distinct eating patterns (low, moderate, and high carb) tailored to specific needs. The accompanying workout plan strategically combines cardio and gym exercises, optimizing weight loss based on the user's unique health profile. The user-centric design extends to dedicated "eat" and "fit" pages, offering recommendations for foods aligned with the diet plan and providing customized exercise routines. This holistic approach positions the software as a comprehensive lifestyle guide, empowering individuals to achieve and maintain a healthy and balanced life.

INTRODUCTION

MOTIVATION:

Our motivation is to introduce a holistic approach that goes beyond weight loss, considering individual factors such as height, weight, and fitness levels. By using fuzzy-logic algorithm, we aim to tailor recommendations to each user's specific requirements. The fuzzy-logic algorithm plays a crucial role by introducing adaptability and precision into diet and workout recommendations, recipe suggestions. The responsive UI and dynamic modifications of the software contribute to an interactive and engaging experience. Ultimately, the motivation behind this software is to guide users toward a healthier and more balanced lifestyle.

PROBLEM STATEMENT:

Our problem statement is to maintain a healthy lifestyle with proper nutrition and physical workouts can be challenging for individuals in this fast-paced world. This software aims to develop an AI-powered software designed to provide Nutritional and Fitness guidance based on the user’s current fitness level.

OBJECTIVE:

The objective is to develop an AI-powered software with a user-friendly web interface that prioritizes accessibility and engagement. The software will integrate fuzzy logic algorithms to accurately identify users' body categories and create a comprehensive knowledge-based system, utilizing databases for diet, recipes, and workouts. The system will dynamically generate personalized meal plans and workout recommendations. Detailed recipe suggestions aligned with individual dietary needs will be provided, accompanied by video demonstrations for diverse workout routines. The software aims to serve as a lifestyle guide, promoting a healthy lifestyle.

DELIVERABLES:

- User-Friendly Web Interface.
- Fuzzy Logic Algorithm Integration and database Integration.
- Knowledge-Based System.
- Dynamic Meal and Workout Recommendations.
- Detailed Recipe Suggestions.
- Responsive UI and Dynamic Modifications.

REQUIREMENTS ENGINEERING

Sprint	Epic	Essential/ Desirable	Description of the Requirement	Remarks
1	Import	Essential	Imports all necessary modules of Matplotlib,Sqlalchemy,Fuzzy logic for diet recommendation system.	Ensure seamless integration of modules.
1	User Input Module	Essential	Create an input page to gather user details (height, weight, gender).	Implement validation checks for input.
1	Fuzzy Logic Algorithm	Essential	Implement fuzzy logic to assess the user's current health accurately.	Thoroughly test the fuzzy logic algorithm.
1	Diet Plan Generation Module	Essential	Generate personalized diet plans based on health goals and preferences. Store user inputs for low, moderate, and high carb preferences.	Explore options for dietary preferences.
2	Workout Recommendation Module	Essential	Recommend dynamic workout plans based on fitness level and preferences.	Provide a variety of workout routines.
2	Data Visualization	Desirable	Provide visual representations like pie charts for user data overview.	Enhance user experience in decision making.
3	Recipe and Steps Details Module	Essential	Store and present detailed recipe and steps information.	Validate correctness of methods.

IMPLEMENTATION AND RISK MANAGEMENT

Individual contribution:

Name: Karthick V

Register Number: 3122215002045

Role in the project: Developer

A. Implementation

Sprint	Epic	Requirement	Essential/ Desirable	Description of the Requirement	Remarks
1	Create a Input page to get user details	To create a personalized profile by providing user's height, weight, gender, and fitness level.	Essential	The user should be able to give input for recommendations based on information includes height, weight, gender, and fitness level (e.g., beginner, intermediate, advanced).	To implement validation checks for user input to maintain data accuracy.
1	Fuzzy Logic Algorithm Implementation	Fuzzy logic algorithm is used to assess user's current state accurately.	Essential	By using fuzzy logic to process user input and determine the user's current health state, considering factors such as fitness level, and specific health goals.	Thoroughly test the fuzzy logic algorithm under various scenarios and user inputs.
1	Personalized Diet Plans	To receive personalized diet plans based on	Essential	To generate daily diet plans with options for low,	Explore options for dietary

		user's health goals and preferences.		moderate, and high carb days, considering the user's health goals, dietary preferences, and any dietary restrictions.	preferences and restrictions to enhance user satisfaction.

Individual contribution:

Name: Karunagaran R

Register Number: 3122215002046

Role in the project: Developer

A. Implementation

Sprint	Epic	Requirement	Essential/ Desirable	Description of the Requirement	Remarks
1	Dynamic Workout Recommendation	Dynamic workout recommendations that adapt to user's fitness level and preferences.	Essential	To recommend workout plans, including cardio and gym exercises, with the ability to dynamically adjust intensity and types of exercises based on the user's fitness level and preferences.	Provide a variety of workout routines to keep users engaged.
2	Nutrition and Diet Management	To input user's stage, body type, sex, and preferences for low, moderate, and high carb diets to receive personalized	Essential	To store user inputs such as stage, body type, sex, low carb, moderate carb, and high carb preferences.	Ensure that can store user preferences accurately.

		nutrition and diet plans.			
2	Workout Plan Manage ment	To input user stage, body type, sex, and preferences for cardio and gym exercises	Essential	To store details about gym exercises, including the day, exercise name, number of sets	Ensure that can store user preferences accurately.

Individual contribution:

Name: Kanitha S A

Register Number: 3122215002044

Role in the project: Developer

A. Implementation

Sprint	Epic	Requirement	Essential/ Desirable	Description of the Requirement	Remarks
2	Exercise Information	Detailed information about individual exercises, including their name, link to demonstrations, overview.	Essential	To store information about an exercise to extract details.	Ensure that is effectively used for providing information to users.
2	Recipe and Steps Details	Detailed information about the recipe and steps involved in preparing a dish.	Essential	To store and present detailed recipe and steps information.	Validate the correctness of methods

RISK MANAGEMENT

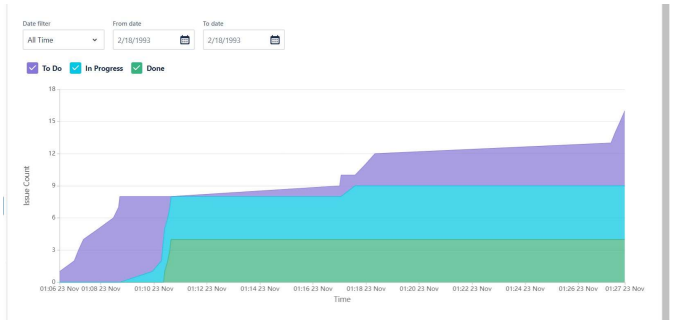
Risk	Risk Description	Impact	Mitigation Plan
Ineffective Fuzzy logic algorithm	Inaccuracies in the fuzzy logic algorithm may result in incorrect health state assessments.	Medium	Conduct extensive testing with diverse scenarios and inputs. Collaborate with health professionals to validate algorithmic decisions.
Inaccurate User Input for Personalized Plans	Users may provide incomplete or incorrect personal details, affecting the accuracy of personalized health and fitness plans.	High	Implement thorough input validation checks, guide users with clear instructions, and ensure the correct format for data entry.

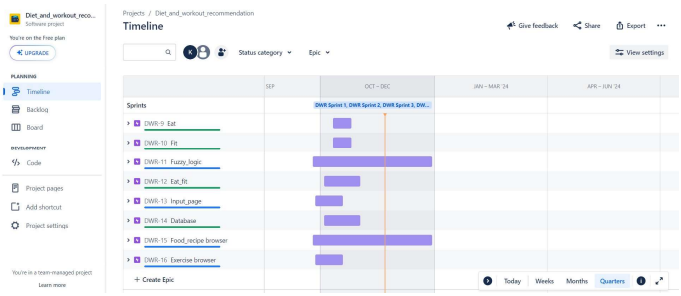
C. Test Log report

TC ID	Test Case Description/Condition	Test Case Input	Expected Output	Result (PASS/FAIL)
TC-01	Height value – 130 - 220	Valid Height value	Successful	PASS
TC-02	Height value - 100	Invalid height value	Value must be equal to or greater than 130	PASS
TC-03	Weight value-30-150	Valid weight	successful	PASS
TC-04	Height value – 250	Invalid Height value	Value must be less than or equal to 220	PASS
TC-05	Weight value- 20	Invalid weight value	Value must be equal to or greater than 30	PASS
TC-06	Weight value-160	Invalid weight value	Value must be less than or equal to 150	PASS

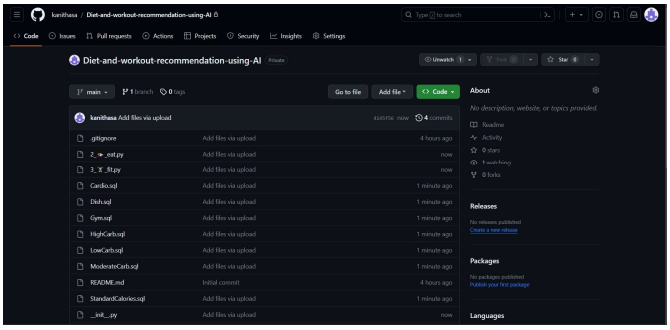
PROJECT MANAGEMENT

JIRA:





GITHUB:



CODE SNIPPETS

Fuzzy logic employs membership values from height and weight to populate a table (membership_values_table) using predefined rules by computing the minimum values across related fuzzy sets.

```
import numpy as np

class FuzzyLogic():

    FUZZY_RULES_TABLE = np.array([[1, 0, 0],
                                  [2, 1, 1],
                                  [4, 3, 2]])

    def __init__(self):
        self.membership_values_table = np.array([[0., 0., 0.],
                                                 [0., 0., 0.],
                                                 [0., 0., 0.]])
        self.fuzzified_decision = np.array([0., 0., 0., 0., 0.])
        self.fuzzy_sets_and_membership_values_of_height = {}
        self.fuzzy_sets_and_membership_values_of_weight = {}
        self.final_decision_on_body = None

    def do_fuzzification_of_height(self, height, sex):
        if sex == 0:
            if height < 160:
                self.fuzzy_sets_and_membership_values_of_height[0] = 1
            elif height >= 160 and height < 167.5:
                p1 = (2 * height - 320) / 15
                p0 = 1 - p1
                self.fuzzy_sets_and_membership_values_of_height[0] = p0
```

It defines membership functions for height and weight, dividing them into fuzzy sets (like short, average, tall, light, average weight, heavy) based on certain thresholds.

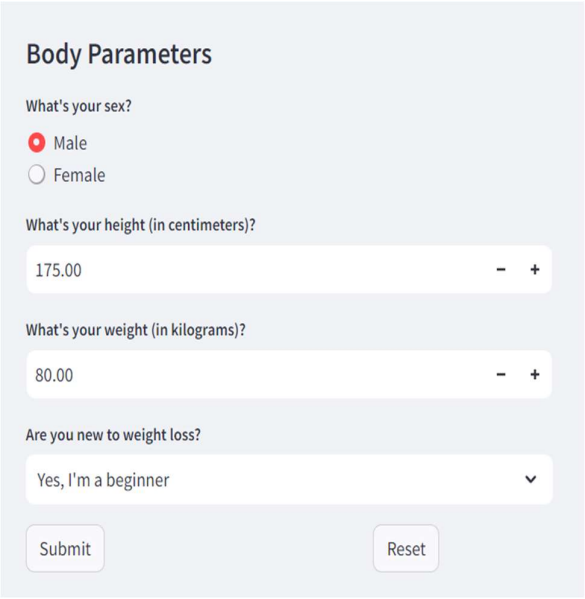
```
def do_fuzzification_of_weight(self, weight, sex):
    if sex == 0:
        if weight < 50:
            self.fuzzy_sets_and_membership_values_of_weight[0] = 1
        elif weight >= 50 and weight < 60:
            p1 = (2 * weight - 100) / 20
            p0 = 1 - p1
            self.fuzzy_sets_and_membership_values_of_weight[0] = p0
            self.fuzzy_sets_and_membership_values_of_weight[1] = p1
        elif weight >= 60 and weight < 70:
            p2 = (2 * weight - 120) / 20
            p1 = 1 - p2
            self.fuzzy_sets_and_membership_values_of_weight[1] = p1
            self.fuzzy_sets_and_membership_values_of_weight[2] = p2
        else:
            self.fuzzy_sets_and_membership_values_of_weight[2] = 1
    else:
        if weight < 45:
            self.fuzzy_sets_and_membership_values_of_weight[0] = 1
        elif weight >= 45 and weight < 50:
            p1 = (2 * weight - 90) / 10
            p0 = 1 - p1
            self.fuzzy_sets_and_membership_values_of_weight[0] = p0
            self.fuzzy_sets_and_membership_values_of_weight[1] = p1
        elif weight >= 50 and weight < 55:
```

```
def do_defuzzification_of_body(self):
    max = 0
    self.final_decision_on_body = 0

    for i in range(0, 5):
        if self.fuzzified_decision[i] > max:
            max = self.fuzzified_decision[i]
            self.final_decision_on_body = i

    print(f'Final Decision on Body: {self.final_decision_on_body}')
    print('_____')
    return self.final_decision_on_body
```

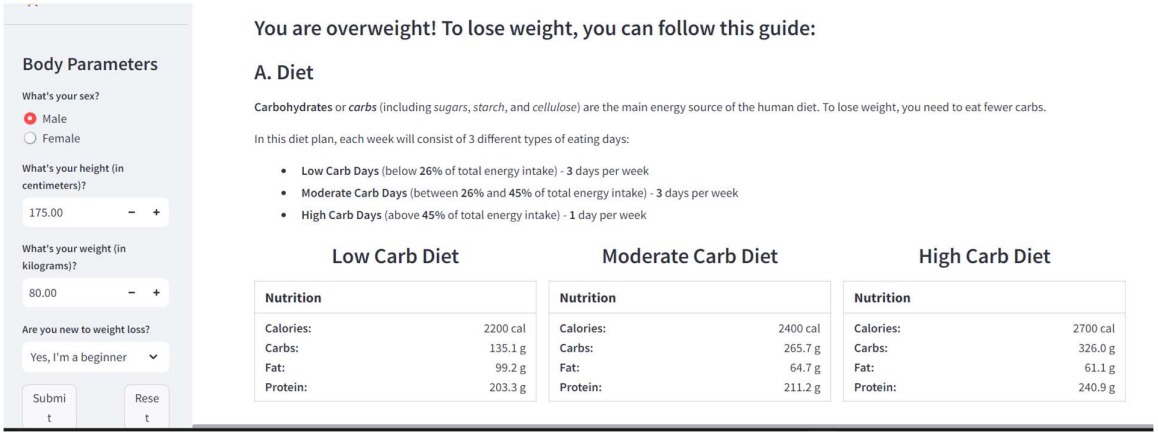
Finally, the algorithm selects the maximum value from the fuzzified decisions as the final decision on the body type. The index of this maximum value corresponds to a specific body type in the decision-making process.

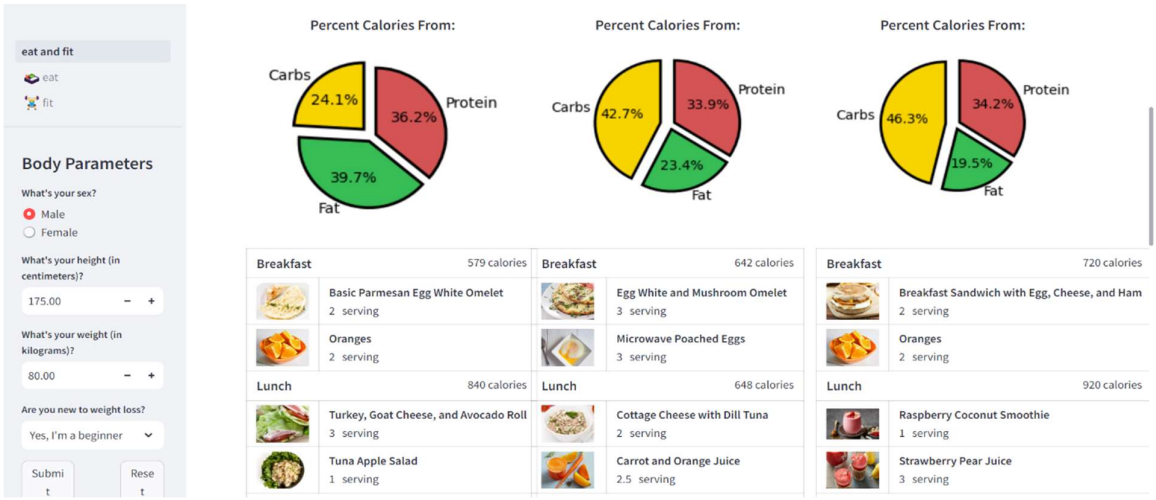
PROJECT OUTCOMES

INPUT PAGE:



RESULT PAGE:





B. Workout

1. Cardio

It doesn't matter which form of cardio you use. Pick something that gets your heart moving, be it treadmill, elliptical, or swimming.

Based on your current state, you should do 3 sessions a week: 12, 15, 12 minutes, respectively.

2. Gym

You will be using an upper/lower workout every week. Rep schemes are merely guidelines.

When a weight becomes manageable using the given set and rep schemes, add weight to the bar. For sake of convenience, use the same weight for each of the sets for a given exercise.

- **Day 1** - Upper
- **Day 2** - Lower
- **Day 3** - *Off*
- **Day 4** - Upper
- **Day 5** - Lower
- **Day 6** - *Off*
- **Day 7** - *Off*

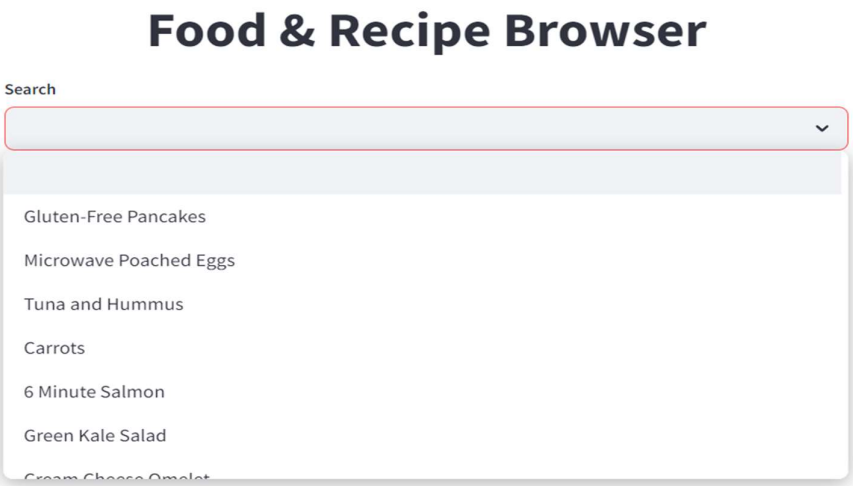
Lower

Exercise	Sets	Reps
Squats	3	8-10
Leg Press	3	15-20
Leg Extension	3	12-15
Leg Press Calf Raise	3	15-20
Stiff Leg Deadlift	3	8-10
Seated Calf Raise	3	15-20

Upper

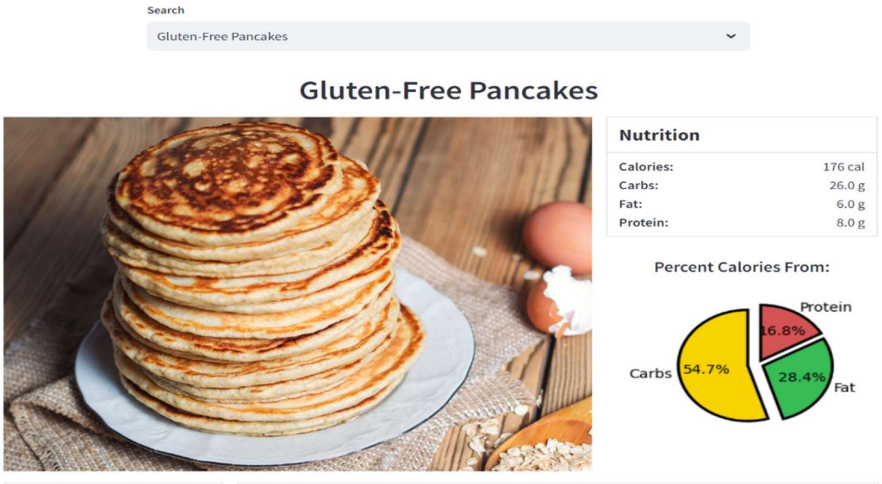
Exercise	Sets	Reps
Incline Bench Press	3	8-10
One Arm Dumbbell Row	3	10-12
Seated Barbell Press	3	8-10
Pull Ups	3	10
Skullcrushers	3	10-12
Dumbbell Bench Press	3	10

FOOD AND RECIPE BROWSER PAGE:



RESULT PAGE:

SAMPLE OUTPUT 1:



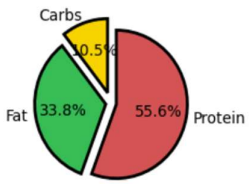
Recipe	
Cream cheese:	0.5 oz
Egg:	1 quả
Honey:	1 thìa cà phê
Cinnamon:	1/2 thìa cà phê
Oatmeal:	1/2 cốc

Steps to cook	
Step 1:	Blend ingredients or hand-whisk vigorously
Step 2:	Heat butter or cooking oil in small skillet and then pour out a single pancake
Step 3:	Once small bubbles appear all over the surface, flip the pancake
Step 4:	Repeat steps 2 and 3 for each pancake, serve immediately

SAMPLE OUTPUT 2:

Calories:	257 cal
Carbs:	7.0 g
Fat:	10.0 g
Protein:	37.0 g

Percent Calories From:



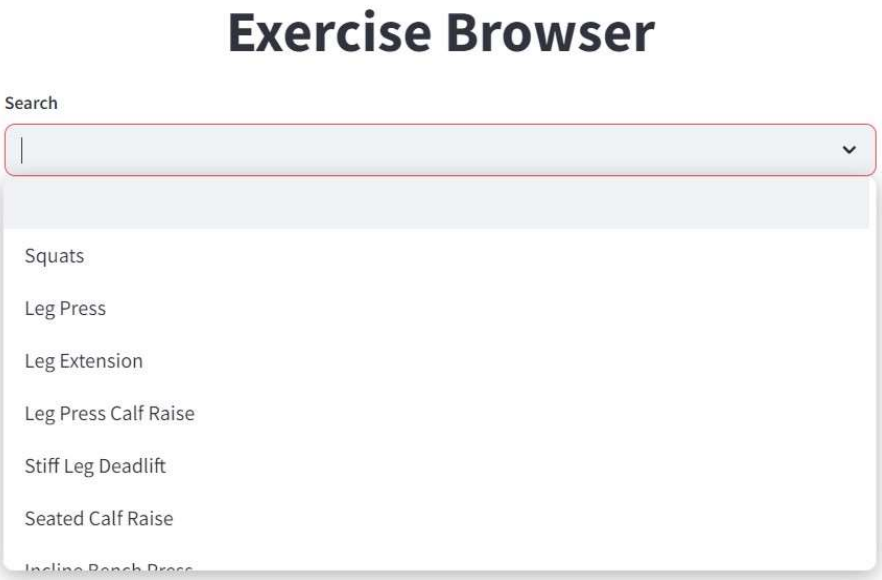
Recipe

Sockeye salmon:	6 oz
Salt:	0.1 g
Garlic powder:	1/2 thìa cà phê
Lemons:	1/2 quả

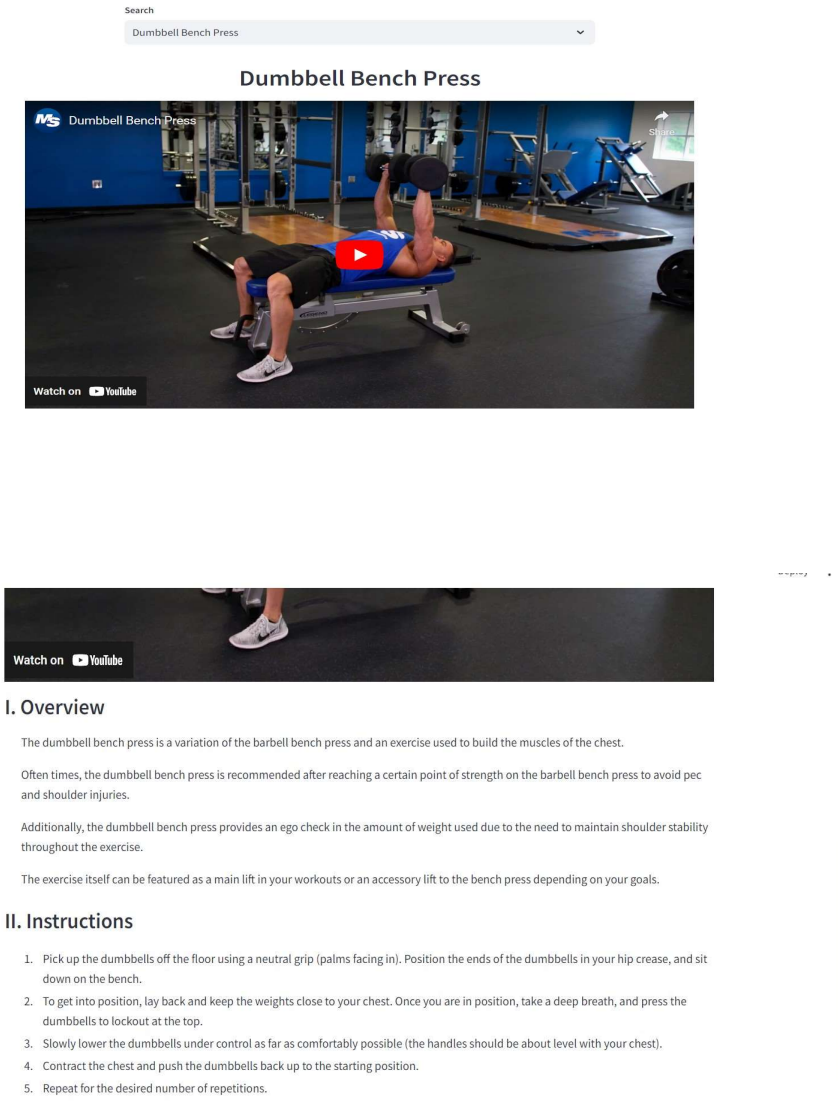
Steps to cook

Step 1:	Preheat a cast iron skillet to high heat and set your oven to the broiler setting
Step 2:	Take the filet of salmon and pat dry with a paper towel. Salt the skin side of the salmon and add it to the pan, skin side down. Set the timer for two minutes. While the skin side is searing, season the other side generously with the garlic powder.Once the salmon is seared, place the cast iron into the oven for 4 minutes under the broiler
Step 3:	Once the 4 minutes are up, serve the salmon with lemon wedges

EXERCISE BROWSER PAGE:



RESULT PAGE:



CONCLUSION

The development of Nutri Fit: Diet Workout Recommendation system using AI has really been an informative and knowledgeable process wherein each of us have learnt a lot of things unknown to us and most importantly gained exposure to HTML, Streamlit, CSS, NumPy, Matplotlib which form the backbone of Web Development. Put all this gained knowledge to use software tool has thus been successfully developed.

FUTURE DIRECTIONS

Continuous expansion of the database to include a broader range of meals, recipes, and workout details. Collaboration with nutritionists, fitness experts, and culinary professionals could contribute to a more diverse and comprehensive repository. Implementation of personalized user profiles to track individual progress, preferences, and feedback. This feature could enhance user engagement and provide more tailored recommendations over time. Providing feedback on lifestyle choices and suggesting improvements based on behavioral patterns could enhance the overall impact on users' well-being.

REFERENCES

[1] jasenka gajdoš kljusurić*, želimir kurtanjek(2012). Fuzzy logic modeling in nutrition planning -Application on meals in boarding schools 36(3), 1453–1464.

[2]Alexa Lauren Kuziel- The Utilization of technology as an approach to improve meal planning and dietary intake improve meal planning and dietary intake(2019)-10.13023.

[3]j. bulka; a. izworski; j. koleszynska; j. lis; i. wochlik

Automatic meal planning using artificial intelligence algorithms in computer aided diabetes therapy(2009) 978-1-4244-2712-3

[4]AI-Driven Meal Planning in the FoodTech Industry -Victor M˚artensson,Master's thesis-2021:E24

[5],Shraddha Mithbavkar;-Diet Planner Using Deep Learning(2023)E2104

[6] Effects of a Personalized Fitness Recommender System Using Gamification and Continuous Player Modeling: System Design and Long-Term Validation Study by Katherine

[7] Rashidi, P., & Mihailidis, A. (2013). A Survey on Ambient-Assisted Living Tools for Older Adults. IEEE Journal of Biomedical and Health Informatics, 17(3), 579–590.

[8] Exercise recommendation based on knowledge concept prediction by Zhengyang Wu , Ming Li , Yong Tang, Qingyu Liang

[9] Gourangi Taware; Reena Kharat; Pratik Dhende; Prathamesh Jondhalekar; Rohit Agrawal. (2016) "AI-Based Workout Assistant and Fitness Guide"

[10] Plan-Cook-Eat: A Meal Planner App with Optimal Macronutrient Distribution of Calories Based on Personal Total Daily Energy Expenditure-Manuel B. Garcia-2019

AI LAB EXERCISES

EXERCISE – 1

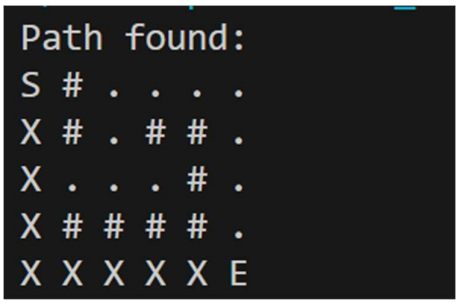
1) ROBOT – MAZE PROBLEM

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

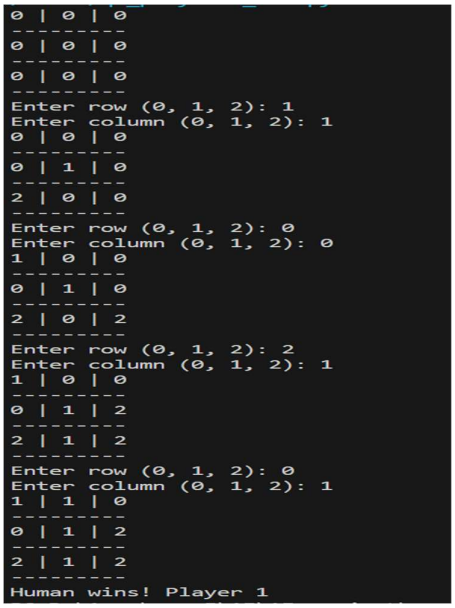
CODE:

```
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

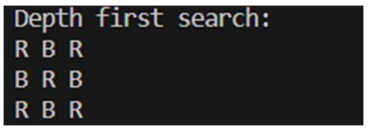
CODE:

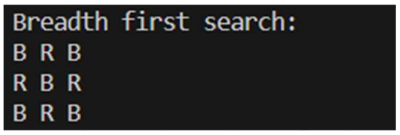
```
class Grid:
    def __init__(self, n):
        self.grid = [[None for _ in range(n // 3)] for _ in range(n // 3)]
        self.goal_patterns = [[[1, 0, 1], [0, 1, 0], [1, 0, 1]], [[0, 1, 0], [1, 0, 1], [0, 1, 0]]]
        self.stack = None
        self.size = 3

    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:





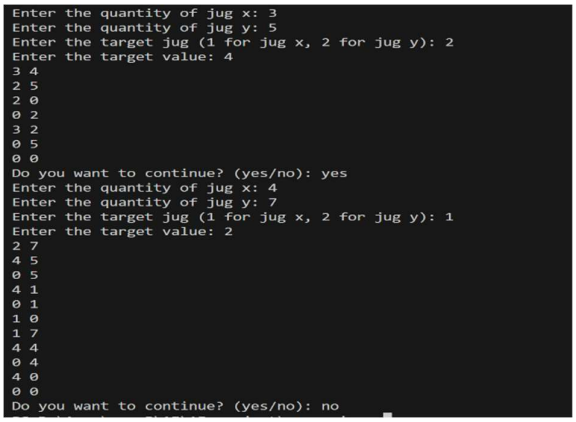
2) WATER JUG PROBLEM:

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:



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
```