



NUTRITION ME

-The ultimate nutrition tracking website

-Krisha Desai

-Mehak Gurbaxani

-Erzhan Okumaliev

-Bach Le Xuan

: We want you to try our application (Very easy to try it out)

- Install streamlit
- Go to cmd, and run the main app file using command

Python -m streamlit run main.py

Introduction:

In the dynamic world of health and wellness, the combination of nutrition science and cutting-edge data analytics has launched a revolution in personalised dietary control. Understanding the importance of data science in nutrition maintenance is critical to recognising the worth of this endeavour. The application of data science facilitates the extraction of meaningful insights from large datasets that include health indicators, lifestyle factors, genetic predispositions, and eating patterns. The complex links between diet and health outcomes can be untangled by researchers using advanced analytics tools, opening the door for evidence-based dietary interventions.

Moreover, data science plays an important role in overcoming the issues associated with traditional nutrition tracking methods. Users frequently find manual entry and standardised criteria to be burdensome, which makes tracking inconsistent and makes it harder to meet dietary objectives. On the other hand, nutrition tracking can be automated, tailored, and easily incorporated into everyday activities by utilising machine learning algorithms.

Maintaining a balanced and healthful lifestyle is mostly dependent on nutritional surveillance. People who carefully track their food intake learn a great deal about their eating patterns and are better equipped to make decisions that support their health goals. For many people, though, keeping track of diet can be an overwhelming undertaking. Maintaining consistency, interpreting complicated nutrition labels, and manually entering data are difficult tasks that frequently cause annoyance and disengagement.

Furthermore, having access to recipes that are in line with particular nutritional objectives, in addition to monitoring intake, improves the effectiveness of dietary control. It can be intimidating to locate these recipes among the plethora of options accessible. Users can filter and rank recipes according to their own dietary needs by incorporating data science into the recipe discovery process. Users are empowered to discover culinary masterpieces that support their nutritional journey with personalised recipe recommendations, regardless of their preference for low-calorie, high-protein, or allergen-free options.

The integration of data analytics with nutrition research presents a means to transform dietary management. To tackle the obstacles associated with nutritional tracking, offering customised advice, and enabling accessibility to specialised recipe collections, we developed a nutritional tracker (“Nutrition Me”) that aims to support people in their pursuit of well-being. We work to reframe the connection between nutrition and wellbeing via innovation and teamwork, pointing people in the direction of a future full of vitality.

Business Problem:

The key business issue is a lack of personalised nutrition management solutions tailored to individual goals and needs. Existing systems frequently offer generic guidance or necessitate significant manual effort from users to measure their nutrition consumption and discover appropriate recipes.

Solution:

Nutrition Me is a recommendation system that uses user information and goals to offer personalised nutrition advice and meal choices. The system uses machine learning algorithms to analyse user data and make recommendations based on nutritional needs, preferences, and dietary restrictions.

Data Acquisition:

Recipes

Initially, we tried web scraping to collect recipe information from numerous online sources. However, we faced various problems during this process:

- 1. Inconsistent Data Format:** The structure and formatting of recipe data differed greatly amongst websites, making it difficult to extract and standardise the information effectively.
- 2. Data Quality Issues:** Due to discrepancies in data formatting, we discovered missing fields, incomplete information, and inaccuracies in recipe specifics.
- 3. Scalability Issues:** Web scraping proved time-consuming and resource-intensive, particularly when dealing with significant amounts of recipe data from many sources.

To tackle these issues, we looked into alternate alternatives and discovered the Edamam API, a comprehensive food and recipe database that provides structured recipe data via a RESTful API interface. Using the Edamam API, we were able to address the following concerns:

- 1. Consistent Data Format:** The Edamam API returns recipe data in a standardised format, ensuring consistency and uniformity across all retrieved recipes. This standardised framework includes necessary information such as ingredients, nutritional information, cooking instructions, and serving sizes.
- 2. High-Quality Data:** Edamam curates its recipe database to guarantee that the material is accurate, thorough, and relevant. This leads in high-quality recipe data with few errors or inconsistencies, which improves the dependability of our recommendations system.
- 3. Scalability and Reliability:** By using the Edamam API, we acquired access to a large database of recipes spanning multiple cuisines, dietary preferences, and nutritional profiles. This flexible and dependable source of data allows us to develop our recipe database and meet a wide range of user preferences and needs.

Integrating the Edamam API into our recommendation system had various benefits:

- 1. Quality Assurance:** The standardised and curated nature of the data obtained from Edamam ensures the correctness and reliability of our recipe recommendations, hence increasing user confidence and happiness.

2. Efficiency and Scalability: By outsourcing the data acquisition process to Edamam, we can streamline the retrieval of recipe data, minimising the need for manual collection and processing. This increases the efficiency of our recommendation engine and allows for seamless scalability as our user base grows.

3. Focus on Core Functionality: Using the Edamam API allows our team to devote more resources and attention to refining recommendation algorithms and improving the user experience, rather than investing time and effort on data acquisition and cleaning.

DataBase

The database is a crucial component of our system, responsible for storing recipe data fetched from external APIs, managing recipe information, and providing necessary data for recommendation algorithms.

The Edamam API integration considerably improves our nutrition recommendation system's functionality and performance. It allows us to provide personalised recipe ideas that are aligned with users' nutritional objectives and preferences by giving us access to a large and dependable recipe library. This strategic cooperation with Edamam ensures the quality, consistency, and scalability of our recommendation system, thereby contributing to its efficacy and usefulness.

```
In [1]: import mysql.connector
import requests
import json

def fetch_data(query, app_id, app_key, offset=0, limit=500, seen_urls=set()):
    request_url = f"https://www.edamam.com/search?q={query}&app_id={app_id}&app_key={app_key}&from={offset}&to={offset+limit}"
    try:
        response = requests.get(request_url)
        response.raise_for_status() # Raise an exception for bad status codes
        data = response.json()
        hits = data.get("hits", [])
        unique_hits = [hit for hit in hits if hit['recipe']['url'] not in seen_urls]
        return unique_hits
    except requests.exceptions.RequestException as e:
        print(f"Failed to fetch data: {e}")
        return []

def create_recipe_table(cursor):
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS recipes (
        id INT AUTO_INCREMENT PRIMARY KEY,
        label VARCHAR(255),
        ingredients TEXT,
        meal_type VARCHAR(255),
        calories FLOAT,
        proteins FLOAT,
        carbs FLOAT,
        fats FLOAT,
        cuisine VARCHAR(255),
        health_labels TEXT,
        cooking_time INT,
        cautions TEXT,
        ingredient_lines TEXT,
        serves INT,
        url VARCHAR(255) UNIQUE
    )
    """)
```

```

def insert_recipe(cursor, recipe):
    meal_type = None
    if "mealType" in recipe and recipe["mealType"]:
        meal_type = recipe["mealType"][0].lower()

    # Check if the URL already exists in the database
    existing_query = "SELECT COUNT(*) FROM recipes WHERE url = %s"
    cursor.execute(existing_query, (recipe.get("url"),))
    count = cursor.fetchone()[0]
    if count > 0:
        print(f"Recipe with URL {recipe.get('url')} already exists. Skipping insertion.")
        return

    query = """
    INSERT INTO recipes (label, ingredients, meal_type, calories, proteins, carbs, fats, cuisine, health_labels, cooking_time
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """
    cursor.execute(query, (
        recipe.get("label", None),
        "\n".join(recipe.get("ingredientLines", [])),
        meal_type,
        recipe.get("calories", None),
        recipe["totalNutrients"].get("PROCNT", {}).get("quantity", None) if "totalNutrients" in recipe else None,
        recipe["totalNutrients"].get("CHOCDF", {}).get("quantity", None) if "totalNutrients" in recipe else None,
        recipe["totalNutrients"].get("FAT", {}).get("quantity", None) if "totalNutrients" in recipe else None,
        recipe.get("cuisineType", [None])[0],
        json.dumps(recipe.get("healthLabels", [])),
        recipe.get("totalTime", None),
        json.dumps(recipe.get("cautions", [])),
        json.dumps(recipe.get("ingredientLines", [])),
        recipe.get("yield", None),
        recipe.get("url", None)
    ))

```

```

def main():
    app_id = '8d453743'
    app_key = '19f0899da368b6406acf9868610f1945'
    query = input("What kind of recipe are you looking for?: ").strip()

    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="Er05jan2002@",
        database="food"
    )

    cursor = conn.cursor()
    create_recipe_table(cursor)

    offset = 0
    limit = 100
    seen_urls = set()

    while True:
        recipe_list = fetch_data(query, app_id, app_key, offset=offset, limit=limit, seen_urls=seen_urls)
        if not recipe_list:
            break
        for recipe_data in recipe_list:
            insert_recipe(cursor, recipe_data["recipe"])
            seen_urls.add(recipe_data["recipe"]["url"])
        offset += limit

    conn.commit()
    conn.close()
    print("Recipes stored successfully.")

if __name__ == "__main__":
    main()

```

```

import pandas as pd

def add_columns_to_table(cursor):
    add_columns_query = """
    ALTER TABLE recipes
    ADD COLUMN proteins_per_serving FLOAT,
    ADD COLUMN carbs_per_serving FLOAT,
    ADD COLUMN fats_per_serving FLOAT,
    ADD COLUMN calories_per_serving FLOAT
    """
    cursor.execute(add_columns_query)
    print("Columns added successfully.")

def calculate_serving_info(cursor):
    select_query = "SELECT id, proteins, carbs, fats, calories, serves FROM recipes"
    cursor.execute(select_query)
    recipes = cursor.fetchall()
    for recipe in recipes:
        id_, proteins, carbs, fats, calories, serves = recipe
        proteins_per_serving = proteins / serves
        carbs_per_serving = carbs / serves
        fats_per_serving = fats / serves
        calories_per_serving = calories / serves

        update_query = """
        UPDATE recipes
        SET
            proteins_per_serving = %,
            carbs_per_serving = %,
            fats_per_serving = %,
            calories_per_serving = %
        WHERE
            id = %
        """
        cursor.execute(update_query, (proteins_per_serving, carbs_per_serving, fats_per_serving, calories_per_serving, id_))
    print("Serving info calculated and updated successfully.")

def main():
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="Er05jan2002@",
        database="food"
    )
    cursor = conn.cursor()

    add_columns_to_table(cursor)

    calculate_serving_info(cursor)

    conn.commit()
    cursor.close()
    conn.close()

if __name__ == "__main__":
    main()

```

User data

To replicate user data, we created a dummy data for testing and development, we used the Faker library, a Python module that generates realistic false data for a variety of information kinds such as names, addresses, dates, and more.

Faker creates realistic identities by mixing common first and last names, guaranteeing that the generated names look like those of actual people while maintaining anonymity and privacy. Age data was produced at random within a specific range to depict a broad user population. This ensures that our recommendation system can accommodate customers of varying ages, each with their own dietary needs and aspirations. Faker was used to create random weight and height numbers, emulating user-specific body composition variability. These characteristics are critical for estimating variables such as basal metabolic rate and recommended daily calorie intake. We randomly assigned a muscle gain goal to each user, expressing their desire to develop muscle mass through nutrition and exercise. This information allows us to modify recipe recommendations to promote muscle growth and recovery. Users were assigned a random number of workout days each week, based on their

degree of physical activity and exercise frequency. This data point affects the calorie and nutrition requirements for peak performance and recuperation. Finally, the user's workout intensity was produced at random to represent the amount of effort and energy expended during exercise sessions. This measure aids in the adjustment of recommended nutritional intake to match the demands of various activity levels.

Dummy Data Using Faker to start with. (Not an integral part of the project)

The use of the Faker library to generate simulated user data improves the reliability and realism of our testing and development process. By building varied user profiles with randomised parameters such as age, weight, height, and exercise goals, we ensure that our recommendation system is tested under realistic settings and can effectively meet the needs of a wide range of users. Furthermore, Faker's versatility and scalability allow us to tailor the data production process to varied testing scenarios and needs, permitting complete evaluation and modification of the recommendation system.

```
] from faker import Faker
import random
import pandas as pd

fake = Faker()

# Generate fake users with random attributes
def generate_fake_users(num_users):
    users = []
    for _ in range(num_users):
        user = {}
        user['name'] = fake.name()
        user['user_id'] = "user20"+str(_ + 1)
        user['password'] = "2024nutrition"
        user['age'] = random.randint(18, 70) # Random age between 18 and 70 years
        user['weight'] = random.uniform(40, 120) # Random weight between 40 and 120 kg
        user['height'] = random.uniform(150, 190) # Random height between 150 and 190 cm
        user['bmi'] = calculate_bmi(user['weight'], user['height'])
        user['gender'] = random.choice(['Male', 'Female'])
        user['muscle_gain_goal'] = random.choice(['Yes', 'No']) # Random choice for muscle gain goal
        user['weight_goal'] = determine_weight_goal(user['age'], user['bmi'], user['gender'])
        user['workout_days_per_week'] = random.randint(1, 7) # Random number of workout days per week
        user['workload_intensity'] = random.choice(['Low', 'Moderate', 'High']) # Random intensity of daily workload
        users.append(user)
    return users
```

In [13]: df.head()

Out[13]:

	name	user_id	password	age	weight	height	bmi	gender	muscle_gain_goal	weight_goal	workout_days_per_week	workload_intensity
0	Devin Carroll	user201	2024nutrition	61	119.217574	162.088778	45.376854	Female	No	Lose Weight (High Priority)	3	High
1	Roger Riggs	user202	2024nutrition	37	78.637811	157.753891	31.598849	Male	Yes	Lose Weight (High Priority)	2	Moderate
2	Rose Baker	user203	2024nutrition	41	91.146430	181.771266	27.586030	Female	Yes	Lose Weight	4	High
3	Angela Schmidt	user204	2024nutrition	34	47.901771	150.753769	21.077311	Female	Yes	Maintain Weight	4	Moderate
4	Lindsay Hayes	user205	2024nutrition	23	77.141750	159.401093	30.360359	Female	No	Lose Weight (High Priority)	2	Moderate

In [14]: df.tail()

Out[14]:

	name	user_id	password	age	weight	height	bmi	gender	muscle_gain_goal	weight_goal	workout_days_per_week	workload_intensity
20496	Candice Nixon	user20496	2024nutrition	38	105.483297	174.551519	34.620747	Female	Yes	Lose Weight (High Priority)	2	Low
20497	Jeremy Turner	user20497	2024nutrition	50	110.893551	177.497806	35.198188	Female	Yes	Lose Weight (High Priority)	2	Moderate
20498	Cheryl Larson	user20498	2024nutrition	64	40.775939	185.949975	11.792658	Female	Yes	Maintain Weight	6	Moderate
20499	James Atkins	user20499	2024nutrition	48	66.623899	177.478484	21.151376	Female	No	Maintain Weight	6	Moderate
20500	William Reyes	user20500	2024nutrition	53	55.142891	180.919660	16.846823	Male	Yes	Maintain Weight	4	Moderate

Transformation and Calculation of Nutritional Metrics

We included the calculation of various critical parameters concerning an individual's nutritional requirements and metabolism. These measurements include BMI, BMR, TDEE, and macronutrient needs (protein, carbs, and fats). The formulas for these computations were derived from scientific journals and literature and chosen for their accuracy and appropriateness to our model.

1. Body Mass Index (BMI): BMI is a regularly used index of body fatness determined from a person's weight and height. It gives a rough approximation of whether a person's weight is suitable for their height.

```
# Calculate BMI
def calculate_bmi(weight, height):
    return weight / ((height/100) ** 2)

# Determine weight goal based on BMI
def determine_weight_goal(age, bmi, gender):
    if gender == 'Male':
        if age < 30:
            if bmi < 18.5:
                return 'Gain Weight (Muscle)'
            elif 18.5 <= bmi < 24.9:
                return 'Maintain Weight'
            elif 24.9 <= bmi < 29.9:
                return 'Lose Weight'
            else:
                return 'Lose Weight (High Priority)'
        else:
            if bmi < 18.5:
                return 'Maintain Weight'
            elif 18.5 <= bmi < 24.9:
                return 'Maintain Weight'
            elif 24.9 <= bmi < 29.9:
                return 'Lose Weight'
            else:
                return 'Lose Weight (High Priority)'
    elif gender == 'Female':
        if age < 30:
            if bmi < 18.5:
                return 'Gain Weight (Muscle)'
            elif 18.5 <= bmi < 24.9:
                return 'Maintain Weight'
            elif 24.9 <= bmi < 29.9:
                return 'Lose Weight'
            else:
                return 'Lose Weight (High Priority)'
        else:
            if bmi < 18.5:
                return 'Maintain Weight'
            elif 18.5 <= bmi < 24.9:
                return 'Maintain Weight'
            elif 24.9 <= bmi < 29.9:
                return 'Lose Weight'
            else:
                return 'Lose Weight (High Priority)'

# Example usage
num_users = 500
dummy_data = generate_fake_users(num_users)
df = pd.DataFrame(dummy_data)
print(dummy_data)
```


2. Basal Metabolic Rate (BMR): BMR is the number of calories necessary to maintain basic body functions at rest. It is affected by elements such as age, gender, weight, height, and body composition.

```
import pandas as pd

# Define a function to calculate BMR based on gender, age, weight, and height
def calculate_bmr(row):
    if row['gender'] == 'Female':
        bmr = 447.593 + (9.247 * row['weight']) + (3.098 * row['height']) - (4.330 * row['age'])
    else:
        bmr = 88.362 + (13.397 * row['weight']) + (4.799 * row['height']) - (5.677 * row['age'])
    return bmr
```

3. Total Daily Energy Expenditure (TDEE): TDEE is the total number of calories burned by an individual in a given day, taking into account both physical activity and BMR. We determined TDEE by multiplying BMR by an activity factor that accounts for the individual's degree of physical activity.

```
# Define a function to calculate TDEE based on BMR and activity level
def calculate_tdee(bmr, workout_days_per_week, workload_intensity):
    if workload_intensity.lower() == 'low' and workout_days_per_week == 0:
        activity_multiplier = 1.2
    elif workload_intensity.lower() == 'low' or (1 <= workout_days_per_week <= 3):
        activity_multiplier = 1.375
    elif workload_intensity.lower() == 'moderate' or (3 <= workout_days_per_week <= 5):
        activity_multiplier = 1.55
    elif workload_intensity.lower() == 'high' and workout_days_per_week == 7:
        activity_multiplier = 1.9
    else:
        activity_multiplier = 1.725

    tdee = bmr * activity_multiplier
    return tdee
```

4. Macronutrient requirements: Macronutrients are nutrients that provide energy, such as protein, carbs, and lipids. We calculated the recommended daily intake for each macronutrient based on the individual's goals and nutritional requirements. The distribution of macronutrients is usually stated as a percentage of total daily calories.

```

# Define a function to calculate macronutrient distribution based on goals
def calculate_macronutrients(weight_goal, muscle_gain_goal, tdee):
    protein_ratio = 0.25
    carb_ratio = 0.4
    fat_ratio = 0.35

    if muscle_gain_goal.lower() == 'yes':
        protein_ratio += 0.15
        carb_ratio -= 0.1
        fat_ratio -= 0.05

    if weight_goal == 'Lose Weight (High Priority)':
        protein_ratio -= 0.05
        carb_ratio -= 0.1
        fat_ratio += 0.15
    elif weight_goal == 'Lose Weight':
        protein_ratio -= 0.05
        carb_ratio -= 0.1
        fat_ratio += 0.05
    elif weight_goal == 'Gain Weight':
        protein_ratio += 0.05
        carb_ratio += 0.1
        fat_ratio += 0.1
    elif weight_goal == 'Maintain Weight':
        pass # No adjustments needed for macronutrients

    protein = protein_ratio * tdee / 4 # 4 calories per gram of protein
    carb = carb_ratio * tdee / 4 # 4 calories per gram of carb
    fat = fat_ratio * tdee / 9 # 9 calories per gram of fat

    return protein, carb, fat

# Load the user dataset
df = user_data # Assuming user_data is your dataset

# Drop the "Unnamed: 0" column
df = df.drop(columns=['Unnamed: 0'])

# Calculate BMR for each user
df['bmr'] = df.apply(calculate_bmr, axis=1)

# Calculate TDEE for each user
df['tdee'] = df.apply(lambda row: calculate_tdee(row['bmr'], row['workout_days_per_week'], row['workload_intensity']), axis=1)

# Calculate macronutrient distribution for each user
df[['protein', 'carb', 'fat']] = df.apply(lambda row: pd.Series(calculate_macronutrients(row['weight_goal'], row['muscle_g

# Display the results
print(df[['name', 'tdee', 'protein', 'carb', 'fat']])

```

By calculating BMI, BMR, TDEE, and macronutrient requirements, we can tailor the nutrition recommendations offered by our system to each user's specific traits, goals, and activity levels. This guarantees that the recommended food plan is matched to each user's unique needs and tastes. These measures enable a more accurate estimate of an individual's dietary needs than generic norms. By considering body composition, metabolism, and activity levels, we can provide more precise and effective nutrition advice that supports the user's health and fitness goals. Calculating BMI, BMR, TDEE, and macronutrient needs helps us understand the user's present situation and aspirations. This information allows us to offer nutritional recommendations that will help you lose weight, increase muscle, or maintain your general health and well-being. Giving people access to their nutritional data allows them to make informed decisions about their diet and lifestyle. Users may optimise their nutritional intake by assessing their calorie demands, macronutrient distribution, and metabolic rate.

Nutrition Me:

"Nutrition Me" seeks to revolutionise the way people approach nutrition by delivering personalised suggestions based on their own needs and aspirations. In today's health-

conscious environment, good diet is essential for overall health and fitness achievement. The "Nutrition Me" website has three primary sections: the User Registration Page, the Dashboard Page, and the Recipe Page. These components work together seamlessly to give users a comprehensive and personalised nutrition experience. The website is developed with Streamlit, a Python web application toolkit, and machine learning techniques are used to provide recommendations based on user data.

```
import streamlit as st
import loginapp
import home
import page2

def main():
    st.title("Streamlit Multi-page App")

    if "is_logged_in" not in st.session_state:
        st.session_state.is_logged_in = False

    if not st.session_state.is_logged_in:
        loginapp.main()
    else:
        # Page selection
        page = st.sidebar.selectbox("Go to", ["Dashboard", "Recipies"])

        if page == "Dashboard":
            home.main()

        elif page == "Recipies":
            page2.main()

if __name__ == "__main__":
    main()
```

User Registration Page:

The users are directed to the login/Registry Page, if it is a new user it asks to register new user otherwise allows direct login with User Id and Password. Upon registration, users are required to enter demographic information such as their name, age, weight, height, and gender. These details are critical for calculating metrics such as BMI and BMR, which are required to provide personalised dietary recommendations. They are also asked about their fitness goals, such as muscle gain(Yes/No), weight management(Lose Weight, Maintain Weight, Gain Weight) , workout Intensity(High, medium, Low), and Workout frequency(Number of days the user works out in a week).This Information is used to tailor recommendations easily. After submitting the registration form, the system estimates the user's BMI and BMR based on the information provided. Basal Metabolic Rate(BMR) is approximated using equations such as the Mifflin-St Jeor Equation, which takes into account age, weight, height, and gender. The user's demographic information, exercise objectives, BMI, and BMR are saved in the user database for future reference and recommendations. After registration or login the user is directed to the dashboard.

NutritionME.

User Registration and Nutrition Calculator

☐ Login

New User Registration

Name

User ID

Password

Age

Weight (kg)

Height (cm)

Gender

- ☒ Male
☐ Female

Muscle Gain

```

import streamlit as st
import pandas as pd
import numpy as np

# Function to calculate BMI
def calculate_bmi(weight, height):
    return weight / ((height/100) ** 2)

def calculate_bmr(gender, weight, height, age):
    if gender == 'Female':
        bmr = 447.593 + (9.247 * weight) + (3.098 * height) - (4.330 * age)
    else:
        bmr = 88.362 + (13.397 * weight) + (4.799 * height) - (5.677 * age)
    return bmr

# Define a function to calculate TDEE based on BMR and activity level
def calculate_tdee(bmr, workout_days_per_week, workload_intensity):
    if workload_intensity.lower() == 'low' and workout_days_per_week == 0:
        activity_multiplier = 1.2
    elif workload_intensity.lower() == 'low' or (1 <= workout_days_per_week <= 3):
        activity_multiplier = 1.375
    elif workload_intensity.lower() == 'moderate' or (3 <= workout_days_per_week <= 5):
        activity_multiplier = 1.55
    elif workload_intensity.lower() == 'high' and workout_days_per_week == 7:
        activity_multiplier = 1.9
    else:
        activity_multiplier = 1.725
    tdee = bmr * activity_multiplier
    return tdee

```

```

# Define a function to calculate macronutrient distribution based on goals
def calculate_macronutrients(weight_goal, muscle_gain_goal, tdee):
    protein_ratio = 0.25
    carb_ratio = 0.4
    fat_ratio = 0.35
    if muscle_gain_goal.lower() == 'yes':
        protein_ratio += 0.15
        carb_ratio -= 0.1
        fat_ratio -= 0.05
    if weight_goal == 'Lose Weight (High Priority)':
        protein_ratio -= 0.05
        carb_ratio -= 0.1
        fat_ratio += 0.15
    elif weight_goal == 'Lose Weight':
        protein_ratio -= 0.05
        carb_ratio -= 0.1
        fat_ratio += 0.05
    elif weight_goal == 'Gain Weight':
        protein_ratio += 0.05
        carb_ratio += 0.1
        fat_ratio += 0.1
    elif weight_goal == 'Maintain Weight':
        pass # No adjustments needed for macronutrients
    protein = protein_ratio * tdee / 4 # 4 calories per gram of protein
    carb = carb_ratio * tdee / 4 # 4 calories per gram of carb
    fat = fat_ratio * tdee / 9 # 9 calories per gram of fat

    return protein, carb, fat

```

```

# Function to calculate required nutrients
def calculate_nutrients(weight, height, age, gender, muscle_gain, weight_goal, workout_days_per_week, workload_intensity):

    # Calculate BMI
    bmi = calculate_bmi(weight, height)

    # Calculate BMR (Basal Metabolic Rate) using Mifflin-St Jeor Equation
    bmr = calculate_bmr(gender, weight, height, age)

    # Calculate TDEE based on BMR based on activity level
    tdee = calculate_tdee(bmr, workout_days_per_week, workload_intensity)

    # Calculate nutrients intake
    proteins, carbs, fats = calculate_macronutrients(weight_goal, muscle_gain, tdee)

    return {
        "BMI": bmi,
        "BMR": bmr,
        "Calories": tdee,
        "Proteins (g)": proteins,
        "Carbs (g)": carbs,
        "Fats (g)": fats
    }

```

```

# Main function
def main():
    st.title("User Registration and Nutrition Calculator")

    user_data = pd.read_csv("userdatafinal.csv", header=0)

    # Login section
    login = st.checkbox("Login")

    if login:
        login_user_id = st.text_input("User ID")
        login_password = st.text_input("Password", type="password")
        login_button = st.button("Login")

        if login_button:
            if login_button:
                login_successful = False
                for index, row in user_data.iterrows():
                    if row['user_id'] == login_user_id and row['password'] == login_password:
                        login_successful = True
                        st.success("Login successful!")
                        st.session_state.is_logged_in = True
                        st.session_state.user_id = login_user_id
                        st.experimental_rerun()
                        st.info("User Dashboard will be displayed here.")
                        st.stop()
                if not login_successful:
                    st.warning("Invalid User ID or Password. Please try again.")

```

```

# Registration section
st.subheader("New User Registration")
name = st.text_input("Name")
user_id = st.text_input("User ID")
password = st.text_input("Password", type="password")
age = st.number_input("Age", min_value=0, max_value=150, step=1)
weight = st.number_input("Weight (kg)", min_value=0.0, max_value=1000.0, step=0.1)
height = st.number_input("Height (cm)", min_value=0.0)
gender = st.radio("Gender", options=["Male", "Female"])
muscle_gain = st.selectbox("Muscle Gain", options=["Yes", "No"])
weight_goals = st.selectbox("Weight Goals", options=["Lose Weight (High Priority)", "Maintain Weight", "Gain Weight (Muscle)"])
workout_days_per_week = st.slider("Number of Workout Days", min_value=0, max_value=7, value=3)
workload_intensity = st.selectbox("Workout Intensity", options=["Low", "Medium", "High"])

register_button = st.button("Register")

if register_button:
    # Calculate BMI and required nutrients
    bmi = calculate_bmi(weight, height)
    nutrients = calculate_nutrients(weight, height, age, gender, muscle_gain, weight_goals, workout_days_per_week, workload_intensity)
    new_row = {
        'name': name, 'user_id': user_id,
        'password': password, 'age': age, 'weight': weight,
        'height': height, 'bmi': bmi, 'gender': gender,
        'muscle_gain_goal': muscle_gain, 'weight_goal': weight_goals, 'workout_days_per_week': workout_days_per_week, 'workload_intensity': workload_intensity,
        'bmr': nutrients["BMR"], 'tdee': nutrients["Calories"], 'protein': nutrients["Proteins (g)"], 'carb': nutrients["Carbs (g)"], 'fat': nutrients["Fats (g)"]}

    user_data.loc[len(user_data)] = new_row
    user_data.to_csv("userdatafinal.csv", index=False)
    st.subheader("Required Nutrients")
    df = pd.DataFrame([nutrients])
    st.write(df)
    st.session_state.is_logged_in = True
    st.session_state.user_id = user_id

    st.experimental_rerun()

# Run the app
if __name__ == "__main__":
    main()

```

Dashboard:

The Dashboard Page is the core portal where users may access their personalised information and recommendations. The dashboard shows the user's name, age, weight, height, and gender. This data gives context for personalised recommendations and enables consumers to confirm their knowledge. Users can view their fitness goals, including muscle gain, weight, training frequency, and intensity. Clear awareness of these goals helps users stay focused and motivated to achieve their intended results.

It also displays the user's current calorie intake and goal.

It allows users to track their calorie consumption in relation to their goals, whether they are losing weight, maintaining it, or gaining weight. Users can track their macronutrient intake, which includes protein, fat, and carbohydrates. The dashboard compares the user's current intake and recommended targets based on their fitness aspirations and dietary preferences.

Users receive personalised health advice based on their goals, activities, and demographic information. These suggestions provide users with vital insights and assistance to help them make smart diet and lifestyle decisions.

The dashboard's dynamic capabilities enable users to change their information and goals as needed. Users can update their demographic information, exercise objectives, or preferences straight from the dashboard, ensuring that their data is correct and up to date. Overall, the dashboard is a comprehensive tool for users to track their progress, receive personalised advice, and stay inspired as they work towards better diet and exercise.

Nutrition and Health Dashboard

User Demographic Information

Name:

Erzhan Ok.

User ID:

erzhan223

Age:

22 years

Weight:

77.0 kg

Height:

178.0 cm

User Goals

Muscle Gain Goal:

Yes

Weight Goal:

Maintain Weight

Workout Days per Week:

5

Workload Intensity:

Medium

	user_id	calories_left	proteins_left	fats_left	carbs_left	calories_goal	proteins_goal	fats_goal
0	erzhan223	1,089.3426	34.4799	37.8684	157.0798	2,866.3515	286.6351	95.545

Calories Consumption and Goals



Proteins Consumption and Goals




Fats Consumption and Goals





Carbs Consumption and Goals




Health Tips Based on User Goals

 **Tip 1:** To support muscle gain, focus on consuming enough protein. Aim for high-quality protein sources like lean meats, eggs, and dairy.

 **Tip 2:** To maintain weight, ensure your calorie intake matches your energy expenditure. Monitor your nutrient consumption regularly.

 **Tip 3:** Consistency is key! Stick to your workout schedule to achieve your fitness goals and support your nutrient needs.

 **Tip 4:** Finding a balanced workout routine that suits your lifestyle and fitness level is key to long-term

```

import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

def main():

    # Load user data from CSV based on user_id stored in session_state
    user_data = pd.read_csv("userdatafinal.csv")
    user_data = user_data[user_data['user_id'] == st.session_state.user_id].iloc[0]
    nutrition_tracking_df = pd.read_csv("nutrition_tracking.csv")
    # Check if the row for the user_id and date exists
    mask = (nutrition_tracking_df['user_id'] == st.session_state.user_id)
    existing_rows = nutrition_tracking_df.loc[mask]
    if not existing_rows.empty:

        nutrition_data = {
            'user_id': [user_data['user_id']],
            'calories_left': user_data['tdee'] - nutrition_tracking_df['calories_left'],
            'proteins_left': user_data['protein'] - nutrition_tracking_df['proteins_left'],
            'fats_left': user_data['fat'] - nutrition_tracking_df['fats_left'],
            'carbs_left': user_data['carb'] - nutrition_tracking_df['carbs_left']
        }
    else:
        nutrition_data = {
            'user_id': [user_data['user_id']],
            'calories_left': 0,
            'proteins_left': 0,
            'fats_left': 0,
            'carbs_left': 0
        }

    nutrition_df = pd.DataFrame(nutrition_data)

    # User goals for the health tips
    user_goals = {
        'calories_goal': user_data['tdee'],
        'proteins_goal': user_data['protein'],
        'fats_goal': user_data['fat'],
        'carbs_goal': user_data['carb']
    }

```

```

# Create Streamlit App
st.title('Nutrition and Health Dashboard')

# Display User Demographic Information and User Goals side by side
col1, col2 = st.columns(2)

# User Demographic Information
with col1:
    st.subheader('User Demographic Information')
    st.markdown("""
<style>
.key {
    font-weight: bold;
    font-style: italic;
}
</style>
""", unsafe_allow_html=True)
    st.text_input('Name:', user_data['name'], key='name', disabled=True)
    st.text_input('User ID:', user_data['user_id'], key='user_id', disabled=True)
    st.text_input('Age:', str(user_data['age']) + ' years', key='age', disabled=True)
    st.text_input('Weight:', str(user_data['weight']) + ' kg', key='weight', disabled=True)
    st.text_input('Height:', str(user_data['height']) + ' cm', key='height', disabled=True)

# User Goals
with col2:
    st.subheader('User Goals')
    st.text_input('Muscle Gain Goal:', user_data['muscle_gain_goal'], key='muscle_gain_goal', disabled=True)
    st.text_input('Weight Goal:', user_data['weight_goal'], key='weight_goal', disabled=True)
    st.text_input('Workout Days per Week:', str(user_data['workout_days_per_week']), key='workout_days_per_week', disabled=True)
    st.text_input('Workload Intensity:', user_data['workload_intensity'], key='workload_intensity', disabled=True)

# Merge nutrition data with user goals
merged_df = pd.merge(nutrition_df, pd.DataFrame([user_goals]), left_index=True, right_index=True)

```

```
# Function to create horizontal bar load
def horizontal_bar_load(value, goal, color):
    percentage = (value / goal) * 100
    fig, ax = plt.subplots(figsize=(8, 1))
    ax.barh([1], percentage, color=color)
    ax.set_xlim(0, 100)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.text(percentage - 5, 1, f'{percentage:.1f}%', color='white', va='center', ha='right', fontsize=10)
    return fig

# Visualizations with Horizontal Bar Load
for nutrient, color in zip(['calories', 'proteins', 'fats', 'carbs'], ['skyblue', 'green', 'purple', 'blue']):
    st.subheader(f'{nutrient.capitalize()} Consumption and Goals')

# Display Horizontal Bar Load
st.pyplot(horizontal_bar_load(merged_df[f'{nutrient}_left'][0], merged_df[f'{nutrient}_goal'][0], color=color))

# Generate Health Tips based on user goals
st.subheader('Health Tips Based on User Goals')

if user_data['muscle_gain_goal'] == 'Yes':
    st.markdown("💪 **Tip 1:** To support muscle gain, focus on consuming enough protein. Aim for high-quality protein sources like lean meats, eggs, and dairy.")
else:
    st.markdown("🌱 **Tip 1:** Even if you're not focusing on muscle gain, including some protein in every meal can help maintain muscle mass and support overall health.")

if user_data['weight_goal'] == 'Gain Weight':
    st.markdown("🍌 **Tip 2:** To gain weight healthily, focus on nutrient-dense foods like whole grains, healthy fats, and protein-rich foods.")
elif user_data['weight_goal'] == 'Lose Weight':
    st.markdown("🥗 **Tip 2:** To lose weight, aim for a balanced diet with a slight calorie deficit. Include plenty of vegetables, lean proteins, and whole grains.")
else:
    st.markdown("⚖️ **Tip 2:** To maintain weight, ensure your calorie intake matches your energy expenditure. Monitor your nutrient consumption regularly.")

if user_data['workout_days_per_week'] >= 3:
    st.markdown("🏋️ **Tip 3:** Consistency is key! Stick to your workout schedule to achieve your fitness goals and support your nutrient needs.")
else:
    st.markdown("🚶 **Tip 3:** Even light exercise like walking can have health benefits. Try to incorporate movement into your daily routine.")

if user_data['workload_intensity'] == 'Low':
    st.markdown("💡 **Tip 4:** Consider increasing your workout intensity gradually to challenge your muscles and boost metabolism.")
elif user_data['workload_intensity'] == 'High':
    st.markdown("🌟 **Tip 4:** High-intensity workouts can be effective but ensure you're giving your body enough rest and recovery time.")
else:
    st.markdown("🧘 **Tip 4:** Finding a balanced workout routine that suits your lifestyle and fitness level is key to long-term success.")

if __name__ == "__main__":
    main()
```

Recipes page

The Application also consisted of a Recipe Page where the user can filter and choose recipes based on their preferences.

Go to

Recipes

Filter recipes based on:

Protein per serving (grams)

0211

Calories per serving

03734

Fats per serving (grams)

0349

Carbohydrates per serving (grams)

0787

Include ingredients (comma-separated)

Filtered Recipes:

	id	label	proteins_per_serving
	0	Recipe: Vegetarian Matzo Ball Soup	8.168
	1	Individual vegetarian lasagnes	62.1694
	2	White bean vegetarian meatballs	33.9706
	3	Farro with Greens, Caramelized Onions and Vegetarian Chorizo	33.0439
	4	Vegetarian Stir-Fried Garlic Scapes	9.5257
	5	Vegetarian Shepherd's Pie	16.8284
	6	Vegetarian Chili Mac-Inspired Baked Ziti Is the Definition of Hands-Off Cooking	28.3887
	7	Veggie toad-in-the-hole	23.142
	8	Vegetarian and Pork Dumpling Fillings	5.422
	9	Honey mustard cocktail sausages (mini toad-in-the-holes)	11.7323

Select a recipe ID to see suggestions:

1

Selected Recipe Details:

0

Recipe: Vegetarian Matzo Ball Soup

ingredients: 2 cups matzo meal (I prefer to buy unsalted crackers and pulse in the food processor)

url: <https://www.thekitchn.com/comforting-recipe-162258>

Top 5 similar recipes:

	id	label	ingredients	
	1,440	1,450	Scrambled Eggs with Truffles	3 large eggs Kosher salt 1 tbsp. olive oil 1 sm.
	2,339	2,340	Vegetarian Matzo Ball Soup	4 large eggs 1/4 cup vegetable oil 1 teaspoon
	4,263	4,264	Matzo Ball Soup for Passover	Homemade Chicken Broth for Matzo Ball Sou
	1,900	1,901	Eating for Two: Peanut Butter Cookies Recipe	2 1/2 cups all purpose flour 1 teaspoon salt 1
	1,928	1,929	Lactation Cookie Recipe To Increase Breast Milk Supply	2 cups organic rolled oats (where to buy) 1/2

```

import streamlit as st
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def main():
    # Load data
    @st.cache(allow_output_mutation=True)
    def load_data():
        return pd.read_csv('recipes_updated.csv')

    # Setup similarity matrix
    @st.cache(allow_output_mutation=True)
    def setup_similarity_matrix(data):
        vectorizer = TfidfVectorizer(stop_words='english')
        ingredient_matrix = vectorizer.fit_transform(data['ingredients'])
        cosine_sim = cosine_similarity(ingredient_matrix)
        return cosine_sim, vectorizer

    data = load_data()
    cosine_sim, vectorizer = setup_similarity_matrix(data)

    # Sidebar for nutritional and ingredient filters
    st.sidebar.header('Filter recipes based on:')
    protein = st.sidebar.slider('Protein per serving (grams)', 0, int(data['proteins_per_serving'].max()), (0, int(data['proteins_per_serving'].max())))
    calories = st.sidebar.slider('Calories per serving', 0, int(data['calories_per_serving'].max()), (0, int(data['calories_per_serving'].max())))
    fats = st.sidebar.slider('Fats per serving (grams)', 0, int(data['fats_per_serving'].max()), (0, int(data['fats_per_serving'].max())))
    carbs = st.sidebar.slider('Carbohydrates per serving (grams)', 0, int(data['carbs_per_serving'].max()), (0, int(data['carbs_per_serving'].max())))
    ingredients_input = st.sidebar.text_input('Include Ingredients (comma-separated)', '')

    # Filter data
    filtered_data = data[(data['proteins_per_serving'] >= protein[0]) & (data['proteins_per_serving'] <= protein[1]) &
                        (data['calories_per_serving'] >= calories[0]) & (data['calories_per_serving'] <= calories[1]) &
                        (data['fats_per_serving'] >= fats[0]) & (data['fats_per_serving'] <= fats[1]) &
                        (data['carbs_per_serving'] >= carbs[0]) & (data['carbs_per_serving'] <= carbs[1])]

    if ingredients_input:
        included_ingredients = [ingredient.strip().lower() for ingredient in ingredients_input.split(',')]
        filtered_data = filtered_data[filtered_data['ingredients'].apply(lambda x: any(ingredient.lower() in x.lower() for ingredient in included_ingredients))]

    # Display recipes in the main page
    st.write('Filtered Recipes:', filtered_data[['id', 'label', 'proteins_per_serving', 'calories_per_serving', 'fats_per_serving', 'carbs_per_serving']])

```

```

# Selecting a recipe
selected_recipe_id = st.selectbox('Select a recipe ID to see suggestions:', filtered_data['id'])
selected_recipe = data[data['id'] == selected_recipe_id]

# Display the selected recipe details
if not selected_recipe.empty:
    st.write('Selected Recipe Details:', selected_recipe[['label', 'ingredients', 'url']].iloc[0])

    # Show similar recipes
    index = selected_recipe.index[0]
    scores = list(enumerate(cosine_sim[index]))
    sorted_scores = sorted(scores, key=lambda x: x[1], reverse=True)
    top_5_recipes = [data.iloc[i[0]] for i in sorted_scores[1:6]] # skip the first one as it is the recipe itself

    st.write('Top 5 similar recipes:', pd.DataFrame(top_5_recipes)[['id', 'label', 'ingredients']])

    # Display steps one by one
    st.subheader("Recipe Preparation Steps:")
    st.markdown("### Ingredients:")
    steps = selected_recipe['ingredient_lines']
    st.write(steps.iloc[0])

    # Display recipe URL
    st.markdown("### Recipe URL:")
    st.write(selected_recipe['url'].iloc[0])

    # Consume Recipe Today button
    if st.button('Consume Recipe Today'):
        st.session_state.selected_recipe_id = selected_recipe_id
        st.session_state.protein = selected_recipe['proteins_per_serving']
        st.session_state.calories = selected_recipe['calories_per_serving']
        st.session_state.carbs = selected_recipe['carbs_per_serving']
        st.session_state.fats = selected_recipe['fats_per_serving']
        st.write(f'Recipe ID {selected_recipe_id} has been selected to consume today!')

```

```

def update_nutrition_tracking(user_id, calories_to_remove, proteins_to_remove, fats_to_remove, carbs_to_remove):
    nutrition_tracking_df = pd.read_csv("nutrition_tracking.csv")

    # Check if the row for the user_id and date exists
    mask = (nutrition_tracking_df['user_id'] == user_id)
    existing_rows = nutrition_tracking_df.loc[mask]

    if not existing_rows.empty:
        # Modify the existing row
        nutrition_tracking_df.loc[mask, 'calories_left'] += calories_to_remove
        nutrition_tracking_df.loc[mask, 'proteins_left'] += proteins_to_remove
        nutrition_tracking_df.loc[mask, 'fats_left'] += fats_to_remove
        nutrition_tracking_df.loc[mask, 'carbs_left'] += carbs_to_remove
    else:
        # Create a new entry
        new_entry = {
            'user_id': user_id,
            'calories_left': calories_to_remove[1],
            'proteins_left': proteins_to_remove[1],
            'fats_left': fats_to_remove[1],
            'carbs_left': carbs_to_remove[1]
        }

        # Append the new entry to the DataFrame
        nutrition_tracking_df.loc[len(nutrition_tracking_df)] = new_entry

    # Write the updated DataFrame back to CSV
    nutrition_tracking_df.to_csv('nutrition_tracking.csv', index=False)

# Example usage
update_nutrition_tracking(st.session_state.user_id, st.session_state.calories,
                           st.session_state.protein, st.session_state.fats, st.session_state.carbs)

if __name__ == "__main__":
    main()

```

A web application called the Streamlit Recipe Page was created to give users an easy way to browse, filter, and choose recipes according to their dietary requirements and ingredient preferences. Users of the app can browse through a variety of recipes, examine the recipe details, and choose a recipe to follow for the day. On the basis of the user's selection, it also makes similar recipe suggestions.

Features

1. Filtering recipes

The following criteria can be used by users to filter recipes:

grams of protein per serving

Calories for every serving

grams of fat in a serving

grams of carbohydrates in each serving

Ingredients listed (separated by commas)

2. Recipe Choosing

From the filtered list, users can choose a recipe to view its details.

The label, ingredients, and URL of the selected recipe are displayed by the application.

3. Comparable Recipe Ideas

Using cosine similarity, the application makes recipe recommendations based on the selected recipe.

4. Use of Recipes

Customers can choose a recipe to eat throughout the day.

The nutritional values of the selected recipe are used by the application to update nutrition tracking.

Execution

Python and Streamlit are used to implement the Streamlit Recipe Page. The libraries that are utilized are as follows:

streamlit: For constructing the online program.

pandas: For analysis and data manipulation.

sklearn: For computing cosine similarity.

mysql.connector: For establishing a MySQL connection.

Loading Data

Data about recipes is imported from a CSV file called recipes_updated.csv.

Recipe labels, ingredients, nutritional values, and URLs are all included in the data.

Recipe Sorting

Recipes can be filtered by users according to ingredients and nutritional preferences.

Protein, calories, fats, carbs, and included ingredients are among the filters.

Similar Recipe Ideas

Using cosine similarity, the application makes recipe recommendations based on the selected recipe.

The components of the recipe are used to calculate similarity.

Utilization of Recipes

Customers can choose a recipe to eat throughout the day.

The nutritional values of the chosen recipe are used to update the nutrition tracking.

Users can explore and choose recipes based on their preferences with ease using the Streamlit Recipe Page's interactive and user-friendly interface. With features like nutrition tracking, filtering and selecting recipes, and suggesting similar recipes, it's a useful tool for anyone looking for meal planning ideas and inspiration.

A web application called the Streamlit Recipe Page was created to give users an easy way to browse, filter, and choose recipes according to their dietary requirements and ingredient preferences. Users of the app can browse through a variety of recipes, examine the recipe

details, and choose a recipe to follow for the day. On the basis of the user's selection, it also makes similar recipe suggestions.

Future Goal

Nutrition Me strives to be more than just calorie counting and food suggestions in the increasing ecosystem of diet guidance apps. We foresee a future in which the user experience goes beyond passive content consumption, creating a dynamic and participatory environment that enables people to reach their health and wellness goals. This research digs into the future of Nutrition Me, examining ways for improving user interaction, empowering recipe development and customisation, and forming collaborative partnerships with fitness centres to provide a genuinely symbiotic user experience.

Improving User Engagement: A Social and Interactive Ecosystem.

Nutrition Me prioritises user interaction by cultivating a social and dynamic ecology, rather than providing static suggestions. This ecosystem will consist of the following elements:

- A robust recipe rating and review system will be an essential component of user involvement. This technology will allow users to not only share their gastronomic experiences, but also influence the decisions of others. Users can provide useful knowledge about dish flavour, ease of preparation, and success in meeting dietary objectives by leaving detailed reviews and ratings. This develops a sense of community and elevates users from passive consumers to active creators of delicious and productive meal plans.
- Social Sharing Features: Integration with social media platforms allows users to share their meal creations and fitness achievements with friends and family. This social affirmation can help individuals feel more accountable and move forward with their wellness initiatives. Furthermore, sharing victories might motivate others in the Nutrition Me community, resulting in a positive feedback loop.
- Dedicated Community Forum: The creation of a dedicated forum will allow members to connect, share recipe modifications, discuss fitness issues, and offer mutual support. This creates a sense of belonging and camaraderie as people embark on their wellness journeys together. The forum can also serve as a helpful knowledge resource, allowing users to learn insights and best practices from their peers.

Empowering Recipe Creation and Customization: A User-Centric Approach to Nutrition

Recognising that a one-size-fits-all approach to nutrition is useless, Nutrition Me will allow users to participate actively in their gastronomic journeys:

- Recipe Customisation Tools: The software will offer user-friendly recipe customisation tools. These capabilities will enable users to change portion sizes, substitute items depending on allergies or preferences, and tailor recipes to their individual nutritional requirements and cultural backgrounds. This creates a sense of ownership and control over meal plans, which promotes long-term adherence.
- My Recipe Creation: Moving beyond basic customisation, the app will provide user-friendly recipe authoring features. Users can enter their own recipes, including full

information on materials, instructions, and nutritional content. This allows users to share their favourite family recipes, cultural delicacies, or unique culinary inventions with the larger Nutrition Me community. This generates a sense of teamwork and knowledge exchange, which improves the overall user experience.

- **Personalised Meal Planning Integration:** A key element will be the seamless incorporation of user-created recipes into personalised meal planning. This allows consumers to create meal plans that not only meet their dietary goals, but also reflect their unique preferences and cultural tastes. This user-centric approach encourages agency and long-term dietary sustainability.

Collaboration with Gyms and Workout routine:

- **Gym Integration:** The app will explore strategic collaborations with clubs to allow users to effortlessly connect their Nutrition Me accounts to their gym memberships. This integration will enable a new level of personalisation in training programmes. Gyms can use user data from Nutrition Me, such as dietary objectives and calorie expenditure, to create personalised workout regimens that meet individual needs and maximise outcomes.
- **Integrated Workout Routine Library:** The app will have a comprehensive library of workout routines geared to various fitness goals. Users can choose routines depending on their gym memberships, preferred training styles, and equipment availability. This allows users to create fitness programmes that work smoothly with their personalised nutrition plans, promoting a holistic approach to wellness.