ДОДАТОК А

Скріншоти UI/UX

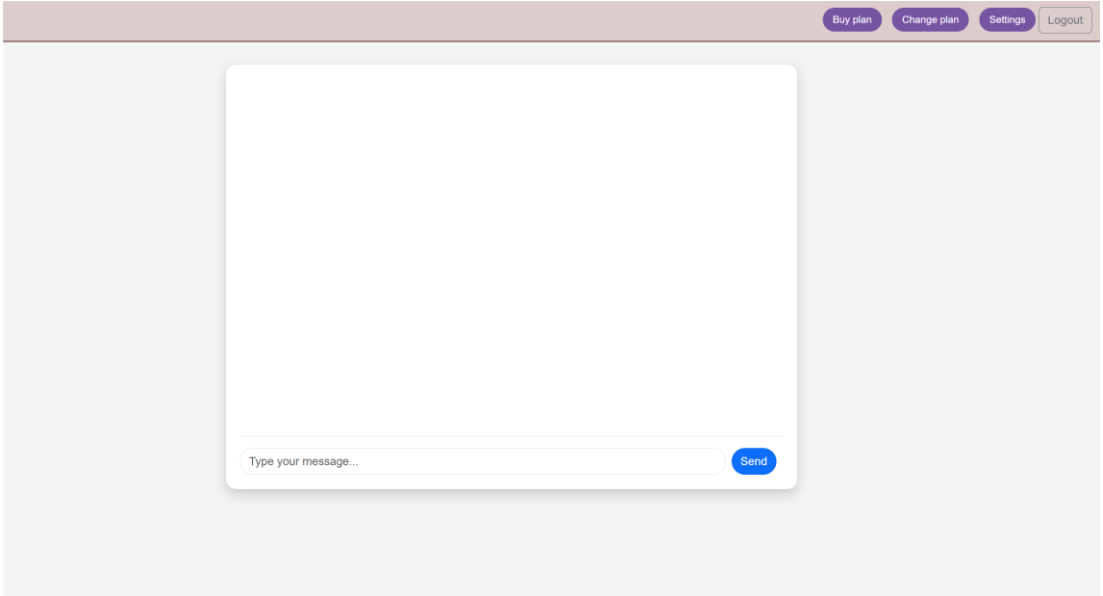

Рисунок А.1 – Головна сторінка

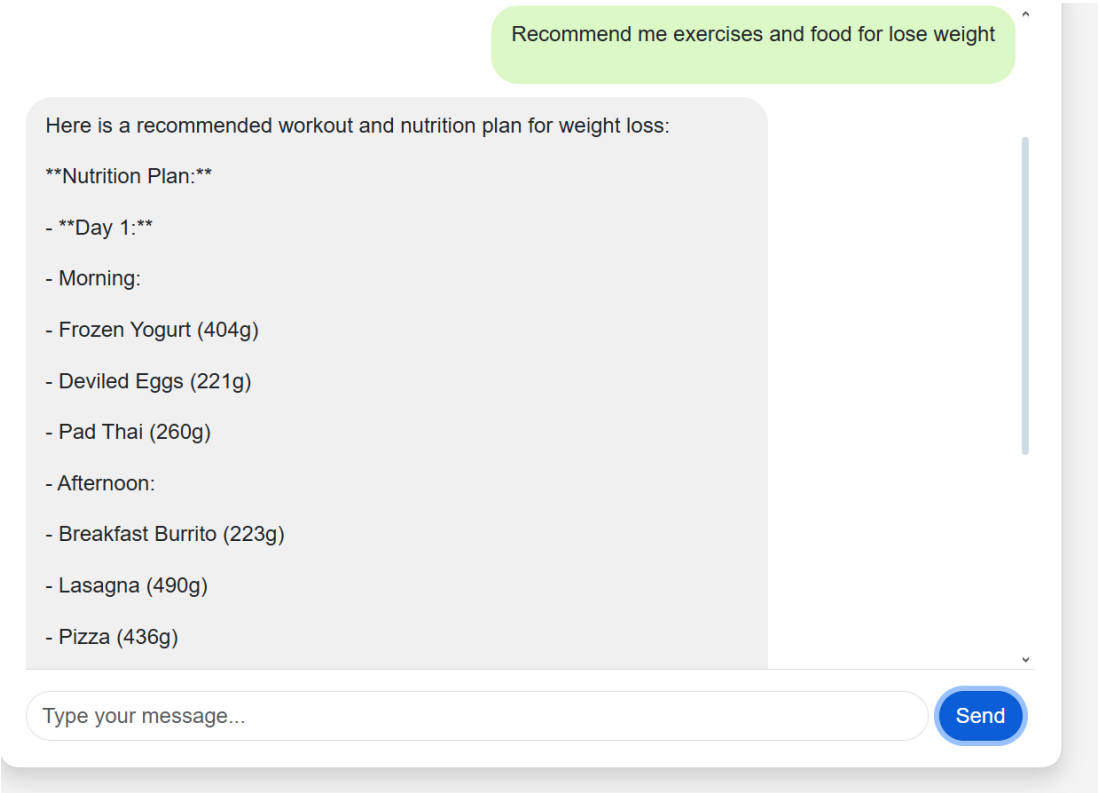

Рисунок А.2 – Процесс чатування

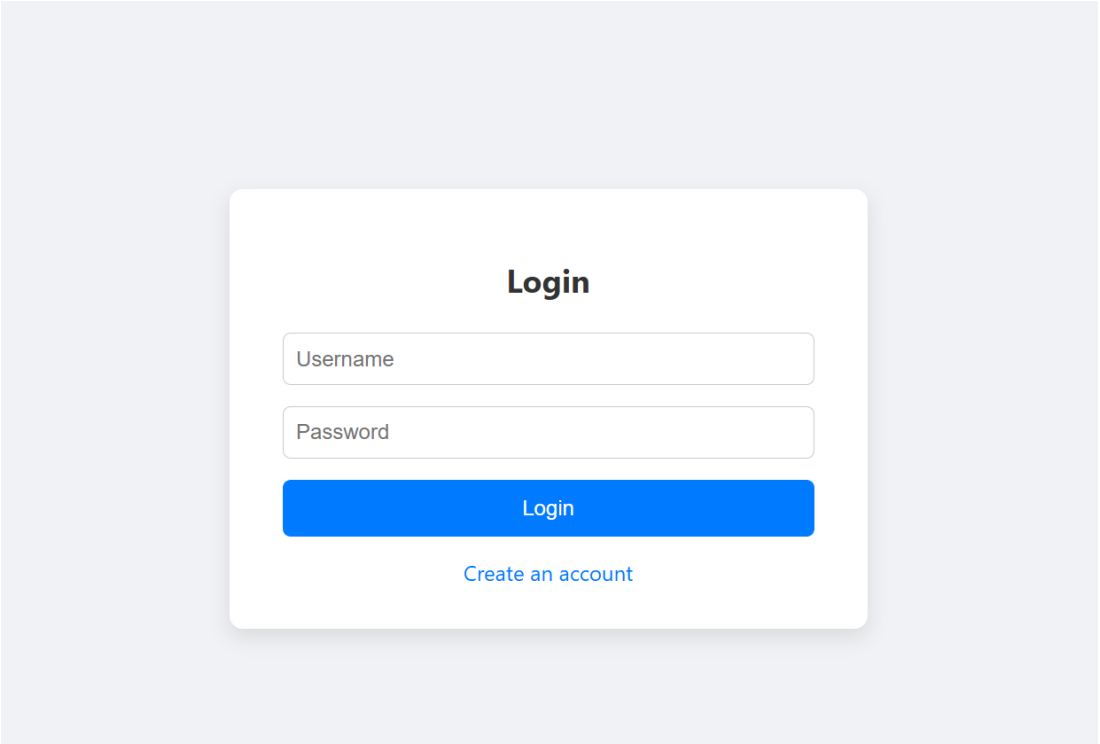

Рисунок А.3 – Процес входу до акаунту

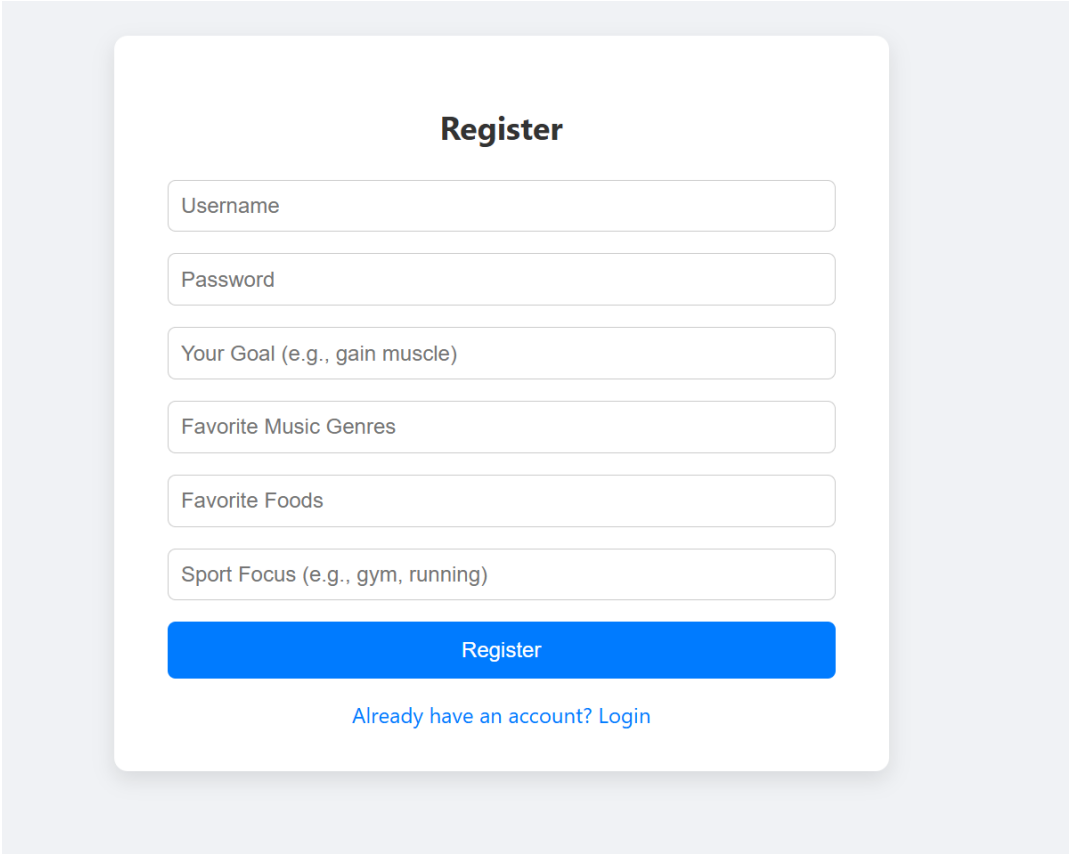

Рисунок А.4 – Процес створення акаунту

ДОДАТОК Б

Коди програми

Код серверної частини:

```
     import pandas as pd
from flask import Flask, request, jsonify, render_template,
redirect, session, url_for, flash
from flask_cors import CORS
import openai
from database import init_db
from graph import agent
from models import User, Preference, db

app = Flask(__name__)
CORS(app)
app.secret_key = "supersecret"
init_db(app)

# Питання для опитування
INITIAL_SURVEY_QUESTIONS = [
    {"question": "Do u like chicken?", "type": "food", "item":
"chicken"},
    {"question": "Do u like rice?", "type": "food", "item":
"rice"},
    {"question": "Do u like broccoli?", "type": "food",
"item": "broccoli"},
    {"question": "DO u like squats?", "type": "exercise",
"item": "squat"},
    {"question": "Do u like bench press?", "type": "exercise",
"item": "bench press"},
    {"question": "Do u like running?", "type": "exercise",
"item": "running"},
    {"question": "Do you like running?", "type": "food",
"item": "turky"},
    {"question": "Do u like buckwheat?", "type": "food",
"item": "buckwheat"},
    {"question": "Do you like to do plank?", "type":
"exercise", "item": "plank"},
    {"question": "Do you like biking?", "type": "exercise",
"item": "bike"},
]

# Функция для загрузки данных из CSV
def load_nutrition_data():
```

```
    return pd.read_csv("cleaned_nutrition.csv")

def load_gym_data():
    return pd.read_csv("cleaned_gym.csv")

# Функция для определения категории по калорийності
def get_food_category(food_name, nutrition_data):
    food_row = nutrition_data[nutrition_data["label"] ==
food_name]
    if not food_row.empty:
        calories = food_row["calories"].values[0]
        return "high calories" if calories > 250 else "low
calories"
    return "unknown"

# Функция для определения категории по интенсивності
def get_exercise_category(exercise_name, gym_data):
    exercise_row = gym_data[gym_data["Exercise Name"] ==
exercise_name]
    if not exercise_row.empty:
        intensity = exercise_row["intensity"].values[0]
        return "high intensity" if intensity > 7 else "low
intensity"
    return "unknown"

# Функція для збереження переваг користувача
def save_initial_preferences(user_id, answers):
    pref = Preference.query.filter_by(user_id=user_id).first()
    if not pref:
        # Создаём новую запись, если не существует
        pref = Preference(user_id=user_id)
        db.session.add(pref)

    sport_weights = {}
    food_weights = {}
    liked_exeicies = {}
    print(f"siska :{answers}")
    for ans in answers:
        print(f"ansewr {ans}")
        if 'category' not in ans:
            continue  # Пропустить некорректные ответы

        item = ans["item"]
        liked = ans["liked"]

        if ans["category"] == "food":
```

```
            food_weights[item] = 3.5 if liked else 1.0
        elif ans["category"] == "exercise":
            liked_exeicies[item] = 3.5 if liked else 1.0

    pref.food_weights = food_weights
    pref.liked_exercises = liked_exeicies

    if pref.liked_exercises["bench press"] > 2:
        pref.sport_weights["bulk"] = 3
    else:
        pref.sport_weights["cut"] = 3

    db.session.commit()

# Ендпоінт для збереження відповідей
@app.route("/api/survey", methods=["POST"])
def submit_survey():
    if "user_id" not in session:
        return jsonify({"error": "Unauthorized"}), 403

    data = request.get_json()
    answers = data.get("answers", [])
    print("Raw answers:", answers)

    if not isinstance(answers, list):
        return jsonify({"error": "Invalid format for answers,
expected a list."}), 400

    # Форматування відповідей
    formatted_answers = []
    for answer in answers:
        if answer['type'] == 'food':
            formatted_answers.append({
                "category": "food",
                "item": answer['item'],
                "liked": answer['liked']
            })
        elif answer['type'] == 'exercise':
            formatted_answers.append({
                "category": "exercise",
                "item": answer['item'],
                "liked": answer['liked']
            })

    user_id = session["user_id"]
    print("User ID:", user_id)
```

```
    print("Formatted answers:", formatted_answers)

    try:
        save_initial_preferences(user_id, formatted_answers)
    except Exception as e:
        print("Error saving preferences:", str(e))
        return jsonify({"error": "Failed to save preferences",
"details": str(e)}), 500

    return jsonify({"status": "preferences saved"})

@app.route("/api/rate", methods=["POST"])
def rate_items():
    if "user_id" not in session:
        return jsonify({"error": "Unauthorized"}), 403

    data = request.get_json()
    item = data.get("item")
    score = float(data.get("score", 0))
    user_id = session["user_id"]

    pref = Preference.query.filter_by(user_id=user_id).first()
    if not pref:
        return jsonify({"error": "No preferences found"}), 404

    if pref.food_weights:
        for food_item in pref.food_weights:
            pref.food_weights[food_item] += score

    if pref.sport_weights:
        for sport_item in pref.sport_weights:
            pref.sport_weights[sport_item] += score

    db.session.commit()
    return jsonify({"status": "ok"})




# Створення сторінки для опитування
@app.route("/survey")
def survey_page():
    if "user_id" not in session:
        return redirect(url_for("login"))

    user_id = session["user_id"]
```

```
    pref = Preference.query.filter_by(user_id=user_id).first()

    # если веса уже заданы, значит опрос пройден
    if pref and pref.food_weights and pref.sport_weights:
        return redirect(url_for("index"))

    return render_template("survey.html",
questions=INITIAL_SURVEY_QUESTIONS, user_id=user_id)

# Логіка для отримання рекомендацій
def get_recommendation(prompt, user_id=None):
    inputs = {"input": prompt}
    if user_id is not None:
        prompt = prompt + f" my user Id is {user_id}"
        inputs = {"input": prompt}
    response = agent.invoke(inputs)
    if isinstance(response, dict) and "agent_outcome" in
response:
        return
response["agent_outcome"].return_values["output"]
    return str(response)

# Головна сторінка
@app.route("/", methods=["GET"])
def index():
    if "user_id" not in session:
        return redirect(url_for("login"))
    return render_template("index.html")

# Логін
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]

        user = User.query.filter_by(username=username).first()
        if user and user.check_password(password):
            session["user_id"] = user.id
            return redirect(url_for("index"))

        flash("Invalid username or password", "error")

    return render_template("login.html")

# Логіка виходу з системи
```

```
@app.route("/logout")
def logout():
    session.clear()
    return redirect(url_for("login"))

# Реєстрація
@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        goal = request.form["goal"]
        music = request.form["music"]
        foods = request.form["foods"]
        sport = request.form["sport"]

        # Проверка, существует ли уже пользователь с таким
именем
        if User.query.filter_by(username=username).first():
            return render_template("register.html",
error="Username already exists")

        # Создаем нового пользователя
        user = User(username=username, goals=goal)
        user.set_password(password)
        db.session.add(user)
        db.session.commit()

        # Пустые веса, чтобы позже заполнить после опроса
        pref = Preference(
            user_id=user.id,
            music_genres=music,
            favorite_foods=foods,
            sport_focus=sport,
            food_weights={},
            sport_weights={}
        )
        db.session.add(pref)
        db.session.commit()

        session["user_id"] = user.id
        return redirect(url_for("survey_page"))

    return render_template("register.html")
```

```
# Рекомендації
@app.route("/api/recommend", methods=["POST"])
def recommend():
    if "user_id" not in session:
        return jsonify({"error": "Unauthorized"}), 403

    data = request.json
    prompt = data.get("prompt", "")
    user_id = session["user_id"]

    pref = Preference.query.filter_by(user_id=user_id).first()
    if pref:
        pref.update_user_goal()

    recommendation = get_recommendation(prompt, user_id)
    return jsonify({"response": recommendation})

# Запуск додатку
if __name__ == "__main__":
    app.run(debug=True)
```

Код моделі:

```
import random
import re

import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from tensorflow.keras import layers, models

from models import User, Preference,db
from flask_sqlalchemy import SQLAlchemy

dba = db

# Загрузка данных
gym_df = pd.read_csv("cleaned_gym.csv")
nutrition_df = pd.read_csv("cleaned_nutrition.csv")
music_df = pd.read_csv("cleaned_music.csv")
# Объединённые текстовые признаки для упражнений
gym_df["combined"] = gym_df["Target_Muscles"] + " " +
gym_df["Main_muscle"] + " " + gym_df["Utility"] + " " +
```

```
gym_df["Equipment"]

goal_keywords = {
    "gain muscles": "bulk",
    "lose weight": "cut",
    "maintain shape": "maintain"
}
period_keywords = {
    "day": 1,
    "week": 7,
    "month": 30,
    "year": 365
}

# ================== #
# === Модель еды ==== #
# ================== #

X_nutrition = nutrition_df[["calories", "protein",
"carbohydrates", "fats", "fiber", "sugars", "sodium"]]
y_nutrition = nutrition_df["label"]

scaler_nutrition = StandardScaler()
X_nutrition_scaled =
scaler_nutrition.fit_transform(X_nutrition)

label_encoder_nutrition = LabelEncoder()
y_nutrition_encoded =
label_encoder_nutrition.fit_transform(y_nutrition)

X_train_nut, X_test_nut, y_train_nut, y_test_nut =
train_test_split(X_nutrition_scaled, y_nutrition_encoded,
test_size=0.2)

model_nutrition = models.Sequential([
    layers.Dense(64, activation='relu',
input_shape=(X_nutrition_scaled.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(len(y_nutrition.unique()),
activation='softmax')
])
model_nutrition.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_nutrition.fit(X_train_nut, y_train_nut, epochs=10,
batch_size=32, validation_data=(X_test_nut, y_test_nut))
```

```
# ====================== #
# === Модель упражнений #
# ====================== #

vectorizer = CountVectorizer()
X_gym = vectorizer.fit_transform(gym_df["combined"])
y_gym = gym_df["Exercise Name"]

label_encoder_gym = LabelEncoder()
y_gym_encoded = label_encoder_gym.fit_transform(y_gym)

X_train_gym, X_test_gym, y_train_gym, y_test_gym =
train_test_split(X_gym.toarray(), y_gym_encoded,
test_size=0.2)

model_gym = models.Sequential([
    layers.Dense(128, activation='relu',
input_shape=(X_gym.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(len(y_gym.unique()), activation='softmax')
])
model_gym.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_gym.fit(X_train_gym, y_train_gym, epochs=10,
batch_size=32, validation_data=(X_test_gym, y_test_gym))

# ====================== #
# === Класс ассистент === #
# ====================== #

class SAssistant:
    def __init__(self, gym_data, nutrition_data, gym_model,
nutrition_model,
                 gym_encoder, nutrition_encoder,
scaler_nutrition, gym_vectorizer,music_data):
        self.gym_data = gym_data
        self.nutrition_data = nutrition_data
        self.gym_model = gym_model
        self.nutrition_model = nutrition_model
        self.gym_encoder = gym_encoder
        self.nutrition_encoder = nutrition_encoder
        self.scaler_nutrition = scaler_nutrition
        self.gym_vectorizer = gym_vectorizer
        self.music_data = music_data

    def extract_goal_and_period(self, text):
```

```
        goal = "bulk"
        period = 1
        for key in goal_keywords:
            if key in text.lower():
                goal = goal_keywords[key]
        for key in period_keywords:
            if key in text.lower():
                print(key)
                period = period_keywords[key]
        return goal, period

    def generate_workout_plan(self, goal, period,
sport_weights=None):
        plan = []

        if goal == "cut":
            filtered = self.gym_data[self.gym_data["Difficulty
(1-5)"] <= 2]
        elif goal == "bulk":
            filtered = self.gym_data[self.gym_data["Difficulty
(1-5)"] >= 4]
        else:
            filtered = self.gym_data[
                (self.gym_data["Difficulty (1-5)"] >= 2) &
(self.gym_data["Difficulty (1-5)"] <= 4)]

        combined = filtered["combined"]
        X_input = self.gym_vectorizer.transform(combined)
        predictions =
self.gym_model.predict(X_input.toarray(), verbose=0)
        preds_mean = predictions.mean(axis=0)

        exercise_scores =
list(zip(self.gym_encoder.inverse_transform(np.arange(len(pred
s_mean))), preds_mean))

        if sport_weights:
            def weight_bonus(name):
                for key, weight in sport_weights.items():
                    if key.lower() in name.lower():
                        return weight
                return 0

            exercise_scores = [(name, score +
weight_bonus(name)) for name, score in exercise_scores]
```

```
        sorted_exercises = sorted(exercise_scores, key=lambda
x: x[1], reverse=True)

        phase_length = 10 if period >= 30 else period
        num_phases = period // phase_length
        reps = 8 if goal == "bulk" else (12 if goal == "cut"
else 10)

        for phase_num in range(num_phases):
            phase_start = phase_num * phase_length + 1
            phase_end = phase_start + phase_length - 1
            phase_exercises = sorted_exercises[phase_num * 10:
(phase_num + 1) * 10]
            phase_names = [name for name, _ in
phase_exercises]

            for day in range(phase_start, phase_end + 1):
                day_exercises = random.sample(phase_names,
min(5, len(phase_names)))
                workout = [f"{ex} 4x{reps}" for ex in
day_exercises]
                plan.append(f"Day {day}: " + ",
".join(workout))

        return "\n".join(plan)

    def generate_nutrition_plan(self, period,
favorite_foods=None, food_weights=None):
        plan = []
        favorite_foods = favorite_foods or []

        features = self.nutrition_data[["calories", "protein",
"carbohydrates", "fats", "fiber", "sugars", "sodium"]]
        X_input = self.scaler_nutrition.transform(features)
        predictions = self.nutrition_model.predict(X_input,
verbose=0)
        top_indices = predictions.argsort(axis=1)[:, -1]

        all_foods = self.nutrition_data["label"].values
        food_index_map = {food: idx for idx, food in
enumerate(all_foods)}

        for day in range(period):
            selected_indices = []

            if favorite_foods:
```

```
                for fav in favorite_foods:
                    idx = food_index_map.get(fav)
                    if idx is not None:
                        selected_indices.append(idx)

            remaining_count = 9 - len(selected_indices)

            if remaining_count > 0:
                food_scores = []
                for idx in top_indices:
                    label =
self.nutrition_encoder.inverse_transform([idx])[0]
                    weight = food_weights.get(label, 0) if
food_weights else 0
                    food_scores.append((idx, weight))

                food_scores = sorted(set(food_scores),
key=lambda x: x[1], reverse=True)
                top_weighted_indices = [idx for idx, _ in
food_scores[:remaining_count]]
                selected_indices.extend(top_weighted_indices)

            selected_indices = selected_indices[:9]
            meals = {
                "Morning": selected_indices[:3],
                "Afternoon": selected_indices[3:6],
                "Evening": selected_indices[6:9]
            }

            formatted_meals = []
            for meal_time, idxs in meals.items():
                foods =
self.nutrition_encoder.inverse_transform(idxs)
                food_with_weights = [f"{food}
{random.randint(200, 500)}g" for food in foods]
                formatted_meals.append(f"{meal_time}: " + ",
".join(food_with_weights))

            plan.append(f"Day {day + 1}:\n" +
"\n".join(formatted_meals))

        return "\n\n".join(plan)

    def random_meal(self):
        features = self.nutrition_data[["calories", "protein",
"carbohydrates", "fats", "fiber", "sugars", "sodium"]]
```

```
        X_input = self.scaler_nutrition.transform(features)
        predictions = self.nutrition_model.predict(X_input,
verbose=0)
        idx = predictions.mean(axis=0).argmax()
        food =
self.nutrition_encoder.inverse_transform([idx])[0]
        return f"{food} {random.randint(200, 500)}g"

    def recommend_plan(self, prompt):
        goal, period = self.extract_goal_and_period(prompt)

        if "workout" in prompt.lower():
            return f"Workout
Plan:\n{self.generate_workout_plan(goal, period)}"

        if "food" in prompt.lower() or "nutrition" in
prompt.lower():
            return f"Nutrition
Plan:\n{self.generate_nutrition_plan(period)}"

        return f"Workout
Plan:\n{self.generate_workout_plan(goal, period)}\n\nNutrition
Plan:\n{self.generate_nutrition_plan(period)}"

    def random_music(self,prompt):
        # Randomly choose a song from the dataset
        random_song = self.music_data.sample(n=1).iloc[0]
        return f"Random Music Recommendation:
{random_song['track_name']} by {random_song['artist_name']}
(Genre: {random_song['genre']})"

    def recommend_music_by_genre(self, user_id, prompt):
        # Fetch user preferences
        prefs = self.get_user_preferences(user_id)

        if not prefs or not prefs['music_genres']:
            return "No music genre preferences found for this
user."

        # Get the preferred music genres (this might be a
string of genres or a list)
        preferred_genres = prefs['music_genres']

        # If it's a single string, convert it into a list of
genres
        if isinstance(preferred_genres, str):
```

```
            preferred_genres = [genre.strip() for genre in
preferred_genres.split(',')]

        # Ensure the list of preferred genres is unique (to
avoid repeating genres in the recommendation process)
        preferred_genres = list(set(preferred_genres))

        # Pick a random genre from the list of preferred
genres
        random_genre = random.choice(preferred_genres)

        # Filter the music dataset by the selected random
genre
        filtered_music =
self.music_data[self.music_data["genre"].isin([random_genre])]

        if filtered_music.empty:
            return f"No music found for the genre
{random_genre}."

        # Randomly recommend a track from the filtered music
        random_music = filtered_music.sample(n=1).iloc[0]
        return f"Recommended Track:
{random_music['track_name']} by {random_music['artist_name']}
from the genre {random_music['genre']}."
    # ▼ Методы с БД ▼

    def get_user_preferences(self, user_id):
        user = User.query.get(user_id)
        if not user or not user.preferences:
            return None
        prefs = user.preferences
        return {
            "goal": user.goals,
            "music_genres": prefs.music_genres,
            "favorite_foods": prefs.favorite_foods,
            "sport_focus": prefs.sport_focus
        }

    def recommend_based_on_user(self, user_id, prompt):
        prefs = self.get_user_preferences(user_id)
        if not prefs:
            return "No preferences found for this user."

        if "preferences" in prompt.lower():
            return (
```

```
                f"Your preferences:\n"
                f"Goal: {prefs['goal']}\n"
                f"Music: {prefs['music_genres']}\n"
                f"Favorite Foods: {prefs['favorite_foods']}\n"
                f"Sport Focus: {prefs['sport_focus']}"
            )

        goal = prefs.get("goal", "maintain")
        period = 7
        match = re.search(r'(\d+)?\s*(day|week|month|year)',
prompt.lower())
        if match:
            num = int(match.group(1) or 1)
            unit = match.group(2)
            if unit.startswith("day"):
                period = num
            elif unit.startswith("week"):
                period = num * 7
            elif unit.startswith("month"):
                period = num * 30
            elif unit.startswith("year"):
                period = num * 365

        user = User.query.get(user_id)
        sport_weights =
user.preferences.get_sport_preferences()
        food_weights = user.preferences.get_food_preferences()

        workout_plan = self.generate_workout_plan(goal,
period, sport_weights)
        nutrition_plan = self.generate_nutrition_plan(period,
prefs.get("favorite_foods", []), food_weights)

        return (
            f"Workout Plan (based on your goal:
{goal}):\n{workout_plan}\n\n"
            f"Nutrition Plan (including your food
preferences):\n{nutrition_plan}"
        )


# Инициализация ассистента
assistant = SAssistant(
    gym_df,
    nutrition_df,
    gym_model=model_gym,
```

```
    nutrition_model=model_nutrition,
    gym_encoder=label_encoder_gym,
    nutrition_encoder=label_encoder_nutrition,
    scaler_nutrition=scaler_nutrition,
    gym_vectorizer=vectorizer,
    music_data=music_df
)
```

Код агента:

```
import json
import re
from langchain import hub
from langchain.agents import create_react_agent
from langchain_core.tools import tool
from langchain_openai.chat_models import ChatOpenAI
from recommendation_model import RecommendationModel
from ss import SAssistant, gym_df, nutrition_df, assistant
from models import db,Preference
RM=RecommendationModel()
SAss = assistant



OpenAI=" "


@tool
def Recommend_workout(prompt: str):
    """
    Generates a workout plan based on the user's request.
    Example: "Recommend me exercises for a week. I want to
gain muscles"
    """
    goal, period = SAss.extract_goal_and_period(prompt)
    print(period)
    return f"Workout Plan:\n{SAss.generate_workout_plan(goal,
period)}"


@tool
def Recommend_nutrition(prompt: str):
    """
    Generates a nutrition plan based on the user's request.
    Example: "Suggest me food for a month"
    """
```

```
    _, period = SAss.extract_goal_and_period(prompt)
    return f"Nutrition
Plan:\n{SAss.generate_nutrition_plan(period)}"


@tool
def Change_food_score(prompt: str):
    """
    Change or add weight to a food category based on user's
preference expression.
    If the prompt includes "less", "dislike", etc. - score
becomes negative.
    In json format!
    Expected prompt format:
        query: natural user message
        food: food item
        score: rating from 1 to 5
        user_id: user's ID
    """
    import json

    data = json.loads(prompt)
    query = data["query"].lower()
    food = data["food"]
    score = int(data["score"])
    user_id = data["user_id"]

    # Detect negative sentiment
    negative_words = ["less", "dislike", "not", "hate",
"prefer other"]
    if any(word in query for word in negative_words):
        score = -abs(score)

    print(f"[FOOD TOOL] prompt={query}, user_id={user_id},
food={food}, score={score}")

    # Fetch preferences
    p = Preference.query.filter_by(user_id=user_id).first()
    if not p:
        return "User preferences not found."

    p.update_food_weight(food, score)
    return f"Changed food score for {food} to {score} points"


@tool
```

```
def Change_sport_score(prompt: str):
    """
    Change or adds weight when user says that he likes
something new or says
    that they dislike or like more already existing sport at
rate from 1 to 5.
    You need to find "cut", "bulk", "maintaining" ways of
sport in users prompt.
    If the prompt contains the word "less" or "dislike" -
score becomes negative.
    In json format!
    Query format in the tool:
        query:{query}
        sport:{sport}
        score:{score}
        user_id:{user_id}
    """
    import json

    data = json.loads(prompt)
    query = data["query"].lower()
    sport = data["sport"]
    score = int(data["score"])
    user_id = data["user_id"]

    # Перевіряємо наявність негативного відтінку
    negative_words = ["less", "dislike", "not", "hate",
"prefer other"]
    if any(word in query for word in negative_words):
        score = -abs(score)  # зробити негативним

    print(f"[TOOL CALL] prompt={query}, user_id={user_id},
score={score}")

    p = Preference.query.filter_by(user_id=user_id).first()
    if not p:
        return "User not found."

    p.update_sport_weight(category=sport, score=score)
    return f"Changed {sport} score at {score} points"


@tool
def Recommend_training_via_user_prefs(prompt: str):
    """
    Generates a training plan based on user preferences using
```

```
their user_id.
    Use this when the user asks for personalized or
preference-based recommendations.
    Requires user_id to be present in the prompt if not passed
explicitly.
    In json format!
    Query format in the tool:
        query: {query}
        user_id: {user_id}
    """
    print(prompt)
    data = json.loads(prompt)
    query = data["query"]
    user_id = data["user_id"]
    print(f"[TOOL CALL] prompt={query}, user_id={user_id}")

    response = SAss.recommend_based_on_user(user_id=user_id,
prompt=query)
    return response

@tool
def Random_meal():
    """
    Suggests a random meal with weight in grams.
    Example: "What should I eat?"
    """
    return SAss.random_meal()


"""
@tool
def RM_call(prompt:str):

    Call when user asks for recommendation music or movie
    You only need to pass such keywords:
    music
    movie

    print("promt " + prompt)
    response = Ass.listen_and_respond(prompt)

    return response

@tool
def Random_call(prompt:str):
```

```
    Call when user asks for random recommenddation
    you can pass any movie or music
    You only can pass such keywords:
    movie
    music

    response = Ass.get_random_recommendation(prompt)
    return response
"""

@tool
def Recommend_random_music(prompt: str):
    """
    User asks for random music
    """
    response = SAss.random_music(prompt)
    print(response)
    return response

@tool
def Recommend_music(prompt: str):
    """
    User asks for music for their preferences using their id
    In json format!
    Query format in the tool:
        query: {query}
        user_id: {user_id}
    """
    data = json.loads(prompt)
    query = data["query"]
    user_id = data["user_id"]
    print(f"[TOOL CALL] prompt={query}, user_id={user_id}")
    response = SAss.recommend_music_by_genre(user_id=user_id,
prompt=query)
    return response

@tool
def Recommend_plan(prompt: str):
    """
    Generates a workout and nutrition plan based on the user's
request.
    Example requests:
    - "Recommend me exercises for today. I want to gain
muscles, also recommend food for my goals"
    - "Give me a workout plan for a month to lose weight"
    """
```

```
    response = SAss.recommend_plan(prompt)
    print(response)
    return response

agent_prompt = hub.pull("hwchase17/react")
tools=[Recommend_training_via_user_prefs,Recommend_plan,Recomm
end_workout,Recommend_nutrition,Random_meal,Change_sport_score
,

Change_food_score,Recommend_music,Recommend_random_music]
llm=ChatOpenAI(model="gpt-4o-mini",api_key=OpenAI)
ReAct_agent = create_react_agent(
    tools=tools,
    llm=llm,
    prompt=agent_prompt
)

if __name__ == "__main__":
    print("hello")
```

ДОДАТОК В

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	88 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

					Планування фізичних тренувань і дозвілля з урахуванням уподобань користувача		
						Шифр групи	Код напр./спец.
						ІТШІ-21-4	122
						ХНУРЕ кафедра ШІ	
.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Суворов І.С.					
Перевірив		Губін В.О.					
Н.контр.		Малєєва І.А.					
Затв.		Золотухін О.В.					