A picture containing text, sign

Description automatically generated

**LiftRight**

Ali salah 221000295

Alzahraa Gwyly 221000968

Jana Mokhtar 221000

Mohamed Nasr 221000089

Yomna Awad Khairy 221001896

**Under the supervision of**

Dr. Mohamed Elgazzar

Eng. Ahmed Jamal

Table of Contents

[Project Idea 3](#_Toc185348394)

[Problem Statement 4](#_Toc185348395)

[User Personas 4](#_Toc185348396)

[System Specifications 5](#_Toc185348397)

[1. Introduction 5](#_Toc185348398)

[2. System Objectives 5](#_Toc185348399)

[3. Functional Specifications 6](#_Toc185348400)

[**3.1. User Management** 6](#_Toc185348401)

[**3.2. Profile Management** 6](#_Toc185348402)

[**3.3. Workout Plan Generation** 6](#_Toc185348403)

[**3.4. Diet Recommendations** 6](#_Toc185348404)

[**3.5. Admin Features** 6](#_Toc185348405)

[4. Non-Functional Specifications 6](#_Toc185348406)

[**4.1. Performance** 6](#_Toc185348407)

[**4.2. Scalability** 6](#_Toc185348408)

[**4.3. Security** 6](#_Toc185348409)

[**4.4. Compatibility** 7](#_Toc185348410)

[**4.5. Usability** 7](#_Toc185348411)

[**4.6. Maintainability** 7](#_Toc185348412)

[5. System Architecture 7](#_Toc185348413)

[**5.1. Client-Side** 7](#_Toc185348414)

[**5.2. Server-Side** 7](#_Toc185348415)

[**5.3. Database** 7](#_Toc185348416)

[6. Use Case Examples 7](#_Toc185348417)

[**6.1. User Registration** 7](#_Toc185348418)

[**6.2. Generate Workout Plan** 8](#_Toc185348419)

[**6.3. Generate Diet Recommendation** 8](#_Toc185348420)

[1. User Authentication: 8](#_Toc185348421)

[2. User Profile Management: 8](#_Toc185348422)

[3. Workout Plan Generation: 8](#_Toc185348423)

[4. Diet Recommendations: 9](#_Toc185348424)

[Non-Functional Requirements 9](#_Toc185348425)

[1. Performance: 9](#_Toc185348426)

[2. Scalability: 9](#_Toc185348427)

[3. Security: 9](#_Toc185348428)

[4. Usability: 9](#_Toc185348429)

[5. Maintainability: 9](#_Toc185348430)

[Deployment Plan 9](#_Toc185348431)

[1. Objective 9](#_Toc185348432)

[2. Pre-Deployment Checklist 9](#_Toc185348433)

[3. Environment Details 10](#_Toc185348434)

[4. Deployment Process 10](#_Toc185348435)

[5. Rollback Plan 10](#_Toc185348436)

[6. Testing and Validation 10](#_Toc185348437)

[7. Communication Plan 11](#_Toc185348438)

[8. Timeline 11](#_Toc185348439)

[9. Resources 11](#_Toc185348440)

[10. Post-Deployment Activities 11](#_Toc185348441)

[1. Acceptance Criteria 11](#_Toc185348442)

[2. Automated Tests 12](#_Toc185348443)

[3. Test Code 12](#_Toc185348444)

[API 14](#_Toc185348445)

[1. Overview 14](#_Toc185348446)

[2. Authentication 14](#_Toc185348447)

[3. Endpoints 14](#_Toc185348448)

[**User Registration** 14](#_Toc185348449)

[**User Login** 15](#_Toc185348450)

[**User Profile** 16](#_Toc185348451)

[**Update Profile** 16](#_Toc185348452)

[**Download Workout Plan** 17](#_Toc185348453)

[**Static Pages** 17](#_Toc185348454)

[4. Error Handling 18](#_Toc185348455)

Project Idea

The Workout Planner App is a Django-based application designed to provide users with personalized workout plans and diet recommendations. It tailors plans based on user inputs such as height, weight, age, body fat percentage, activity level, and fitness goals. The app aims to simplify fitness planning by offering easily accessible and actionable advice for users with diverse fitness levels and objectives.

**Problem Statement**

Many individuals struggle with planning effective workouts and diets tailored to their specific needs and goals. Existing solutions are often overly complex, generic, or require significant expertise to use effectively. The Workout Planner App addresses this gap by providing personalized fitness and nutrition plans in a user-friendly platform, enabling users to focus on their fitness journey without unnecessary barriers.

**User Personas**

1. Ali – The Beginner

- Age: 25  
- Occupation: Student  
- Goals: Lose weight and get fit.  
- Pain Points: Doesn’t know how to start with workouts or diet planning.  
- Needs: Easy-to-follow workout plans and diet suggestions that fit a student budget.

2. Sara – The Fitness Enthusiast

- Age: 30  
- Occupation: Marketing Professional  
- Goals: Build muscle and track progress.  
- Pain Points: Wants advanced plans tailored to her needs.  
- Needs: Plans that focus on specific muscle groups and advanced workout suggestions.

3. Omar – The Busy Parent

- Age: 40  
- Occupation: Accountant  
- Goals: Stay healthy with limited time.  
- Pain Points: Doesn’t have time for complex workout routines.   
( Will be pinpointed by days/week available )  
- Needs: Short and effective workout plans and simple meal recommendations.

**System Specifications**

**1. Introduction**

The Workout Planner App is a web-based application designed to generate personalized workout and diet plans for users based on their fitness goals. The system provides a user-friendly platform for beginners, fitness enthusiasts, and busy professionals to simplify fitness planning.

**2. System Objectives**

- Provide tailored plans based on user inputs.  
- Maintain a simple and intuitive interface.  
- Ensure the app is mobile-friendly and works on all modern browsers.  
- Deliver results quickly, even with a high number of users.  
- Protect sensitive user data with robust security measures.

**3. Functional Specifications**

**3.1. User Management**

- Users can register, log in, log out, and reset passwords.  
- Secure authentication for all user accounts.

**3.2. Profile Management**

- Users can input and update personal details: age, gender, weight, height, activity level, and fitness goals.

**3.3. Workout Plan Generation**

- The app generates workout plans tailored to fitness level, user goals, and available days per week.

**3.4. Diet Recommendations**

- Calorie calculation based on BMR and user goals (maintenance, surplus, deficit).  
- Macronutrient breakdown (protein, fats, carbohydrates).

**3.5. Admin Features**

- Manage user accounts.  
- Monitor system logs for performance and security.

**4. Non-Functional Specifications**

**4.1. Performance**

- Generate workout and diet plans within 10 seconds.

**4.2. Scalability**

- Handle up to 1000 concurrent users effectively.

**4.3. Security**

- Encrypt user passwords using PBKDF2 or bcrypt.  
- Prevent unauthorized data access with secure authentication mechanisms.

**4.4. Compatibility**

- Ensure responsiveness for mobile devices (smartphones, tablets).  
- Support major modern browsers: Chrome, Firefox, Safari, Edge.

**4.5. Usability**

- Maintain an intuitive design with clear instructions.

**4.6. Maintainability**

- Use modular and well-documented code.

**5. System Architecture**

The system consists of client-side components (frontend), server-side components (backend), and a database for storing user data and plans.

**5.1. Client-Side**

- Frontend Framework: HTML, CSS, JavaScript.  
- Responsive Design: Adapt layout for mobile and desktop views.

**5.2. Server-Side**

- Backend Framework: Django (Python).  
- Authentication: Django's built-in authentication system.  
- APIs: RESTful endpoints for managing user data, workout plans, and diet recommendations.

**5.3. Database**

- Database Type: PostgreSQL or MySQL (preferred for scalability).  
- Tables:  
 - Users: Stores user information (e.g., email, hashed password).  
 - Profiles: Stores user inputs (e.g., height, weight, goals).  
 - Plans: Stores generated workout and diet plans.

**6. Use Case Examples**

**6.1. User Registration**

Actors: User, System  
Steps:  
1. User enters email and password.  
2. System validates input and creates a new account.  
3. User receives a confirmation email.

**6.2. Generate Workout Plan**

Actors: User, System  
Steps:  
1. User selects a goal (e.g., weight loss).  
2. User inputs details (e.g., weight, height, activity level).  
3. System generates a workout plan tailored to the user.

**6.3. Generate Diet Recommendation**

Actors: User, System  
Steps:  
1. User selects a goal (e.g., calorie deficit).  
2. System calculates calorie needs and macro distribution.  
3. User receives a detailed diet recommendation.

**Functional Requirements**

**1. User Authentication:**

* + Users can register, log in, log out, and reset passwords.

**2. User Profile Management:**

* + Users can input and update their personal details: age, weight, height, activity level, and fitness goals.

**3. Workout Plan Generation:**

* + Personalized workout plans are generated based on user inputs such as weight, height, and goals.

**4. Diet Recommendations:**

* + Calculate calorie needs, including maintenance, surplus, and deficit.
  + Provide macronutrient breakdown based on user goals.

Non-Functional Requirements

**1. Performance:**

* Generate workout and diet plans within 10 seconds or less.

**2. Scalability:**

* Support up to 1000 concurrent users.

**3. Security:**

* Passwords must be hashed and stored securely.
* Prevent unauthorized access to user data.

**4. Usability:**

* Maintain a simple, intuitive interface with a clean design.

**5. Maintainability:**

* Provide clear documentation for developers.

**Deployment Plan**

**1. Objective**

The purpose of this deployment plan is to outline the steps required to deploy the Workout Planner application to the production environment. This deployment includes the implementation of new features, bug fixes, and performance enhancements, ensuring minimal downtime and disruption to users.

**2. Pre-Deployment Checklist**

Ensure the following steps are completed before deployment:

Verify that the production environment is ready and configured.

Ensure that all dependencies and packages are installed.

Take a complete backup of the existing database and application.

Verify the integrity of the new release package.

Inform the stakeholders and set up a communication channel for updates.

**3. Environment Details**

The deployment will take place in the following environments:

1. \*\*Development Environment\*\*: For testing and bug fixing.

2. \*\*Staging Environment\*\*: For final verification before production.

3. \*\*Production Environment\*\*: For end-user access.

All environments use the following technology stack: Django, PostgreSQL, and a Linux-based deployment server.

**4. Deployment Process**

The deployment process involves the following steps:

Pull the latest code from the repository.

Verify the code and run unit tests.

Package the application using Docker (if applicable).

Deploy the code to the production server.

Run database migrations and restart the application services.

**5. Rollback Plan**

If deployment fails or issues arise, the following steps will be taken:

1. Stop the deployed application.

2. Restore the backup of the previous application version.

3. Notify stakeholders of the rollback and the expected resolution timeline.

**6. Testing and Validation**

After deployment, perform the following tests to validate the application:

Smoke tests to ensure basic functionality is working.

Integration tests for critical workflows.

End-user tests to confirm UI and API functionality.

**7. Communication Plan**

Stakeholders will be notified at each stage of the deployment process. Key updates will be provided through a shared communication channel (e.g., Slack or email).

**8. Timeline**

The deployment will be scheduled as follows:

1. \*\*Preparation\*\*: [Date/Time]

2. \*\*Deployment\*\*: [Date/Time]

3. \*\*Validation and Monitoring\*\*: [Date/Time]

**9. Resources**

The following resources will be required for deployment:

Deployment Team: Responsible for executing the deployment process.

Backup Tools: For database and application backups.

Monitoring Tools: To track application performance post-deployment.

**10. Post-Deployment Activities**

After deployment, the following steps will be performed:

Monitor system performance and address any issues.

Validate all key functionalities with the stakeholders.

Archive deployment logs and update the documentation.

**Automated Testing**

**1. Acceptance Criteria**

The acceptance criteria for this project define the expected functionality and requirements that must be fulfilled for the application to be considered complete. These include:

- The user should be able to sign up with a username, email, and password, and the system should validate all inputs.

- Users should be able to log in and log out of their accounts.

- Users should be redirected to their profile page after logging in, where their BMI is displayed.

- Users should be able to download workout plans based on their BMI.

- Users should be able to update their profile information such as age, weight, and height.

- The system should render static pages like Home, About, and Sign-Up without errors.

**2. Automated Tests**

The following automated tests were implemented to verify the functionality of the application:

1. `test\_render\_signup\_view\_get`: Tests that the Sign-Up page renders correctly.

2. `test\_render\_signup\_view\_post`: Tests that users can successfully sign up with valid data.

3. `test\_render\_login\_view\_get`: Tests that the Log-In page renders correctly.

4. `test\_render\_login\_view\_post`: Tests that users can log in with correct credentials.

5. `test\_render\_home\_page`: Tests that the Home page renders correctly.

6. `test\_render\_about\_page`: Tests that the About page renders correctly.

7. `test\_load\_profile\_user\_data`: Tests that user profile data is loaded and BMI is calculated correctly.

8. `test\_download\_workout\_plan`: Tests that the workout plan download feature works as expected.

9. `test\_update\_profile`: Tests that users can update their profile data.

**3. Test Code**

Below is the automated test code implemented for the project:

import pytest  
from django.urls import reverse  
from django.contrib.auth import get\_user\_model  
import json  
  
User = get\_user\_model()  
  
@pytest.mark.django\_db  
def test\_render\_signup\_view\_get(client):  
 response = client.get(reverse('SignUp'))  
 assert response.status\_code == 200  
 assert 'SignUp.html' in [t.name for t in response.templates]  
  
@pytest.mark.django\_db  
def test\_render\_signup\_view\_post(client):  
 response = client.post(reverse('SignUp'), data={  
 'username': 'testuser',  
 'email': 'testuser@example.com',  
 'password1': 'testpassword123',  
 'password2': 'testpassword123',  
 'age': 25,  
 'height': 170,  
 'weight': 70,  
 })  
 assert response.status\_code == 302  
 assert response.url == reverse('Home')  
 user = User.objects.get(username='testuser')  
 assert user.email == 'testuser@example.com'  
  
# Additional tests follow similar structure...

API

**1. Overview**

This API documentation outlines the endpoints, request/response formats, and error handling for the Workout Planner application. The API allows users to sign up, log in, manage profiles, download workout plans, and access static pages.

**2. Authentication**

The API uses session-based authentication. Users must log in to access authenticated endpoints. For simplicity, no token-based authentication is used in this implementation.

**3. Endpoints**

**User Registration**

\*\*URL:\*\* /signup/

\*\*Method:\*\* POST

\*\*Description:\*\* Creates a new user.

\*\*Request Body:\*\*

- username: string

- email: string

- password1: string

- password2: string

- age: integer

- height: integer

- weight: integer

\*\*Response:\*\*

- message: User registered successfully

- redirect\_url: /home/

\*\*Error Responses:\*\*

- 400: Invalid input data.

**User Login**

\*\*URL:\*\* /login/

\*\*Method:\*\* POST

\*\*Description:\*\* Logs a user in.

\*\*Request Body:\*\*

- username: string

- password: string

\*\*Response:\*\*

- message: Login successful

- redirect\_url: /profile/

\*\*Error Responses:\*\*

- 401: Invalid credentials.

**User Profile**

\*\*URL:\*\* /profile/

\*\*Method:\*\* GET

\*\*Description:\*\* Fetches user profile data.

\*\*Response:\*\*

- username: string

- email: string

- age: integer

- height: integer

- weight: integer

- bmi: float

\*\*Error Responses:\*\*

- 403: Unauthorized access.

**Update Profile**

\*\*URL:\*\* /update-profile/

\*\*Method:\*\* POST

\*\*Description:\*\* Updates user profile data.

\*\*Request Body:\*\*

- age: integer

- height: integer

- weight: integer

\*\*Response:\*\*

- message: Profile updated successfully

\*\*Error Responses:\*\*

- 400: Invalid input data.

**Download Workout Plan**

\*\*URL:\*\* /download/

\*\*Method:\*\* GET

\*\*Description:\*\* Provides a downloadable workout plan based on the user's BMI.

\*\*Response:\*\*

PDF File with headers: Content-Type: application/pdf, Content-Disposition: attachment; filename="Workout\_Plan.pdf"

\*\*Error Responses:\*\*

- 404: File not found.

**Static Pages**

\*\*URL:\*\* /home/ (GET): Renders the Home page., /about/ (GET): Renders the About page.

\*\*Method:\*\* GET

\*\*Description:\*\* No description provided.

\*\*Response:\*\*

HTML content with appropriate status codes (200 or 404).

**4. Error Handling**

The API follows standard HTTP status codes for indicating errors. Below are the common error codes used across the API:

\*\*General Error Codes:\*\*

- 400: Bad Request – Invalid data provided.

- 401: Unauthorized – User is not authenticated.

- 403: Forbidden – User does not have permission.

- 404: Not Found – Endpoint or resource does not exist.

- 500: Internal Server Error – Server-side issue.