Project Specifications and Requirements

Project Name: CoreX Fitness **Version:** 1.0 **Date:** December 16, 2025

1. Introduction

CoreX Fitness is a full-stack web application designed to help users manage their fitness journey. The system provides tools for tracking body statistics, accessing curated workout and diet plans, and managing user profiles securely. The application follows a decoupled architecture with a responsive frontend and a robust RESTful API backend.

2. Technology Stack

The system is built using the following technologies:

- **Frontend:**
  - **Languages:** HTML5, CSS3, JavaScript.
  - **Design:** Custom Dark Mode UI with Neon Blue (#3aa7ff) accents.
  - **Communication:** fetch API for asynchronous HTTP requests.
- **Backend:**
  - **Framework:** ASP.NET Core Web API (.NET 8).
  - **Security:** JWT (JSON Web Tokens) for authentication and BCrypt for password hashing.
  - **Data Access:** Entity Framework Core.
- **Database:**
  - **Type:** SQL Server (implied by Entity Framework usage).

3. Detailed Frontend Specifications

3.1 Design System & UI Components

The user interface is built on a cohesive design language defined in the global CSS files.

- **Color Palette:**
  - **Primary Accent:** Neon Blue (#3aa7ff) used for buttons, borders, hover effects, and active states.
  - **Backgrounds:** Deep Black (#121212) for the body and Dark Gray (#1e1e1e) for content cards and headers.
  - **Text:** Pure White (#ffffff) for primary text and Light Gray (#cfd8e3, #aaaaaa) for placeholders and secondary descriptions.
- **Typography:**
  - **Primary Font:** 'Poppins', sans-serif is used for body text to ensure readability.

    - **Heading Font:** 'Montserrat', sans-serif (italicized) is used for high-impact headings like "Register Now!".

- **Interactive Elements:**
    - **Buttons:** All actionable buttons feature a 0.3-second transition on background color. On hover, they shift to a darker blue (#1e90ff).
    - **Cards:** Program and feature cards utilize a scale(1.1) transform effect on hover, creating a "pop-out" 3D effect.

3.2 Page-Specific Functionality

3.2.1 Landing Page (Home)

- **Hero Section:** Features a large welcome message ("Keep Your Body Fit & Strong") and a hero image with a dynamic Neon Blue border that reacts to mouse hover.
- **Feature Grid:** A "Why Choose Us" section utilizes a CSS Grid layout (grid-template-columns: repeat(2, 1fr)) to display key selling points. Each feature box has a thick white border that turns blue on hover.
- **Navigation:** A persistent top navigation bar includes links to sections and a circular "Profile" button that links directly to profile page.html.

3.2.2 Authentication Interfaces

- **Login Page:**
    - **Input Fields:** Captures Username, Email, and Password.
    - **Validation:** JavaScript enforces a minimum password length of 6 characters and checks for valid email formatting (must contain "@" and ".") before sending data.
    - **Feedback:** Uses browser alerts to notify users of success ("Login Successful") or backend errors.
- **Registration Page:**
    - **Extended Data Collection:** In addition to credentials, this form collects physical stats (Weight, Height, Age) required for the BMI and diet algorithms.
    - **Validation:** Prevents submission of negative numbers for age/weight/height.

3.2.3 Programs & Systems Page

- **Dynamic Filtering:** The page uses a JavaScript-based filter (filterSelection) that allows users to toggle between "All," "Training," and "Diet" views without reloading the page.
- **DOM Manipulation:** Program cards are generated programmatically via JavaScript objects, allowing for easy updates to the program catalog (e.g., changing "Bulking" to "Advanced Bulking") without touching the HTML structure.
- **Localization Support:** The internal architecture supports switching between English (en) and other languages (e.g., ar), including automatic text direction changes (ltr to rtl).

3.2.4 User Profile Dashboard

- **Sidebar Navigation:** A sticky sidebar provides quick access to "Profile Info," "Programs Registered," and "Log Out".
- **Security-First Update Flow:**
    - The update form includes a specific field for Current Password.
    - **Step 1:** The frontend sends the Current Password and Email to the /PasswordChecker API endpoint.
    - **Step 2:** Only if the backend returns a 200 OK does the frontend proceed to send the updated profile data to /updateUserInformation.
    - **Step 3:** A final check ensures the "New Password" and "Confirm Password" fields match locally before any request is made.

3.3 Client-Side Logic & State Management

- **Asynchronous Communication:** The frontend utilizes async/await syntax with the fetch API for all server operations, ensuring the UI remains responsive during network requests.
- **Event Handling:** Forms utilize e.preventDefault() to stop standard HTML submission, allowing JavaScript to sanitize and validate data first.
- **Error Handling:** The application captures HTTP 400/500 errors from the API and displays the specific text message returned by the server (e.g., "User already exists") directly to the user via alerts.

4. Backend Functional Requirements

4.1 User Authentication

The system must allow users to create accounts and log in securely.

- **Registration:** Users must provide a Username, Email, Password, Weight, Height, and Age. Validation must ensure the email format is correct and the password meets length requirements.
- **Login:** Users authenticating with valid credentials must receive a JSON Web Token (JWT) for session management.
- **Password Hashing:** All user passwords must be hashed using BCrypt before being stored in the database. Plain text passwords must never be stored.

4.2 User Profile Management

Users must be able to view and update their personal information.

- **View Profile:** The system shall display the user's current Username, Email, Age, Weight, and Height.
- **Security Check:** Critical profile updates (Email, Password) require the user to re-enter their current password for verification via the /PasswordChecker endpoint before changes are saved.
- **Update Info:** Validated users can update their body statistics (Weight, Height) and contact info.

4.3 Fitness Programs Content

The application must provide curated fitness content.

- **Program Types:**
    - **Training:** Includes "Gym" (Advanced) and "Body Weight" (Beginner) routines.
    - **Nutrition:** Includes "Bulking" (Muscle Gain) and "Cutting" (Fat Loss) plans.

5. Non-Functional Requirements

5.1 Security

- **Encryption:** Communication between the client and server must occur over HTTPS.
- **Token Expiry:** Authentication tokens (JWT) shall expire after 14 days, requiring re-login to ensure security.
- **Input Validation:** The backend must reject requests with missing or invalid data (e.g., null values in API testing endpoints).

5.2 Performance

- **Asynchronous Operations:** All database interactions and API calls must be asynchronous (async/await) to prevent blocking the main execution thread and ensure a responsive UI.
- **Asset Optimization:** Images and styles should be loaded efficiently to minimize First Contentful Paint (FCP).

5.3 User Interface (UI/UX)

- **Theme:** The application relies on a "Dark Mode" aesthetic (#121212 background) to reduce eye strain and provide a modern look.
- **Feedback:** The system must provide immediate visual feedback (alerts) for actions such as successful login, registration errors, or password mismatches.

6. Data Specifications

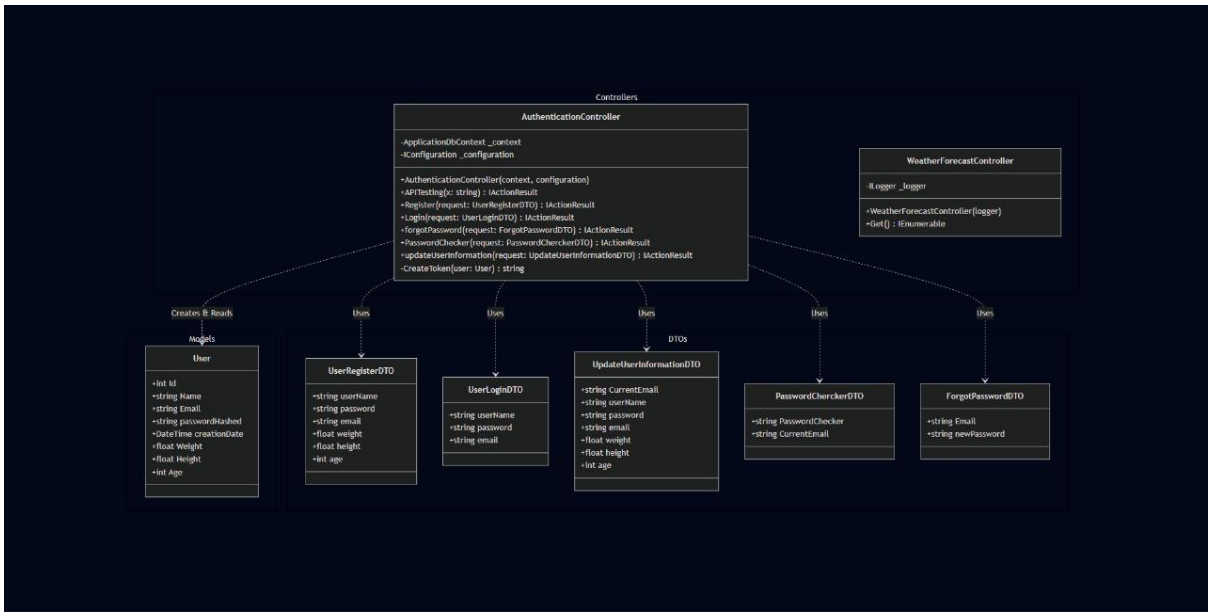
- The system manages a primary User entity with the following attributes:

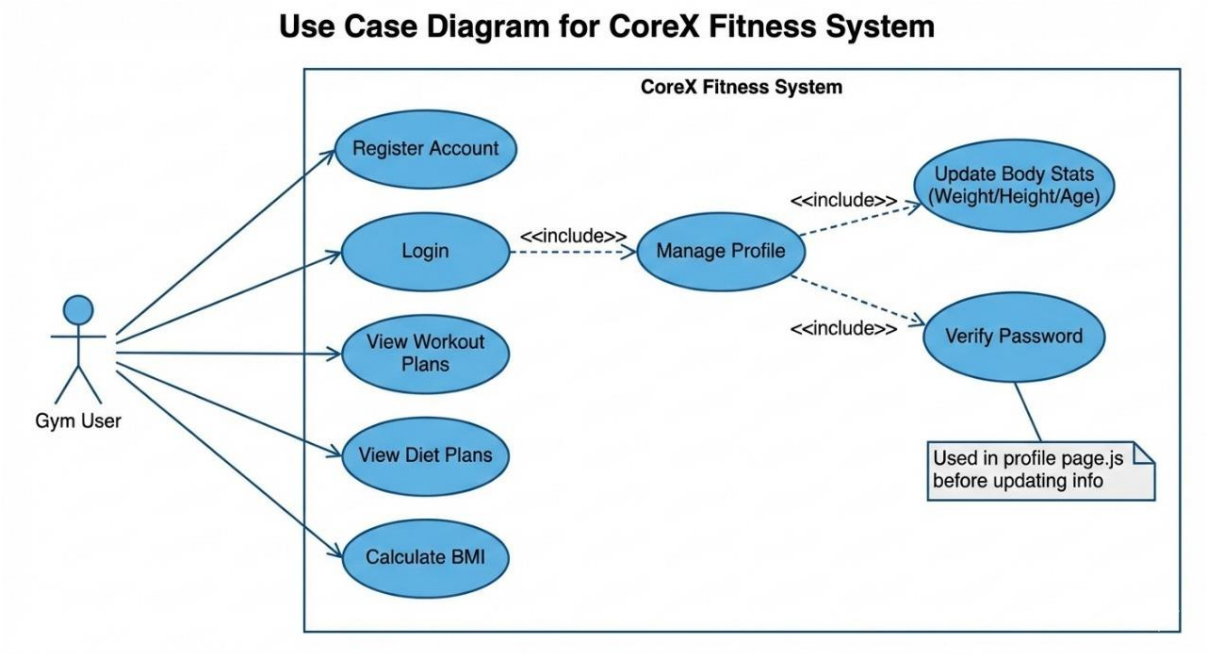
Attribute	Data Type	Description
Id	int	Unique identifier for the user (Primary Key)
Name	string	User's display name
Email	string	User's unique email address
PasswordHashed	string	The BCrypt hash of the user's password
Weight	float	Physical weight (kg/lbs)
Height	float	Physical height (cm)
Age	int	User's age in years
CreationDate	DateTime	Timestamp of account creation

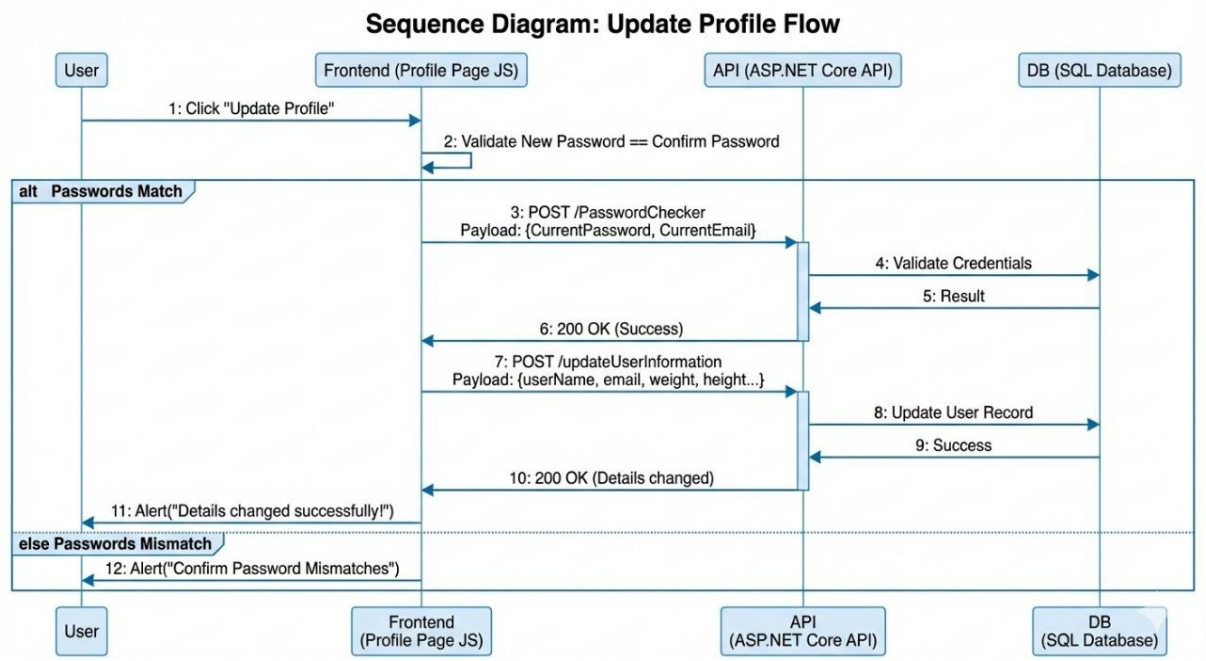
7. Assumptions and Constraints

- **Browser Support:** The application is designed for modern web browsers (Chrome, Edge, Firefox) utilizing standard ES6 JavaScript features.
- **Internet Connection:** An active internet connection is required to communicate with the Azure-hosted (or local) backend API.
- **Backend Hosting:** The system is configured to run on a .NET 8 runtime environment.

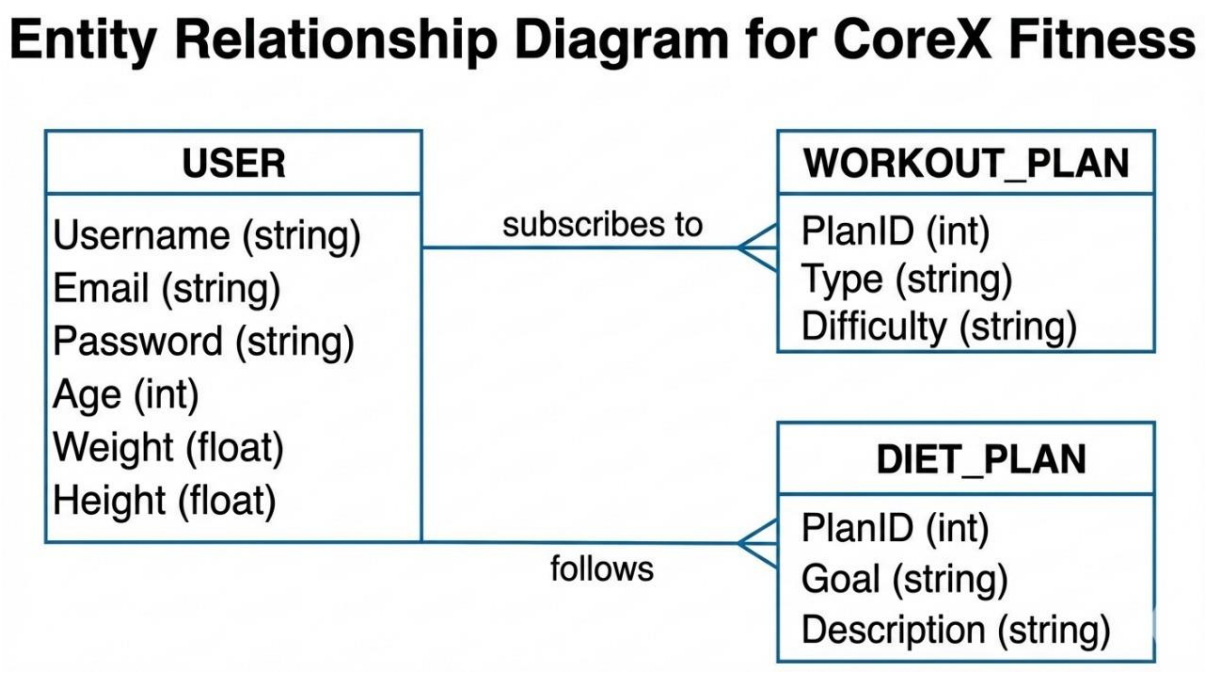
8. Behavioural & Functional Diagrams







Structural & Data Diagrams:



Test Cases:

Authentication Module Test Cases

This module verifies the security and functionality of user access, specifically implementing JWT and BCrypt standards.

Test ID	Test Case	Preconditions	Test Steps	Expected Result
REG-001	Valid Registration	No user exists with test email	1. Navigate to Registration page 2. Enter valid data (Name, Email, Password, Weight=70, Height=175, Age=25) 3. Click "Register"	User account created; redirect to login/profile with success message. (Validates required fields [8]).
REG-002	Duplicate Email Registration	User with test@example.com exists	1. Use existing email in registration form 2. Submit form	Alert "User already exists"; form not submitted.
REG-003	Invalid Email Format	None	1. Enter invalid-email in email field	Client-side validation prevents submission; checks for "@" and "."[9].

Test ID	Test Case	Preconditions	Test Steps	Expected Result
			2. Submit form	
REG-004	Password Too Short	None	1. Enter password 12345 (5 chars) 2. Submit form	Client validation error: "Password must be at least 6 characters"[10].
REG-005	Negative Body Stats	None	1. Enter Weight: -5, Height: -10, Age: -1 2. Submit form	Validation prevents submission; shows "Values must be positive"[11].

Test ID	Test Case	Preconditions	Test Steps	Expected Result
LOGIN-001	Valid Credentials	User account exists	1. Enter correct email/password 2. Click "Login"	JWT token received; redirect to dashboard[12].
LOGIN-002	Invalid Password	User account exists	1. Enter correct email with wrong password	Alert "Invalid credentials"; user remains on login page.

Test ID	Test Case	Preconditions	Test Steps	Expected Result
			2. Submit	
LOGIN-003	Non-existent User	No account with email	1. Enter non-existent email 2. Submit	Alert "User not found" or "Invalid credentials".
LOGIN-004	Empty Fields	None	1. Leave email/password empty 2. Submit	Client validation prevents submission.
LOGIN-005	JToken Storage	Successful login	1. Login successfully 2. Check browser storage	JWT token stored in localStorage/sessionStorage.

User Profile Module Test Cases

This module tests the "Security-First Update Flow" required by the specifications

Test ID	Test Case	Preconditions	Test Steps	Expected Result
PROF-001	Load Profile Data	User logged in	1. Navigate to Profile page	All user data displayed correctly (Name, Email, Stats)[14].
PROF-002	Sidebar Navigation	User logged in	1. Click "Programs	Redirects to programs page[15].

Test ID	Test Case	Preconditions	Test Steps	Expected Result
			Registered" in sidebar	
PROF-003	Logout Function	User logged in	1. Click "Log Out" in sidebar 2. Try to access profile	Session cleared; redirect to login; JWT invalidated.

Test ID	Test Case	Preconditions	Test Steps	Expected Result
UPDATE-001	Successful Update with Password Check	User logged in, knows current password	1. Click "Update Profile" 2. Enter correct current password 3. Update weight to 75 4. Submit	1. /PasswordChecker returns 200 [16] 2. /updateUserInformation updates data 3. Success alert shown.
UPDATE-002	Incorrect Current Password	User logged in	1. Enter wrong current password in update flow 2. Attempt to update	/PasswordChecker returns 401; update blocked; error shown.

Test ID	Test Case	Preconditions	Test Steps	Expected Result
UPDATE-003	New Password Mismatch	User logged in	1. Enter correct current password 2. Enter newpass123 and newpass124 in confirm 3. Submit	Client validation: "Confirm Password Mismatches" alert; no API call[17].
UPDATE-004	Email Update Verification	User logged in	1. Attempt to change email 2. Provide correct current password 3. Submit	Email updated securely after password check[18].
UPDATE-005	Invalid Body Stats Update	User logged in	1. Try to update weight to -10 2. Submit	Backend validation rejects with 400 Bad Request.

Test ID	Endpoint	Method	Request	Expected Response
API-001	/api/auth/register	POST	Valid user JSON	201 Created, user data (no password).
API-002	/api/auth/register	POST	Duplicate email	409 Conflict, "User already exists"[27].
API-003	/api/auth/login	POST	Valid credentials	200 OK, JWT token in response.
API-004	/api/auth/login	POST	Invalid credentials	401 Unauthorized.
API-005	/api/PasswordChecker	POST	Correct password/email	200 OK[28].
API-006	/api/PasswordChecker	POST	Wrong password	401 Unauthorized.

Test ID	Test Case	Preconditions	Test Steps	Expected Result
SEC-001	Password Hashing	User registration	1. Register new user 2. Check database	PasswordHashed field contains BCrypt hash, not plain text[37373737].
SEC-002	JWT Expiry	User logged in	1. Wait 14+ days	401 Unauthorized; redirect to login[38].

Test ID	Test Case	Preconditions	Test Steps	Expected Result
			2. Try to access protected endpoint	
SEC-003	HTTPS Enforcement	Deployed environment	1. Try HTTP access	Redirect to HTTPS (if configured)[39].
SEC-004	XSS Protection	User input fields	1. Enter <script>alert('xss')</script> in any form 2. Submit	Input sanitized; script not executed.
SEC-005	SQL Injection	API endpoints	1. Send SQL injection payload in email field	400 Bad Request; query parameterized in backend[40].

Organization Link:

Link

Team Members and IDs:

Yousef Mahmoud Ali – 231001086

Salah Eldin Mohamed – 231001778

Mostafa Abd Elhamied Ismael – 231000842

Mahmoud Khaled - 231000616