Ingeniería Web I

Proyecto Final: TrackFit

Jairo Hernández Noguera

1. Descripción del proyecto

Este proyecto consiste en el desarrollo de una aplicación web orientada al seguimiento de actividades deportivas. El objetivo principal es implementar una plataforma donde los usuarios puedan registrar sus actividades, definirse objetivos y obtener un historial de estas.

La aplicación ha sido concebida siguiendo buenas prácticas de desarrollo web, poniendo especial énfasis en la validación de datos, la seguridad de la información personal y la estructura modular del código, de forma que el proyecto sea escalable y fácilmente mantenible.

2. Tecnologías utilizadas

Para el desarrollo del proyecto se han utilizado las siguientes tecnologías y herramientas:

- **HTML**

Utilizado para la estructura semántica de la aplicación web y la creación de los formularios de registro.

- **CSS**

Encargado del diseño visual, la maquetación de la interfaz y la mejora de la experiencia de usuario mediante un diseño claro y responsive.

Se ha seguido una línea estética coherente en cuanto a colores y tipografía. Para el diseño responsivo básico se han utilizado:

  - Unidades relativas.
  - Flexbox para la distribución de los elementos principales.
  - Media queries sencillas para adaptar el formulario y el contenido a distintos tamaños de pantalla.

- **JavaScript**

Se ha incorporado JavaScript para implementar interactividad con el usuario y validaciones en el lado del cliente, reduciendo errores antes de enviar los datos al servidor.

Las validaciones implementadas incluyen:

- Comprobación del formato del nombre de usuario.
- Verificación de la longitud y complejidad mínima de la contraseña.
- Confirmación de que ambas contraseñas coinciden.
- Validación edad mínima del usuario.

La interactividad se gestiona mediante el uso de event listeners, principalmente sobre el evento submit del formulario y eventos de entrada en los campos más relevantes.

A través del DOM se:

- Accede dinámicamente a los valores introducidos por el usuario.
- Muestran mensajes de error o confirmación en tiempo real.
- Modifican clases CSS para resaltar campos incorrectos.
- **PHP**

Lenguaje principal del lado servidor, utilizado para el procesamiento de formularios, lógica de negocio y conexión con la base de datos. El proyecto está preparado para conectarse a una base de datos MySQL mediante PHP, utilizando una conexión centralizada que facilita la reutilización del código y su mantenimiento.

Para mejorar la estructura y escalabilidad del proyecto, se ha aplicado Programación Orientada a Objetos en PHP. La lógica de la aplicación se organiza en clases que encapsulan responsabilidades concretas, esto se ha implementado en la clase Activity y User.

- **MySQL**

Base de datos relacional destinada a almacenar la información de los usuarios. Se han implementado las siguientes tablas:

  - Users: Contiene la información de usuario:
    - id
    - Username
    - Full_name
    - Email
    - Bio
    - Avatar_path
    - Password_hash
    - Dob
    - Created_at
  - activities: Contiene las actividades de los usuarios:
    - Id
    - User_id
    - Type
    - Minutes
    - Activity_date
    - Notes
    - Created_at

- goals: Contiene los objetivos de los usuarios:
  - Id
  - User_id
  - Goal_type
  - Target_value
  - Activity_type
  - period
  - Created_at
  - Updated_at
  - Is_active
  - Start_date

- **Fetch API**

Se ha implementado comunicación asíncrona mediante Fetch API, permitiendo enviar y recibir datos sin recargar la página.

- **Servidor local (XAMPP)**

Empleado como entorno de desarrollo para ejecutar PHP y simular un entorno web real.

- **Node.js**

Se ha implementado una Micro-API como Prueba de Concepto de esta tecnología.

- **Buenas prácticas de seguridad**

Se han aplicado medidas de seguridad fundamentales:

- Prevención de inyección SQL mediante consultas preparadas.
- Protección frente a XSS mediante sanitización de datos de entrada.
- Almacenamiento seguro de contraseñas utilizando la función password_hash(), evitando

3. Funcionalidades implementadas

1) Landing

La landing actúa como punto de entrada del usuario. Presenta el valor de FitTrack y dirige a las acciones principales: registrarse o iniciar sesión.

Cómo se ha implementado

- **HTML5 semántico**: uso de `<header>`, `<nav>`, `<main>`, `<footer>` para una estructura clara, accesible y fácil de mantener.
- **Diseño responsive**: se adapta a escritorio y móvil mediante media queries.
- **Menú móvil**: en pantallas pequeñas, se reemplaza la navegación horizontal por un botón "burger" que despliega/oculta el menú.

Tecnologías

- HTML5 + CSS (responsive)
- JavaScript para interacción del menú

2) Registro de usuarios con validaciones y mayoría de edad

Permite crear una cuenta con los siguientes datos:

- Nombre de usuario
- Contraseña + confirmación
- Fecha de nacimiento (para validar mayoría de edad)

Cómo se ha implementado

- **Validación en cliente (JS)**:
  - Username con formato permitido (longitud y caracteres).
  - Contraseña segura (mínimo, mayúsculas, minúsculas, dígitos).
  - Confirmación de contraseña.
  - Mayoría de edad (+18) calculada con la fecha de nacimiento.
  - Mensajes de error claros junto al campo correspondiente.
- **Procesamiento seguro en servidor (PHP)**:
  - Comprobación de método POST.
  - Validación/sanitización en backend.
  - Inserción en base de datos con prepared statements (PDO), evitando inyección SQL.
  - Contraseña almacenada con **`password_hash()`**.

- **Tecnologías**:
  - JavaScript
  - PHP
  - MySQL

3) Login y sesión

Permite autenticarse y acceder a páginas protegidas (dashboard, perfil). Si no hay sesión activa, el usuario no puede entrar.

Cómo se ha implementado

- **Login (PHP)**:
  - Obtención de usuario por username.
  - Verificación de contraseña con `password_verify()`.
  - Creación de sesión (`$_SESSION`) con datos mínimos necesarios: `user_id`, `username` y `avatar_path`.
- **Protección de rutas**:
  - `auth.php` incluye un helper, `require_login()` que bloquea el acceso al dashboard/perfil si no hay sesión válida.
- **Logout**:
  - Endpoint que destruye sesión y redirige a la landing/login.

4) Dashboard: registro de actividades y métricas

Desde el dashboard, el usuario puede:

- Añadir una actividad (tipo, duración, fecha, notas)
- Consultar su historial
- Ver métricas agregadas (actividades, minutos, última actividad) según el rango mostrado

Cómo se ha implementado

- **Formulario con validaciones en JS** (minutos válidos, fecha obligatoria, tipo obligatorio).
- **Guardado asíncrono con Fetch API (AJAX)**:
  - El formulario no recarga la página.
  - Se envía JSON por POST a `php/activities_add.php`.
  - El servidor devuelve respuesta JSON con estado OK/ERROR.
- **Recarga del historial**:
  - Tras guardar o borrar, se vuelve a pedir el listado con `activities_list.php` y se re-renderiza la UI.
- **Arquitectura POO en PHP**:
  - La clase `Activity` encapsula la lógica (validación, alta, listados, borrado).
  - Los endpoints (add/list/delete) son simples controladores que delegan en la clase.

- **Tecnologías**:
  - JavaScript (DOM + Fetch API)
  - PHP (endpoints)
  - MySQL (tabla `activities`)
  - Programación orientada a objetos (clase `Activity`)
  - JSON como formato de intercambio

5) Historial con filtro temporal

El historial de actividades muestra por defecto las actividades del día actual y permite cambiar el rango (Hoy, últimos 3 días, última semana, intervalo personalizado):

Cómo se ha implementado

- **UI**: selector de rango y, si es "personalizado", inputs de fecha.
- **Frontend**: calcula `from/to` en formato `YYYY-MM-DD` y los envía como query params.
- **Backend con POO**: `Activity::list()` soporta `from/to` y filtrar por `activity_date`,

Tecnologías

- JavaScript (cálculo de fechas + refresco)
- PHP + PDO
- MySQL (filtrado por `activity_date`)

6) Perfil de usuario: datos, avatar y cambio de contraseña

Se ha implementado una sección de "Mi perfil" donde el usuario puede modificar sus datos. Para ello se muestra los datos del usuario como una "tarjeta" con un menú que permite:

- Editar datos del perfil
- Subir/cambiar foto de perfil
- Permite el cambio de contraseña

Cómo se ha implementado

- **Vista tipo tarjeta**: muestra datos principales y un botón "Modificar". Solo al entrar en modo edición aparecen controles de guardado, lo que reduce errores y hace la UI más clara.
- **Subida de Foto de perfil**:
  - Formulario con `multipart/form-data`.
  - Validación básica de archivo (tipo/tamaño).
  - Guardado físico en `uploads/avatars/`.
  - Persistencia de la ruta en BD (`avatar_path`) y actualización de sesión para reflejarlo en la UI.
- **Cambio de contraseña**:
  - verificación de reglas mínimas.
  - actualización con `password_hash()`.

Tecnologías

- PHP (procesamiento)
- MySQL (users)
- Manejo de ficheros en servidor (`uploads/avatars`)
- Validaciones en JS + servidor
- Seguridad de contraseñas (`password_hash()`)

7) Objetivos: creación, validez, progreso y “objetivo cumplido”

Se ha implementado una funcionalidad que permite al usuario crear objetivos de actividad para fomentar el deporte.

Qué hace

- El usuario define un objetivo (por ejemplo: “300 minutos” o “10 actividades”)
- Opcionalmente filtra por tipo de actividad
- Se define un periodo (semana/mes/sin límite)
- El dashboard muestra progreso y rango de validez
- Al cumplirse: aparece indicador “ ✔️ Objetivo cumplido 👏 ”

Cómo se ha implementado

- **Modelo de datos (`goals`)**:
  - Se guarda el objetivo activo por usuario.
- **Progreso dinámico**:
  - `goal_get.php` calcula el progreso en tiempo real consultando `activities`.
  - La lógica está pensada para que el progreso cuente solo actividades posteriores a la creación del objetivo (y/o según el criterio de fecha de actividad).
- **Validez del objetivo**:
  - Se muestra “desde/hasta” en función del periodo.
- **Feedback visual al cumplirlo**:
  - El frontend compara `progress >= target_value` y muestra el badge de éxito mediante CSS y JS.

Tecnologías

- PHP + MySQL + PDO
- JSON endpoints (goal_get / goal_save)
- JavaScript (render dinámico en dashboard)
- CSS (badge de estado)

8) AJAX / Fetch API: comunicación asíncrona

Para mejorar la experiencia de usuario y cumplir con el alcance del proyecto se ha implementado Fetch API para poder realizar modificaciones en el DOM sin necesidad de recargar la página en acciones como:

- Añadir actividades
- Borrar actividades
- Cargar historial
- Cargar objetivo
- Actualizar secciones del perfil

Cómo se ha implementado

- Llamadas `fetch()` con `Content-Type: application/json`
- Endpoints PHP devolviendo `{"ok": true/false, ...}`
- Manejo de errores:
    - HTTP status + JSON de error
    - Mensajes amigables en interfaz

Tecnologías

- Fetch API
- JSON
- PHP endpoints

9) Parte opcional: Micro-API en Node.js

Como prueba de concepto se ha implementado un servidor Node.js básico como demostración de backend, que expone un endpoint de estadísticas de los usuarios en JSON.

Cómo se ha implementado

- Node.js + Express
- `mysql2` para conexión a MySQL
- Endpoint simple `GET /stats/today/:userId` que devuelve datos agregados

Tecnologías

- Node.js
- Express

4. Entrega del proyecto

El código esta disponible en para su evaluación en el siguiente repositorio de GitHub.

Las instrucciones para montar el proyecto se encuentran en el archivo readme.md