1. Application Outline

The Workout Tracker application is a lightweight health and fitness logging system that allows users to record, view, and search their personal workouts.

The system provides essential CRUD-style functionality by enabling users to log workout sessions with details such as the date, activity type, duration, intensity, and optional notes.

Users can register for an account, log in, and access personalised features such as viewing their own workout history and adding new entries.

The application also includes a global search function that allows users to discover activities logged by other users, making the system both personal and community-focused.

The application is built with simplicity and usability in mind, featuring a clear navigation bar, secure session-based authentication, and a clean interface.

Dynamic content is rendered using EJS templates, generating individual pages based on the user's state (logged in or not).

All workout data and user accounts are stored securely in a MySQL database. The app aims to provide a straightforward digital fitness log that can be extended further with additional analytics or multimedia features.

2. High-Level Architecture

Architecture Description

The application follows a classic two-tier architecture consisting of an application tier and a data tier.

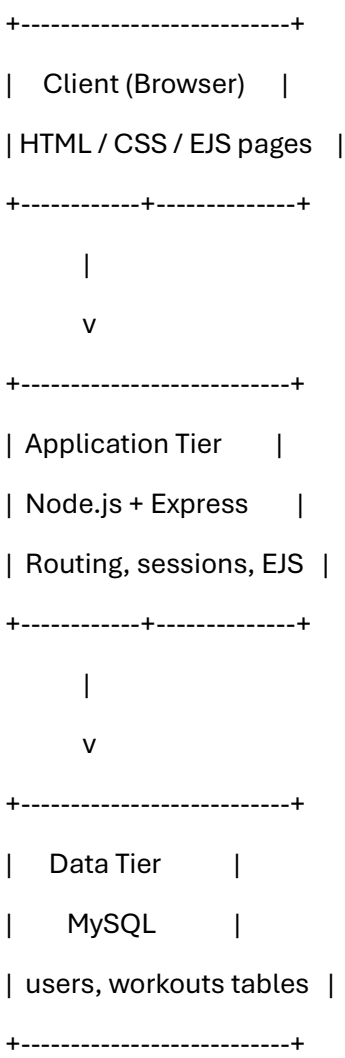
The application tier uses Node.js, Express.js, and EJS templates to handle routing, authentication, form processing, and rendering views.

User sessions are stored server-side using express-session.

The data tier is implemented with MySQL, which stores user accounts and workout records. Communication between tiers uses the mysql2 library, allowing the app to perform parameterised queries securely.

All dynamic pages are generated server-side through EJS, and static assets such as CSS are served through the Express public directory.

Architecture Diagram



3. Data Model

Data Model Description

The database contains two main tables: users and workouts.

The users table stores login credentials, including a unique ID, username, and password.

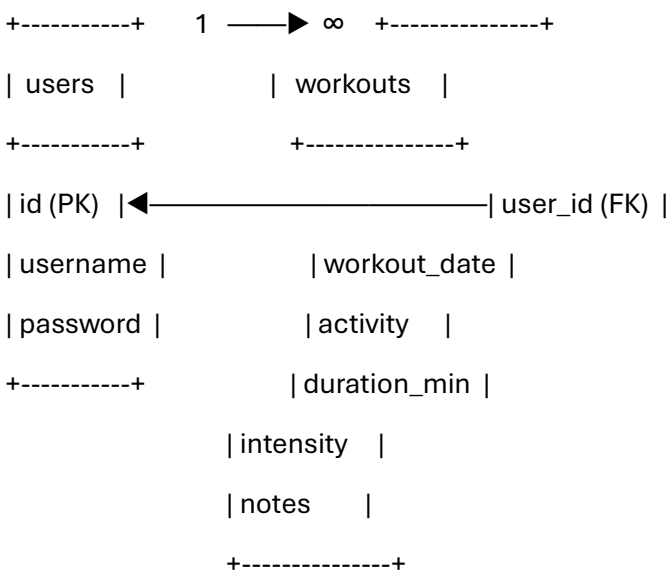
The workouts table stores workout entries, each linked to a specific user via a foreign key.

Each workout includes a date, activity type, duration, intensity level, and optional notes.

The relationship between users and workouts is one-to-many, meaning each user may have multiple workouts logged.

This structure supports personalised workout tracking and enables global search across all users.

Data Model Diagram



4. User-Facing Functionality

The application provides a simple, intuitive interface for interacting with workout data.

Upon loading the homepage, users are greeted with a clean welcome section that introduces the platform and offers clear call-to-action buttons.

Logged-out users see options to register or log in, while logged-in users see shortcuts to add a new workout or view their existing logs.

Registration & Login

Users can create an account via the registration page, filling out a simple form.

Login uses session-based authentication; once logged in, the navigation bar updates dynamically to display the user's name and workout-related options.

After logging in, users are taken to a dedicated “logged in” page confirming their session.

Workout Management

The “My Workouts” page displays all workouts belonging to the logged-in user.

Each entry includes the date, activity name, duration, intensity, and notes.

The “Add Workout” page provides a form for submitting new activities.

Required fields are validated server-side to maintain clean data.

Search Functionality

Users can search globally across all workouts in the system.

The search page allows queries by activity or username.

Results are displayed in a table, enabling users to explore workouts added by others.

Navigation Behaviour

The navigation bar updates based on authentication status using currentUser provided by middleware.

Logged-out users see Login/Register, while logged-in users gain access to personalised pages like “My Workouts” and “Add Workout”.

Screenshots

(You may capture and insert screenshots here: homepage, login form, registration, workouts list, add form, search.)

5. Advanced Techniques

1. Dynamic Navigation Rendering Using Middleware

The application uses a small middleware function to pass the logged-in user to all templates:

```
app.use((req, res, next) => {
  res.locals.currentUser = req.session.user || null;
  next();
});
```

This allows EJS templates (e.g., header.ejs) to dynamically alter navigation links depending on authentication state without repeating code in each route.

2. Parameterised Queries for Security

All database interactions use parameterised queries to prevent SQL injection:

```
const [rows] = await db.query(
  "SELECT id FROM users WHERE username = ? AND password = ?",
  [username, password]
);
```

3. Server-side Rendering with EJS

Each page is generated server-side using EJS, which helps maintain consistent structure and reduces client-side complexity.

```
<% if (currentUser) { %>
  <span>Logged in as <%= currentUser.username %></span>
<% } %>
```

4. Modular Architecture

Database logic is separated into a db.js module, ensuring maintainability and separation of concerns.

6. AI Declaration

AI tools, including ChatGPT, were used to assist in generating explanations, clarifying code behaviour, and helping structure parts of the assignment.

All coding and final design decisions were performed by myself, and AI assistance was used only for support and improvement, not full automation.