final documentation

Outline

This project is a health and fitness web application designed to help users record, manage, and review their workout activities in a simple and structured way. The aim of the application is to bring together all the techniques learned throughout the module into a single, fully functioning system that demonstrates database interaction, form handling, server-side logic, user sessions, and dynamic content rendering.

The application allows users to register, log in, and securely manage their workout records. Each workout entry includes the workout date, title, and detailed exercise information such as sets, repetitions, weight, duration, and notes. The system stores data across three related tables—workout, exercises, and workout_entries—allowing users to track the exact structure of each session. Search functionality enables users to filter workouts by date range or keywords, helping them review progress efficiently.

In addition, the application integrates real-time weather information from an external API. This feature displays the current weather on the homepage and provides a suggestion to exercise indoors or outdoors depending on conditions, making the application more practical and responsive to daily user needs.

Architecture

The application follows a three-tier architecture.
The presentation tier is the web browser, which renders HTML pages generated from EJS templates and sends form data via HTTP. The application tier is a Node.js and Express server running on the Goldsmiths virtual machine. It handles routing, input validation, session management, weather API requests, and communication with the database.
The data tier is a MySQL database called `health`, which stores users, workouts, exercises, and workout_entries in related tables. The application tier uses parameterised queries to read and write data securely.

Datamodel

The data model consists of three main entities.
The `workout` table stores each training session, including its title and date.
The `exercises` table stores all possible exercise types with an optional muscle group label.
The `workout_entries` table links workouts to specific exercises and records detailed information such as sets, reps, weight, duration, and notes.
Each workout can contain multiple entries, and each entry references exactly one exercise, forming a one-to-many relationship between workout and workout_entries.
The `users` table stores account credentials but is kept separate from workout data.

User Functionality

The application provides a set of user-facing features that allow individuals to manage their workout records in an intuitive and structured way. The home page introduces the system and provides navigation links to registration, login, workout management, and the About page. It also displays real-time weather information retrieved through an external API and offers a suggestion to exercise indoors or outdoors depending on the current conditions.

Users can register for an account through a form that collects basic details and stores a securely hashed password in the database. After logging in, session management ensures that only authenticated users can access the workout-related features. If a user attempts to view restricted pages without being logged in, the system redirects them to the login page.

Once logged in, users can access the main workout dashboard. This page displays all previously recorded workouts along with the date, title, and the exercises included in each session. Exercise names are aggregated and shown through a database JOIN query, allowing users to quickly see the structure of each workout without viewing the full details.

Users may add a new workout through a structured form that collects the workout date, title, exercise name, muscle group, and detailed exercise metrics such as sets, repetitions, weight, duration, and notes. When the form is submitted, the application inserts a new workout into the database, verifies whether the exercise already exists, and finally creates a corresponding workout entry that links the two. This enables users to build a complete log of their training sessions.

Each workout can also be viewed individually through the “View” button on the dashboard. The workout details page shows the basic workout information and a full table of entries belonging to that session, including all exercise parameters. This structured layout allows users to review their training performance in detail.

The application also provides a search feature that allows users to filter past workouts by date range and optional keywords. The results are displayed in a list format, enabling users to locate specific sessions or track training progress over time.

Finally, users can log out at any time, which clears their session and returns them to the public-facing pages. Throughout the system, appropriate validation and sanitisation ensure that user input is processed safely and consistently.

Advanced Techniques

In addition to the core functionality taught in the module, the application incorporates several advanced techniques to improve efficiency, maintainability, and data accuracy. These techniques involve multi-table SQL operations, dynamic filtering patterns, and extended validation logic beyond what was demonstrated in class.

One advanced feature is the use of SQL JOIN operations to combine information across multiple related tables. Since each workout consists of several exercise entries, the application uses a LEFT JOIN to merge `workout`, `workout_entries`, and `exercises` into a single query. This allows the system to display both workout details and the names of the exercises associated with each session without requiring multiple separate queries. The following example illustrates this approach:

```
SELECT
  w.*,
  GROUP_CONCAT(DISTINCT e.name SEPARATOR ', ') AS exercise_names
FROM workout w
LEFT JOIN workout_entries we ON w.id = we.workout_id
LEFT JOIN exercises e ON we.exercise_id = e.id
GROUP BY w.id;
```

The use of `GROUP_CONCAT` is another advanced SQL technique. It aggregates multiple exercise names into a single formatted string, enabling the application to present a clean summary of each workout on the dashboard.

The search functionality also demonstrates an advanced SQL pattern using the dynamic condition `WHERE 1=1`. This structure simplifies the process of conditionally extending the query based on user input. Instead of constructing multiple separate query branches, additional filters can be appended safely and consistently:

```
let sql = "SELECT * FROM workout WHERE 1=1";
if (from) sql += " AND workout_date >= ?";
if (to) sql += " AND workout_date <= ?";
if (keyword) sql += " AND title LIKE ?";
```

This pattern is flexible and scalable, allowing the application to support multi-parameter searching without introducing unnecessary complexity.

Input validation uses features of `express-validator` that go beyond the examples covered in class. Specifically, the `{ checkFalsy: true }` option allows optional fields such as muscle group, duration, or notes to be validated only when they contain actual input. This prevents validation errors from appearing when users intentionally leave optional fields blank:

```
check('muscle_group')
  .optional({ checkFalsy: true })
  .isLength({ max: 50 });
```

Furthermore, the application implements conditional creation of exercise records. When a user adds a workout with a new exercise name, the system first checks whether the exercise already exists and creates it only when necessary. This preserves data integrity and avoids unnecessary duplication:

```
db.query("SELECT id FROM exercises WHERE name = ?", [exerciseName], ...);
```

AI declaration

I used AI assistance during the development of this assignment for two purposes.
First, I consulted AI tools to help interpret and resolve error messages that I was unable to understand on my own.
Second, I used AI to translate and polish parts of my written documentation to ensure clarity and accuracy.
All core design decisions, code implementation, and testing were carried out by me, and AI was used only as a supportive tool rather than a source of generated solutions.