Outline

FitTrack is a dynamic web application designed to help users monitor their health and fitness journey. The application allows users to register for an account, securely log in, and record their daily workout activities, including the type of exercise, duration, calories burned, and personal notes.

The core goal of FitTrack is to simplify data persistence for fitness tracking. Users can view a history of their past workouts, search for specific activities (e.g., "Yoga" or "Running"), and manage their fitness data through a secure interface. The application also integrates with an external Weather API, allowing users to check local weather conditions before planning outdoor activities.

Key features include secure user authentication (with hashed passwords), session management, input sanitisation to prevent security vulnerabilities, and an administrative audit log that tracks login attempts for security monitoring. The application is built using a robust Model-View-Controller (MVC) approach, ensuring a clean separation between the user interface, business logic, and database storage.

Architecture

The application follows a standard **Three-Tier Architecture** pattern:

1. **Presentation Tier (Client):** The user interface is rendered using **EJS (Embedded JavaScript)** templates, HTML5, and CSS. It runs in the user's browser and communicates with the server via HTTP requests.
2. **Application Tier (Server):** Built with **Node.js** and **Express**. This layer handles routing, business logic, session management, input validation, and communication with external APIs (OpenWeather).
3. **Data Tier (Database):** Uses **MySQL** to store persistent data. It manages relational data for users, workouts, and security logs.

Description: The client sends requests to the Node.js server. The server processes these requests, interacts with the MySQL database via the mysql2 driver, or fetches data from the external Weather API, and returns rendered HTML views to the client.

Data Model

The database consists of three relational tables designed to maintain data integrity and user privacy.

1. **users**: Stores user credentials and profile information. It uses a PRIMARY KEY on the ID and a UNIQUE constraint on the username. Passwords are stored as bcrypt hashes, not plain text.

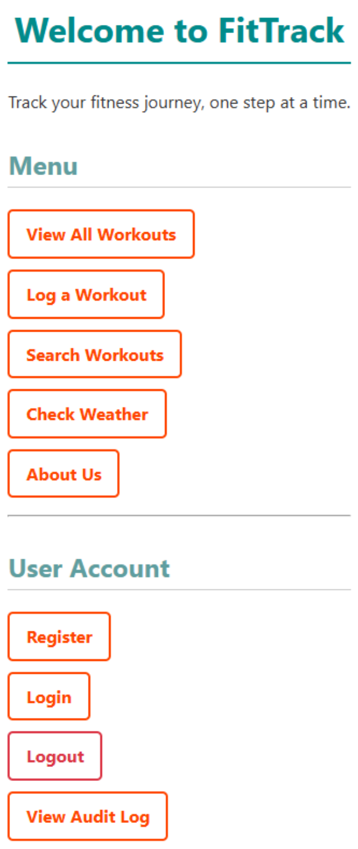
2. **workouts**: Stores workout logs. It includes a FOREIGN KEY (user_id) that links each workout to a specific user in the users table, creating a **One-to-Many** relationship (one user can have many workouts).
3. **audit_log**: A standalone table that records security events, such as successful or failed login attempts, along with timestamps.

Description: The users table is the central entity. The workouts table relies on the users table for referential integrity. The audit_log operates independently to track system-wide access events.

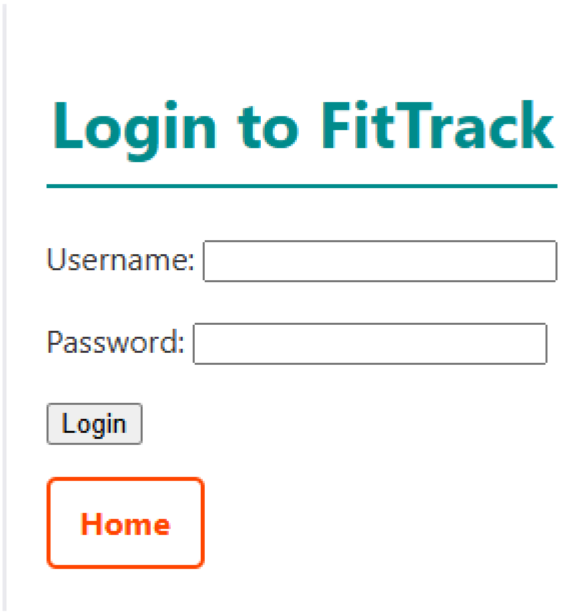
User Functionality

The application provides a seamless experience for tracking fitness data. Below is the breakdown of the user-facing features:

1. **Home & Navigation** The landing page provides a clear menu for navigation. Users can access public pages (Home, About, Weather) immediately but must log in to access private data.



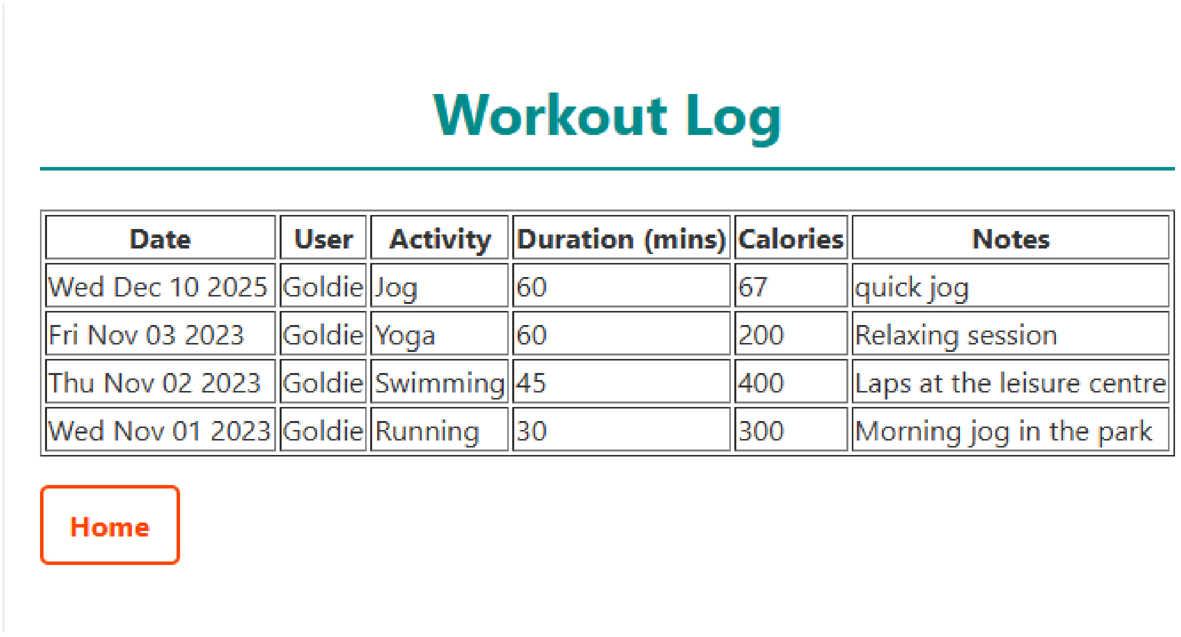
2. **User Authentication (Register/Login)** New users can create an account via the registration page. The application enforces validation (e.g. email format) and hashes the passwords for security. Existing users log in via the Login page, if a login fails, it is recorded in the audit log.



Login to FitTrack

Username:	
Password:	

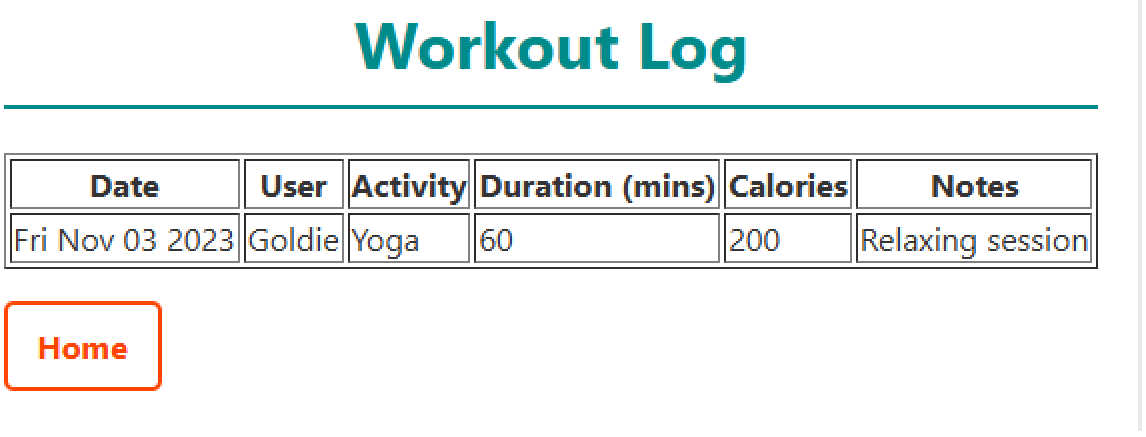
3. **Workout Management** Once logged in, users can access the "Log a Workout" page. This form captures the date, activity type, duration, and calories. Upon submission, the data is saved to the database. Users can then view their history on the "View All Workouts" page, which displays a table of all recorded exercises.



Workout Log

Date	User	Activity	Duration (mins)	Calories	Notes
Wed Dec 10 2025	Goldie	Jog	60	67	quick jog
Fri Nov 03 2023	Goldie	Yoga	60	200	Relaxing session
Thu Nov 02 2023	Goldie	Swimming	45	400	Laps at the leisure centre
Wed Nov 01 2023	Goldie	Running	30	300	Morning jog in the park

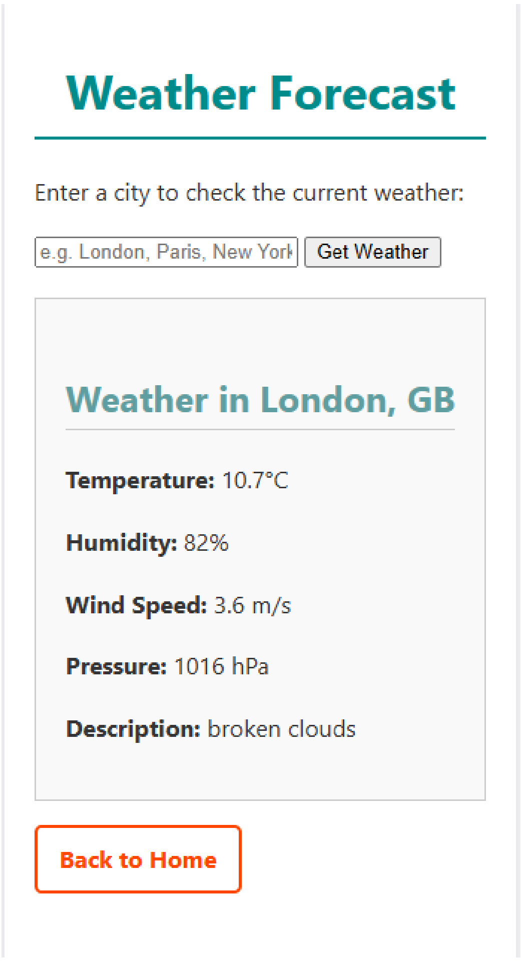
4. **Search Functionality** The Search page allows users to filter their workout history. By entering a keyword (e.g., "Run"), the application queries the database and returns only the matching records.



Workout Log

Date	User	Activity	Duration (mins)	Calories	Notes
Fri Nov 03 2023	Goldie	Yoga	60	200	Relaxing session

5. **Weather Integration** Users can visit the weather page and enter a city name. Then the app contacts the OpenWeather API and displays the current temperature, humidity, and conditions, helping users decide if they should exercise outdoors.



Weather Forecast

Enter a city to check the current weather:

Weather in London, GB

Temperature: 10.7°C

Humidity: 82%

Wind Speed: 3.6 m/s

Pressure: 1016 hPa

Description: broken clouds

Advanced Techniques

I have implemented several advanced development techniques to enhance the security, reliability, and functionality of the application.

1. Machine-Readable API Endpoint I created a JSON API endpoint to allow external applications to consume workout data. This demonstrates knowledge of RESTful standards beyond simple HTML rendering. *File: routes/api.js*

```
router.get('/workouts', function (req, res, next) {
     let sqlquery = "SELECT id, date, activity, duration,
     calories FROM workouts"
     db.query(sqlquery, (err, result) => {
          if (err) return next(err);
          res.json(result) // Returns data as JSON instead of
     HTML }) })
```

2. Security Audit Logging To improve security monitoring, I implemented an audit logging system. Every time a user attempts to log in, the attempt (whether successful or failed) is recorded in the audit_log database table with a timestamp. This allows administrators to detect potential brute-force attacks. *File: routes/users.js*

```
// Log the failure let auditQuery = "INSERT INTO audit_log
(username, action) VALUES (?, ?)";
db.query(auditQuery, [req.body.username, "Failed - User not
found"]);
```

3. Input Sanitisation & Security I used express-sanitizer to scrub user input before it interacts with the database. This prevents Cross-Site Scripting (XSS) attacks by removing dangerous HTML tags. Additionally, I used bcrypt to hash passwords, ensuring that even if the database is compromised, user passwords remain secure. *File: routes/workouts.js*

```
let newrecord = [
req.sanitize(req.body.activity), // Sanitised to prevent XSS
req.body.duration, req.body.calories,
req.sanitize(req.body.notes)
];
```

4. External API Integration I demonstrated the ability to consume third-party services by integrating the OpenWeather API using the request module. The application handles the asynchronous HTTP request, parses the JSON response, and handles errors (e.g., if an invalid city is entered). *File: routes/main.js*

```
request(url, function (err, response, body) {
     let weather = JSON.parse(body);
     if (weather.main !== undefined) {
     res.render('weather.ejs', { weather: weather, error: null
     });
     }
});
```

AI declaration

AI was not used the during this coursework