**Outline**

FitTrack is a modern web application that aims to assist users in tracking their health and fitness progress. Users can sign up for an account, securely log in, and subsequently, input their workout information on a daily basis, including exercises accomplished, durations, calories burned, and notes.

The main purpose of FitTrack is to facilitate data persistence to record fitness-related information. Users can browse their past workouts over time or search by a specific workout type (i.e., "Yoga," "Running") and maintain exercise records through a secure web interface. Additionally, the application connects to an external Weather API that lets users check the current conditions in their area before they make a decision about outdoor exercise.

Other features include secure login (passwords are hashed), session tracking, input sanitisation to prevent noSQL injections or SQL errors, and an admin audit log of security logins attempted and completed throughout time. The application is created using a standard Model-View-Controller (MVC) framework so that presentation, business logic and database storage layers are appropriately separated.

**Architecture**

The application follows a standard Three-Tier Architecture Pattern.

1. Presentation Layer (Client): The client renders HTML using EJS (Embedded JavaScript) templates, HTML5, and CSS. The client operates on the end user's side, requesting information from the server via HTTP.
2. Application Layer (Server): Built from Node.js and Express, this layer routes requests between the client and the mysql2 database in addition to business logic and OpenWeather.
3. Data Layer (Database): The persistent data layer uses MySQL for relational information to be stored across users, workouts, and audit logs.

**Description**: The client requests via the Node.js server. The server responds with either information to render via the mysql2 database or OpenWeather.

**Data Model**

The database contains three relational tables to keep data integrity and user privacy intact.

1. users: users table keeps user credentials. It has a PRIMARY KEY on ID and UNIQUE constraints on username only. Passwords are stored in bcrypt hashes, not text.

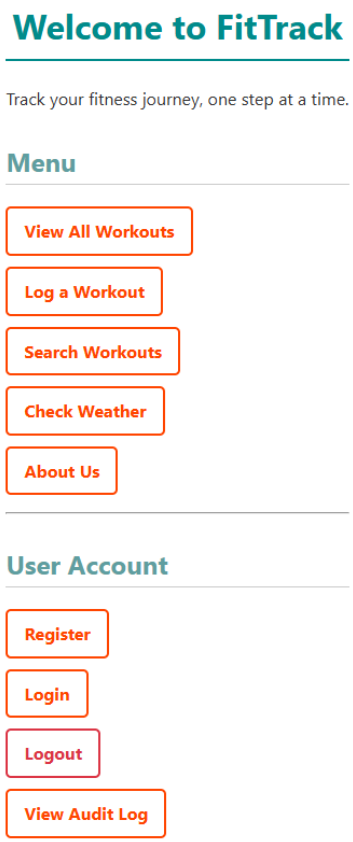
2. workouts: workouts table keeps the workouts logged for each user. It has a FOREIGN KEY (user_id) that links it back to users; this is part of a One-to-Many (one user has many workouts) relationship.
3. audit_log: audit log table is unique for security purposes to record specific security events that occur with timestamps.

**Description**: Workouts table uses users table for referential integrity while the audit_log does not depend on any other table to track system-level access events.

**User Functionality**

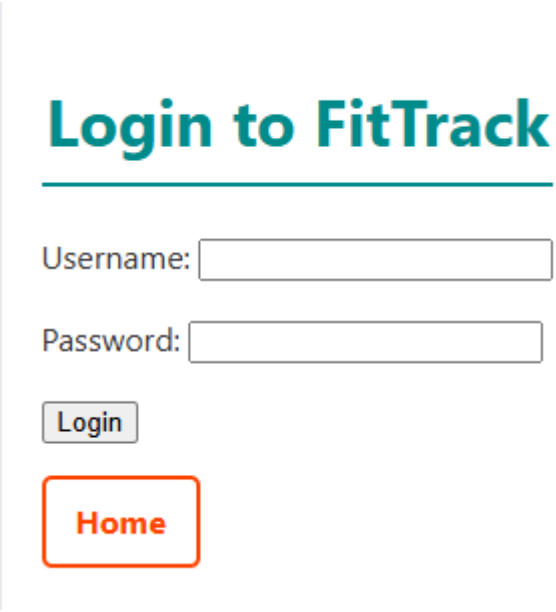
The application provides a seamless experience for tracking fitness data. Below is the breakdown of the user-facing features:

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 the private data.

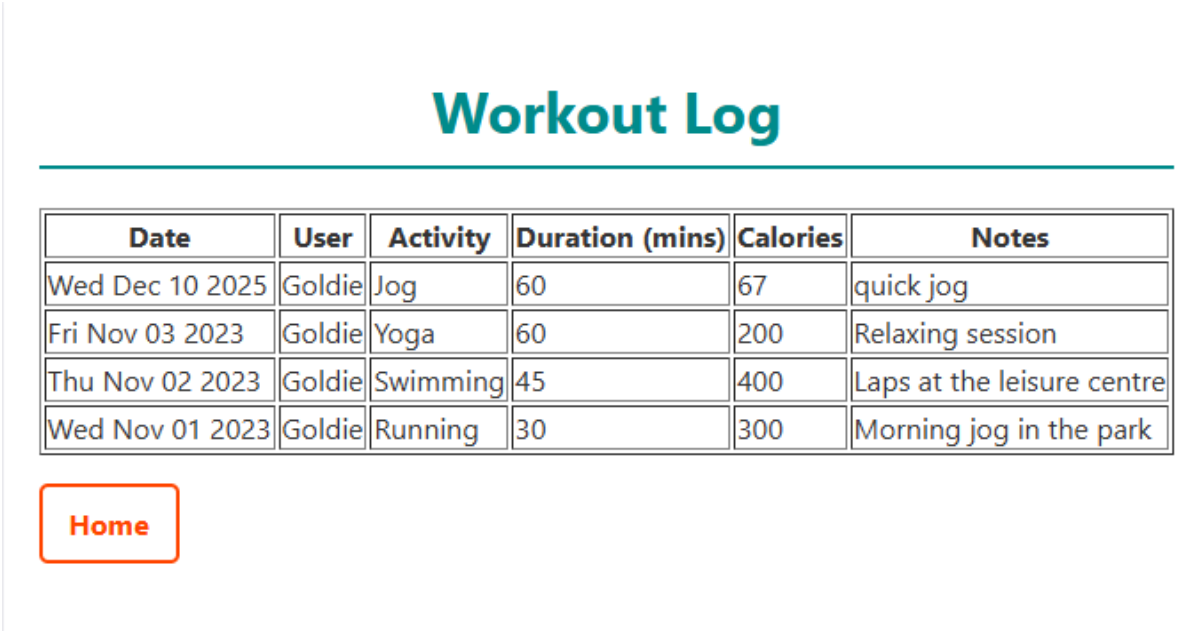



2. **User Authentication (Register/Login)** New users can create an account from the registration page, then the application enforces validation (e.g. email format)

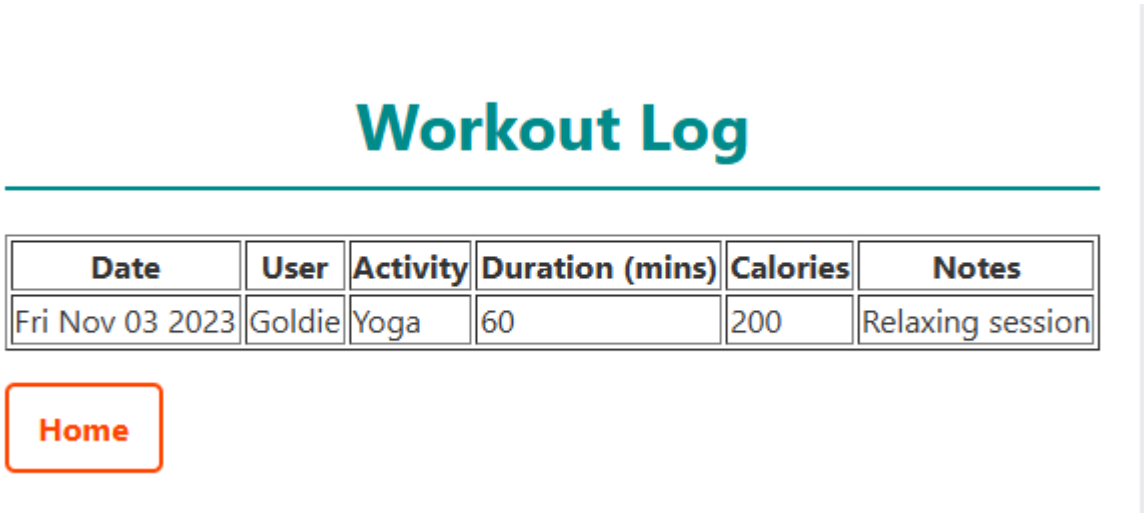
and hashes the passwords for security. Existing users log in via the login page, if a login fails, it's recorded in the audit log.




3. **Workout Management,** Once logged in, users can access the "Log a Workout" page, this is a form that 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 their recorded exercises.



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

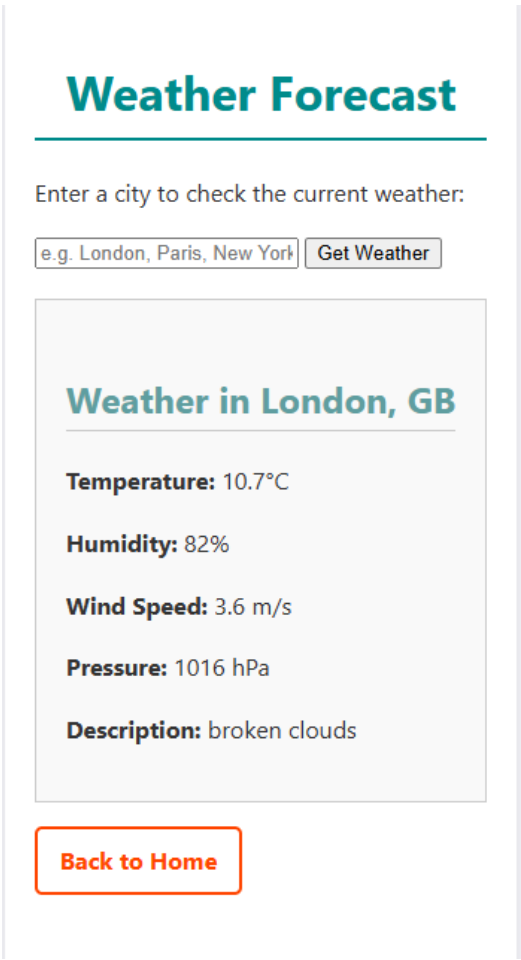


# Workout Log

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

Home

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 or not.



# Weather Forecast

Enter a city to check the current weather:

e.g. London, Paris, New York | Get Weather

## Weather in London, GB

**Temperature:** 10.7°C

**Humidity:** 82%

**Wind Speed:** 3.6 m/s

**Pressure:** 1016 hPa

**Description:** broken clouds

Back to Home

**Advanced Techniques**

Advanced development techniques used to improve app security, reliability, and functionality:

**1. Machine Readable API Endpoint** A JSON API endpoint is established to allow other applications to pull workout information from this app. This illustrates a knowledge of RESTful standards beyond basic 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** For increased security monitoring, I added an audit logging feature so that whenever users try to log in, the attempt is recorded in the audit_log table of the database with a timestamp (successful or unsuccessful). This is meant to notify admins of 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 incorporated express-sanitizer to cleanse user input before it goes to the database. Thus, this prevents cross-site scripting (XSS) from happening by eliminating harmful HTML tags. Furthermore, I incorporated bcrypt to hash passwords, so even if the database is vulnerable, user passwords are safe and sound. *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 reflected my ability to work with third-party services by incorporating the OpenWeather API through the request module where my application handles the asynchronous call to the HTTP request, parses JSON responses and handles errors (i.e. if a user inputs a city that doesn't exist). *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