# Health and fitness app report

## Outline

The Health Tracker App is a full-stack web application. Its purpose is to help users record, manage, and analyse their fitness and wellbeing activities in a secure and user friendly environment. Users can create accounts, log in securely, and record workouts and achievements such as personal bests or fitness milestones.

The application supports searching and filtering workouts, viewing short workouts, and receiving weather based exercise suggestions via an external API. A key feature is the Analytics Dashboard, which visualises workout data using charts to provide meaningful insights into user activity patterns over time as users add more workouts.

The system demonstrates core dynamic web concepts including routing, server-side rendering with EJS, database integration using MySQL, authentication and session management, validation and sanitisation, API endpoints, and the integration of third-party services. The application is deployed on the Goldsmiths virtual server and the full source code is available via GitHub, with installation instructions provided in the README.

## Architecture

The application follows a three-tier architecture.

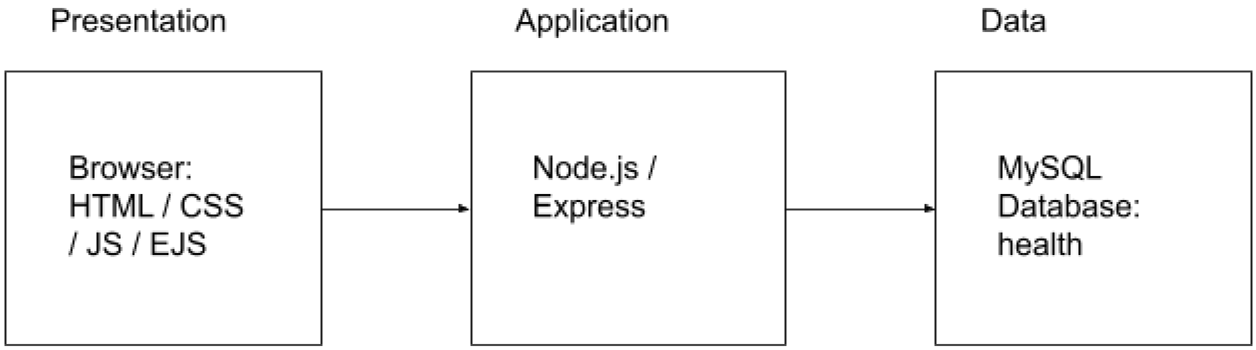
1) The presentation layer - consists of EJS templates, HTML, CSS, and client side JavaScript, which render dynamic pages and handle user interaction.

2) The application layer - built with Node.js and Express. These handle routing, business logic, authentication, and API endpoints. EJS is used for server side rendering of views.

3) The data layer -  uses a MySQL relational database. The databases store information such as users, workouts and achievements, and an Audit login log.

External services e.g OpenWeatherMap API are accessed through HTTP. Client side libraries e.g Chart.js are used for data visualisation.

**High-level architecture diagram:**



Data Model

The data model is relational and designed to support user specific fitness tracking. The users table stores account credentials and profile data. Each user can have many workouts and achievements, linked using foreign keys. The login_audit table records authentication attempts for security monitoring. This structure ensures data integrity, supports reporting queries using GROUP BY, and allows secure separation of user data.

## Data model diagram

```
users
 ├── id (Primary Key)
 ├── username
 ├── first
 ├── last
 ├── email
 ├── password
 └── hashedPassword

workouts
 ├── id (Primary Key)
 ├── title
 ├── activity_type
 ├── duration_minutes
 ├── intensity
 ├── notes
 └── performed_at
```

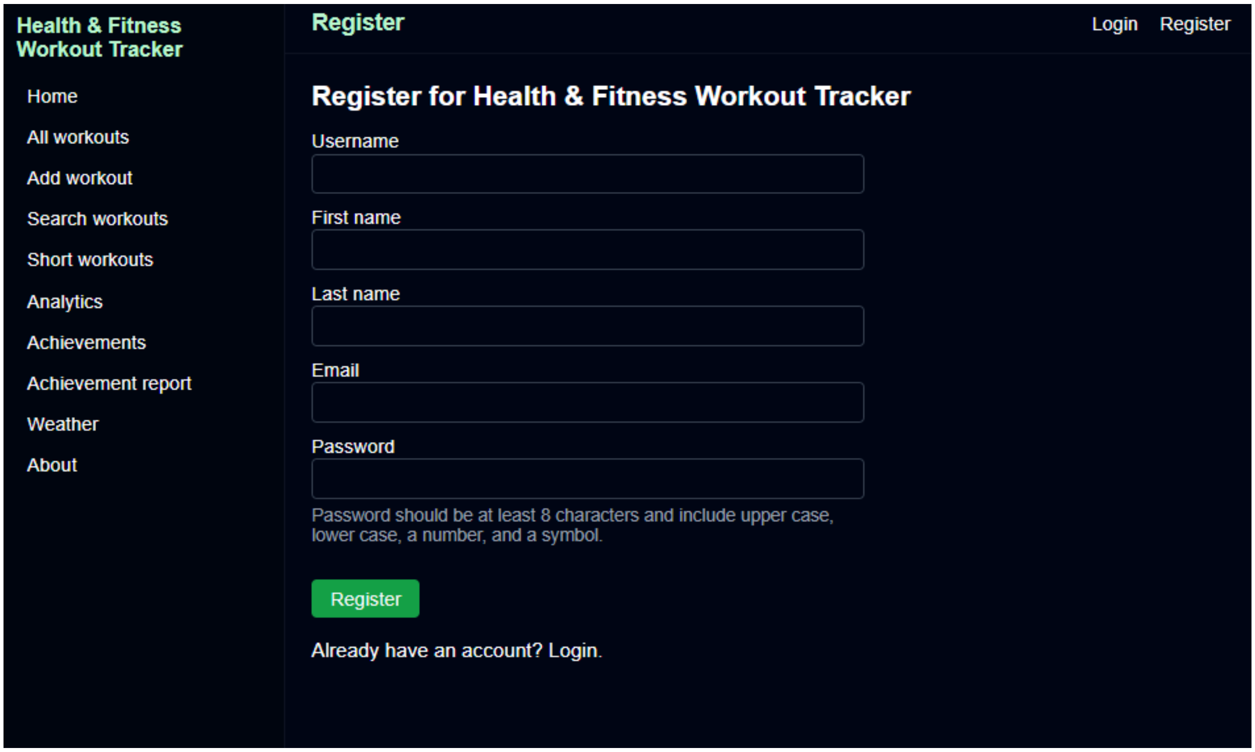
```
achievements
 ├── id (Primary Key)
 ├── title
 ├── description
 ├── category
 ├── metric_value
 ├── metric_unit
 ├── achieved_at
 └── created_by

login_audit
 ├── id (Primary Key)
 ├── identifier
 ├── success
 ├── reason
 ├── ip
 └── created_at
```

## User Functionality

The application provides a clear and structured user experience centred around fitness tracking and analysis.

**1) User accounts and security:**

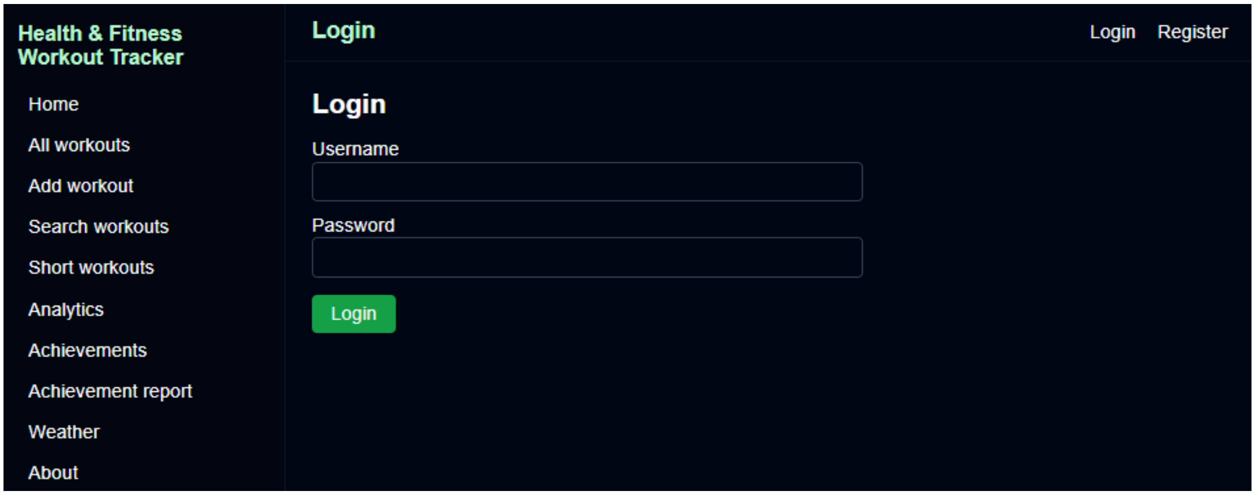
Users can register and log in using a secure authentication system. Passwords are hashed using bcrypt, and login sessions are managed with express-session. Protected routes ensure that only authenticated users can access private pages such as workouts, achievements, analytics, and audit logs.



Register for Health & Fitness Workout Tracker

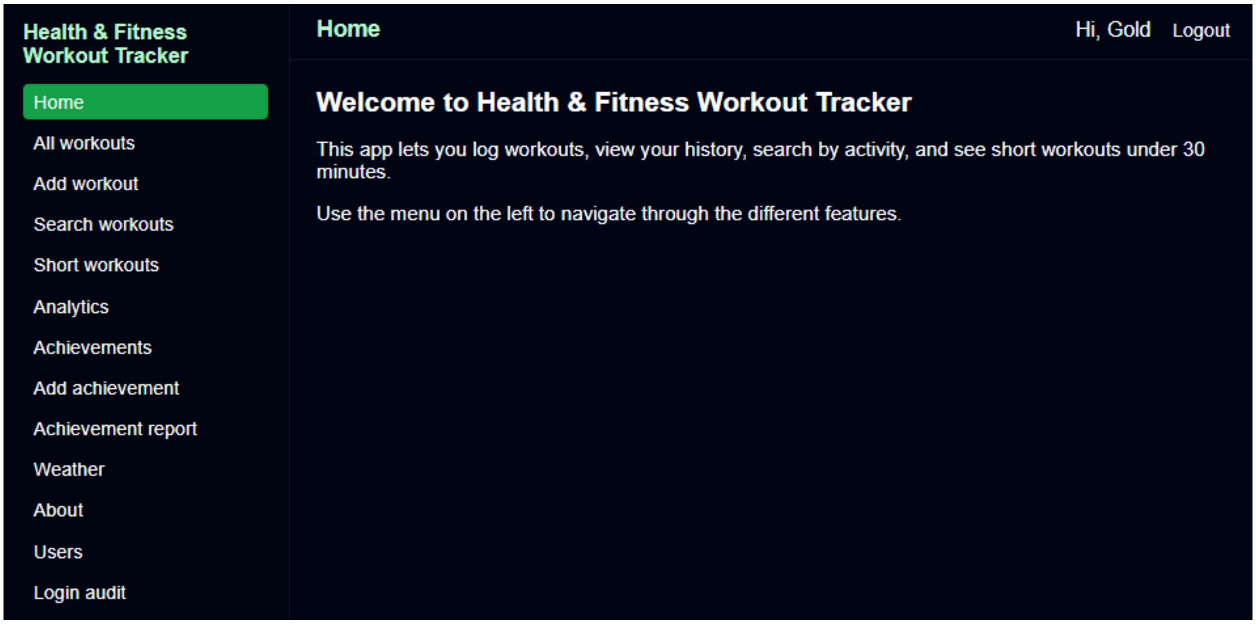
Password should be at least 8 characters and include upper case, lower case, a number, and a symbol.

Already have an account? Login.



Login

After users are logged in, they will now have access to the audit log of login attempts, add achievements and also see a list of all users saved in the database. These are shown on the menu on the left side of the page.



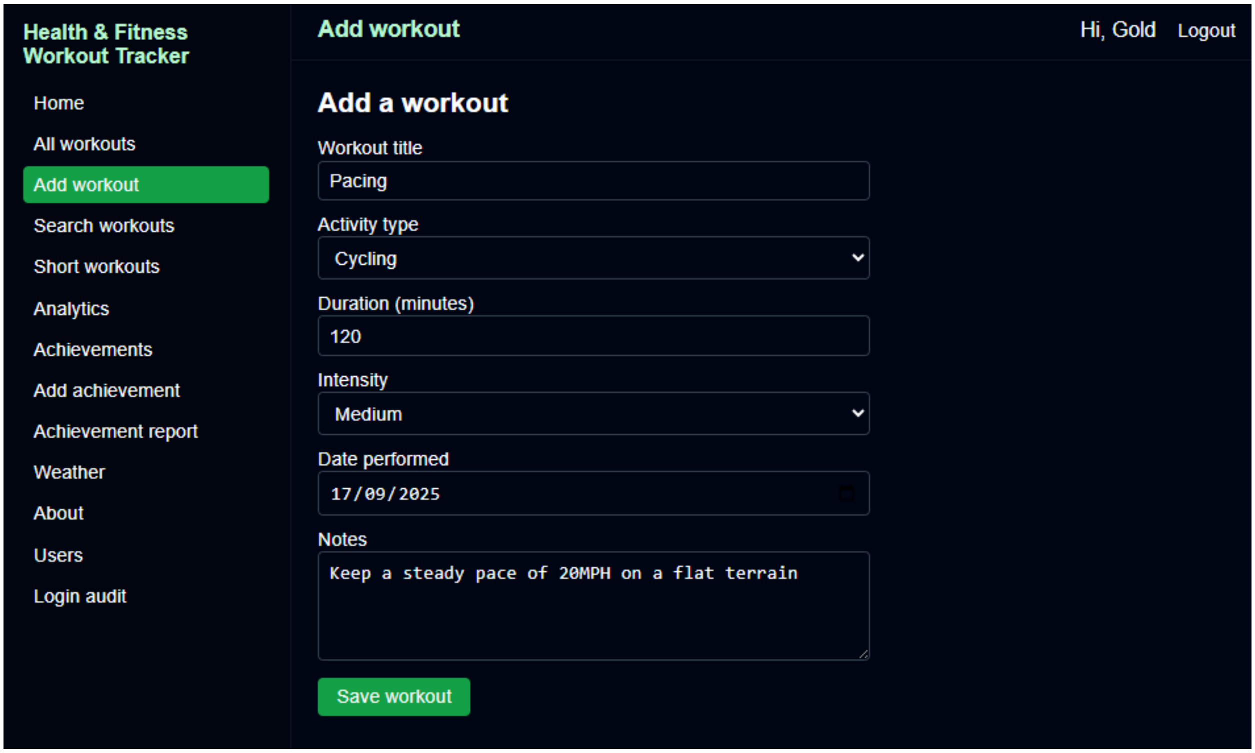
Welcome to Health & Fitness Workout Tracker

This app lets you log workouts, view your history, search by activity, and see short workouts under 30 minutes.

Use the menu on the left to navigate through the different features.

## 2) Workout management

Logged-in users can add workouts by providing details such as activity type, duration, intensity, and date. Users can view all their workouts, search by keyword, filter short workouts (less than 30 minutes), and sort results. Input validation ensures correct data entry, while sanitisation protects against XSS attacks.



Add a workout

Workout title: Pacing

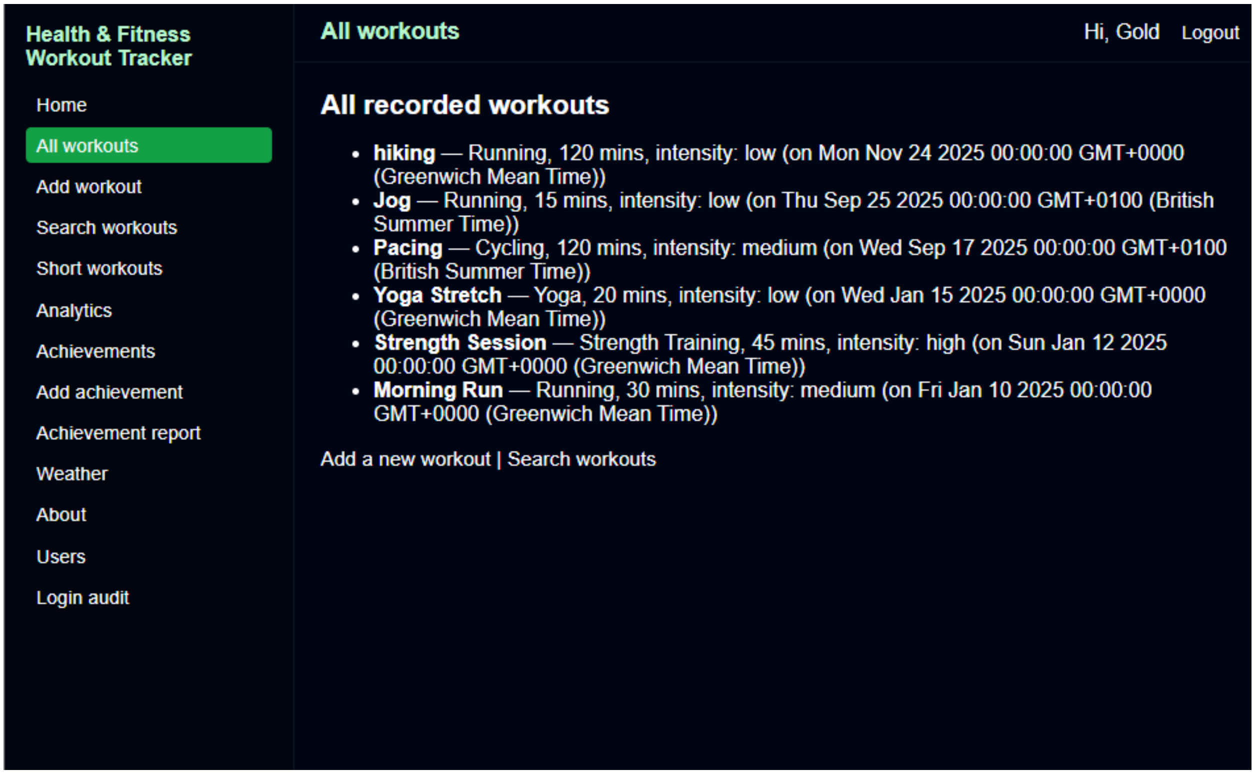
Activity type: Cycling

Duration (minutes): 120

Intensity: Medium

Date performed: 17/09/2025

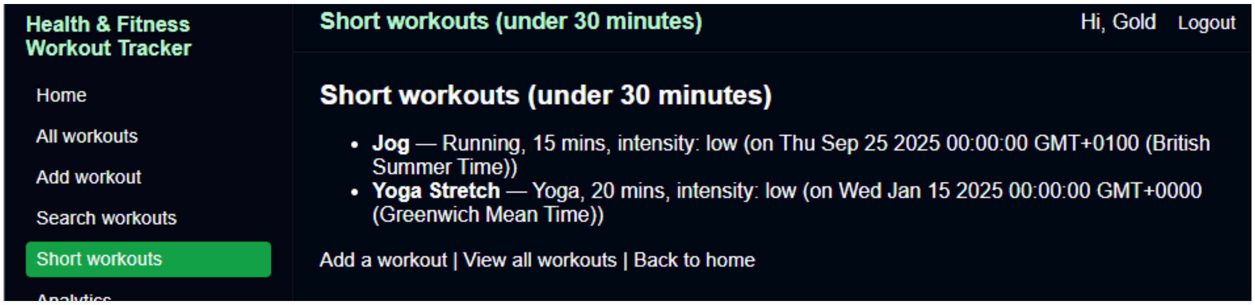
Notes: Keep a steady pace of 20MPH on a flat terrain



## All recorded workouts

- **hiking** — Running, 120 mins, intensity: low (on Mon Nov 24 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
- **Jog** — Running, 15 mins, intensity: low (on Thu Sep 25 2025 00:00:00 GMT+0100 (British Summer Time))
- **Pacing** — Cycling, 120 mins, intensity: medium (on Wed Sep 17 2025 00:00:00 GMT+0100 (British Summer Time))
- **Yoga Stretch** — Yoga, 20 mins, intensity: low (on Wed Jan 15 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
- **Strength Session** — Strength Training, 45 mins, intensity: high (on Sun Jan 12 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
- **Morning Run** — Running, 30 mins, intensity: medium (on Fri Jan 10 2025 00:00:00 GMT+0000 (Greenwich Mean Time))

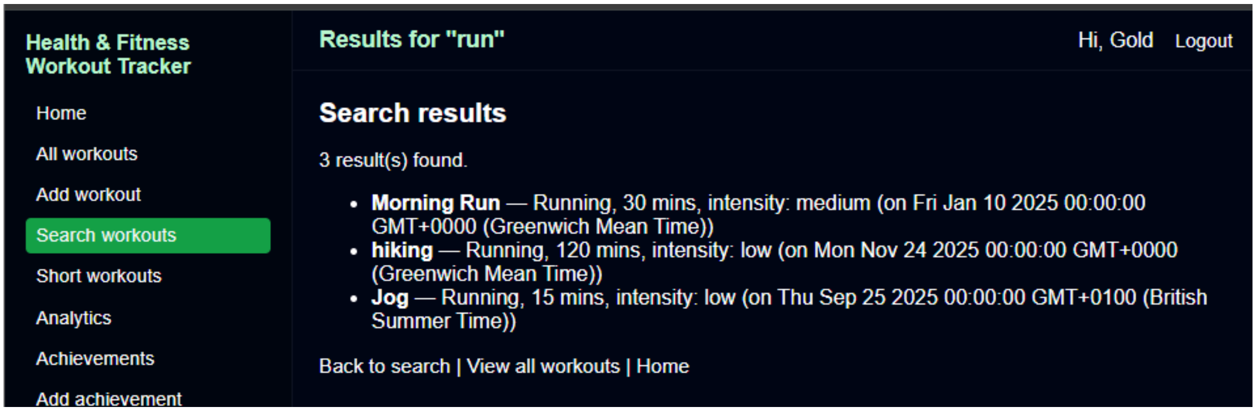
Add a new workout | Search workouts



## Short workouts (under 30 minutes)

- **Jog** — Running, 15 mins, intensity: low (on Thu Sep 25 2025 00:00:00 GMT+0100 (British Summer Time))
- **Yoga Stretch** — Yoga, 20 mins, intensity: low (on Wed Jan 15 2025 00:00:00 GMT+0000 (Greenwich Mean Time))

Add a workout | View all workouts | Back to home



## Search results

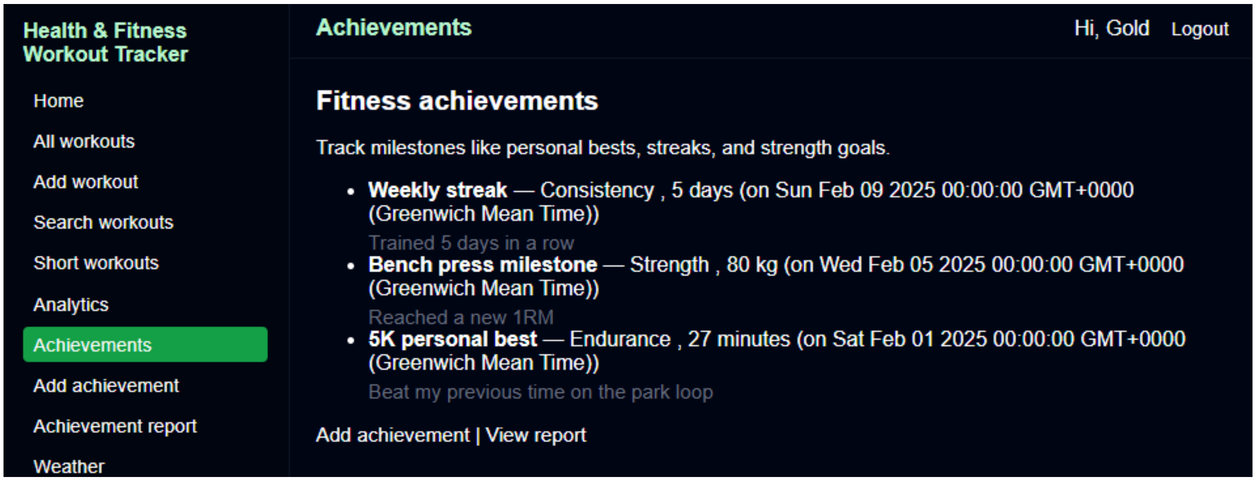
3 result(s) found.

- **Morning Run** — Running, 30 mins, intensity: medium (on Fri Jan 10 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
- **hiking** — Running, 120 mins, intensity: low (on Mon Nov 24 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
- **Jog** — Running, 15 mins, intensity: low (on Thu Sep 25 2025 00:00:00 GMT+0100 (British Summer Time))

Back to search | View all workouts | Home

### 3) Achievements tracking

Users can record achievements such as personal bests or fitness milestones. Achievements support optional metric values and units (e.g. kg, minutes). The system provides reports that summarise achievements by category and highlight best recorded values, as well as a recent activity feed.

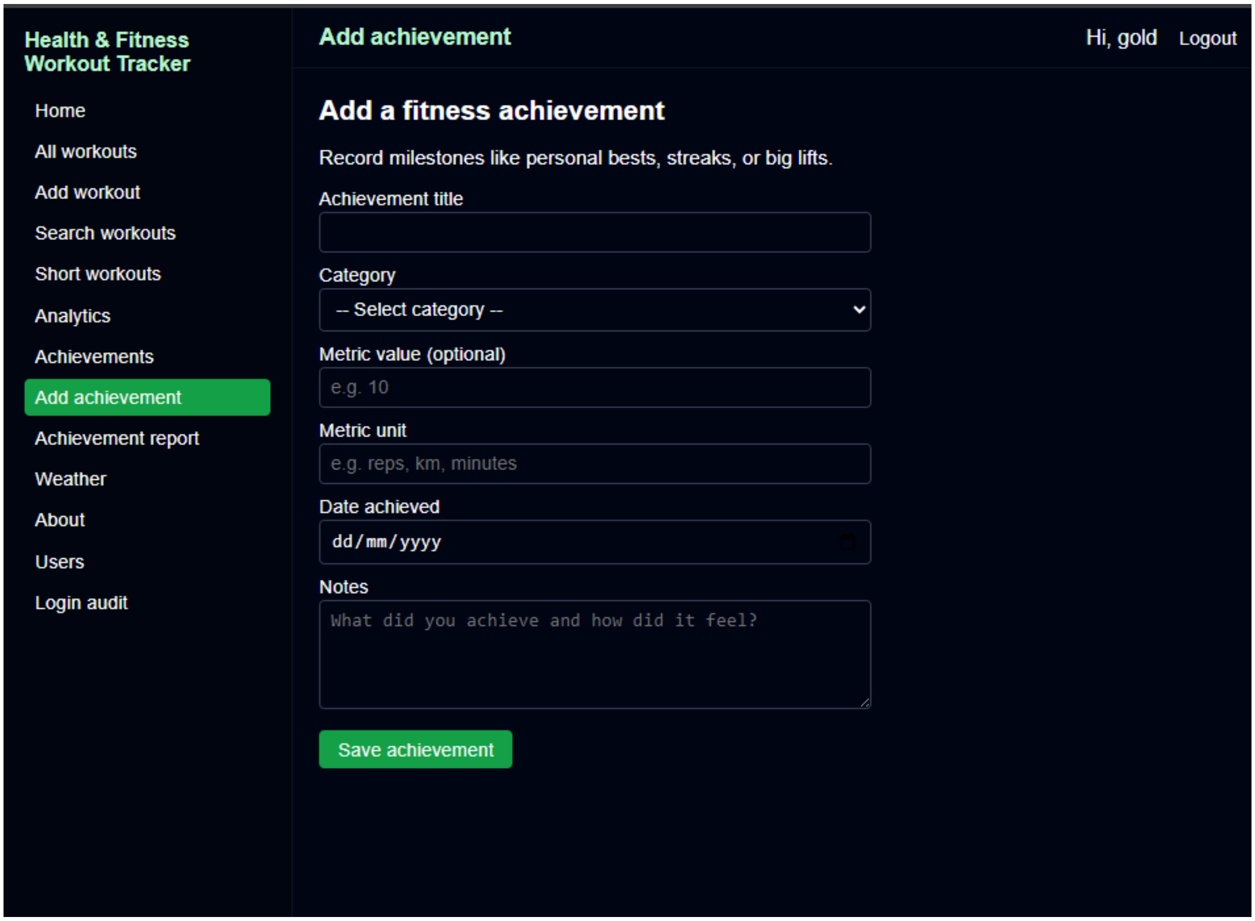


## Fitness achievements

Track milestones like personal bests, streaks, and strength goals.

- **Weekly streak** — Consistency , 5 days (on Sun Feb 09 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
  Trained 5 days in a row
- **Bench press milestone** — Strength , 80 kg (on Wed Feb 05 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
  Reached a new 1RM
- **5K personal best** — Endurance , 27 minutes (on Sat Feb 01 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
  Beat my previous time on the park loop

Add achievement | View report



## Add a fitness achievement

Record milestones like personal bests, streaks, or big lifts.

Achievement title

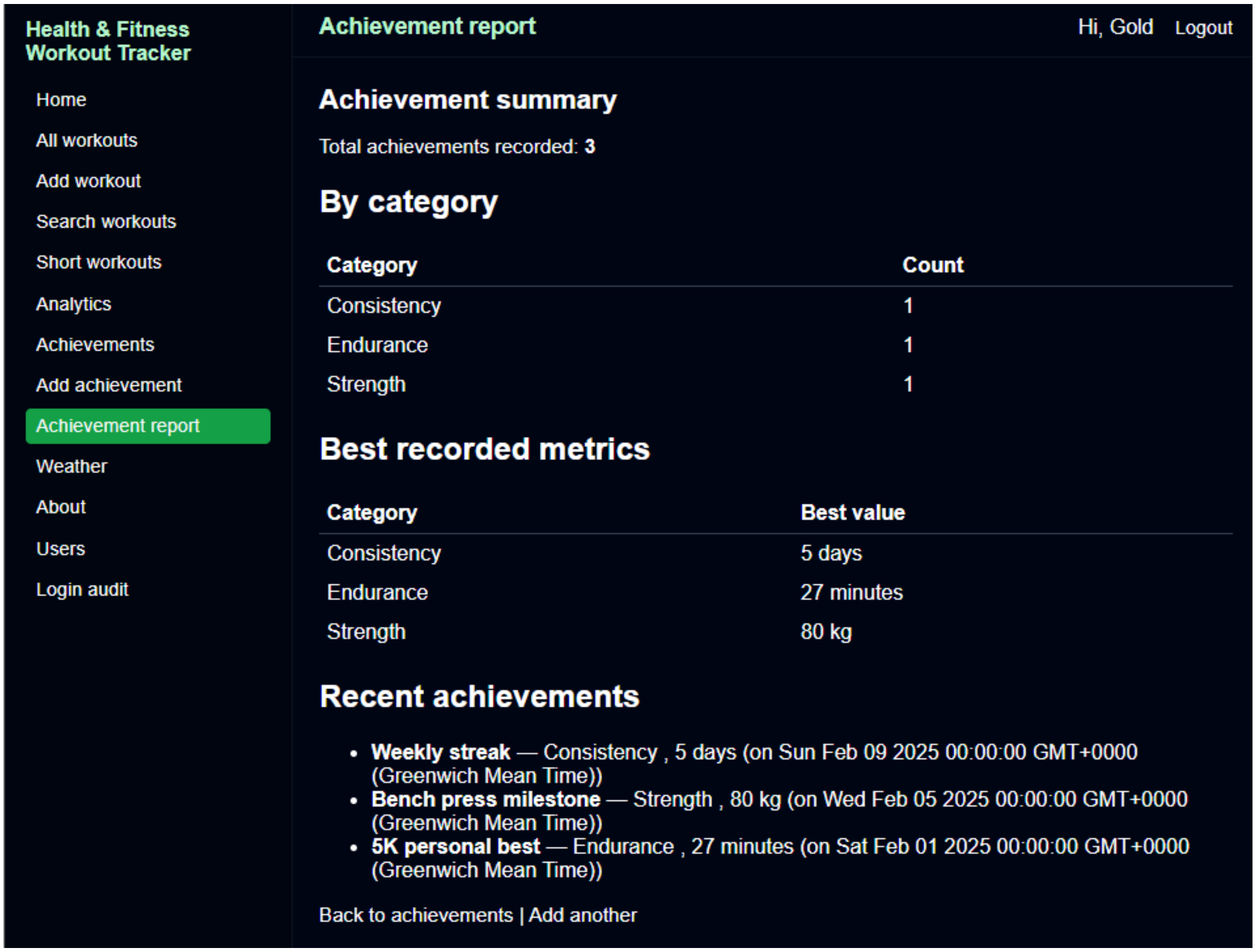
Category
-- Select category --

Metric value (optional)
e.g. 10

Metric unit
e.g. reps, km, minutes

Date achieved
dd/mm/yyyy

Notes
What did you achieve and how did it feel?



Achievement report

Achievement summary

Total achievements recorded: **3**

By category

| Category | Count |
|---|---|
| Consistency | 1 |
| Endurance | 1 |
| Strength | 1 |

Best recorded metrics

| Category | Best value |
|---|---|
| Consistency | 5 days |
| Endurance | 27 minutes |
| Strength | 80 kg |

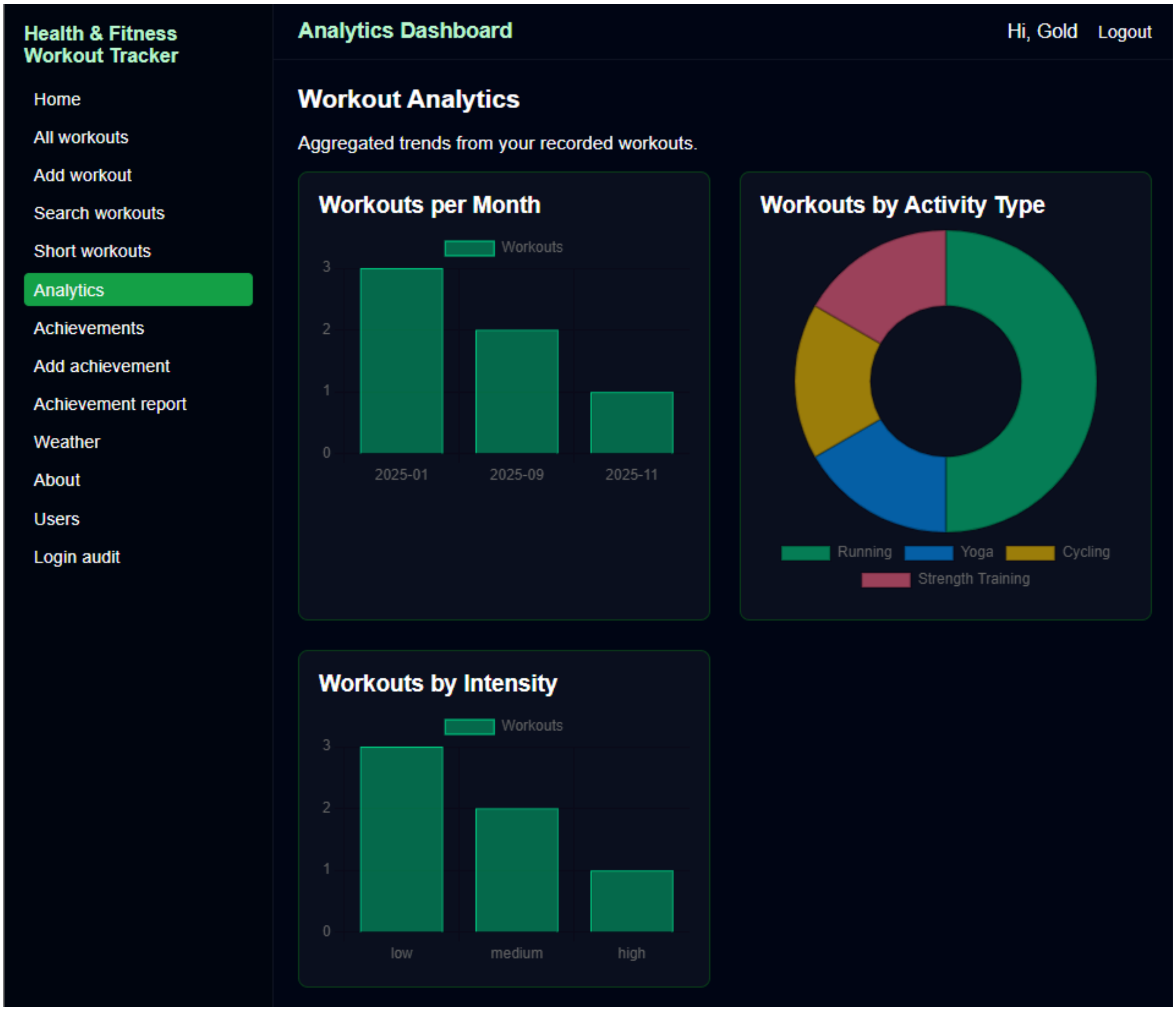
Recent achievements

- **Weekly streak** — Consistency , 5 days (on Sun Feb 09 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
- **Bench press milestone** — Strength , 80 kg (on Wed Feb 05 2025 00:00:00 GMT+0000 (Greenwich Mean Time))
- **5K personal best** — Endurance , 27 minutes (on Sat Feb 01 2025 00:00:00 GMT+0000 (Greenwich Mean Time))

Back to achievements | Add another

### 4) Analytics dashboard

The Analytics page presents visual summaries of workout data using Chart.js. It displays workouts per month, by activity type, and by intensity. Data is retrieved asynchronously from a protected internal API endpoint. If no data exists, a friendly message is shown instead of empty charts.



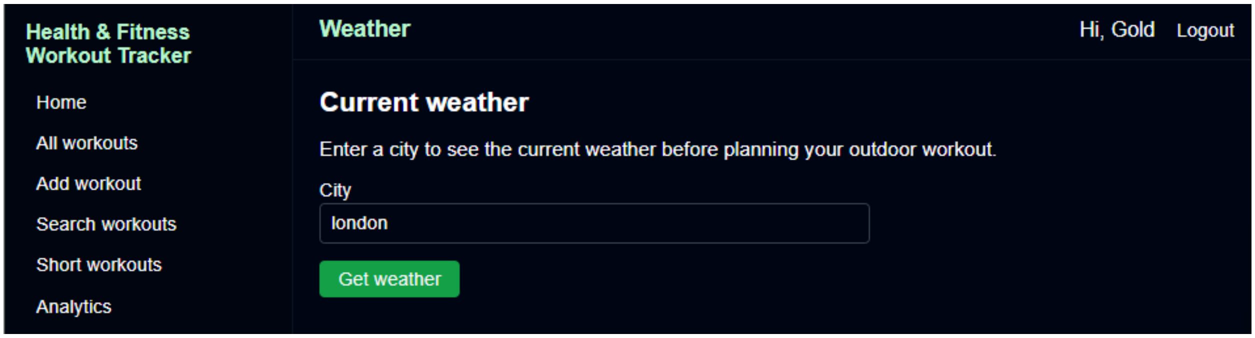
Analytics Dashboard

Workout Analytics

Aggregated trends from your recorded workouts.

### 5) Weather integration

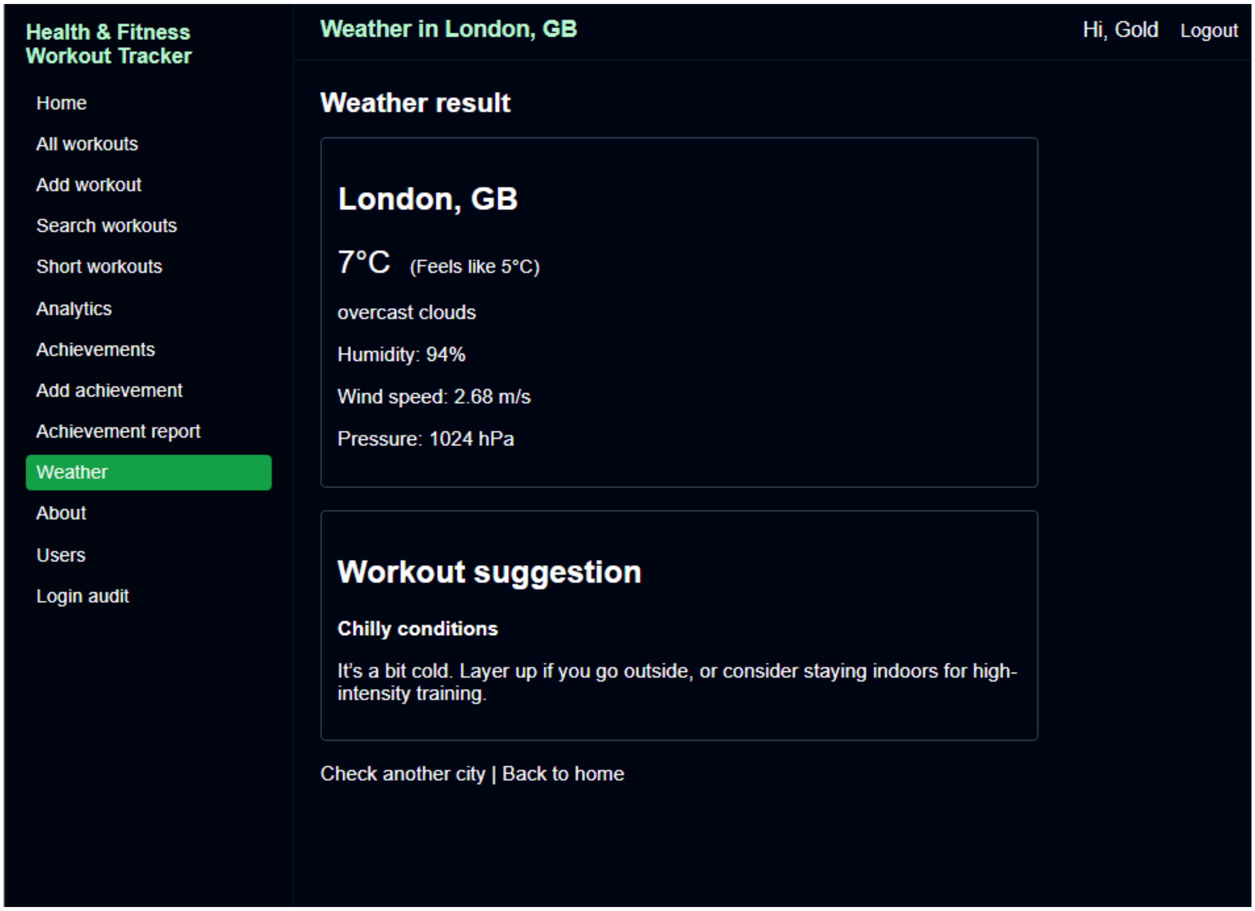
The Weather page uses the OpenWeatherMap API to show live weather data and suggest suitable exercises based on current conditions, demonstrating external API integration. As shown below, if the conditions of the location change, it will suggest different workouts. This could be based on temperature, rain and other weather conditions.



Current weather

Enter a city to see the current weather before planning your outdoor workout.

City: london



Weather in London, GB

Weather result

London, GB

7°C (Feels like 5°C)

overcast clouds

Humidity: 94%

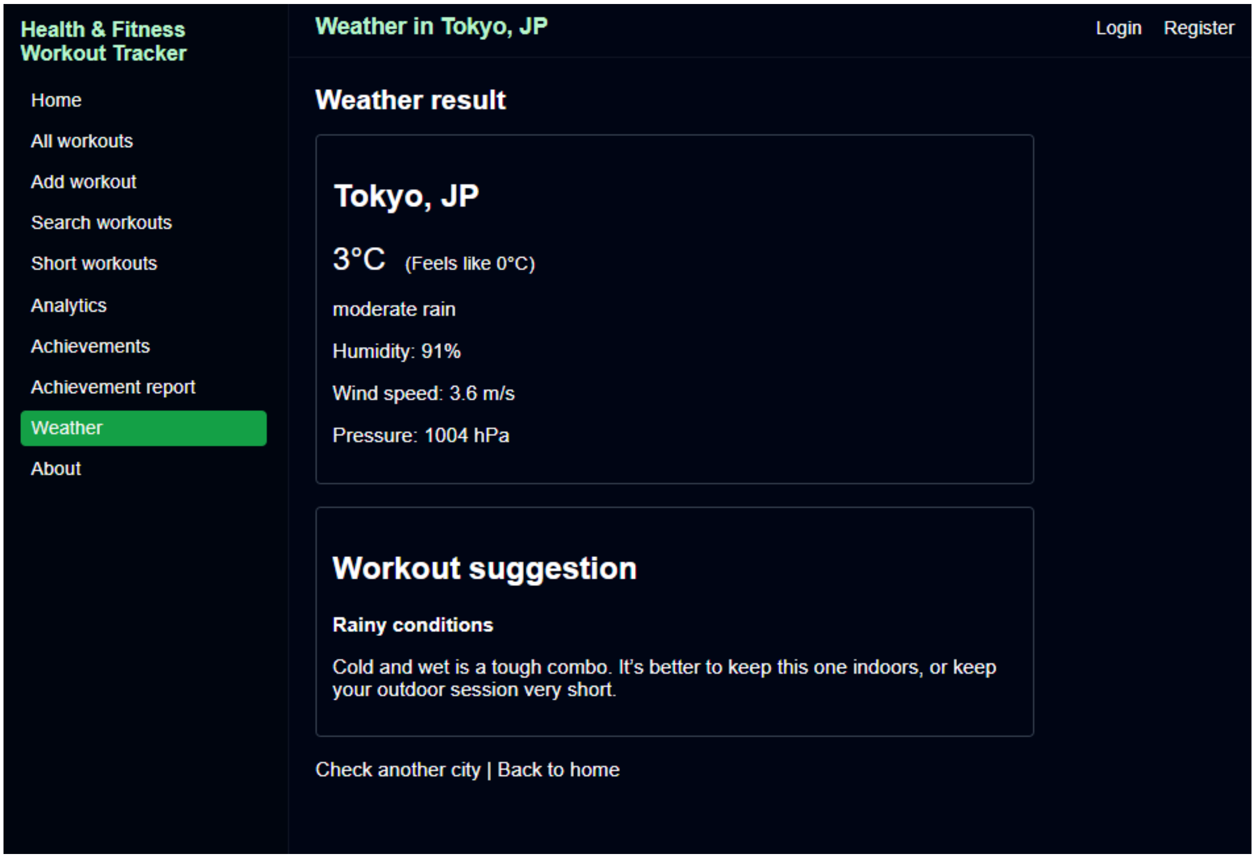
Wind speed: 2.68 m/s

Pressure: 1024 hPa

Workout suggestion

**Chilly conditions**

It's a bit cold. Layer up if you go outside, or consider staying indoors for high-intensity training.



Weather in Tokyo, JP

## Weather result

### Tokyo, JP

3°C (Feels like 0°C)

moderate rain

Humidity: 91%

Wind speed: 3.6 m/s

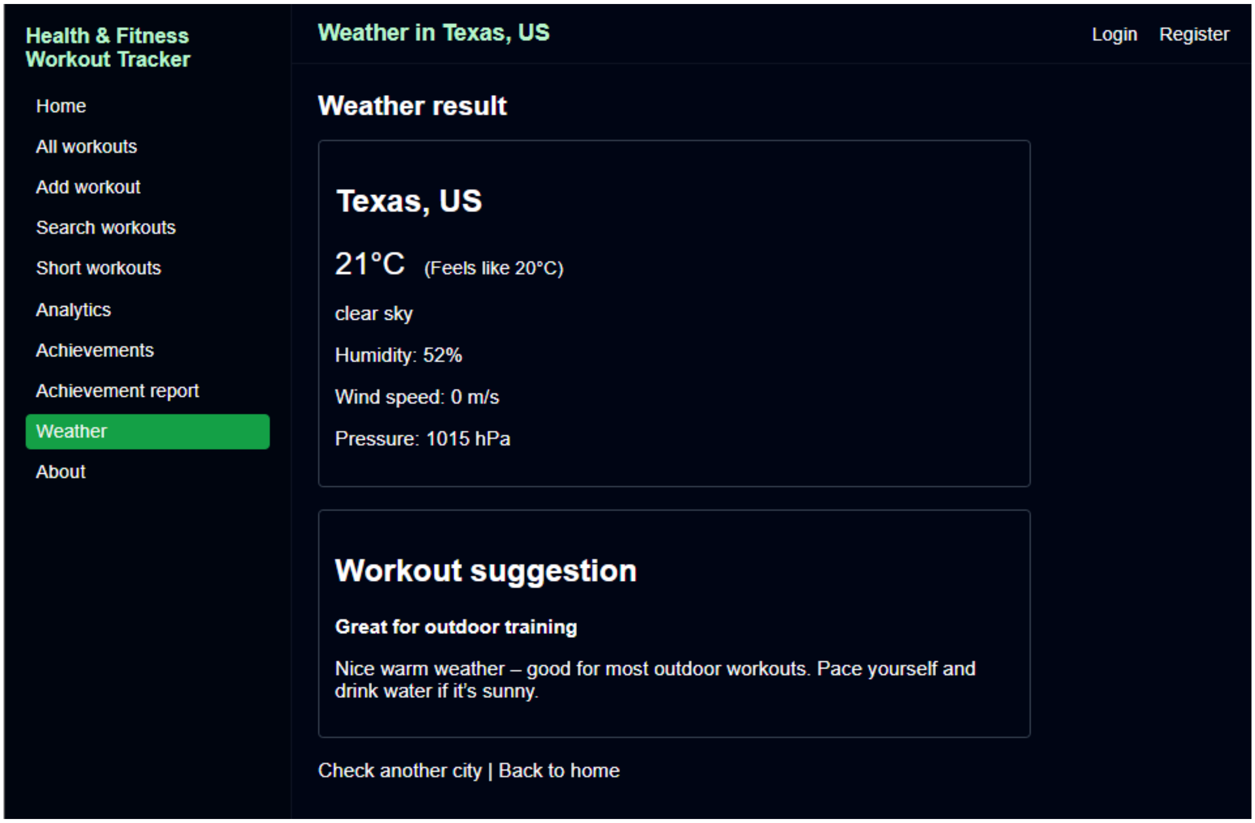
Pressure: 1004 hPa

### Workout suggestion

**Rainy conditions**

Cold and wet is a tough combo. It's better to keep this one indoors, or keep your outdoor session very short.

Check another city | Back to home



Weather in Texas, US

## Weather result

### Texas, US

21°C (Feels like 20°C)

clear sky

Humidity: 52%

Wind speed: 0 m/s

Pressure: 1015 hPa

### Workout suggestion

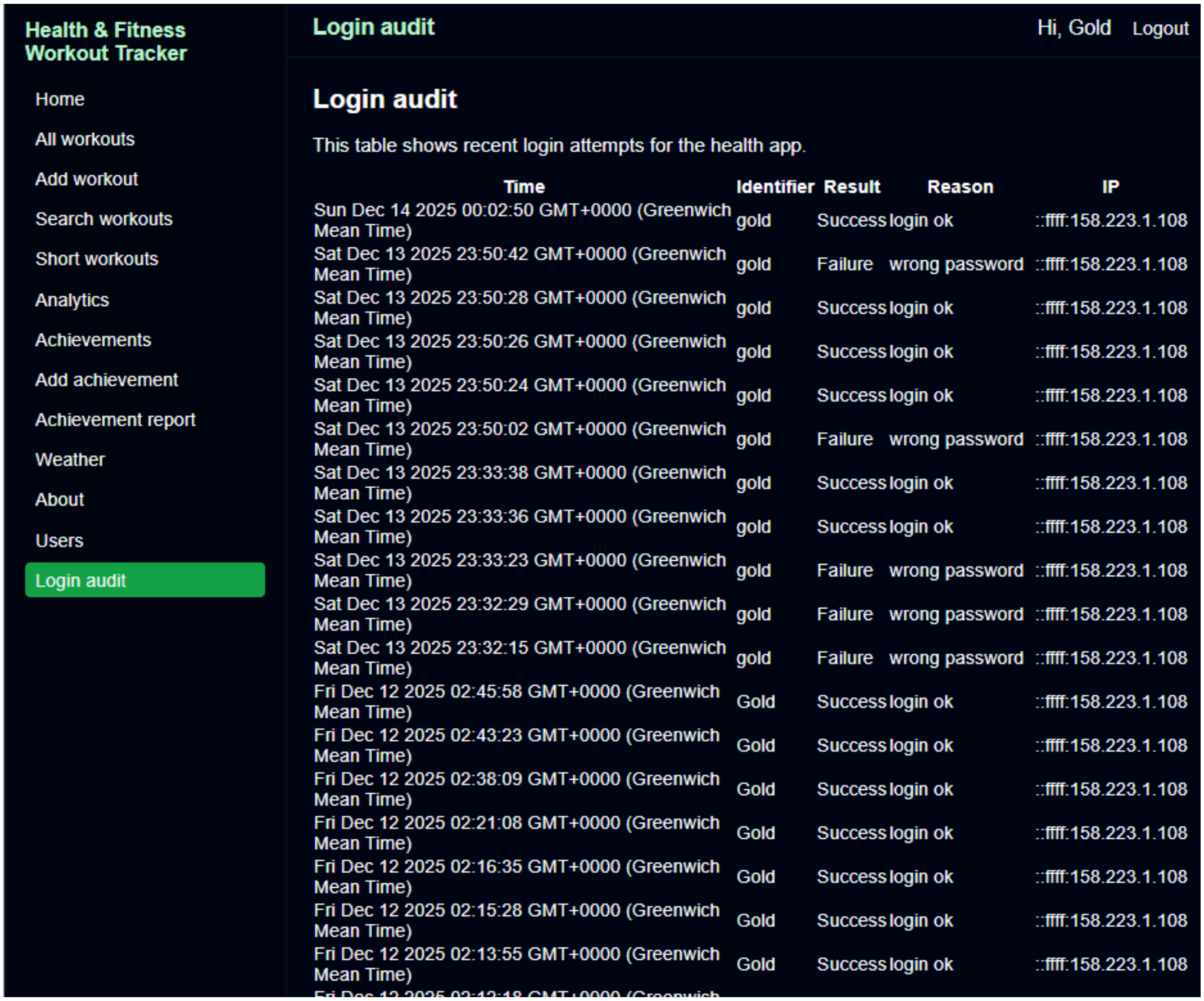
**Great for outdoor training**

Nice warm weather – good for most outdoor workouts. Pace yourself and drink water if it's sunny.

Check another city | Back to home

## 6) Administrative and audit features

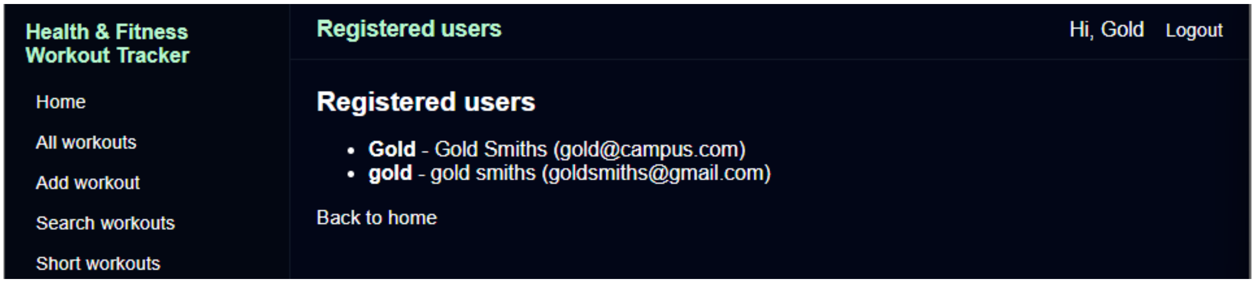
An audit log records all login attempts with timestamps and IP addresses. This supports transparency and security monitoring.There is also a users log that lists all users saved in the database.



### Login audit

This table shows recent login attempts for the health app.

| Time | Identifier | Result | Reason | IP |
|---|---|---|---|---|
| Sun Dec 14 2025 00:02:50 GMT+0000 (Greenwich Mean Time) | gold | Success | login ok | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:50:42 GMT+0000 (Greenwich Mean Time) | gold | Failure | wrong password | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:50:28 GMT+0000 (Greenwich Mean Time) | gold | Success | login ok | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:50:26 GMT+0000 (Greenwich Mean Time) | gold | Success | login ok | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:50:24 GMT+0000 (Greenwich Mean Time) | gold | Success | login ok | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:50:02 GMT+0000 (Greenwich Mean Time) | gold | Failure | wrong password | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:33:38 GMT+0000 (Greenwich Mean Time) | gold | Success | login ok | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:33:36 GMT+0000 (Greenwich Mean Time) | gold | Success | login ok | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:33:23 GMT+0000 (Greenwich Mean Time) | gold | Failure | wrong password | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:32:29 GMT+0000 (Greenwich Mean Time) | gold | Failure | wrong password | ::ffff:158.223.1.108 |
| Sat Dec 13 2025 23:32:15 GMT+0000 (Greenwich Mean Time) | gold | Failure | wrong password | ::ffff:158.223.1.108 |
| Fri Dec 12 2025 02:45:58 GMT+0000 (Greenwich Mean Time) | Gold | Success | login ok | ::ffff:158.223.1.108 |
| Fri Dec 12 2025 02:43:23 GMT+0000 (Greenwich Mean Time) | Gold | Success | login ok | ::ffff:158.223.1.108 |
| Fri Dec 12 2025 02:38:09 GMT+0000 (Greenwich Mean Time) | Gold | Success | login ok | ::ffff:158.223.1.108 |
| Fri Dec 12 2025 02:21:08 GMT+0000 (Greenwich Mean Time) | Gold | Success | login ok | ::ffff:158.223.1.108 |
| Fri Dec 12 2025 02:16:35 GMT+0000 (Greenwich Mean Time) | Gold | Success | login ok | ::ffff:158.223.1.108 |
| Fri Dec 12 2025 02:15:28 GMT+0000 (Greenwich Mean Time) | Gold | Success | login ok | ::ffff:158.223.1.108 |
| Fri Dec 12 2025 02:13:55 GMT+0000 (Greenwich Mean Time) | Gold | Success | login ok | ::ffff:158.223.1.108 |
| Fri Dec 12 2025 02:12:18 GMT+0000 (Greenwich | | | | |



### Registered users

- **Gold** - Gold Smiths (gold@campus.com)
- **gold** - gold smiths (goldsmiths@gmail.com)

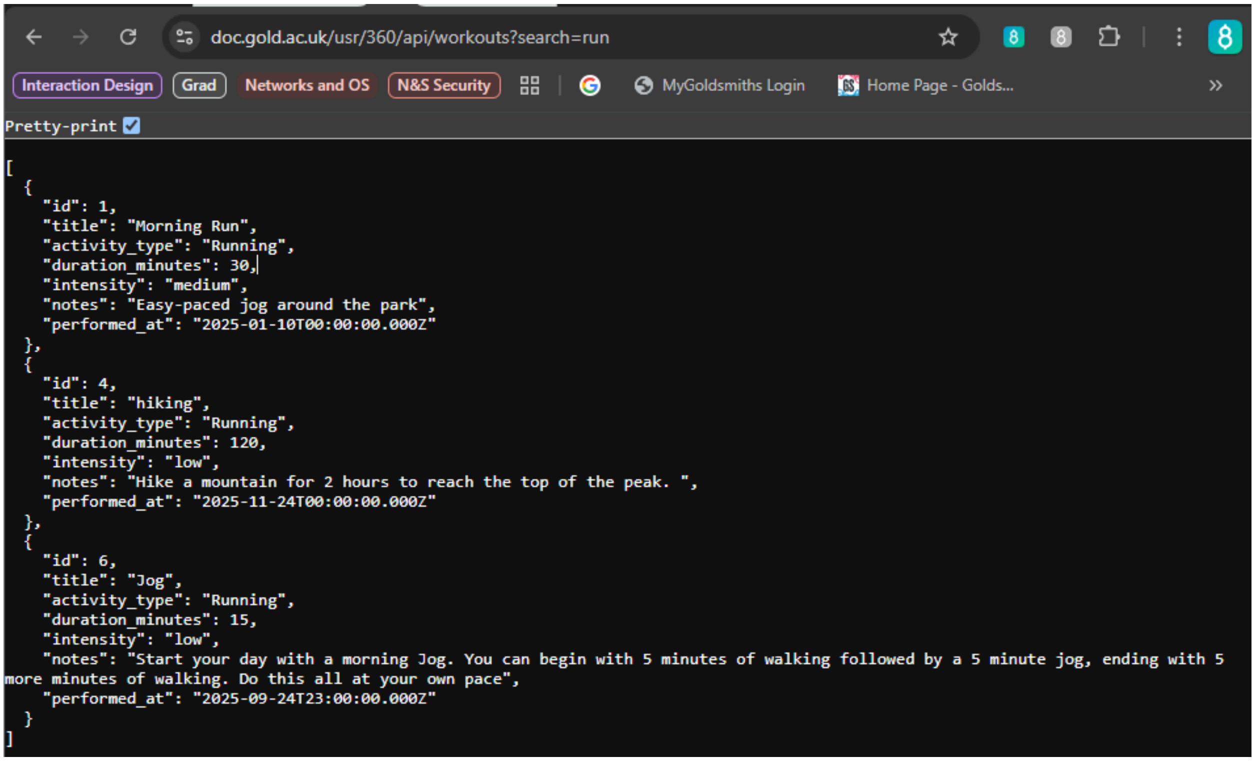
Back to home

## 7) JSON api

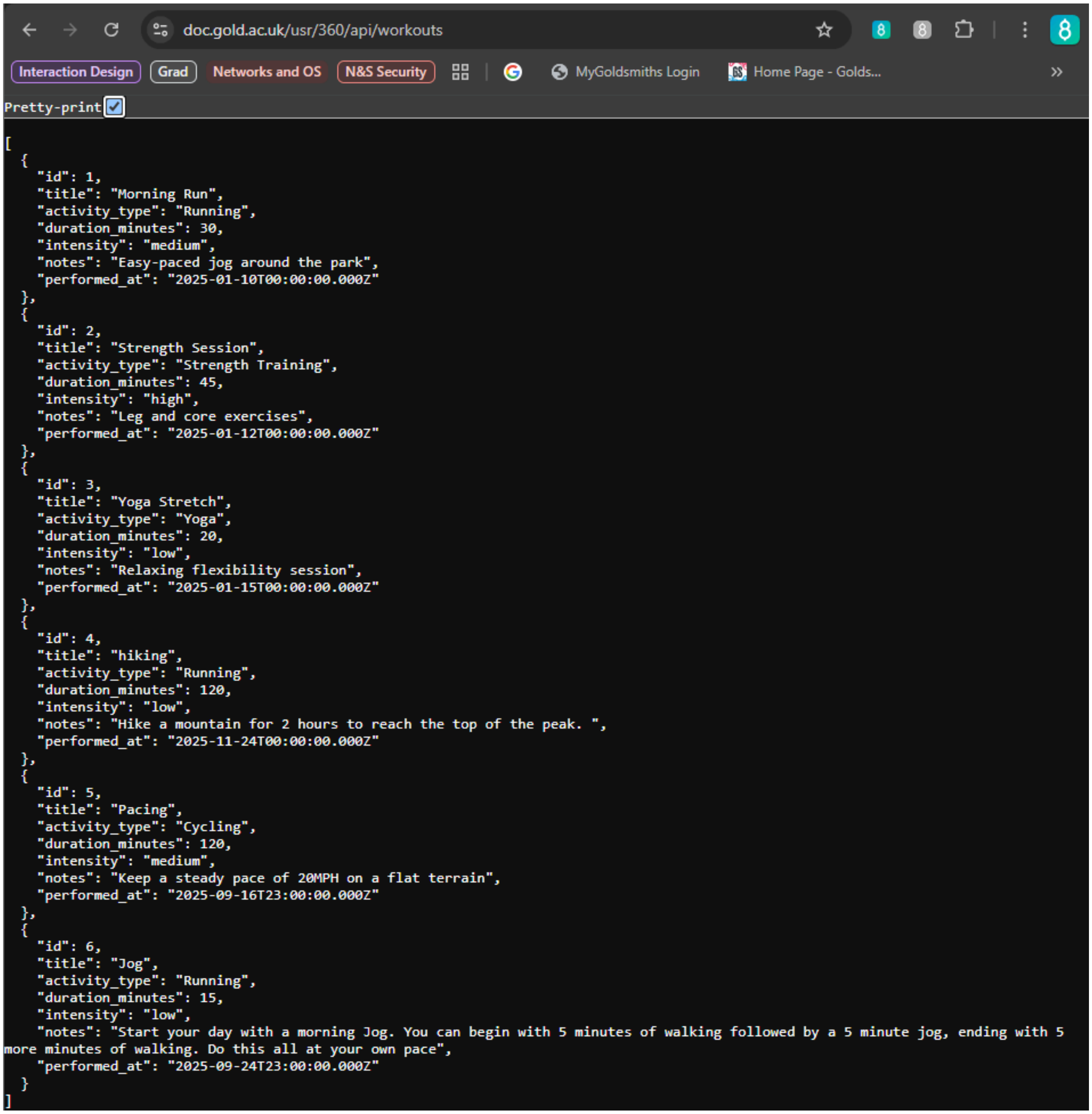
In addition to the standard user pages, the application provides a JSON based API that exposes workout data in a structured format. This API allows workout information to be retrieved directly from the database without rendering an HTML view, demonstrating separation between data access and presentation.

The API endpoint /api/workouts returns a list of workouts as JSON objects. It supports multiple query parameters that allow clients to filter and customise the results. Users can perform text searches on workout titles and activity types, filter workouts by minimum or maximum duration, filter by intensity level, and sort results by title, duration, or date. These filters are processed server-side using SQL queries to ensure efficient data retrieval.

This API can be accessed directly via the browser or developer tools, and screenshots of the returned JSON responses are provided to illustrate the structure of the data.



```
[
  {
    "id": 1,
    "title": "Morning Run",
    "activity_type": "Running",
    "duration_minutes": 30,
    "intensity": "medium",
    "notes": "Easy-paced jog around the park",
    "performed_at": "2025-01-10T00:00:00.000Z"
  },
  {
    "id": 4,
    "title": "hiking",
    "activity_type": "Running",
    "duration_minutes": 120,
    "intensity": "low",
    "notes": "Hike a mountain for 2 hours to reach the top of the peak. ",
    "performed_at": "2025-11-24T00:00:00.000Z"
  },
  {
    "id": 6,
    "title": "Jog",
    "activity_type": "Running",
    "duration_minutes": 15,
    "intensity": "low",
    "notes": "Start your day with a morning Jog. You can begin with 5 minutes of walking followed by a 5 minute jog, ending with 5
more minutes of walking. Do this all at your own pace",
    "performed_at": "2025-09-24T23:00:00.000Z"
  }
]
```



Pretty-print ☑

```
[
  {
    "id": 1,
    "title": "Morning Run",
    "activity_type": "Running",
    "duration_minutes": 30,
    "intensity": "medium",
    "notes": "Easy-paced jog around the park",
    "performed_at": "2025-01-10T00:00:00.000Z"
  },
  {
    "id": 2,
    "title": "Strength Session",
    "activity_type": "Strength Training",
    "duration_minutes": 45,
    "intensity": "high",
    "notes": "Leg and core exercises",
    "performed_at": "2025-01-12T00:00:00.000Z"
  },
  {
    "id": 3,
    "title": "Yoga Stretch",
    "activity_type": "Yoga",
    "duration_minutes": 20,
    "intensity": "low",
    "notes": "Relaxing flexibility session",
    "performed_at": "2025-01-15T00:00:00.000Z"
  },
  {
    "id": 4,
    "title": "hiking",
    "activity_type": "Running",
    "duration_minutes": 120,
    "intensity": "low",
    "notes": "Hike a mountain for 2 hours to reach the top of the peak. ",
    "performed_at": "2025-11-24T00:00:00.000Z"
  },
  {
    "id": 5,
    "title": "Pacing",
    "activity_type": "Cycling",
    "duration_minutes": 120,
    "intensity": "medium",
    "notes": "Keep a steady pace of 20MPH on a flat terrain",
    "performed_at": "2025-09-16T23:00:00.000Z"
  },
  {
    "id": 6,
    "title": "Jog",
    "activity_type": "Running",
    "duration_minutes": 15,
    "intensity": "low",
    "notes": "Start your day with a morning Jog. You can begin with 5 minutes of walking followed by a 5 minute jog, ending with 5
more minutes of walking. Do this all at your own pace",
    "performed_at": "2025-09-24T23:00:00.000Z"
  }
]
```

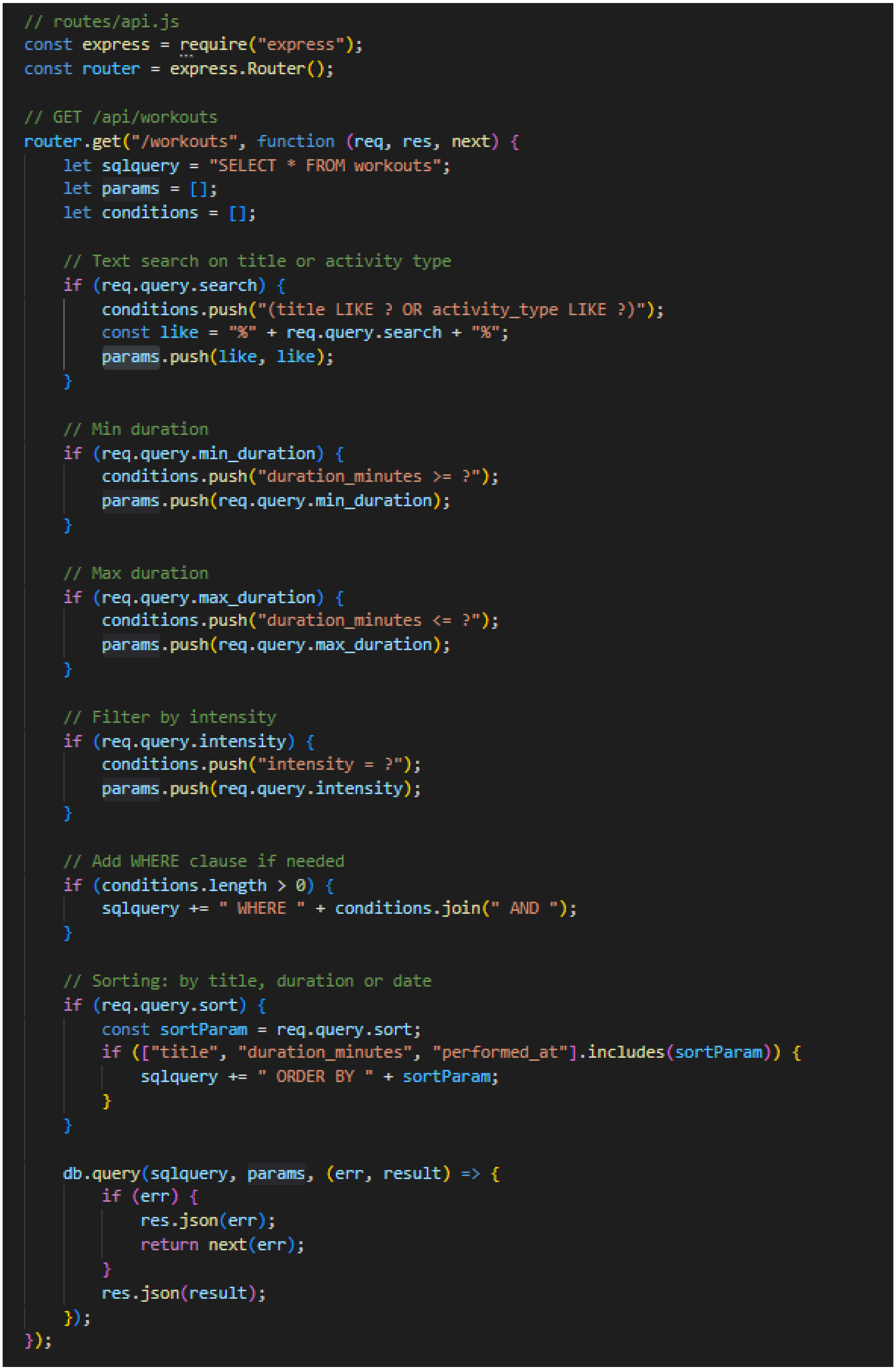
## Advanced Techniques

Several advanced techniques beyond core lab requirements are implemented.

### 1) Internal JSON API with query parameters

The application includes an internal JSON API that exposes workout data independently of the user interface. The /api/workouts endpoint returns workout records in JSON format and supports server-side filtering, searching, and sorting via query parameters.

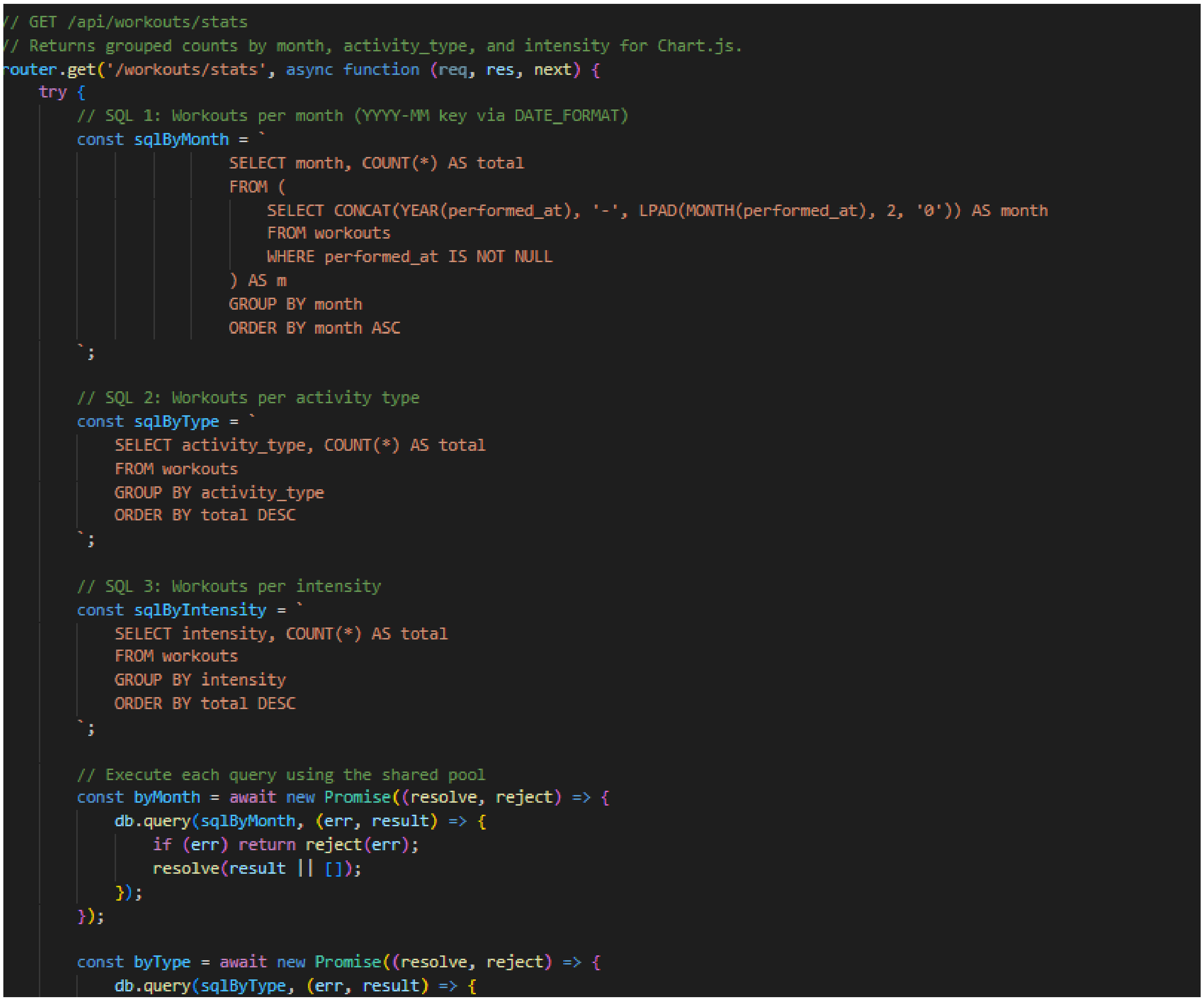
This design separates data access from presentation and allows the same endpoint to be reused by multiple clients, such as browser requests and developer tools.



```
// routes/api.js
const express = require("express");
const router = express.Router();

// GET /api/workouts
router.get("/workouts", function (req, res, next) {
    let sqlquery = "SELECT * FROM workouts";
    let params = [];
    let conditions = [];

    // Text search on title or activity type
    if (req.query.search) {
        conditions.push("(title LIKE ? OR activity_type LIKE ?)");
        const like = "%" + req.query.search + "%";
        params.push(like, like);
    }

    // Min duration
    if (req.query.min_duration) {
        conditions.push("duration_minutes >= ?");
        params.push(req.query.min_duration);
    }

    // Max duration
    if (req.query.max_duration) {
        conditions.push("duration_minutes <= ?");
        params.push(req.query.max_duration);
    }

    // Filter by intensity
    if (req.query.intensity) {
        conditions.push("intensity = ?");
        params.push(req.query.intensity);
    }

    // Add WHERE clause if needed
    if (conditions.length > 0) {
        sqlquery += " WHERE " + conditions.join(" AND ");
    }

    // Sorting: by title, duration or date
    if (req.query.sort) {
        const sortParam = req.query.sort;
        if (["title", "duration_minutes", "performed_at"].includes(sortParam)) {
            sqlquery += " ORDER BY " + sortParam;
        }
    }

    db.query(sqlquery, params, (err, result) => {
        if (err) {
            res.json(err);
            return next(err);
        }
        res.json(result);
    });
});
```

### 2) REST-style internal API for analytics

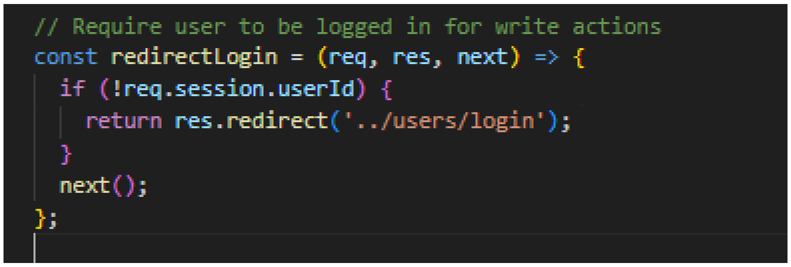
A dedicated API endpoint (/api/workouts/stats) aggregates workout data using **SQL GROUP BY** queries. This cleanly separates data logic from presentation. This helps with analytics and reporting, and is used in the charts.js.



```
/ GET /api/workouts/stats
/ Returns grouped counts by month, activity_type, and intensity for Chart.js.
outer.get('/workouts/stats', async function (req, res, next) {
    try {
        // SQL 1: Workouts per month (YYYY-MM key via DATE_FORMAT)
        const sqlByMonth = `
                        SELECT month, COUNT(*) AS total
                        FROM (
                            SELECT CONCAT(YEAR(performed_at), '-', LPAD(MONTH(performed_at), 2, '0')) AS month
                            FROM workouts
                            WHERE performed_at IS NOT NULL
                        ) AS m
                        GROUP BY month
                        ORDER BY month ASC
        `;

        // SQL 2: Workouts per activity type
        const sqlByType = `
            SELECT activity_type, COUNT(*) AS total
            FROM workouts
            GROUP BY activity_type
            ORDER BY total DESC
        `;

        // SQL 3: Workouts per intensity
        const sqlByIntensity = `
            SELECT intensity, COUNT(*) AS total
            FROM workouts
            GROUP BY intensity
            ORDER BY total DESC
        `;

        // Execute each query using the shared pool
        const byMonth = await new Promise((resolve, reject) => {
            db.query(sqlByMonth, (err, result) => {
                if (err) return reject(err);
                resolve(result || []);
            });
        });

        const byType = await new Promise((resolve, reject) => {
            db.query(sqlByType, (err, result) => {
```

### 3) Centralised base-path–safe routing

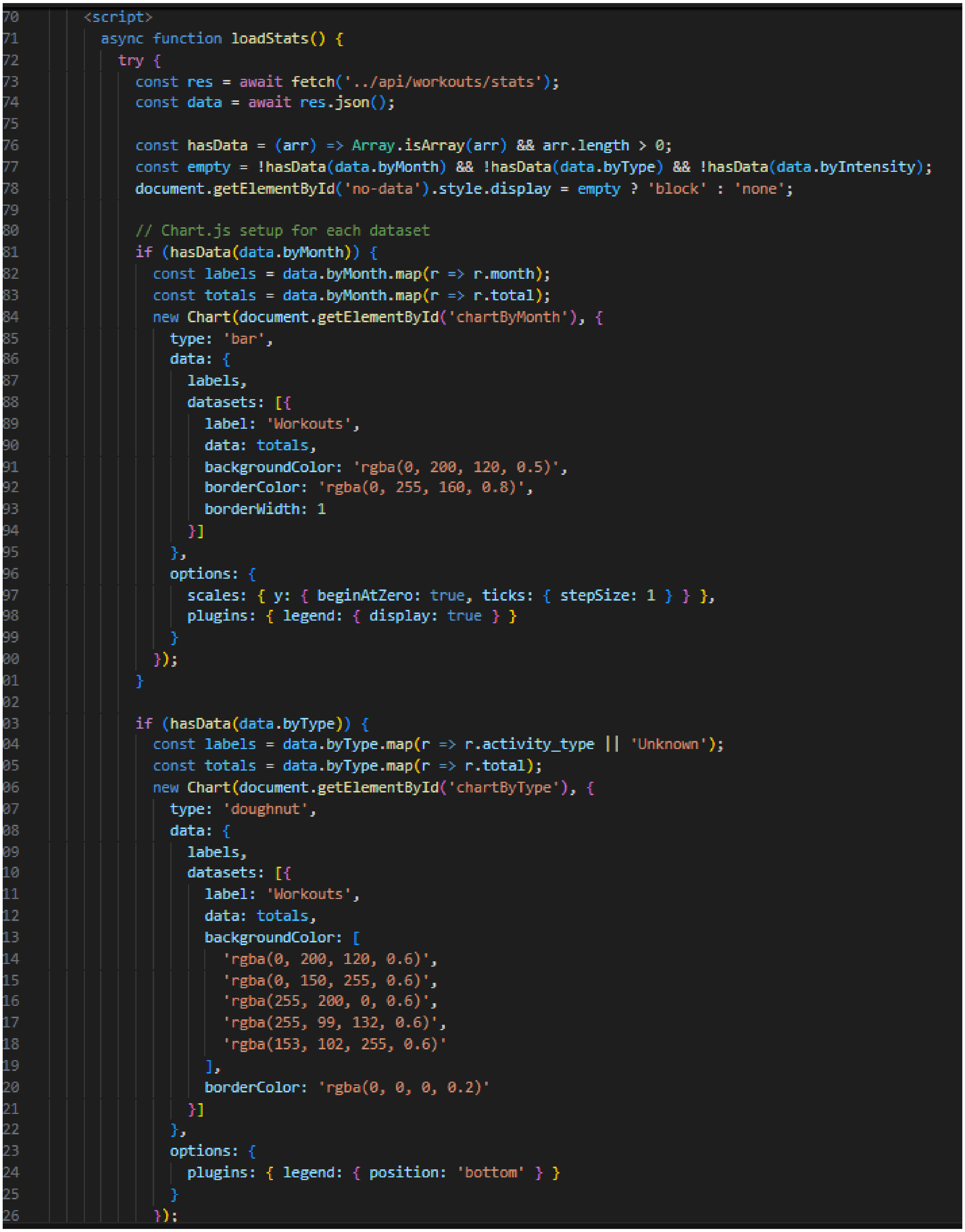
The application was designed to run both locally and on the Goldsmiths virtual server, where the app is deployed under a non root base path (/usr/360). To ensure consistent behaviour across environments, routing and redirects were implemented using base-path–safe and relative paths rather than hard-coded absolute URLs.This approach prevents broken routes and incorrect redirects. By avoiding absolute paths such as /users/login, the application remains robust when deployed under /usr/360, demonstrating practical problem solving



```
// Require user to be logged in for write actions
const redirectLogin = (req, res, next) => {
  if (!req.session.userId) {
    return res.redirect('../users/login');
  }
  next();
};
```

### 4) Client-side data visualisation with chats.js

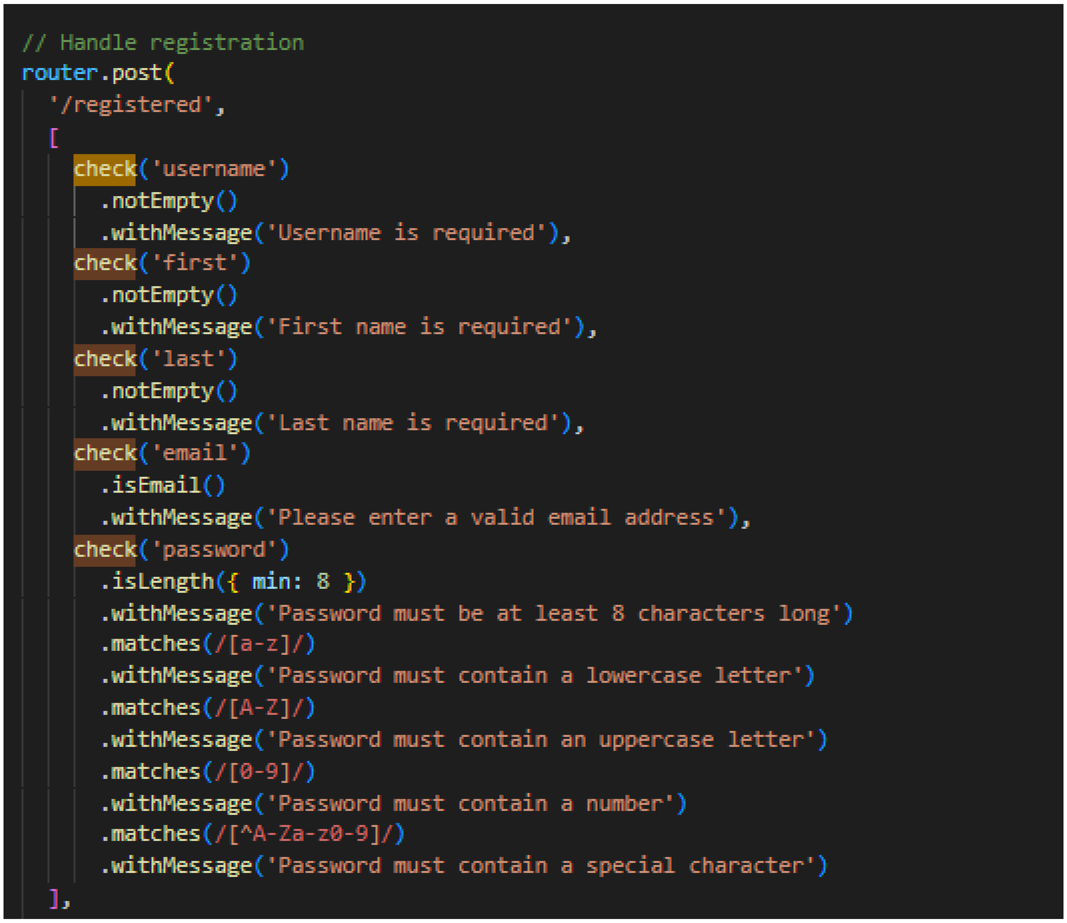
Charts are rendered dynamically using data fetched from the internal API, demonstrating asynchronous client-server interaction.



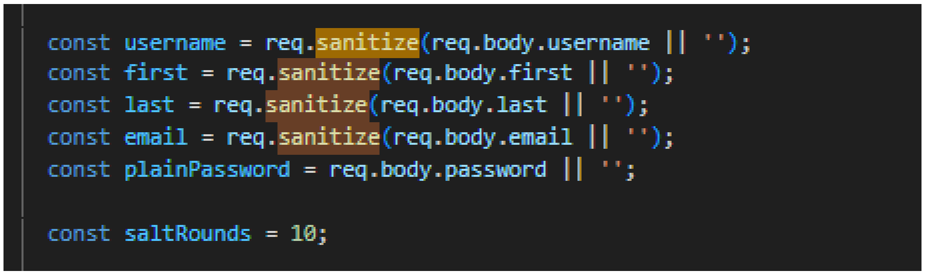
```
    <script>
      async function loadStats() {
        try {
          const res = await fetch('../api/workouts/stats');
          const data = await res.json();

          const hasData = (arr) => Array.isArray(arr) && arr.length > 0;
          const empty = !hasData(data.byMonth) && !hasData(data.byType) && !hasData(data.byIntensity);
          document.getElementById('no-data').style.display = empty ? 'block' : 'none';

          // Chart.js setup for each dataset
          if (hasData(data.byMonth)) {
            const labels = data.byMonth.map(r => r.month);
            const totals = data.byMonth.map(r => r.total);
            new Chart(document.getElementById('chartByMonth'), {
              type: 'bar',
              data: {
                labels,
                datasets: [{
                  label: 'Workouts',
                  data: totals,
                  backgroundColor: 'rgba(0, 200, 120, 0.5)',
                  borderColor: 'rgba(0, 255, 160, 0.8)',
                  borderWidth: 1
                }]
              },
              options: {
                scales: { y: { beginAtZero: true, ticks: { stepSize: 1 } } },
                plugins: { legend: { display: true } }
              }
            });
          }

          if (hasData(data.byType)) {
            const labels = data.byType.map(r => r.activity_type || 'Unknown');
            const totals = data.byType.map(r => r.total);
            new Chart(document.getElementById('chartByType'), {
              type: 'doughnut',
              data: {
                labels,
                datasets: [{
                  label: 'Workouts',
                  data: totals,
                  backgroundColor: [
                    'rgba(0, 200, 120, 0.6)',
                    'rgba(0, 150, 255, 0.6)',
                    'rgba(255, 200, 0, 0.6)',
                    'rgba(255, 99, 132, 0.6)',
                    'rgba(153, 102, 255, 0.6)'
                  ],
                  borderColor: 'rgba(0, 0, 0, 0.2)'
                }]
              },
              options: {
                plugins: { legend: { position: 'bottom' } }
              }
            });
```

## 5) Security-focused validation and sanitisation

The application uses express-validator and express-sanitizer to ensure data integrity and prevent XSS attacks. I also added messages for users to see what is the issue when they are completing their forms. This makes it easier for them to know how to successfully complete the form without trial and error.



```
// Handle registration
router.post(
  '/registered',
  [
    check('username')
      .notEmpty()
      .withMessage('Username is required'),
    check('first')
      .notEmpty()
      .withMessage('First name is required'),
    check('last')
      .notEmpty()
      .withMessage('Last name is required'),
    check('email')
      .isEmail()
      .withMessage('Please enter a valid email address'),
    check('password')
      .isLength({ min: 8 })
      .withMessage('Password must be at least 8 characters long')
      .matches(/[a-z]/)
      .withMessage('Password must contain a lowercase letter')
      .matches(/[A-Z]/)
      .withMessage('Password must contain an uppercase letter')
      .matches(/[0-9]/)
      .withMessage('Password must contain a number')
      .matches(/[^A-Za-z0-9]/)
      .withMessage('Password must contain a special character')
  ],
```



```
  const username = req.sanitize(req.body.username || '');
  const first = req.sanitize(req.body.first || '');
  const last = req.sanitize(req.body.last || '');
  const email = req.sanitize(req.body.email || '');
  const plainPassword = req.body.password || '';

  const saltRounds = 10;
```

These techniques demonstrate modular design, security awareness, and professional development practices.

## AI Declaration

AI tools like ChatGPT were utilized as a support resource throughout development. The assistance provided included:

- Clarifying concepts
- Debugging issues
- Enhancing the clarity of documentation

While AI played a key role in supporting various aspects of the development process, all final application logic, design decisions, and code were actively overseen, refined, and tested by me.