

Outline

This project is a compact Health & Fitness web application for recording personal achievements (runs, workouts, sleep), tracking menstrual cycles, scheduling medications, and providing fitness calculators (BMI, BMR/TDEE, heart-rate zones, macros, water). It supports per-user entries, searching and CSV export, a calendar view for periods, next-due medication suggestions, and lightweight audit logging and rate limiting. The app is implemented as a single Express server with EJS templates and a MySQL backend and is intended as a coursework prototype with clear separation between application and data tiers.

Architecture

Application tier: Node.js + Express serving EJS views and JSON endpoints (see `index.js`). Data tier: MySQL accessed via `mysql2/promise` connection pool (`src/db.js`). Sessions use `express-session` ; passwords hashed with `bcrypt` . Observability via `src/audit.js` .

Diagram (high level):

Browser <--HTTP--> Express (EJS, routes: `index.js`) <--SQL--> MySQL (pool from `src/db.js`)

Data Model

Core relational tables are `users` , `achievements` , `period_logs` , `medications` , and `audit_logs` (see `sql/create_db.sql`). Achievements reference users; `period_logs` and `medications` are per-user. The schema is intentionally narrow to keep operations simple and indexed for list/search.

Diagram (ER sketch):

users 1---* achievements users 1---* period_logs users 1---* medications users 1---* audit_logs

User Functionality

Users can register and sign in (session-based). Signed-in users may add achievements (title, category, metric, amount, notes), view a paginated list, filter and search entries, and export their results as CSV. The Period Tracker records cycle start dates and shows a month calendar with an estimated next window. The Medication Tracker accepts interval/daily/weekly schedules and computes next-due times for display. Fitness tools calculate BMI, BMR/TDEE, heart-rate zones, macros and daily water recommendations.

Screenshots (place these files under `screenshots/`):

- `screenshots/login.png` — login form and error hints.
- `screenshots/achievements_list.png` — paginated achievements, search bar, CSV export button.
- `screenshots/period_calendar.png` — month view with logged starts and highlighted next-window.
- `screenshots/meds_list.png` — medication list showing next due timestamps.
- `screenshots/tools_bmi.png` — BMI form and result card.

Each screenshot should show the form or list and a visible example row/result. Together these cover the main user journeys: register → add achievement → view/export; add period → inspect calendar; add medication → check next due; use tools.

Advanced Techniques

This section highlights selected technical choices and code references used to demonstrate development skills.

1. Database pool (file: `src/db.js`)

```
const mysql = require('mysql2/promise');
const pool = mysql.createPool({
  host: process.env.DB_HOST || process.env.HEALTH_HOST || 'localhost',
  user: process.env.DB_USER || process.env.HEALTH_USER || 'health_app',
  password: process.env.DB_PASSWORD || process.env.HEALTH_PASSWORD || 'qwerty',
  database: process.env.DB_NAME || process.env.HEALTH_DATABASE || 'health',
});
module.exports = pool;
```

2. Input sanitization (file: `src/sanitize.js`) — server-side sanitizer to reduce stored XSS:

```
function sanitizeText(input, { maxLength = 500 } = {}) {
  if (typeof input !== 'string') return '';
  let s = input.trim();
  s = s.replace(/<\s*script[>]*[\s\S]*?<\s*\/\s*script\s*>/gi, '');
  s = s.replace(/on[a-z]+\s*=\s*"[\"]*"\/gi, '');
  s = s.replace(/[<>]/g, '');
  if (s.length > maxLength) s = s.slice(0, maxLength);
  return s;
}
module.exports = { sanitizeText };
```

3. Rate limiting and audit logging (file: `index.js` , `src/audit.js`) — a compact per-IP+route limiter plus audit hooks:

```
const rateLimits = new Map();
function rateLimit({ windowMs=60000, max=10 }) {
  return (req,res,next) => { /* increment Map by `${req.ip}:${req.path}` and block if
exceeded */ };
}
// audit.log(req,'event', meta) used throughout routes
```

4. Example server-side search (file: `index.js`) — parameterized LIKE queries using `mysql2/promise` to prevent injection:

```
const like = `%${q}%`;
const [rows] = await db.query(
  'SELECT id,title,category,metric,amount,notes,created_at FROM achievements WHERE title
LIKE ? OR category LIKE ? OR notes LIKE ? ORDER BY created_at DESC',
  [like, like, like]
);
```

These techniques are intentionally simple and auditable for a coursework project while addressing common web-app concerns (input validation, DB pooling, auditability, and basic throttling).

AI Declaration

I used an AI assistant to help reword my readme and documentation. I also used an AI assistant to help with reasearching how to fix my code when trying to upload to the VM as the base url kept removing the original URL when uploading to the VM.