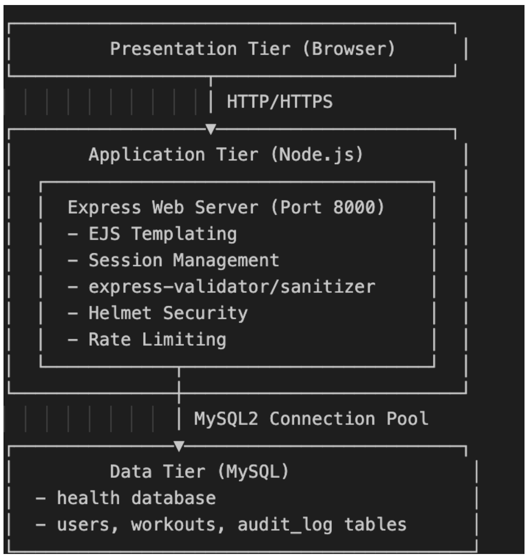
Health Fitness Tracker - Report Lab 10

Outline

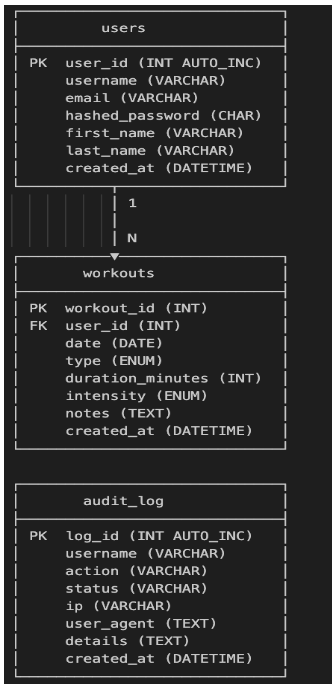
This application is a web-based health and fitness tracker that allows users to log their workout activities, search and filter workout history, and view personalized workout lists. It is built with Node.js, Express, EJS, and MySQL, providing both a user-friendly web interface and a JSON API for programmatic access. Users can register accounts, securely log in, add workouts with details like type, duration, and intensity, and search across all workouts in the database with advanced filtering and sorting options.

Architecture


Presentation Tier (Browser)
HTTP/HTTPS
Application Tier (Node.js)
Express Web Server (Port 8000)
- EJS Templating
- Session Management
- express-validator/sanitizer
- Helmet Security
- Rate Limiting
MySQL2 Connection Pool
Data Tier (MySQL)
- health database
- users, workouts, audit_log tables

The application uses a three-tier architecture. The presentation tier is rendered server-side using EJS templates. The application tier runs on Node.js with Express, handling routing, authentication, input validation/sanitization, and security middleware (Helmet, rate limiting). The data tier is a MySQL database accessed via connection pooling, storing user accounts, workout records, and audit logs with referential integrity constraints. The application also uses API-Ninjas for data enrichment.

Data Model

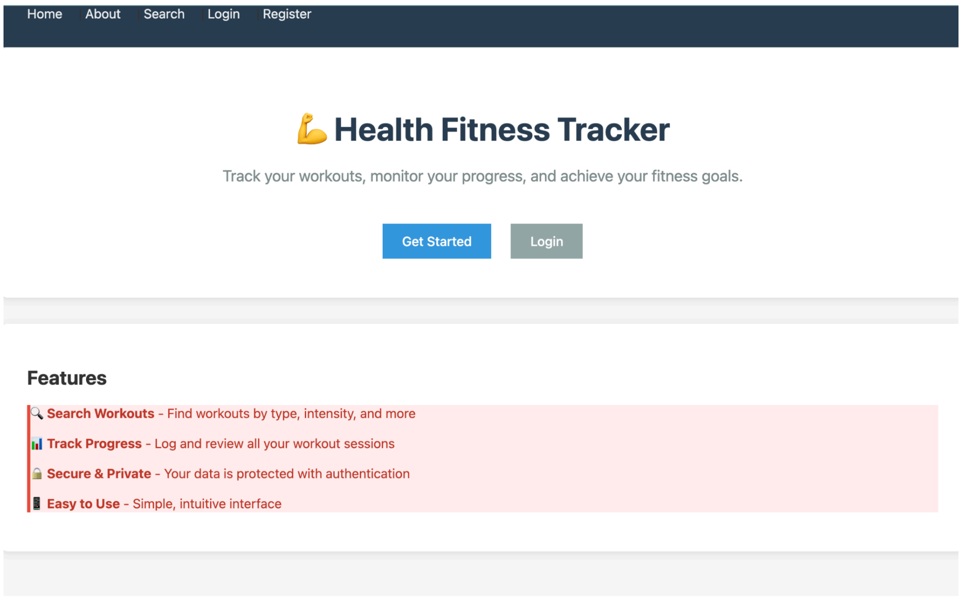


```
┌──────────────────────────────────┐
│             users                │
├──────────────────────────────────┤
│ PK  user_id (INT AUTO_INC)       │
│     username (VARCHAR)           │
│     email (VARCHAR)              │
│     hashed_password (CHAR)       │
│     first_name (VARCHAR)         │
│     last_name (VARCHAR)          │
│     created_at (DATETIME)        │
└──────────────────────────────────┘
             │ 1
             │
             │ N
┌────────────▼─────────────────────┐
│           workouts               │
├──────────────────────────────────┤
│ PK  workout_id (INT)             │
│ FK  user_id (INT)                │
│     date (DATE)                  │
│     type (ENUM)                  │
│     duration_minutes (INT)       │
│     intensity (ENUM)             │
│     notes (TEXT)                 │
│     created_at (DATETIME)        │
└──────────────────────────────────┘

┌──────────────────────────────────┐
│           audit_log              │
├──────────────────────────────────┤
│ PK  log_id (INT AUTO_INC)        │
│     username (VARCHAR)           │
│     action (VARCHAR)             │
│     status (VARCHAR)             │
│     ip (VARCHAR)                 │
│     user_agent (TEXT)            │
│     details (TEXT)               │
│     created_at (DATETIME)        │
└──────────────────────────────────┘
```

Description: The data model consists of three tables. The users table stores account credentials with bcrypt-hashed passwords. The workouts table has a many-to-one relationship with users (each workout belongs to one user), storing workout metadata with ENUM types for type (cardio, strength, etc.) and the different intensity (low, medium, high). The audit_log records authentication events independently for security monitoring, tracking login attempts with IP addresses and user agents.

User Functionality

1. Home Page
The landing page displays a section with the app title, tagline, and context-aware call-to-action buttons. Unauthenticated users see "Get Started" and "Login" buttons, while authenticated users see "Add Workout" and "View My Workouts". A features section highlights key capabilities: search, progress tracking, security, and ease of use.



💪 Health Fitness Tracker

Track your workouts, monitor your progress, and achieve your fitness goals.

Features

🔍 **Search Workouts** - Find workouts by type, intensity, and more

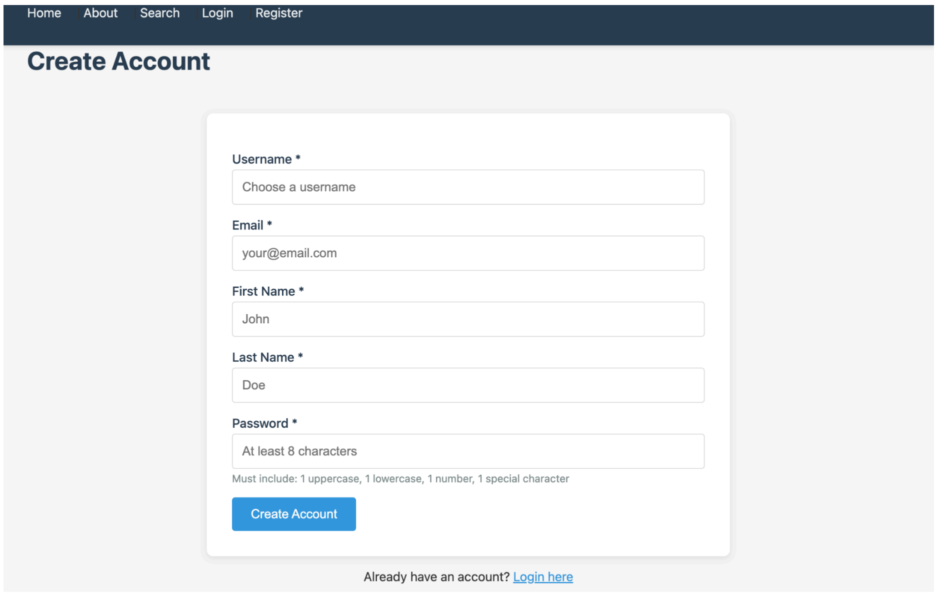
📊 **Track Progress** - Log and review all your workout sessions

🔒 **Secure & Private** - Your data is protected with authentication

📱 **Easy to Use** - Simple, intuitive interface

2. User Registration & Login

Users can create accounts via the registration form, which validates username (min 3 chars), email format, first/last names, and enforces strong password requirements (8+ chars, uppercase, lowercase, number, special char). The form preserves input values and displays specific validation errors. Login accepts username/password, compares against bcrypt hashes, creates sessions, and logs audit records of all attempts (success/failure).



Create Account

Username *
Choose a username

Email *
your@email.com

First Name *
John

Last Name *
Doe

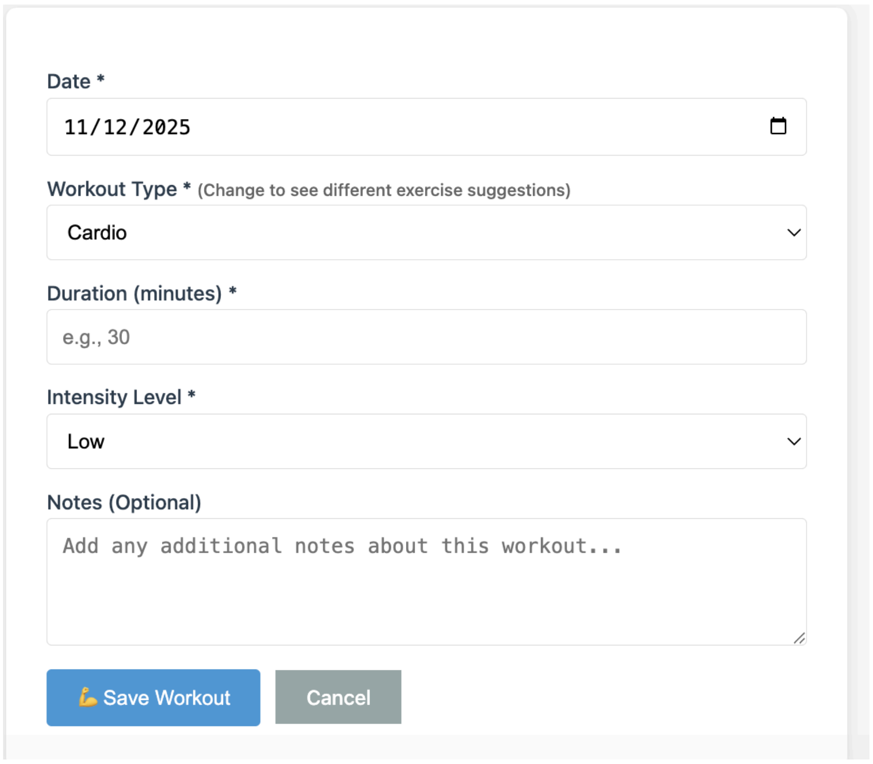
Password *
At least 8 characters

Must include: 1 uppercase, 1 lowercase, 1 number, 1 special character

Already have an account? Login here

3. Add Workout

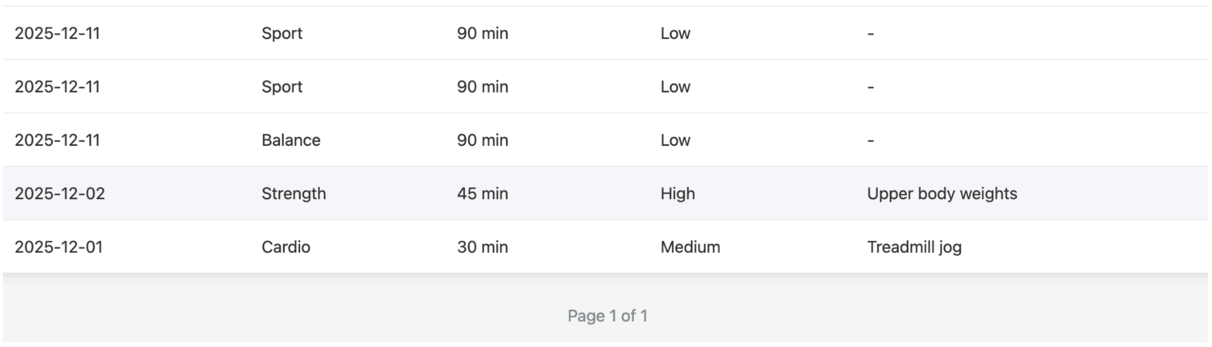
Authenticated users access a form to log workouts. Fields include date (defaults to today), type (dropdown: cardio, strength, flexibility, balance, sport, other), duration in minutes, intensity level (low/medium/high), and optional notes. The form validates all required fields and displays errors if validation fails. On success, the workout is saved under the logged-in user's account and redirects to "My Workouts".



Date *	11/12/2025
Workout Type * (Change to see different exercise suggestions)	Cardio
Duration (minutes) *	e.g., 30
Intensity Level *	Low
Notes (Optional)	Add any additional notes about this workout...

4. My Workouts (List)

This protected page displays the logged-in user's workout history in a table (20 per page) with columns for date, type, duration, intensity, and notes. Workouts are sorted by date (newest first). Pagination controls appear at the bottom with "Previous" and "Next" links. If no workouts exist, an empty state message encourages users to add their first workout.

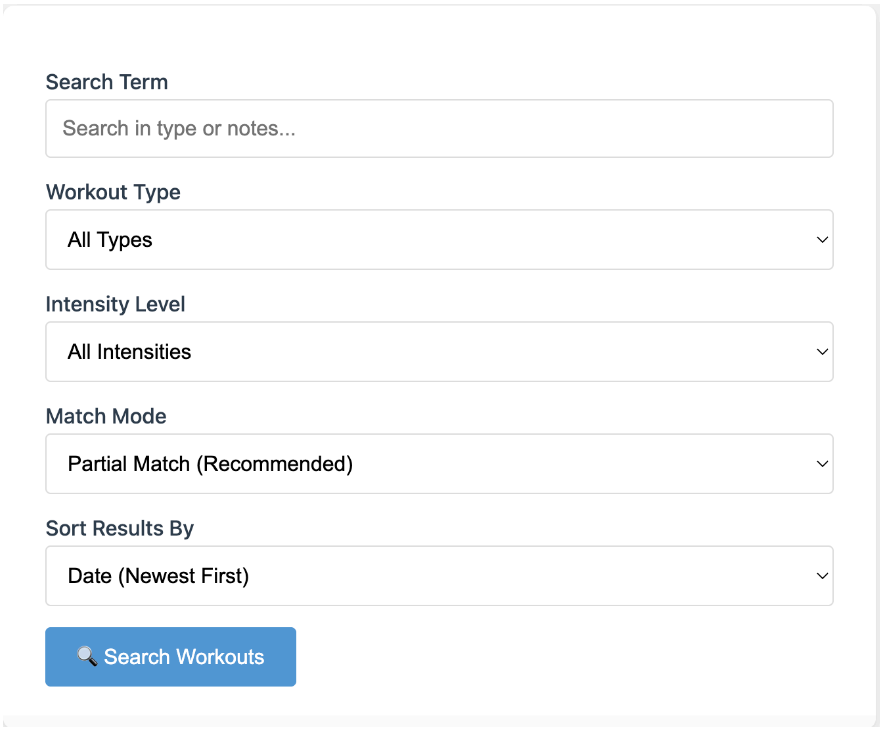


2025-12-11	Sport	90 min	Low	-
2025-12-11	Sport	90 min	Low	-
2025-12-11	Balance	90 min	Low	-
2025-12-02	Strength	45 min	High	Upper body weights
2025-12-01	Cardio	30 min	Medium	Treadmill jog

Page 1 of 1

5. Search Workouts

A public search page allows filtering across all workouts in the database. Users can enter search terms (matches type or notes), filter by workout type and intensity, choose exact or partial matching, and sort results by date, duration, or intensity.


Search Term

Search in type or notes...

Workout Type

All Types

Intensity Level

All Intensities

Match Mode

Partial Match (Recommended)

Sort Results By

Date (Newest First)

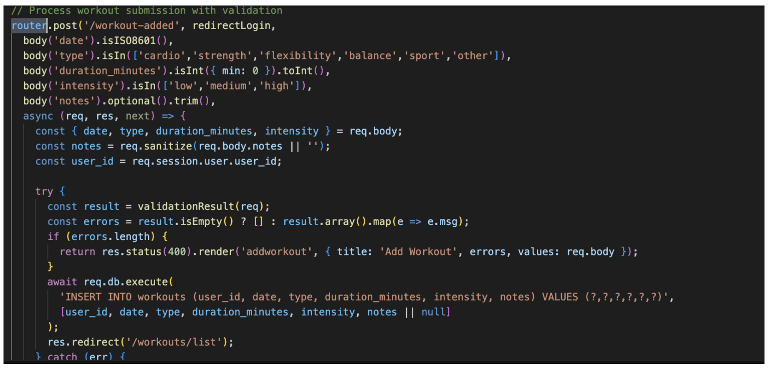
6. Navigation

A persistent navigation bar appears on all pages with links to Home, About, and Search. When logged in, it additionally shows Add Workout, My Workouts, a greeting with the username, and a Logout link. When logged out, it displays Login and Register links instead.

Advanced Techniques

1. Comprehensive Input Validation & Sanitization

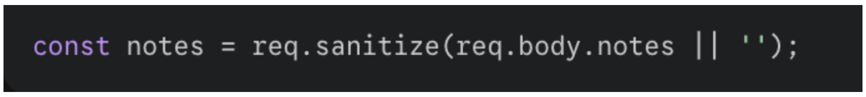
The application uses express-validator for declarative validation chains on all form submissions. For example, workout creation validates:



```
// Process workout submission with validation
router.post('/workout-added', redirectLogin,
  body('date').isISO8601(),
  body('type').isIn(['cardio','strength','flexibility','balance','sport','other']),
  body('duration_minutes').isInt({ min: 0 }).toInt(),
  body('intensity').isIn(['low','medium','high']),
  body('notes').optional().trim(),
  async (req, res, next) => {
    const { date, type, duration_minutes, intensity } = req.body;
    const notes = req.sanitize(req.body.notes || '');
    const user_id = req.session.user.user_id;

    try {
      const result = validationResult(req);
      const errors = result.isEmpty() ? [] : result.array().map(e => e.msg);
      if (errors.length) {
        return res.status(400).render('addworkout', { title: 'Add Workout', errors, values: req.body });
      }
      await req.db.execute(
        'INSERT INTO workouts (user_id, date, type, duration_minutes, intensity, notes) VALUES (?,?,?,?,?,?)',
        [user_id, date, type, duration_minutes, intensity, notes || null]
      );
      res.redirect('/workouts/list');
    } catch (err) {
```

File: routes/workouts.js

Additionally, express-sanitizer is applied globally to prevent XSS attacks:

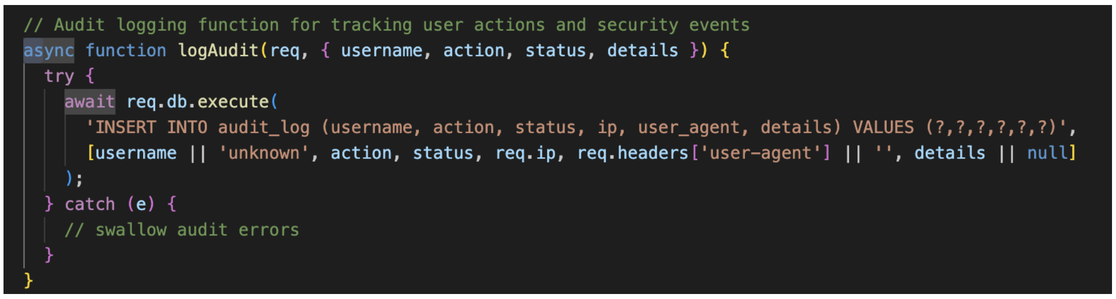


```
const notes = req.sanitize(req.body.notes || '');
```

This ensures user-supplied content cannot execute malicious scripts.

2. Security Audit Logging

All login attempts (successful and failed) are logged to an audit_log table with contextual information:

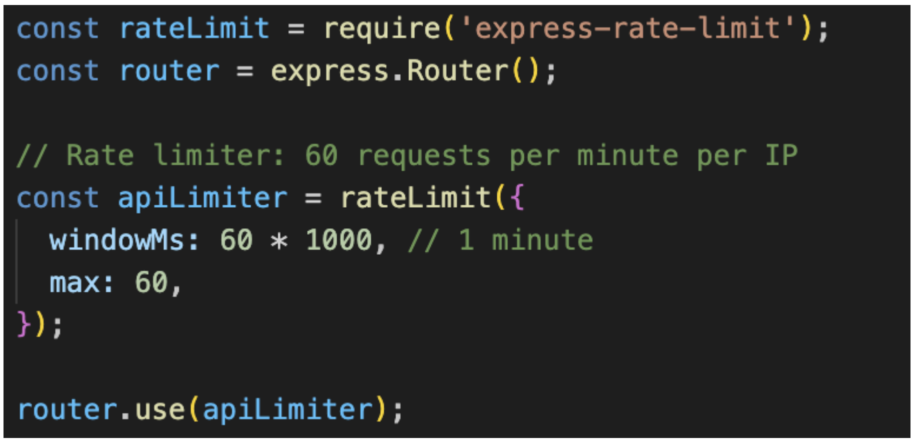


```
// Audit logging function for tracking user actions and security events
async function logAudit(req, { username, action, status, details }) {
  try {
    await req.db.execute(
      'INSERT INTO audit_log (username, action, status, ip, user_agent, details) VALUES (?,?,?,?,?,?)',
      [username || 'unknown', action, status, req.ip, req.headers['user-agent'] || '', details || null]
    );
  } catch (e) {
    // swallow audit errors
  }
}
```

This provides a tamper-evident record for security monitoring and forensic analysis.

3. API Rate Limiting

The public JSON API endpoint implements rate limiting to prevent abuse:

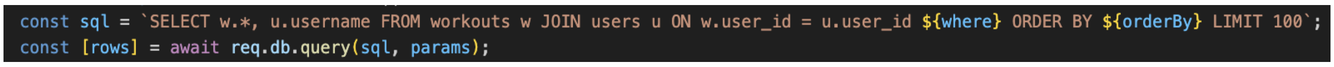


```
const rateLimit = require('express-rate-limit');
const router = express.Router();

// Rate limiter: 60 requests per minute per IP
const apiLimiter = rateLimit({
  windowMs: 60 * 1000, // 1 minute
  max: 60,
});

router.use(apiLimiter);
```

File: routes/api.js

This protects against denial-of-service attacks and excessive scraping while allowing legitimate usage.

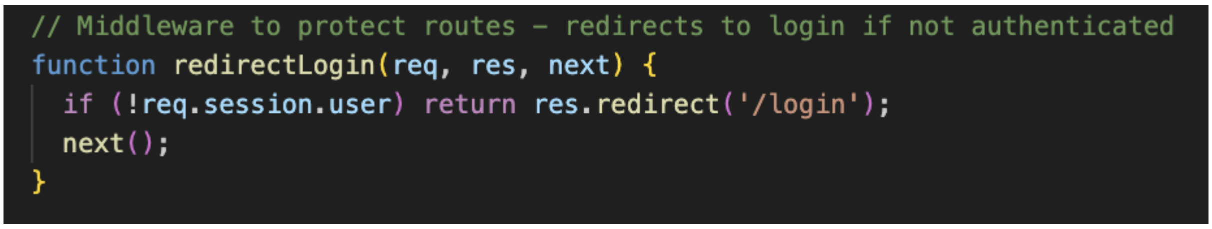
4. SQL Injection Prevention with Parameterized Queries
All database queries use parameterized statements via mysql2/promise, never concatenating user input:



```
const sql = `SELECT w.*, u.username FROM workouts w JOIN users u ON w.user_id = u.user_id ${where} ORDER BY ${orderBy} LIMIT 100`;
const [rows] = await req.db.query(sql, params);
```

The where clause is built programmatically, but all user values are passed separately in the params array, ensuring they're properly escaped by the MySQL driver.

5. Session-Based Authentication with Middleware Guards
Authentication is implemented using express-session with bcrypt password hashing. Protected routes use a reusable middleware:

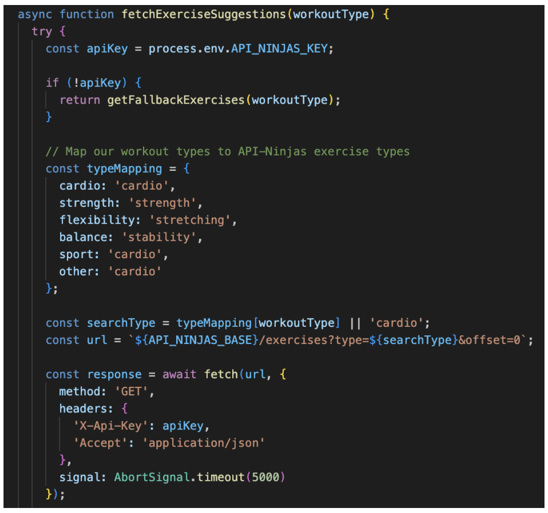


```
// Middleware to protect routes - redirects to login if not authenticated
function redirectLogin(req, res, next) {
  if (!req.session.user) return res.redirect('/login');
  next();
}
```

File: routes/workouts.js

This follows the DRY principle and ensures consistent authorization checks across all protected endpoints.

6. External API Integration with Graceful Degradation
The application integrates with the API-Ninjas Exercise Database to provide exercise suggestions when users add workouts, demonstrating the ability to consume third-party RESTful services:



```
async function fetchExerciseSuggestions(workoutType) {
  try {
    const apiKey = process.env.API_NINJAS_KEY;

    if (!apiKey) {
      return getFallbackExercises(workoutType);
    }

    // Map our workout types to API-Ninjas exercise types
    const typeMapping = {
      cardio: 'cardio',
      strength: 'strength',
      flexibility: 'stretching',
      balance: 'stability',
      sport: 'cardio',
      other: 'cardio'
    };

    const searchType = typeMapping[workoutType] || 'cardio';
    const url = `${API_NINJAS_BASE}/exercises?type=${searchType}&offset=0`;

    const response = await fetch(url, {
      method: 'GET',
      headers: {
        'X-Api-Key': apiKey,
        'Accept': 'application/json'
      },
      signal: AbortSignal.timeout(5000)
    });
```

File: services/exerciseService.js

Key advanced concepts demonstrated:

Asynchronous programming: Uses modern async/await with external HTTP requests.

Error handling: Try-catch blocks with fallback behavior (getFallbackExercises) ensure the app works even if the API is down or the key is missing.

Timeout handling: AbortSignal.timeout(5000) prevents hanging requests that could block the UI.

Data transformation: Maps and sanitizes the API response to fit the application's needs.

The integration is visible when users visit the "Add Workout" page, and they are presented with relevant exercise suggestions based on the selected workout type in an info box.

AI Declaration

I declare that I utilized AI tools to assist with the research and development of this project. The AI's role was strictly limited to providing me support and clarification; it did not write the core application code. My usage was as follows:

Clarification of Requirements: I used AI to help interpret specific constraints within the assignment instructions, ensuring my implementation aligned with the "Advanced Techniques" criteria.

Debugging & Troubleshooting: When encountering specific error messages (e.g., MySQL connection refusals or EJS rendering errors), I pasted logs into the AI to understand the root cause, which I then fixed myself.

Research & Syntax Reference: I used the AI as a dynamic documentation tool to look up specific syntax for library integrations (such as the fetch API options and helmet middleware configuration) and to understand best practices for secure password storage.