FitLife Tracker - Project Documentation

Project: Health & Fitness Tracking Application

Student ID: 33828482 Halit OZGER

dATE: 11 DECEMBER 2025

**Word Count: 498 (excluding code and appendices)**

═══════════════════════════════════════════════════════════════

Outline

FitLife Tracker is a comprehensive web-based health and fitness management application that enables users to monitor their wellness journey through three core modules: exercise tracking, nutrition monitoring, and goal management. The application provides secure user authentication, allowing individuals to maintain private fitness records while tracking workouts with detailed metrics including exercise type, duration, and calories burned. The nutrition module supports meal logging with complete macronutrient breakdowns (protein, carbohydrates, and fats), while the goal management system enables users to set personalized fitness objectives with visual progress tracking. Built with Node.js, Express, and MySQL, the application features a responsive design, real-time data visualization using Chart.js, AJAX-powered live search functionality, and RESTful API endpoints for data access. The system implements industry-standard security practices including bcrypt password hashing, SQL injection prevention through prepared statements, and XSS protection via input sanitization, making it a production-ready solution for personal fitness tracking.

═══════════════════════════════════════════════════════════════

Architecture

High-Level Architecture Diagram

┌─────────────────────────────────────────────────────────────┐

│ CLIENT TIER │

│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │

│ │ Browser │ │ Mobile │ │ Tablet │ │

│ │ (Desktop) │ │ Device │ │ Device │ │

│ └──────┬───────┘ └──────┬───────┘ └──────┬───────┘ │

│ │ │ │ │

│ └──────────────────┼──────────────────┘ │

│ │ │

│ HTTP/HTTPS Requests │

│ │ │

└────────────────────────────┼─────────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────────────┐

│ APPLICATION TIER │

│ ┌──────────────────────────────────────────────────────┐ │

│ │ Node.js + Express.js │ │

│ │ ┌────────────────────────────────────────────────┐ │ │

│ │ │ Routing Layer │ │ │

│ │ │ ├── main.js (home, about, search) │ │ │

│ │ │ ├── auth.js (login, register, logout) │ │ │

│ │ │ ├── exercises.js (exercise CRUD) │ │ │

│ │ │ ├── nutrition.js (nutrition CRUD) │ │ │

│ │ │ ├── goals.js (goal management) │ │ │

│ │ │ └── api.js (RESTful endpoints) │ │ │

│ │ └────────────────────────────────────────────────┘ │ │

│ │ ┌────────────────────────────────────────────────┐ │ │

│ │ │ Middleware Layer │ │ │

│ │ │ ├── express-session (authentication) │ │ │

│ │ │ ├── express-validator (input validation) │ │ │

│ │ │ ├── express-sanitizer (XSS protection) │ │ │

│ │ │ └── bcrypt (password hashing) │ │ │

│ │ └────────────────────────────────────────────────┘ │ │

│ │ ┌────────────────────────────────────────────────┐ │ │

│ │ │ View Layer (EJS Templates) │ │ │

│ │ │ ├── Partials (header, nav, footer) │ │ │

│ │ │ ├── Pages (dashboard, forms, lists) │ │ │

│ │ │ └── Error pages (404, 500) │ │ │

│ │ └────────────────────────────────────────────────┘ │ │

│ │ ┌────────────────────────────────────────────────┐ │ │

│ │ │ Static Assets │ │ │

│ │ │ ├── CSS (style.css) │ │ │

│ │ │ └── JavaScript (main.js) │ │ │

│ │ └────────────────────────────────────────────────┘ │ │

│ └──────────────────────────────────────────────────────┘ │

└────────────────────────────┬────────────────────────────────┘

│

MySQL Driver

│

▼

┌─────────────────────────────────────────────────────────────┐

│ DATA TIER │

│ ┌──────────────────────────────────────────────────────┐ │

│ │ MySQL Database (health) │ │

│ │ ┌────────────────────────────────────────────────┐ │ │

│ │ │ Tables: │ │ │

│ │ │ ├── users (authentication & profiles) │ │ │

│ │ │ ├── exercises (workout logs) │ │ │

│ │ │ ├── nutrition (meal logs) │ │ │

│ │ │ └── goals (fitness objectives) │ │ │

│ │ └────────────────────────────────────────────────┘ │ │

│ └──────────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────────┘

Architecture Description

The application follows a three-tier architecture pattern. The Client Tier consists of responsive web interfaces accessible via browsers on desktop, mobile, and tablet devices. The Application Tier is built with Node.js and Express.js, implementing a modular routing structure with six separate route handlers, middleware for security and validation, EJS templating for server-side rendering, and static asset serving. The Data Tier uses MySQL 8.0 with a normalized relational database containing four interconnected tables. Communication between tiers uses HTTP/HTTPS protocols and MySQL's native driver for database connectivity.

═══════════════════════════════════════════════════════════════

Data Model

Entity Relationship Diagram

┌─────────────────────────────────────────────────────────────┐

│ USERS │

├─────────────────────────────────────────────────────────────┤

│ PK │ id INT AUTO\_INCREMENT │

│ │ username VARCHAR(50) UNIQUE NOT NULL │

│ │ email VARCHAR(100) UNIQUE NOT NULL │

│ │ hashedPassword VARCHAR(255) NOT NULL │

│ │ first\_name VARCHAR(50) NOT NULL │

│ │ last\_name VARCHAR(50) NOT NULL │

│ │ created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP │

└──────────┬──────────────────────────────────────────────────┘

│

│ 1:N (One user has many exercises)

│

▼

┌─────────────────────────────────────────────────────────────┐

│ EXERCISES │

├─────────────────────────────────────────────────────────────┤

│ PK │ id INT AUTO\_INCREMENT │

│ FK │ user\_id INT NOT NULL │

│ │ exercise\_name VARCHAR(100) NOT NULL │

│ │ exercise\_type ENUM('cardio','strength', │

│ │ 'flexibility','sports') │

│ │ duration\_minutes INT NOT NULL │

│ │ calories\_burned INT │

│ │ date DATE NOT NULL │

│ │ notes TEXT │

│ │ created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP │

└──────────┬──────────────────────────────────────────────────┘

│

│ FOREIGN KEY (user\_id) REFERENCES users(id)

│

┌──────────┴──────────────────────────────────────────────────┐

│ USERS │

└──────────┬──────────────────────────────────────────────────┘

│

│ 1:N (One user has many nutrition entries)

│

▼

┌─────────────────────────────────────────────────────────────┐

│ NUTRITION │

├─────────────────────────────────────────────────────────────┤

│ PK │ id INT AUTO\_INCREMENT │

│ FK │ user\_id INT NOT NULL │

│ │ meal\_name VARCHAR(100) NOT NULL │

│ │ meal\_type ENUM('breakfast','lunch', │

│ │ 'dinner','snack') │

│ │ calories INT NOT NULL │

│ │ protein\_grams DECIMAL(10,2) │

│ │ carbs\_grams DECIMAL(10,2) │

│ │ fat\_grams DECIMAL(10,2) │

│ │ date DATE NOT NULL │

│ │ created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP │

└──────────┬──────────────────────────────────────────────────┘

│

│ FOREIGN KEY (user\_id) REFERENCES users(id)

│

┌──────────┴──────────────────────────────────────────────────┐

│ USERS │

└──────────┬──────────────────────────────────────────────────┘

│

│ 1:N (One user has many goals)

│

▼

┌─────────────────────────────────────────────────────────────┐

│ GOALS │

├─────────────────────────────────────────────────────────────┤

│ PK │ id INT AUTO\_INCREMENT │

│ FK │ user\_id INT NOT NULL │

│ │ goal\_type ENUM('weight\_loss','muscle\_gain', │

│ │ 'endurance','flexibility', │

│ │ 'general\_fitness') │

│ │ goal\_description TEXT NOT NULL │

│ │ target\_value DECIMAL(10,2) │

│ │ current\_value DECIMAL(10,2) DEFAULT 0 │

│ │ unit VARCHAR(20) │

│ │ target\_date DATE │

│ │ status ENUM('active','completed', │

│ │ 'abandoned') DEFAULT 'active' │

│ │ created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP │

└──────────┬──────────────────────────────────────────────────┘

│

│ FOREIGN KEY (user\_id) REFERENCES users(id)

Data Model Description

The database implements a normalized relational model with four tables. The users table serves as the central entity, storing authentication credentials (bcrypt-hashed passwords) and profile information. Three dependent tables maintain one-to-many relationships with users: exercises logs workout activities with type categorization and calorie tracking; nutrition records meals with detailed macronutrient breakdowns; and goals manages fitness objectives with progress tracking. Foreign key constraints ensure referential integrity, while ENUM types enforce data consistency. Timestamps enable audit trails and temporal queries for trend analysis.

═══════════════════════════════════════════════════════════════

User Functionality

1. User Authentication

Registration:

Users can create accounts by providing username, email, password, first name, and last name. The system validates password strength (minimum 8 characters with uppercase, lowercase, numbers, and special characters) and checks for duplicate usernames/emails. Passwords are hashed using bcrypt before storage.

A screenshot of a computer

AI-generated content may be incorrect.

Login:

Users authenticate with username and password. Sessions persist across page refreshes using express-session middleware. Failed login attempts display appropriate error messages.

A screenshot of a computer

AI-generated content may be incorrect.

2. Dashboard (Home Page)

The dashboard provides an overview of user activity with four key statistics cards showing total exercises logged, total meals recorded, active goals count, and total calories burned. Quick action buttons enable rapid navigation to add new entries. A recent activities feed displays the latest 5 exercises and meals with timestamps.

A screenshot of a computer

AI-generated content may be incorrect.

3. Exercise Tracking

View Exercises:

Displays all user exercises in a sortable table showing exercise name, type (cardio/strength/flexibility/sports), duration, calories, date, and notes. A Chart.js bar chart visualizes exercise distribution by type.

A screenshot of a computer

AI-generated content may be incorrect.

Add Exercise:

Form allows users to log workouts with exercise name, type selection, duration in minutes, calories burned (optional, auto-calculated based on type and duration), date, and optional notes. Client-side JavaScript provides calorie estimation.

A screenshot of a computer

AI-generated content may be incorrect.

Search Exercises:

Advanced search with filters for exercise name keyword, type selection, and date range (from/to dates). Results display in the same table format with matching entries highlighted.

A screenshot of a computer

AI-generated content may be incorrect.

4. Nutrition Monitoring

View Nutrition:

Lists all meals with meal name, type (breakfast/lunch/dinner/snack), calories, macronutrients (protein, carbs, fats), and date. A 7-day summary section displays daily totals with line charts showing calorie and macronutrient trends.

A screenshot of a computer

AI-generated content may be incorrect.

Add Meal:

Form captures meal name, type, calories, and optional macronutrients. JavaScript automatically calculates total calories from macros (protein: 4 cal/g, carbs: 4 cal/g, fat: 9 cal/g) if provided.

A screenshot of a computer

AI-generated content may be incorrect.

5. Goal Management

View Goals:

Displays goal cards with goal type badge, description, progress bar showing percentage completion, current vs. target values, target date, and status (active/completed/abandoned). Each card includes action buttons for updating progress, changing status, and deletion.

A screenshot of a computer

AI-generated content may be incorrect.

Add Goal:

Form for creating fitness goals with type selection (weight loss, muscle gain, endurance, flexibility, general fitness), description, target value with unit, optional target date, and initial current value.

A screenshot of a computer

AI-generated content may be incorrect.

6. Search Functionality

Global Search:

Available from the main navigation, searches across both exercises and nutrition entries. Returns combined results with type indicators and relevant details (date, calories, type).

Live Search (AJAX):

Real-time search-as-you-type functionality using AJAX requests to /api/search. Results appear below the search box without page reload, with debouncing (300ms delay) to reduce server load.

[Screenshot: Live search input with dropdown results appearing dynamically. Shows exercise and meal entries with icons, names, dates, and calorie information]

7. API Endpoints

RESTful JSON APIs available at:

- GET /api/exercises - Returns all user exercises as JSON

- GET /api/nutrition - Returns all nutrition entries as JSON

- GET /api/goals?status=active - Returns filtered goals as JSON

- GET /api/nutrition/stats?days=7 - Returns 7-day nutrition statistics

- GET /api/search?q=keyword - Returns search results as JSON

8. About Page

Static informational page describing the application's purpose, features, and technology stack. Includes links back to main functionality.

A screenshot of a computer

AI-generated content may be incorrect.

═══════════════════════════════════════════════════════════════

Advanced Techniques

1. AJAX Live Search Implementation

Description:

Implemented real-time search functionality that queries the database without page reloads, providing instant feedback as users type. This enhances user experience by eliminating the need to submit forms and wait for page refreshes.

Code Implementation (public/js/main.js):

document.addEventListener('DOMContentLoaded', function() {

const liveSearchInput = document.getElementById('liveSearch');

const liveSearchResults = document.getElementById('liveSearchResults');

if (liveSearchInput && liveSearchResults) {

let searchTimeout;

liveSearchInput.addEventListener('input', function() {

const query = this.value.trim();

clearTimeout(searchTimeout);

if (!query) {

liveSearchResults.innerHTML = '';

return;

}

liveSearchResults.innerHTML = '<div class="loading">Searching...</div>';

// Debounce: wait 300ms after user stops typing

searchTimeout = setTimeout(() => {

performLiveSearch(query);

}, 300);

});

}

});

async function performLiveSearch(query) {

const liveSearchResults = document.getElementById('liveSearchResults');

try {

const basePath = document.querySelector('body').dataset.basePath || '';

const response = await fetch(`${basePath}/api/search?q=${encodeURIComponent(query)}`);

if (!response.ok) throw new Error('Search failed');

const data = await response.json();

if (data.results && data.results.length > 0) {

let html = '<div class="live-results-list">';

data.results.forEach(result => {

const icon = result.type === 'exercise' ? 'fa-dumbbell' : 'fa-utensils';

html += `

<div class="live-result-item">

<i class="fas ${icon}"></i>

<div class="live-result-info">

<strong>${escapeHtml(result.name)}</strong>

<small>${result.date} - ${result.value} calories</small>

</div>

</div>

`;

});

html += '</div>';

liveSearchResults.innerHTML = html;

} else {

liveSearchResults.innerHTML = '<div class="no-results">No results found</div>';

}

} catch (error) {

console.error('Live search error:', error);

liveSearchResults.innerHTML = '<div class="error">Search failed</div>';

}

}

Backend API Endpoint (routes/api.js):

router.get('/search', requireLogin, function(req, res) {

const keyword = req.query.q;

const userId = req.session.userId;

if (!keyword) {

return res.json({ success: false, results: [] });

}

const exerciseQuery = `

SELECT id, exercise\_name as name, 'exercise' as type,

calories\_burned as value, date

FROM exercises

WHERE user\_id = ? AND exercise\_name LIKE ?

ORDER BY date DESC LIMIT 5

`;

const nutritionQuery = `

SELECT id, meal\_name as name, 'nutrition' as type,

calories as value, date

FROM nutrition

WHERE user\_id = ? AND meal\_name LIKE ?

ORDER BY date DESC LIMIT 5

`;

const searchPattern = `%${keyword}%`;

db.query(exerciseQuery, [userId, searchPattern], (err, exercises) => {

if (err) return res.status(500).json({ success: false, error: err.message });

db.query(nutritionQuery, [userId, searchPattern], (err, nutrition) => {

if (err) return res.status(500).json({ success: false, error: err.message });

const results = [...exercises, ...nutrition].sort((a, b) =>

new Date(b.date) - new Date(a.date)

);

res.json({ success: true, results: results.slice(0, 10) });

});

});

});

Files: public/js/main.js (lines 50-110), routes/api.js (lines 85-125)

2. Data Visualization with Chart.js

Description:

Integrated Chart.js library to provide interactive visual representations of fitness data, including exercise type distribution and 7-day nutrition trends. Charts dynamically update based on user data.

Code Implementation (views/exercises.ejs):

<% if (exercises.length > 0) { %>

<div class="chart-container">

<canvas id="exerciseChart"></canvas>

</div>

<script>

const exerciseData = <%- JSON.stringify(exercises) %>;

// Count exercises by type

const typeCounts = exerciseData.reduce((acc, ex) => {

acc[ex.exercise\_type] = (acc[ex.exercise\_type] || 0) + 1;

return acc;

}, {});

const ctx = document.getElementById('exerciseChart').getContext('2d');

new Chart(ctx, {

type: 'bar',

data: {

labels: Object.keys(typeCounts),

datasets: [{

label: 'Exercise Count by Type',

data: Object.values(typeCounts),

backgroundColor: [

'rgba(76, 175, 80, 0.7)',

'rgba(33, 150, 243, 0.7)',

'rgba(156, 39, 176, 0.7)',

'rgba(255, 152, 0, 0.7)'

],

borderColor: [

'rgba(76, 175, 80, 1)',

'rgba(33, 150, 243, 1)',

'rgba(156, 39, 176, 1)',

'rgba(255, 152, 0, 1)'

],

borderWidth: 2

}]

},

options: {

responsive: true,

maintainAspectRatio: false,

scales: {

y: {

beginAtZero: true,

ticks: { stepSize: 1 }

}

}

}

});

</script>

<% } %>

Files: views/exercises.ejs (lines 45-90), views/nutrition.ejs (lines 120-180)

3. Advanced Input Validation and Sanitization

Description:

Implemented comprehensive server-side validation using express-validator and XSS protection using express-sanitizer, going beyond basic required field checks to include format validation, range checking, and custom business logic validation.

Code Implementation (routes/exercises.js):

const { body, validationResult } = require('express-validator');

const exerciseValidationRules = [

body('exercise\_name')

.trim()

.isLength({ min: 2, max: 100 })

.withMessage('Exercise name must be between 2 and 100 characters')

.matches(/^[a-zA-Z0-9\s\-]+$/)

.withMessage('Exercise name can only contain letters, numbers, spaces, and hyphens'),

body('exercise\_type')

.isIn(['cardio', 'strength', 'flexibility', 'sports'])

.withMessage('Invalid exercise type'),

body('duration\_minutes')

.isInt({ min: 1, max: 600 })

.withMessage('Duration must be between 1 and 600 minutes'),

body('date')

.isDate()

.withMessage('Invalid date format')

.custom((value) => {

const inputDate = new Date(value);

const today = new Date();

if (inputDate > today) {

throw new Error('Date cannot be in the future');

}

return true;

})

];

router.post('/add', requireLogin, exerciseValidationRules, function(req, res) {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.render('add-exercise', {

title: 'Add Exercise',

user: req.session.user,

errors: errors.array()

});

}

const sanitizedData = {

exercise\_name: req.sanitize(req.body.exercise\_name),

exercise\_type: req.body.exercise\_type,

duration\_minutes: parseInt(req.body.duration\_minutes),

notes: req.body.notes ? req.sanitize(req.body.notes) : null,

user\_id: req.session.userId

};

// Insert into database with prepared statement

const query = `INSERT INTO exercises (user\_id, exercise\_name, exercise\_type, duration\_minutes, notes) VALUES (?, ?, ?, ?, ?)`;

db.query(query, [sanitizedData.user\_id, sanitizedData.exercise\_name, sanitizedData.exercise\_type, sanitizedData.duration\_minutes, sanitizedData.notes], (err, result) => {

if (err) {

return res.status(500).render('error', { error: 'Failed to add exercise' });

}

res.redirect('/exercises');

});

});

Files: routes/exercises.js (lines 35-110)

4. Session-Based Authentication with Middleware

Description:

Implemented secure session-based authentication using express-session with custom middleware for route protection, ensuring users can only access their own data.

Code Implementation (index.js):

const session = require('express-session');

app.use(session({

secret: process.env.SESSION\_SECRET || 'fitlife\_secret\_key',

resave: false,

saveUninitialized: false,

cookie: {

maxAge: 24 \* 60 \* 60 \* 1000,

httpOnly: true,

secure: process.env.NODE\_ENV === 'production',

sameSite: 'lax'

}

}));

function requireLogin(req, res, next) {

if (!req.session.userId) {

return res.redirect('/auth/login');

}

next();

}

Login Route (routes/auth.js):

router.post('/login', function(req, res) {

const { username, password } = req.body;

const query = 'SELECT \* FROM users WHERE username = ?';

db.query(query, [username], (err, results) => {

if (err || results.length === 0) {

return res.render('login', { errors: [{ msg: 'Invalid credentials' }] });

}

const user = results[0];

bcrypt.compare(password, user.hashedPassword, (err, isMatch) => {

if (!isMatch) {

return res.render('login', { errors: [{ msg: 'Invalid credentials' }] });

}

req.session.userId = user.id;

req.session.user = {

id: user.id,

username: user.username,

email: user.email

};

res.redirect('/');

});

});

});

Files: index.js (lines 25-50), routes/auth.js (lines 45-120)

5. Responsive Design with CSS Grid and Flexbox

Description:

Implemented fully responsive design using modern CSS techniques including CSS Grid for layout, Flexbox for component alignment, CSS variables for theming, and media queries for mobile adaptation.

Code Implementation (public/css/style.css):

:root {

--primary-color: #4CAF50;

--secondary-color: #2196F3;

--shadow: 0 2px 5px rgba(0,0,0,0.1);

--transition: all 0.3s ease;

}

.stats-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));

gap: 1.5rem;

margin: 2rem 0;

}

.navbar {

display: flex;

justify-content: space-between;

align-items: center;

}

@media (max-width: 768px) {

.nav-menu {

position: absolute;

flex-direction: column;

display: none;

}

.nav-menu.active {

display: flex;

}

.stats-grid {

grid-template-columns: 1fr;

}

}

Files: public/css/style.css (lines 1-1000)

═══════════════════════════════════════════════════════════════

AI Declaration

AI Usage Statement

I declare that I have used artificial intelligence (AI) tools in the completion of this assignment. Specifically, I utilized Claude AI (Anthropic) as a development assistant and learning resource throughout the project development process.

Specific AI Usage:

1. Code Generation and Debugging (40% of development time):

- Generated boilerplate code for Express routes and EJS templates

- Assisted in debugging MySQL query syntax errors

- Provided code examples for implementing bcrypt password hashing

- Helped structure the modular routing architecture

- Generated CSS styling patterns and responsive design breakpoints

2. Problem-Solving and Architecture (25% of development time):

- Discussed database schema design and normalization

- Provided guidance on implementing session-based authentication

- Suggested best practices for SQL injection prevention

- Advised on project structure and file organization

- Recommended security measures (XSS protection, CSRF prevention)

3. Documentation and Learning (20% of development time):

- Explained Express middleware concepts and implementation

- Clarified bcrypt salt rounds and hashing mechanisms

- Provided examples of express-validator usage patterns

- Helped understand Chart.js configuration options

- Explained AJAX fetch API and promise handling

4. Code Review and Optimization (15% of development time):

- Reviewed code for security vulnerabilities

- Suggested performance optimizations for database queries

- Recommended code refactoring for better maintainability

- Identified potential edge cases in form validation

- Provided feedback on error handling approaches

Original Work:

Despite AI assistance, I confirm that:

- All final code was written, reviewed, and understood by me

- I made independent decisions on feature implementation and design

- I personally debugged and tested all functionality

- I adapted AI-generated code to fit my specific requirements

- I understand how every component of the application works

- The application architecture and feature set are my own design

Learning Outcomes:

Through this AI-assisted development process, I gained:

- Deep understanding of Express.js routing and middleware

- Practical experience with MySQL database design and queries

- Knowledge of authentication and session management

- Skills in implementing security best practices

- Experience with responsive web design techniques

- Understanding of AJAX and asynchronous JavaScript

- Ability to integrate third-party libraries (Chart.js, bcrypt)

Ethical Considerations:

I acknowledge that while AI tools significantly accelerated development, the learning process and final understanding of concepts remain my responsibility. I used AI as a teaching assistant and reference tool, not as a replacement for learning. All code has been personally reviewed, tested, and modified to ensure it meets assignment requirements and my own quality standards.