# FITNESS TRACKER APPLICATION - TECHNICAL REPORT

Project: Fitness Tracker Web Application
Student ID: 33801956
Technology Stack: Node.js, Express.js, MySQL, EJS Templates
Deployment URL: http://www.doc.gold.ac.uk/usr/112/

## Table of Contents

# 1. PROJECT OVERVIEW

FitTracker is a full-stack web application designed to help users track their fitness journey. The application provides a comprehensive platform for fitness enthusiasts to monitor their progress, set goals, and maintain workout records.

Key Features:
- User registration and secure account management
- Workout logging with detailed exercise tracking
- Goal setting with progress monitoring and auto-completion
- Searchable exercise library with categories and difficulty levels
- Personal statistics dashboard showing workout history and achievements
- Responsive design for desktop and mobile access

The application follows the Model-View-Controller (MVC) architecture pattern, separating data logic, user interface, and control flow for maintainable and scalable code. Server-side rendering with EJS templates ensures fast page loads and SEO-friendly content.

# 2. APPLICATION URLs & NAVIGATION

The application is deployed at the Goldsmiths university server and accessible via the following URLs:

PUBLIC PAGES (No login required):

Home Page
URL: https://www.doc.gold.ac.uk/usr/112/
Description: The main landing page displaying platform statistics (total users, workouts, exercises) and a welcome message. For logged-in users, it also shows their recent workouts and personal statistics.

Login Page
URL: https://www.doc.gold.ac.uk/usr/112/auth/login
Description: User authentication page where existing users enter their username and password to access their account. Includes error handling for invalid credentials.

Register Page
URL: https://www.doc.gold.ac.uk/usr/112/auth/register
Description: New user registration form requiring username, email, password (with strength validation), and optional profile information. Passwords must meet security requirements.

Exercise Library
URL: https://www.doc.gold.ac.uk/usr/112/exercises
Description: Browse all available exercises in the database. Users can filter by category (Cardio, Strength, Flexibility, etc.) and view exercise details including muscle groups, difficulty level, and calories burned.

Search Page
URL: https://www.doc.gold.ac.uk/usr/112/search
Description: Global search functionality allowing users to find exercises by name, description, or muscle group. Logged-in users can also search their own workouts.

About Page
URL: https://www.doc.gold.ac.uk/usr/112/about
Description: Information about the FitTracker application, its features, and purpose.

PROTECTED PAGES (Login required):

My Workouts
URL: https://www.doc.gold.ac.uk/usr/112/workouts
Description: Displays a paginated list of all workouts logged by the current user, showing workout name, date, duration, calories burned, and exercise count.

Log New Workout
URL: https://www.doc.gold.ac.uk/usr/112/workouts/new
Description: Form to create a new workout entry. Users can specify workout name, date, duration, add multiple exercises with sets/reps/weight, and rate their session.

My Goals
URL: https://www.doc.gold.ac.uk/usr/112/goals
Description: View and manage fitness goals. Shows all goals with progress bars, status (active/completed/abandoned), and target dates. Goals can be filtered by status.

Create New Goal
URL: https://www.doc.gold.ac.uk/usr/112/goals/new
Description: Form to set a new fitness goal with title, type (weight loss, muscle gain, endurance, etc.), target value, unit of measurement, and deadline.

User Profile
URL: https://www.doc.gold.ac.uk/usr/112/auth/profile
Description: View and edit personal profile information including name, date of birth, height, weight, and activity level. Users can also change their password here.

# 3. SYSTEM ARCHITECTURE

Technology Stack:

| Component | Description |
|---|---|
| Runtime | Node.js - JavaScript runtime for server-side execution |
| Framework | Express.js v4.18.2 - Web application framework |
| Database | MySQL 5.7 - Relational database management system |
| Template Engine | EJS v3.1.9 - Embedded JavaScript templating |
| Authentication | bcrypt v5.1.1 - Password hashing library |
| Session Management | express-session v1.17.3 - Session middleware |
| Environment Config | dotenv v16.3.1 - Environment variable management |

Architecture Overview:

The application uses a three-tier architecture:

1. Presentation Layer: EJS templates render HTML pages with dynamic content. The header and footer partials provide consistent navigation across all pages.

2. Application Layer: Express.js handles HTTP requests, routing, middleware processing, and business logic. Routes are organised into modules (auth, workouts, goals, main, api).

3. Data Layer: MySQL database stores all persistent data. The mysql2 library provides connection pooling for efficient database access.

Request Flow:
Browser Request → Express Router → Middleware (auth check) → Route Handler → Database Query → EJS Template → HTML Response

Code Example - Application Entry Point (index.js):

```
const express = require("express");
const session = require("express-session");
const app = express();

// Session configuration
app.use(session({
  secret: process.env.SESSION_SECRET,
  resave: false,
  saveUninitialized: false,
  cookie: { maxAge: 24 * 60 * 60 * 1000 }  // 24 hours
}));

// Route mounting
app.use("/", mainRoutes);
app.use("/auth", authRoutes);
app.use("/workouts", workoutRoutes);
app.use("/goals", goalRoutes);
```

# 4. DATABASE DESIGN

The database uses a relational schema with seven tables and two views for aggregated data.

Entity Relationship Overview:

- Each USER can have one PROFILE (1:1 relationship)
- Each USER can have many WORKOUTS (1:N relationship)
- Each USER can have many GOALS (1:N relationship)
- Each WORKOUT can have many EXERCISES via WORKOUT_EXERCISES junction table (N:M relationship)
- Each EXERCISE belongs to one CATEGORY (N:1 relationship)

Table Descriptions:

USERS - Stores core authentication data including username, email, and hashed password. Timestamps track account creation and updates.

USER_PROFILES - Extended user information such as name, date of birth, physical measurements (height, weight), and activity level for personalised recommendations.

EXERCISE_CATEGORIES - Classification system for exercises (e.g., Cardio, Strength, Flexibility, Balance, Sports). Each category has a name, description, and icon.

EXERCISES - The exercise library containing name, description, category reference, calories burned per minute, target muscle group, and difficulty level (beginner/intermediate/advanced).

WORKOUTS - User workout sessions with name, date, total duration, calories burned, personal notes, and a 1-5 star rating.

WORKOUT_EXERCISES - Junction table linking workouts to exercises with specific details: sets, reps, weight used, duration, and calories burned for that exercise.

GOALS - User fitness goals with title, description, type, target/current values, unit of measurement, start/target dates, and status tracking.

Database Views:

WORKOUT_SUMMARY - Aggregates workout data with exercise counts for quick dashboard display.

USER_STATS - Calculates user statistics including total workouts, minutes exercised, calories burned, average rating, and goal completion rate.

Code Example - Users Table Schema:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Code Example - Database Connection (config/database.js):

```
const mysql = require("mysql2/promise");

const pool = mysql.createPool({
  host: process.env.HEALTH_HOST,
  user: process.env.HEALTH_USER,
  password: process.env.HEALTH_PASSWORD,
  database: process.env.HEALTH_DATABASE,
  connectionLimit: 10
});

module.exports = { pool };
```

# 5. APPLICATION STRUCTURE

The project follows a modular structure separating concerns:

Root Files:
- index.js: Application entry point, Express configuration, middleware setup
- package.json: Dependencies and npm scripts
- .env: Environment variables (database credentials, session secret)
- create_db.sql: Database schema creation script
- insert_test_data.sql: Sample data for testing

Directories:

/config - Database connection pool configuration

/middleware - Authentication middleware (isAuthenticated, isNotAuthenticated, validatePassword)

/routes - Route handlers organised by feature:
  - main.js: Home, about, search, exercises
  - auth.js: Login, register, logout, profile
  - workouts.js: CRUD operations for workouts
  - goals.js: CRUD operations for goals
  - api.js: REST API endpoints for AJAX requests

/views - EJS templates:
  - /partials: Reusable header and footer components
  - /auth: Login, register, profile pages
  - /workouts: List, form, detail views

- /goals: List, form, detail views

/public - Static assets (CSS stylesheets)

## 6. AUTHENTICATION SYSTEM

The authentication system provides secure user registration and login using industry-standard practices.

Password Security:
Passwords are hashed using bcrypt with 10 salt rounds before storage. This one-way hashing ensures passwords cannot be recovered even if the database is compromised.

Password Requirements:
- Minimum 8 characters
- At least one lowercase letter
- At least one uppercase letter
- At least one number
- At least one special character (!@#$%^&* etc.)

Session Management:
User sessions are stored server-side using express-session. Sessions expire after 24 hours of inactivity. The session contains user ID, username, email, and profile data for quick access.

Route Protection:
The isAuthenticated middleware checks for a valid session before allowing access to protected routes. Unauthenticated users are redirected to the login page with an error message.

Login Process:
1. User submits username and password
2. System queries database for matching username
3. bcrypt compares submitted password with stored hash
4. On success, session is created with user data
5. User is redirected to home page with welcome message

Code Example - Authentication Middleware (middleware/auth.js):

```
function isAuthenticated(req, res, next) {
  if (req.session.user) return next();
  req.session.error = "Please log in to access this page";
  res.redirect("/auth/login");
}
```

Code Example - Password Hashing:

```
const bcrypt = require("bcrypt");

// Hashing a password during registration
```

```
const hashedPassword = await bcrypt.hash(password, 10);

// Verifying password during login
const validPassword = await bcrypt.compare(password, user.password_hash);
```

# 7. WORKOUT MANAGEMENT

The workout system allows users to log, view, edit, and delete their exercise sessions.

Features:
- Paginated workout list (10 per page) sorted by date
- Detailed workout view showing all exercises performed
- Create workouts with multiple exercises
- Edit existing workouts
- Delete workouts with confirmation
- Rate workouts 1-5 stars
- Add notes for personal reference

Workout Creation:
Users select exercises from the library and specify sets, reps, weight, or duration for each. The system calculates total calories based on exercise data and duration.

Data Validation:
- Workout name is required
- Date is required and validated
- Numeric fields (sets, reps, weight) are validated
- Users can only access their own workouts

Code Example - Fetching User Workouts (routes/workouts.js):

```
router.get("/", isAuthenticated, async (req, res) => {
  const [workouts] = await pool.query(
    `SELECT w.*, COUNT(we.id) as exercise_count
     FROM workouts w
     LEFT JOIN workout_exercises we ON w.id = we.workout_id
     WHERE w.user_id = ?
     GROUP BY w.id
     ORDER BY w.workout_date DESC`,
    [req.session.user.id]
  );
  res.render("workouts/list", { workouts });
})
```

# 8. GOALS MANAGEMENT

The goals system helps users set and track fitness objectives with progress monitoring.

Goal Types:
- Weight Loss
- Muscle Gain
- Endurance
- Flexibility
- General Fitness
- Other (custom)

Features:
- Create goals with target values and deadlines
- Update progress manually
- Automatic status change to "completed" when target is reached
- Filter goals by status (active/completed/abandoned)
- Progress bar visualisation
- Goal history tracking

Progress Tracking:
Users update their current value, and the system calculates percentage completion. When current value meets or exceeds the target, the goal is automatically marked as completed with a congratulatory message.

Code Example - Auto-Complete Goal (routes/goals.js):

```
// Check if target reached and auto-complete
let status = goal.status;
if (goal.target_value && currentValue >= goal.target_value) {
  status = "completed";
}

await pool.query(
  "UPDATE goals SET current_value = ?, status = ? WHERE id = ?",
  [currentValue, status, goalId]
);
```

# 9. SEARCH FUNCTIONALITY

The search feature provides a unified way to find content across the application.

Search Capabilities:
- Search exercises by name, description, or muscle group
- Search personal workouts by name or notes (logged-in users only)
- Filter results by type (exercises, workouts, or all)
- Results limited to 20 items per category for performance

Implementation:
The search uses SQL LIKE queries with wildcards for partial matching. Results are displayed in categorised sections with links to detailed views.

Code Example - Search Query (routes/main.js):

```
const searchTerm = `%${query}%`;

const [exercises] = await pool.query(
  `SELECT e.*, c.name as category_name
   FROM exercises e
   LEFT JOIN exercise_categories c ON e.category_id = c.id
   WHERE e.name LIKE ? OR e.description LIKE ? OR e.muscle_group LIKE ?
   LIMIT 20`,
  [searchTerm, searchTerm, searchTerm]
);
```

# 10. SECURITY CONSIDERATIONS

The application implements multiple security measures:

Password Security:
- bcrypt hashing with salt prevents rainbow table attacks
- Strong password policy enforced at registration
- Passwords never logged or displayed

SQL Injection Prevention:
All database queries use parameterised statements. User input is never concatenated directly into SQL strings, preventing injection attacks.

Session Security:
- Sessions stored server-side (not in cookies)
- Session secret configurable via environment variable
- 24-hour expiration limits exposure window
- Session destroyed on logout

Access Control:
- Protected routes require authentication

- Users can only view/edit their own data
- Ownership verified on all CRUD operations

Input Validation:
- Server-side validation on all form submissions
- Email format validation
- Required field enforcement
- Numeric range checks

Code Example - Parameterised Query (prevents SQL injection):

```
// SAFE - User input passed as parameter
const [users] = await pool.query(
  "SELECT * FROM users WHERE username = ?",
  [username]
);

// UNSAFE - Never do this (vulnerable to injection)
// "SELECT * FROM users WHERE username = '" + username + "'"
```

# 11. DEPLOYMENT CONFIGURATION

Environment Variables:
The application uses a .env file for configuration:
- HEALTH_HOST: Database server hostname
- HEALTH_USER: Database username
- HEALTH_PASSWORD: Database password
- HEALTH_DATABASE: Database name
- SESSION_SECRET: Secret key for session encryption
- BASE_PATH: URL prefix for reverse proxy deployment

VM Deployment:
The application runs on the Goldsmiths university VM behind a reverse proxy at /usr/112/. The forever package keeps the Node.js process running persistently.

Deployment Steps:
1. SSH into the VM
2. Navigate to project directory
3. Install dependencies with npm install
4. Create MySQL database and user
5. Run schema and test data scripts
6. Start application with forever start index.js

Reverse Proxy Handling:
All internal links use the basePath variable to prepend the correct URL prefix. Redirects use relative paths (../ and ./) to work correctly regardless of the base URL.

# 12. TESTING & DEFAULT CREDENTIALS

Test User Accounts:

Username: gold
Password: smiths
Email: gold@example.com

Username: testuser
Password: Test123!@#
Email: test@example.com

Local Development:
1. Clone the repository
2. Run npm install to install dependencies
3. Create MySQL database using create_db.sql
4. Insert test data using insert_test_data.sql
5. Create .env file with database credentials
6. Run npm start to launch the server
7. Visit http://localhost:8000

Package Dependencies:
- bcrypt: Password hashing
- dotenv: Environment variable management
- ejs: Template engine
- express: Web framework
- express-session: Session management
- express-validator: Input validation
- mysql2: MySQL database driver
- nodemon: Development auto-restart (dev dependency)