

# FINAL LAB – HEALTH APP

## Intro

The Health Macro Tracker is a web application designed to help users explore and understand the nutritional values of everyday foods. The core purpose of the app is to allow users to browse macro-nutrient information, search for foods by name, and, if logged in, contribute new items to the shared database. The app follows the key concepts introduced throughout the module, including server-side routing, form handling, templating, database connectivity and deployment to the virtual server.

The application is built using Node.js and Express for its backend logic, EJS for dynamic HTML templates, and MySQL for persistent data storage. A simple login system allows authenticated users to access restricted functionality, helping to demonstrate control flow between public and protected routes. The app includes dedicated SQL scripts for both database creation and initial data loading, allowing the marker to easily set up the HEALTH database.

This project consolidates the practical skills developed during the labs while extending them through a cleaner structure, improved navigation, and enhanced user interaction. The final result is a clear, functional, and maintainable web application that reflects understanding of both the taught material and good development practices.

## Architecture

The application follows a two-tier architecture consisting of an Express-based application tier and a MySQL-based data tier. The app tier handles routing, request processing, validation, authentication, and page rendering using EJS templates. The data tier stores user accounts and food macro-nutrient records and is accessed through the `mysql2` connection pool.

Browser ---> Node.js + Express (routes, EJS templates) ---> MySQL (health database)

## Data Model

The HEALTH database contains two tables: `users` and `foods`. The `users` table stores login credentials used for restricting access to food creation. The `foods` table stores the nutritional data shown within the application. Each table uses an auto-increment primary key and appropriate data types for reliability.

## Diagram

USERS

id (PK)

username

password

FOODS

id (PK)

name

calories

protein

carbs

fats

No foreign keys are required, as foods are not tied to individual users.

## User Functionality

The Health Macro Tracker provides clear and accessible functionality for both casual browsing and data contribution. The **Home Page** introduces the purpose of the app and links to key sections: Foods List, Search Page, About Page, Login Page, and Add Food. This consistent navigation appears in the header across all EJS templates.

### Browsing Data

At /foods, users can view the full set of food items stored in the HEALTH database. Each entry is displayed in a structured table showing calories, protein, carbohydrates, and fats. This table is rendered using foods.ejs, which loops over the results returned from the SQL query.

# Food Macro Tracker

[Home](#) | [About](#) | [Foods](#) | [Search](#) | [Login](#)

## All Foods

Name	Calories	Protein	Carbs	Fats
Almonds (30g)	174	6	6.1	15.2
Banana (100g)	89	1.1	23	0.3
Brown Rice (100g cooked)	111	2.6	23	0.9
Chicken Breast (100g)	165	31	0	3.6

## Searching

The Search page (/foods/search) allows users to filter foods by name using a text input. The backend performs a LIKE query to match partial text, supporting flexible user input. Search results overwrite the page content dynamically to maintain a consistent user experience.

# Food Macro Tracker

[Home](#) | [About](#) | [Foods](#) | [Search](#) | [Login](#)

## Search Foods

Name	Calories	Protein	Carbs	Fats
Almonds (30g)	174	6	6.1	15.2
Banana (100g)	89	1.1	23	0.3
Brown Rice (100g cooked)	111	2.6	23	0.9
Chicken Breast (100g)	165	31	0	3.6

## Authentication

The app includes a simple but functional login system implemented via sessions. Only logged-in users can access the Add Food form. Unauthenticated users attempting to access /foods/add are redirected to the login page. This demonstrates controlled access and reinforces the use of sessions taught in the labs.

# Food Macro Tracker

[Home](#) | [About](#) | [Foods](#) | [Search](#) | [Login](#)

## Login

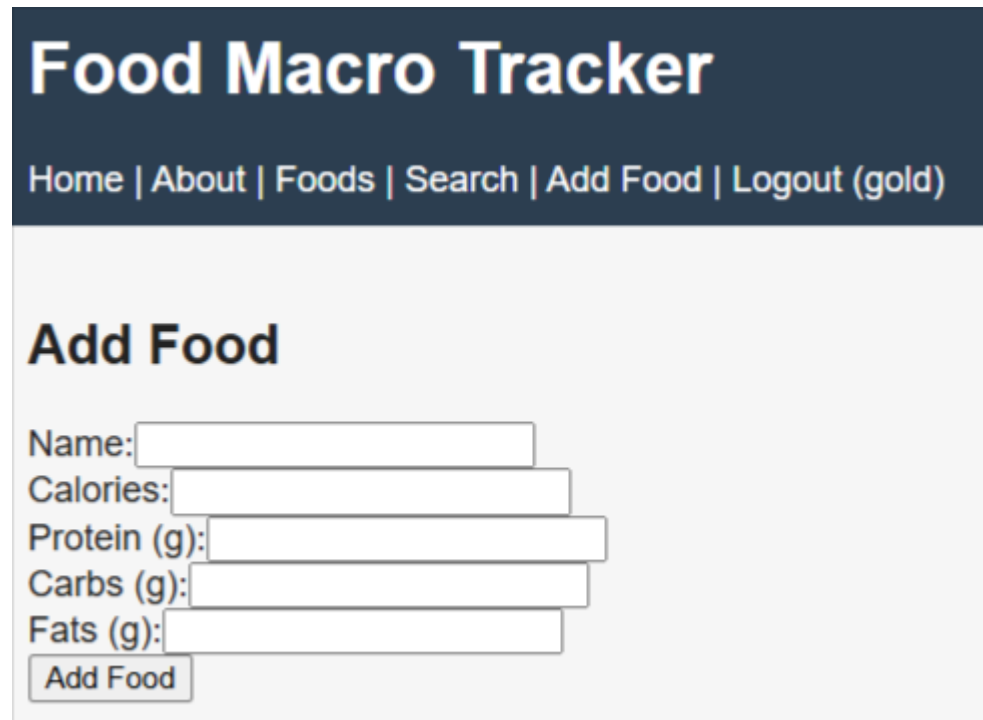
Username:

Password:

For marking, use username **gold** and password **smiths**.

## Adding Foods

The Add Food page demonstrates form handling, server-side validation, SQL insertion, and protected routes. When users submit a new food, the values are inserted into the database using a parameterised SQL query. This ensures safe and predictable data handling.



**Food Macro Tracker**

Home | About | Foods | Search | Add Food | Logout (gold)

### Add Food

Name:

Calories:

Protein (g):

Carbs (g):

Fats (g):

## About Page and Deployment

A simple About page communicates the purpose of the application. The app was successfully deployed to the Goldsmiths virtual server, and the HEALTH database was installed using the provided SQL scripts (create\_db.sql and insert\_test\_data.sql). This ensures the marker can run the app exactly as intended.

## Advanced Techniques

Although the core of the application follows the material taught in the labs, I introduced a few additional techniques to extend functionality in a structured and maintainable way.

### 1. Modular Route Organisation

Instead of keeping all routes inside index.js, the app separates them into routes/pages.js, routes/foods.js, and routes/auth.js. This improves maintainability and divides the app into logical sections.

### Example (routes/foods.js):

```
router.get('/', async (req, res) => {  
  const [rows] = await pool.query('SELECT * FROM foods ORDER BY name ASC');  
  res.render('foods', { foods: rows });  
});
```

## 2. Use of a Connection Pool

The database connection uses a connection pool rather than a single connection. This is more efficient for multiple users and prevents connection errors.

```
const pool = mysql.createPool({  
  host: 'localhost',  
  user: 'food_app',  
  password: 'qwertyuiop',  
  database: 'HEALTH',  
  connectionLimit: 10  
});
```

## 3. Conditional Rendering in Templates

Navigation links change depending on whether a user is logged in, demonstrating dynamic rendering in EJS:

```
<% if (user) { %>  
  <a href="/foods/add">Add Food</a>  
  <a href="/auth/logout">Logout</a>  
<% } else { %>  
  <a href="/auth/login">Login</a>  
<% } %>
```

## 4. Server-Side Validation

Before inserting foods, the app checks that required fields exist and are valid. This prevents accidental invalid inserts.

```
if (!name || !calories) {  
  return res.render('add_food', { error: 'All fields required' });  
}
```

## Ai Declaration

AI assistance was used for guidance during development, including help with debugging MySQL setup issues, clarifying deployment procedures for the virtual server, and generating ideas for documentation layout. The final implementation and all code were written, tested, and adapted manually by me. AI was used only as a supportive learning tool, not a replacement for completing the work.