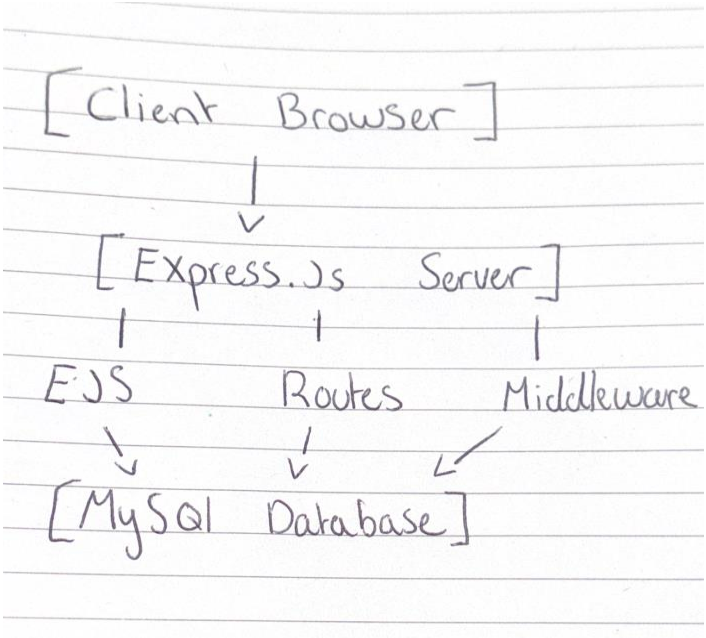
Outline

The Health Tracker application is a user-friendly web platform designed to help individuals monitor and manage their personal health data efficiently. It allows users to keep track of key metrics such as weight, height, BMI, blood pressure, and heart rate, providing a clear overview of their health over time. Users can create a secure account and log in to a personal dashboard, where they can easily add new records, review past entries, and monitor trends in their health metrics. The application prioritises simplicity and accessibility, featuring a responsive design that works smoothly across devices while ensuring the privacy and security of user data. On the technical side, the application is built using Node.js and Express.js for robust server-side functionality, MySQL for reliable and scalable data storage, and EJS templates to render dynamic and interactive pages. Security measures such as bcrypt-hashed passwords and session-based authentication protect user accounts, while input sanitization and server-side validation prevent common errors and vulnerabilities. Overall, this application showcases essential web development practices, including CRUD operations, secure session management, and careful handling of user input, making it a practical and secure tool for personal health tracking.

Architecture

The application follows a standard client-server model. The application tier uses Node.js with Express.js to handle HTTP requests, routing, and server-side logic. The data tier consists of a MySQL database that stores user information and health records. EJS templates render dynamic HTML pages, while CSS provides styling. Session management is implemented with express-session and express-sanitizer ensures safe user input.



Data Model

The database contains two primary tables: users and health_records. The users table stores user credentials (username, hashed password, first name, last name, email) and a unique id.

The health_records table stores health metrics for each user, including weight, height, BMI, blood pressure, heart rate, record date, and a foreign key user_id linking to the user's table.

Diagram:

Users table:

Field	Type	Null	Key	Default	Extra
username	varchar(50)	NO	UNI	NULL	
id	int	NO	PRI	NULL	auto_increment
first	varchar(50)	NO		NULL	
last	varchar(50)	NO		NULL	
email	varchar(100)	NO	UNI	NULL	
hashedPassword	varchar(255)	NO		NULL	

health_records

```
mysql> describe health_records;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
user_id	int	NO	MUL	NULL	
record_date	date	NO		NULL	
weight	decimal(5,2)	YES		NULL	
height	decimal(5,2)	YES		NULL	
bmi	decimal(5,2)	YES		NULL	
blood_pressure	varchar(20)	YES		NULL	
heart_rate	int	YES		NULL	
notes	text	YES		NULL	

User Functionality

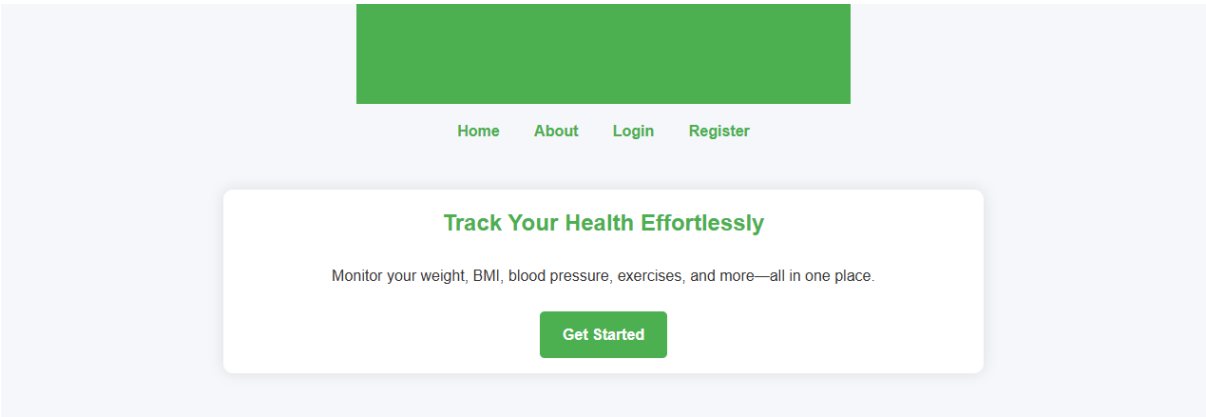
Users interact with the Health Tracker through a browser interface:

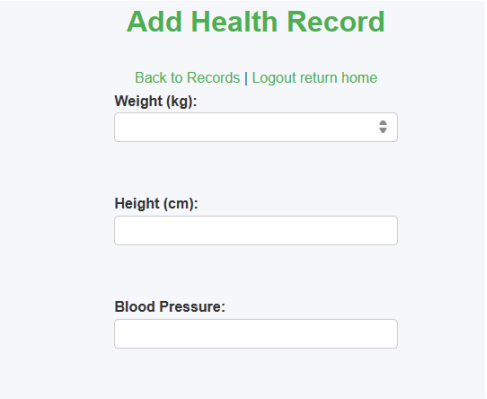
1. **Registration and Login:** Users can create an account via /users/register and log in at /users/login. Validation ensures correct email format, sufficient password length, and required fields. Passwords are hashed with bcrypt (users.js, POST /register).
2. **Home:** After login, users are redirected to /users/loggedin or / showing personalized greetings. Navigation links adapt based on login status (index.ejs).
3. **Health Records Management:** Users can add new records (/health/add, add_record.ejs) specifying weight, height, blood pressure, and heart rate. BMI is calculated automatically in health.js during POST /add. Records are stored in MySQL and displayed in a table (/health, health.ejs) with sortable fields such as date, weight, and BMI.

4. **Session and Logout:** Session management ensures users remain logged in securely. The logout route (/users/logout) destroys the session and clears cookies.
5. **About Page:** /about provides an overview of app features and mission (about.ejs).

Screenshots:

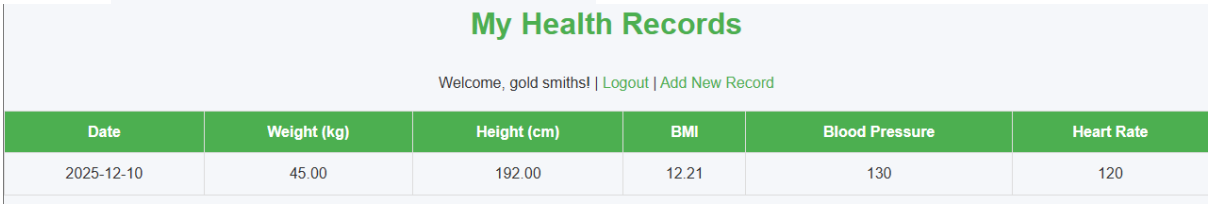
- Home page with login/register/about links





- Screen shot here shows when a user wants to add a health Record to their account

Below shows how when the user “Gold Smiths” added a record it gets saved in their personal records:



Below is a screenshot from a test user implementing all the records that are personal to him and not shared with other users like gold smiths etc.

Welcome, momo Hacksmith! | Logout | Add New Record

Date	Weight (kg)	Height (cm)	BMI	Blood Pressure	Heart Rate
2025-12-09	23.00	23.00	434.78	222	323
2025-12-09	232.00	232.00	43.10	2323	255
2025-12-09	234.00	564.00	7.36	45	453
2025-12-09	434.00	343.00	36.89	434	32
2025-12-09	43.00	34.00	371.97	23	23
2025-12-09	33.00	564.00	1.04	66	5
2025-12-09	23.00	32.00	224.61	44	343

Advanced Techniques

1. **Password Security:** Implemented bcrypt hashing with a configurable salt round (10) for secure password storage.

```
bcrypt.hash(password, saltRounds, (err, hashedPassword) => {
    if (err) return next(err);

    const sql = `
        INSERT INTO users (username, first, last, email, hashedPassword)
```

BMI Calculation and Input Sanitization: User-submitted weight and height are sanitized using express-sanitizer, and BMI is automatically calculated during record insertion.

```
const weight = req.sanitize(req.body.weight);
const height = req.sanitize(req.body.height);
const blood_pressure = req.sanitize(req.body.blood_pressure);
const heart_rate = req.sanitize(req.body.heart_rate);
const bmi = (weight / ((height / 100) ** 2)).toFixed(2);
```

3. **Session-based Authentication:** Middleware checks for active sessions to protect routes such as /health/add and /users/loggedin.

```
const redirectLogin = (req, res, next) => {
    if (!req.session.userId) {
        res.redirect('/users/login');
    } else {
        next();
    }
};
```

These techniques demonstrate secure and robust handling of user authentication, input validation, and server-side logic

AI Declaration:

I have used AI, specifically ChatGPT, as a supportive tool throughout the development of this project. ChatGPT helped me understand my code by explaining how different routes, middleware, and database operations work, and guided me in identifying and resolving errors. It also provided advice and suggestions for improving the CSS styling and layout of my EJS templates, enhancing the user interface and responsiveness. Importantly, AI was not used to generate the code for the application itself; all coding, design decisions, and testing were performed independently. AI served solely as a reference and learning aid to clarify concepts, debug issues, and improve presentation.