**Introduction to Modelling and Animation**

**Mobina Sharafi (33777254)**

**Final Lab instructions report:**

**Outline**

This application is a full-stack health tracking system designed to give users a structured system with a calm interface, that allows them to gain an understanding on the patterns regarding their health over time. This application allows users to log and monitor their activities and reflect on their long-term goals.

New users can create an account by entering their personal details including age, height, weight and a chosen health goal, once the user is registered, the application calculates their BMI. The BMI value calculated is paired with its own suited classification such as severely underweight, normal, obesity class II, giving user an immediate understanding of their current status.

After logging in, users can add and track health activities, which include specifications such as activity type, duration, caloric expenditure and intensity. There is a built-in link to an External AI system (ChatGPT), that will help users who need to specify their calorie estimates. These records can be searched, sorted (by their date, alphabetically, duration, and calories burnt), edited and deleted, enabling useful interactions with the stored data. The application also includes a personal vision board where users can upload motivational images and articulate their health intentions.

**Architecture**

This web-application uses a three-layered structure:

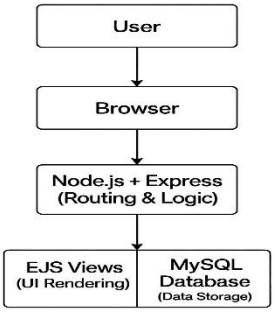
Express runs on Node.js which handles all routing and application logic.

EJS renders the user interface.

MySQL stores the data.

The server is organized into different route files for user account actions, health record management, and the vision board features.

Each route processes requests and preforms validations when necessary and interacts with the database through SQL queries. Dynamic pages are generated using EJS templates in the views directory, while static assets and uploaded images are served from public. This clear separation of routing, interface rendering and data storage keeps this application organized and well-maintained.



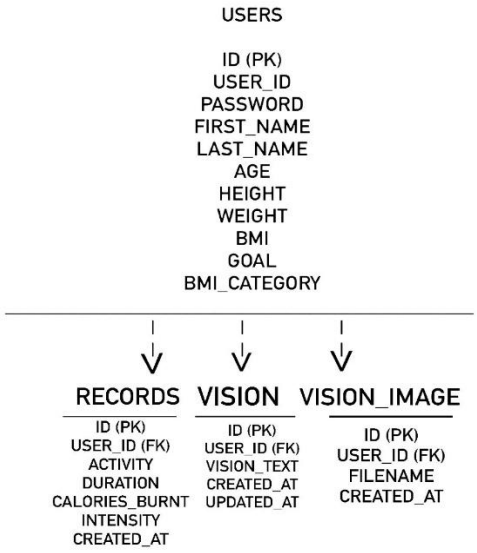
**Data Model**

This application uses MySQL database with four tables: users, records, vision, and vision_images.

The users table stores account and profile information including personal details (age, height, weight, first name, last name) BMI, goal and BMI category.

The records table stores individual health activity logs linked to a user through user_id. The vision table stores each user's written health vision text with created and updated timestamps.

The vision_images table stores filenames of images uploaded to each user's vision board.
All three tables reference users.user_id, creating many relationships from users to their records, vision and images.

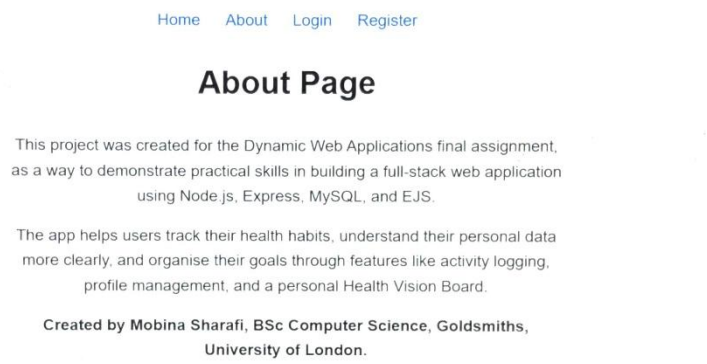



User Functionality:

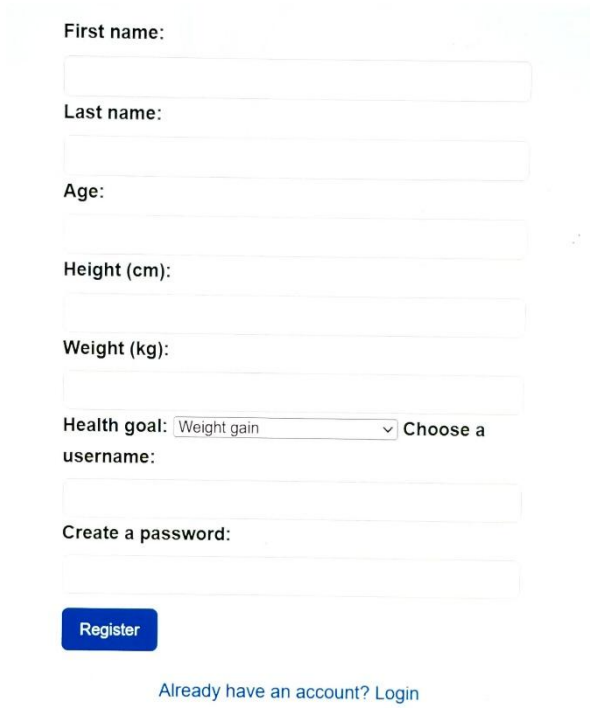
This web application provides a clean and clear interface that supports users manage their health data, track their activities and maintain a personal vision board of their long-term goals. When visiting the site without having logged in, users see a navigation bar with links to Home, About, Login and Register. Once logged in, the navigation bar updates to to Home, About, View My Records, Profile, Vision Board and Logout.

The Home Page welcomes users with a message describing the purpose of the app, followed by inspirational health and fitness images.

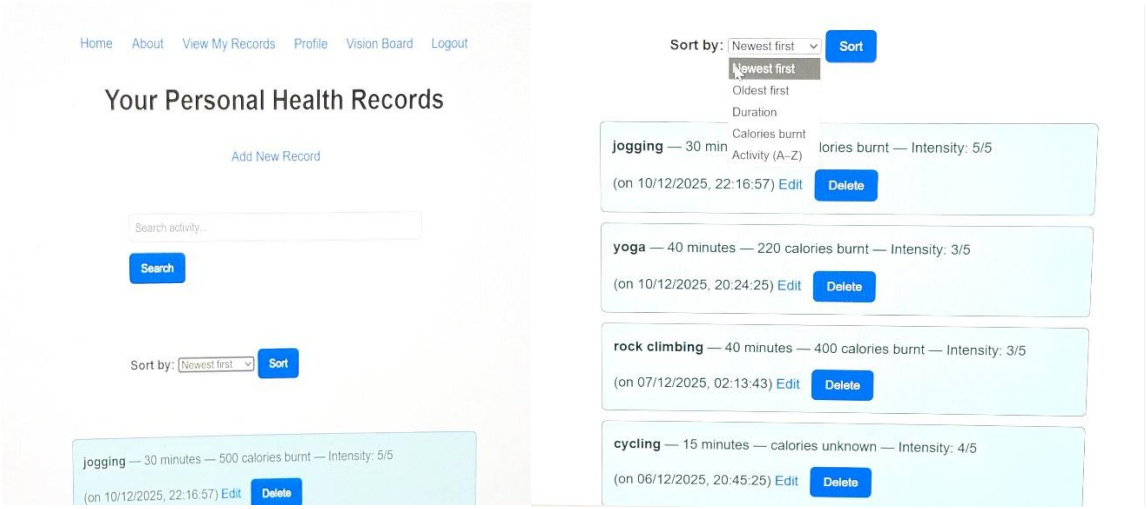
The About Page explains that the project was built for the DWA final assignment and outlines the goals of the application and the technologies used.



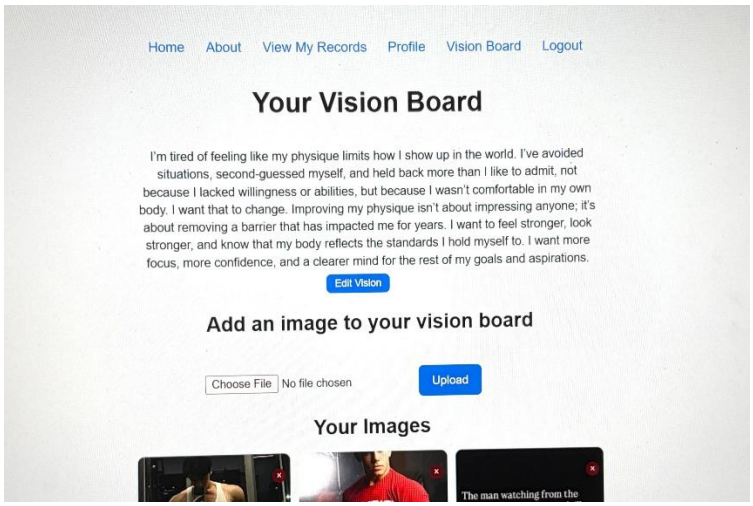

Users can create an account through Register Page, where they can enter their personal details such as name, last name, age, height, weight and their health goal (selected from a dropdown or entered manually when "Other" is chosen). Their BMI will be automatically calculated and shown once they are registered as a part of their profile, and they will also be assigned a BMI category. They also create their username and password (the password must be minimum 8 characters long and have at least one alphabetical word and one number). After this, they are shown a welcome message and can now log in through the login page.

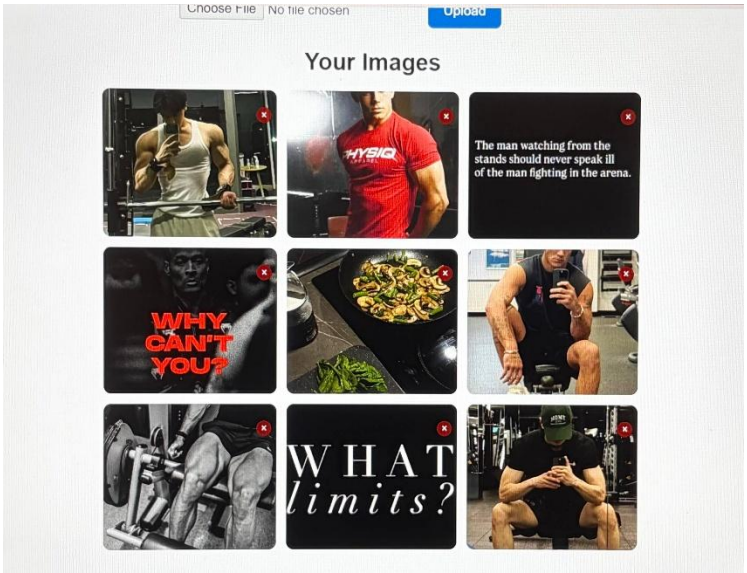



The View My Records page displays all logged health activities. Each entry includes its own data such as activity name, duration, calories burnt, intensity level (0-5), and timestamps. Users can search for specific activities, sort their records (by the following orders: newest, oldest, duration, calories burned, activity A-Z), delete or edit their current records and add new ones. The calorie burned section is optional, however there is a link that allows the users to consult AI to get an estimation of it.




The Vision Board page allows users to write and update their health visions, describing their long-term goals and motivations. They can also add images which will appear beneath the text as a part of their vision board, each of these images can be removed, giving the users full access to add, delete or change any material within their vision board.

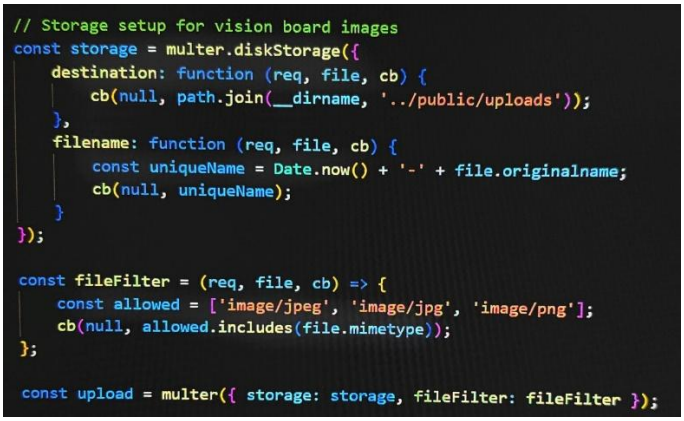


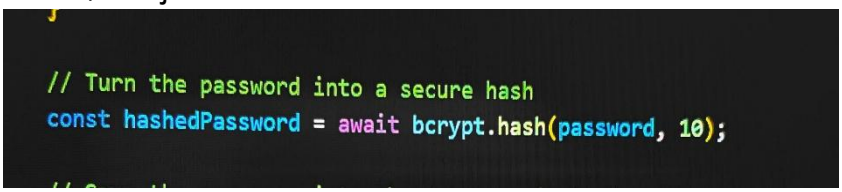
And at last, the log out function securely ends the session and returns users to the public log in page.

**Advanced Techniques:**

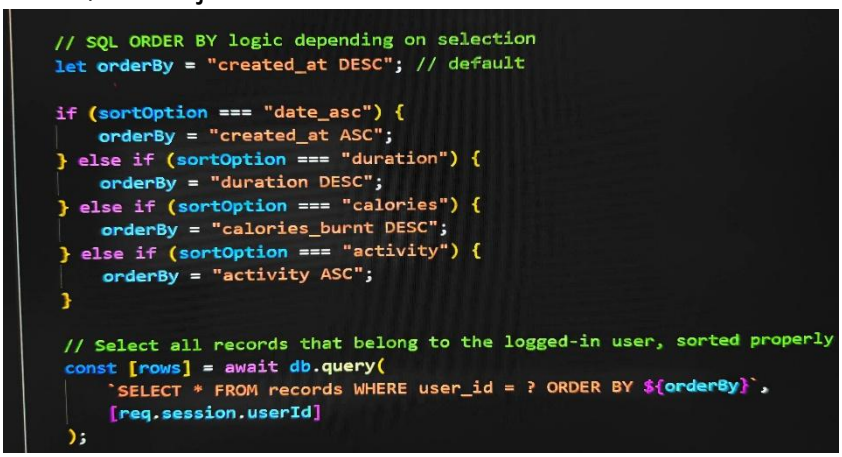
1- Image upload handling with Multer: this application uses Multer with a custom disk storage configuration to handle user image upload for the vision board, images will be saved and filenames will be generated.
Routes/vision.js




2- Password hashing with bcrypt: User passwords are securely hashed before being stored, ensuring the database never contains plain-text passwords.
Routes/auth.js




3- Dynamic SQL sorting and searching: The records page supports multiple sorting options and keyword search using dynamically constructed SQL queries.
Routes/records.js




4- Session-based authentication and protected routes: Only logged-in users can access sensitive/personal features such as records, the vision board, and the profile page.

Files: routes/records.js, routes/vision.js, routes/auth.js

```
if (!req.session.userId) {
    return res.redirect('/login');
}
```

5- RESTful PUT and DELETE request support using method-override: The application supports editing and deleting records through PUT and DELETE HTTP methods, even though HTML forms normally only support GET and POST.
Index.js routes/records.js

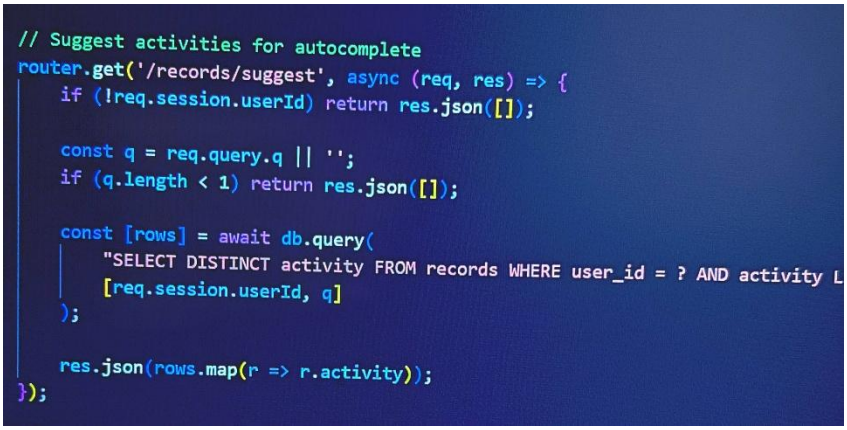
```
// this allows us to use HTTP verbs such as PUT or DELETE in
app.use(methodOverride('_method'));
```

```
// Delete a record
router.delete('/records/delete/:id', async (req, res) => {
    try {
        if (!req.session.userId) {
            return res.redirect('/login');
        }

        const recordId = req.params.id;

        await db.query(
            "DELETE FROM records WHERE id = ? AND user_id = ?",
            [recordId, req.session.userId]
        );

        res.redirect('/records');

    } catch (error) {
        console.error(error);
        res.send("Something went wrong");
    }
});
```

6- Automatic BMI calculation/classification: (Not Necessarily considered as advanced, however inventory and worth a mention.) When users register or update their profile credentials, their MBTI is calculated and categorised based on medical BMI criteria.

```
// Simple helper that returns a BMI classification based on the BMI value
function classifyBMI(bmi) {
    if (!bmi) return null;

    if (bmi < 16) return "Severely Underweight";
    if (bmi < 18.5) return "Underweight";
    if (bmi < 25) return "Normal";
    if (bmi < 30) return "Overweight";
    if (bmi < 35) return "Obesity Class I";
    if (bmi < 40) return "Obesity Class II";
    return "Obesity Class III";
}
```

7- Autocomplete suggestion endpoint (JSON API): this web application provides an endpoint that returns activity suggestions according to what the user types and enables autocomplete functionality.
You can see the code snippet in the image attached below:



```
// Suggest activities for autocomplete
router.get('/records/suggest', async (req, res) => {
    if (!req.session.userId) return res.json([]);

    const q = req.query.q || '';
    if (q.length < 1) return res.json([]);

    const [rows] = await db.query(
        "SELECT DISTINCT activity FROM records WHERE user_id = ? AND activity L
        [req.session.userId, q]
    );

    res.json(rows.map(r => r.activity));
});
```

**AI Declaration:**
AI was used as a supportive tool in a limited way during the development of this application for. For example, I used AI to help interpret a MySQL “access denied” error, which turned out to be caused by incorrect database user permissions rather than any issue with my code. With that explanation, I was able to configure a dedicated database user with the correct privileges. I also used AI to clarify how to set up Multer’s storage engine for image uploads, including MIME-type filtering and creating unique filenames. In addition, I referred to AI to better understand asynchronous MySQL queries using async/await and how the mysql2 promise-based connection pool works, along with confirming best practices for using environment variables in database configuration. The AI suggestions built into my code editor occasionally helped with completing or rephrasing comments, but weren’t used to generate the application logic. Overall, whenever I used AI, it was not solely to solve an issue, but also for to deepen my understanding of the underlying concepts.

*Thank you for taking the time to read my report.*
*Mobina Sharafi 33777254*