-- Outline –

The Health & Fitness Tracker is a full-stack web application designed to help users log, visualize, and track their personal wellness data. Built using Node.js, Express, and EJS templates, the application provides a secure, session-based user experience. Upon registration and login, users gain access to personalized data entry forms. Key features include logging specific daily workouts, tracking body metrics , and viewing a centralized dashboard of their personal history. The application uses a robust MySQL database to ensure data persistence and integrity. Security is prioritized through the use of environment variables for database credentials, bcrypt for password hashing, and session management for authentication, ensuring that each user's data is private and restricted to their session ID. Additionally, a public search library allows users to browse pre-defined workout types without needing to log in.

-- Data model --

- The database schema uses five main tables connected via one-to-many relationships.

- users: Stores user credentials and basic profile information.

- login_attempts: An audit table tracking login successes and failures.

- exercises: A static library of defined activities (Cardio, Strength, etc.).

- workouts: Logs specific instances of user activity, linked to users via user_id and potentially to
- exercise types via workout_type_id.

- body_metrics: Stores historical measurements like weight, linked to users via user_id.

-- Architecture --

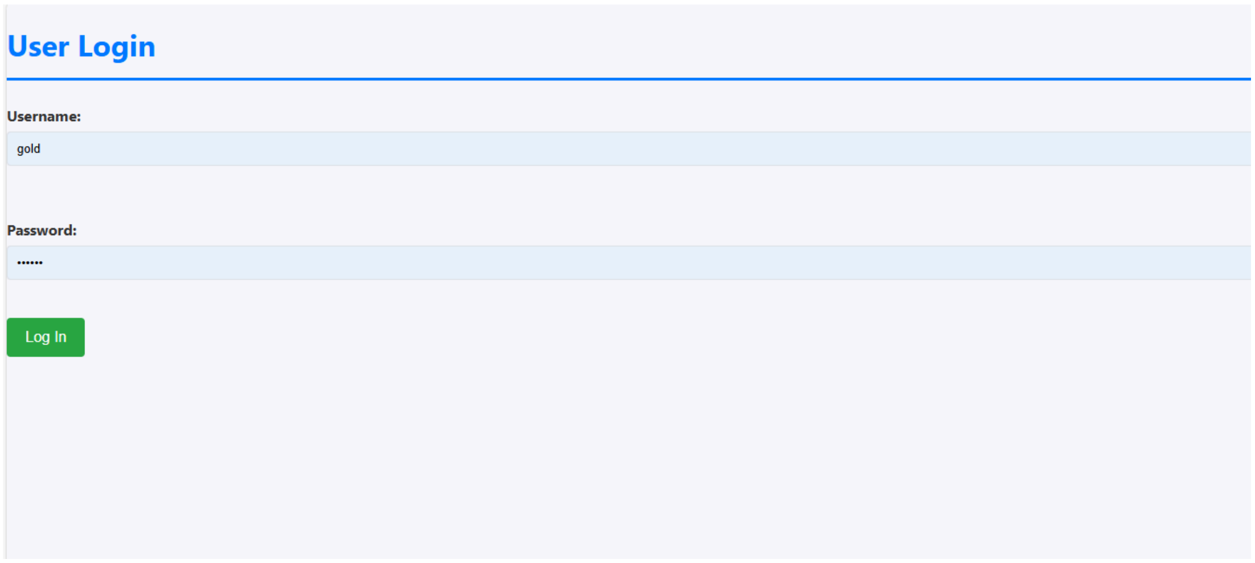
- Presentation Tier: Handled by EJS (Embedded JavaScript) templates rendered by the Express server.

- Application Tier: Managed by Node.js using the Express framework for routing, middleware (sessions,
- body parsing), and business logic (login, data validation, database queries).

- Data Tier: A MySQL database securely stores all persistent data (user accounts, workout logs, body metrics).
- The application connects to this tier using the mysql2 driver.

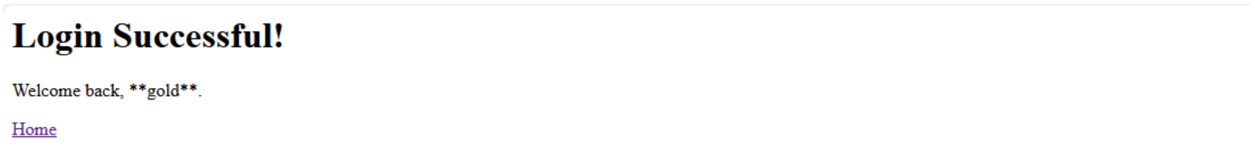
-- Functionality --
1. User Authentication and Access Control
Access to personalized features is restricted via robust authentication:

Login/Logout: Users must log in via the dedicated form using their username and password. The application utilizes Express Sessions, ensuring that once logged in,
a user's ID is securely stored.


User Login

Username:
gold

Password:
••••••


Login Successful!

Welcome back, **gold**.

Access Control: The redirectLogin middleware protects sensitive routes like /workout and /progress-chart. Unauthenticated users are automatically redirected to the
login page, maintaining data privacy.

Audit Logging: Every login attempt (success or failure) is logged to the login_attempts table for security monitoring.

Login Audit History

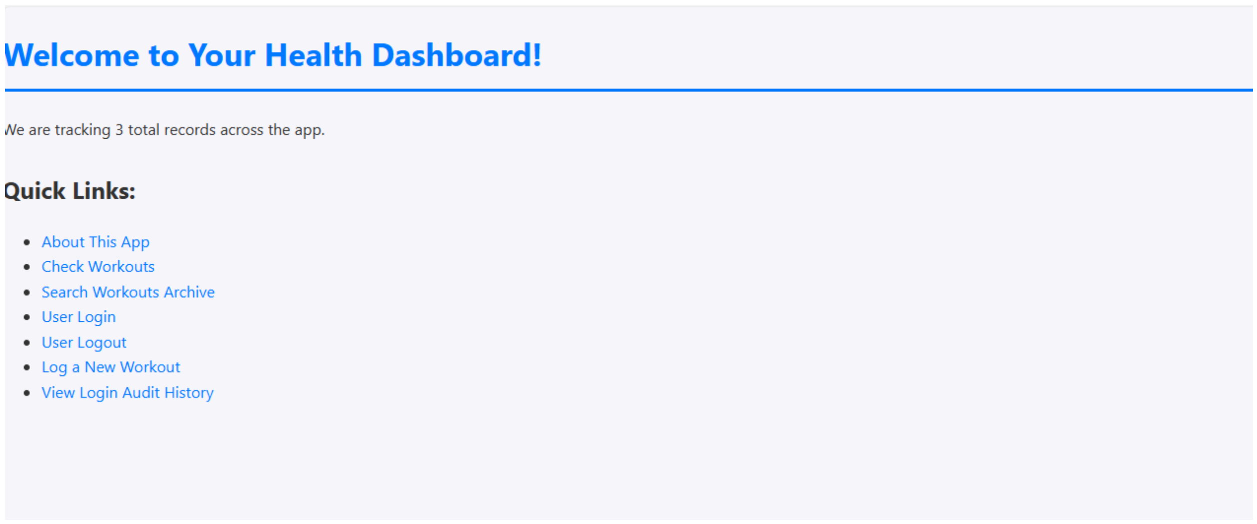
A log of all successful and failed user login attempts.

Username	Time of Attempt	Status
gold	11/12/2025, 17:20:39	SUCCESS
gold	11/12/2025, 17:19:25	SUCCESS
gold	11/12/2025, 17:19:12	SUCCESS
gold	11/12/2025, 17:17:30	SUCCESS
gold	11/12/2025, 17:13:19	SUCCESS
gold	11/12/2025, 16:51:10	SUCCESS
gold	11/12/2025, 16:45:53	SUCCESS
gold	11/12/2025, 16:29:20	SUCCESS
gold	11/12/2025, 16:17:41	SUCCESS
gold	11/12/2025, 16:16:40	SUCCESS
gold	11/12/2025, 15:15:04	SUCCESS
gold	11/12/2025, 14:42:10	SUCCESS
gold	11/12/2025, 10:36:41	SUCCESS
gold	11/12/2025, 10:36:27	SUCCESS
gold	11/12/2025, 10:23:41	SUCCESS
gold	11/12/2025, 10:23:31	SUCCESS
gold	11/12/2025, 10:14:26	SUCCESS
gold	11/12/2025, 10:10:03	SUCCESS
gold	11/12/2025, 10:07:56	SUCCESS
gold	11/12/2025, 10:07:22	SUCCESS
gold	11/12/2025, 10:05:31	SUCCESS
gold	11/12/2025, 10:05:27	SUCCESS
gold	11/12/2025, 10:05:13	SUCCESS
gold	11/12/2025, 10:03:20	SUCCESS
gold	11/12/2025, 09:45:20	SUCCESS
gold	11/12/2025, 09:38:29	SUCCESS
gold	11/12/2025, 09:38:15	SUCCESS
gold	11/12/2025, 09:37:56	SUCCESS
gold	11/12/2025, 09:29:26	SUCCESS
gold	11/12/2025, 09:29:19	SUCCESS

2. Home Dashboard

The home page acts as the central hub, displaying:

- **Summary Statistics:** A count of the total records stored in the database.
- **Quick Links:** Navigation links to all primary features (Log Workout, Search, Progress Chart).



Welcome to Your Health Dashboard!

We are tracking 3 total records across the app.

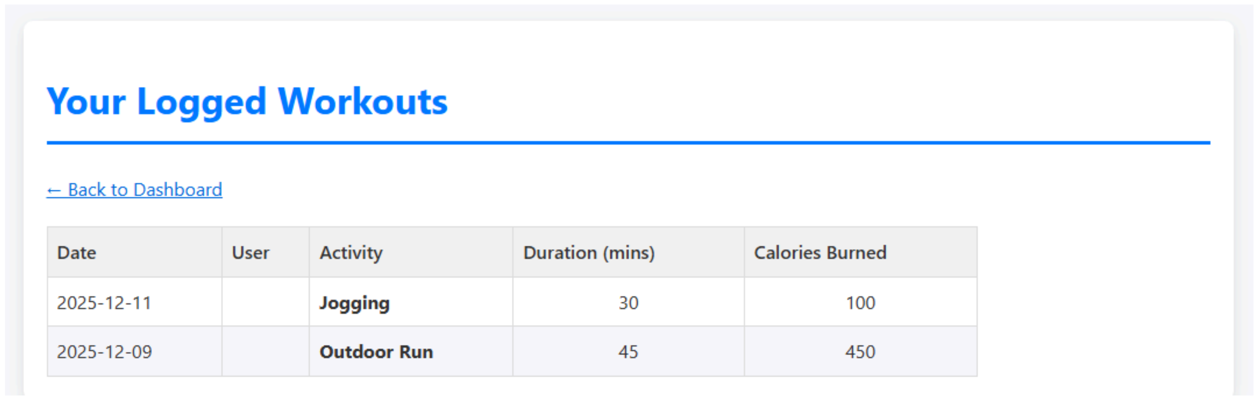
Quick Links:

- About This App
- Check Workouts
- Search Workouts Archive
- User Login
- User Logout
- Log a New Workout
- View Login Audit History

3. Data Entry Forms

Users can input personal data via dedicated forms, with data automatically linked to their session ID (req.session.userId).

- **Log Workout** (/workout-log): This form allows the user to record specific activity details:
  - Activity Name
  - Duration (minutes)
  - Calories Burned The submitted data is inserted into the workouts table.



Your Logged Workouts

← Back to Dashboard

Date	User	Activity	Duration (mins)	Calories Burned
2025-12-11		**Jogging**	30	100
2025-12-09		**Outdoor Run**	45	450

4. Data Visualization and Viewing

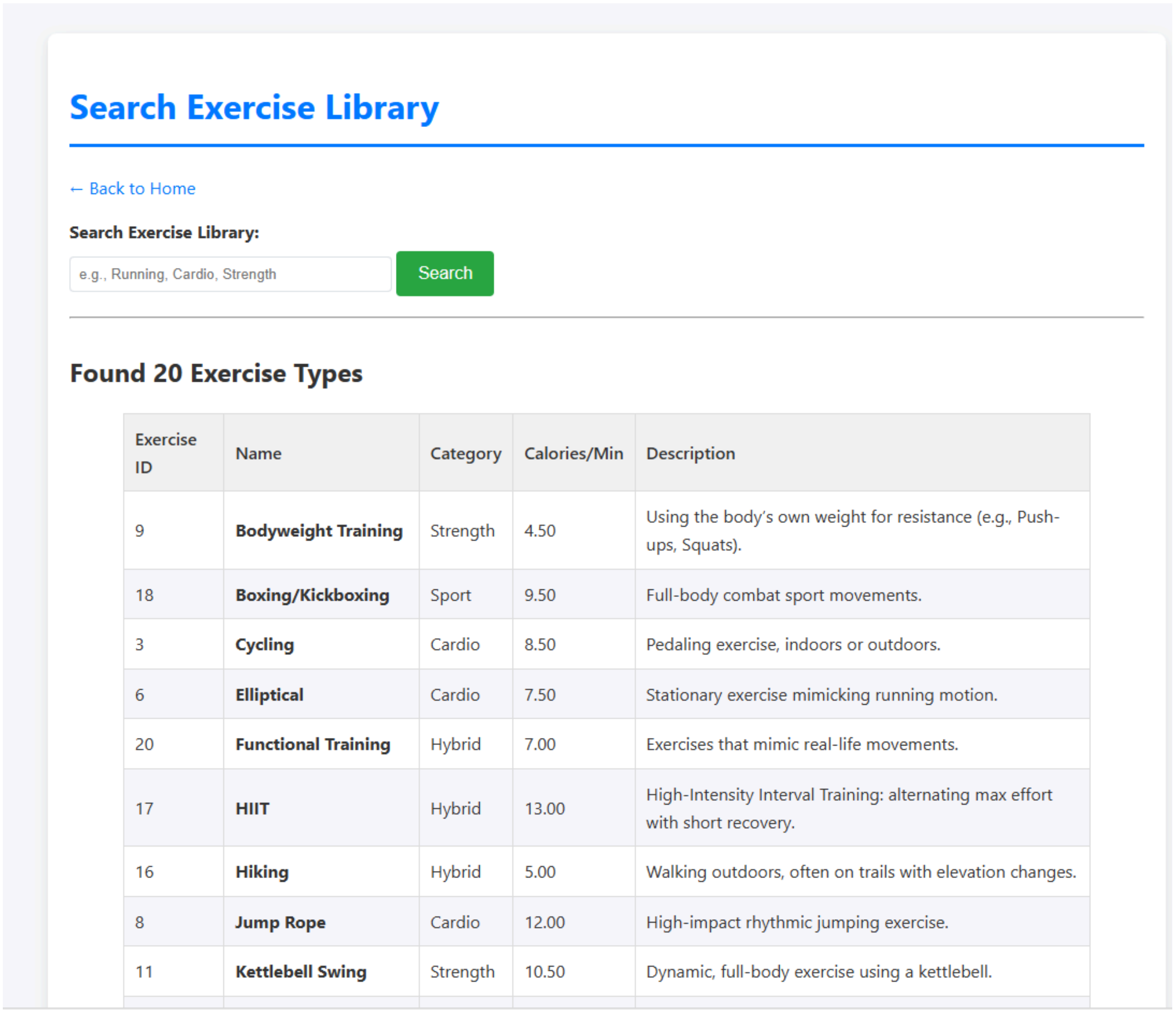
The application provides two main methods for viewing historical data:

- **Personal Workout History** (/workout): This page displays a table listing only the workouts logged by the current session user. The data is pulled from the workouts table, restricted by a WHERE user_id = ? clause for privacy.
- **Progress Chart** (/progress-chart): This feature displays a visual line graph of the user's historical weight over time. The backend fetches data from the body_metrics table, and the frontend uses the Chart.js library to render an interactive progress chart.

5. Exercise Library Search (/search)

The search page provides a public, unauthenticated mechanism to explore the static library of activities stored in the exercise exercise types table.

- **Functionality:** Users can search the library by type_name or category. If no query is entered, the page lists all available exercise types.
- **Output:** Results are presented in a clear table showing the name, category, and base calorie rate for each activity.



Search Exercise Library

← Back to Home

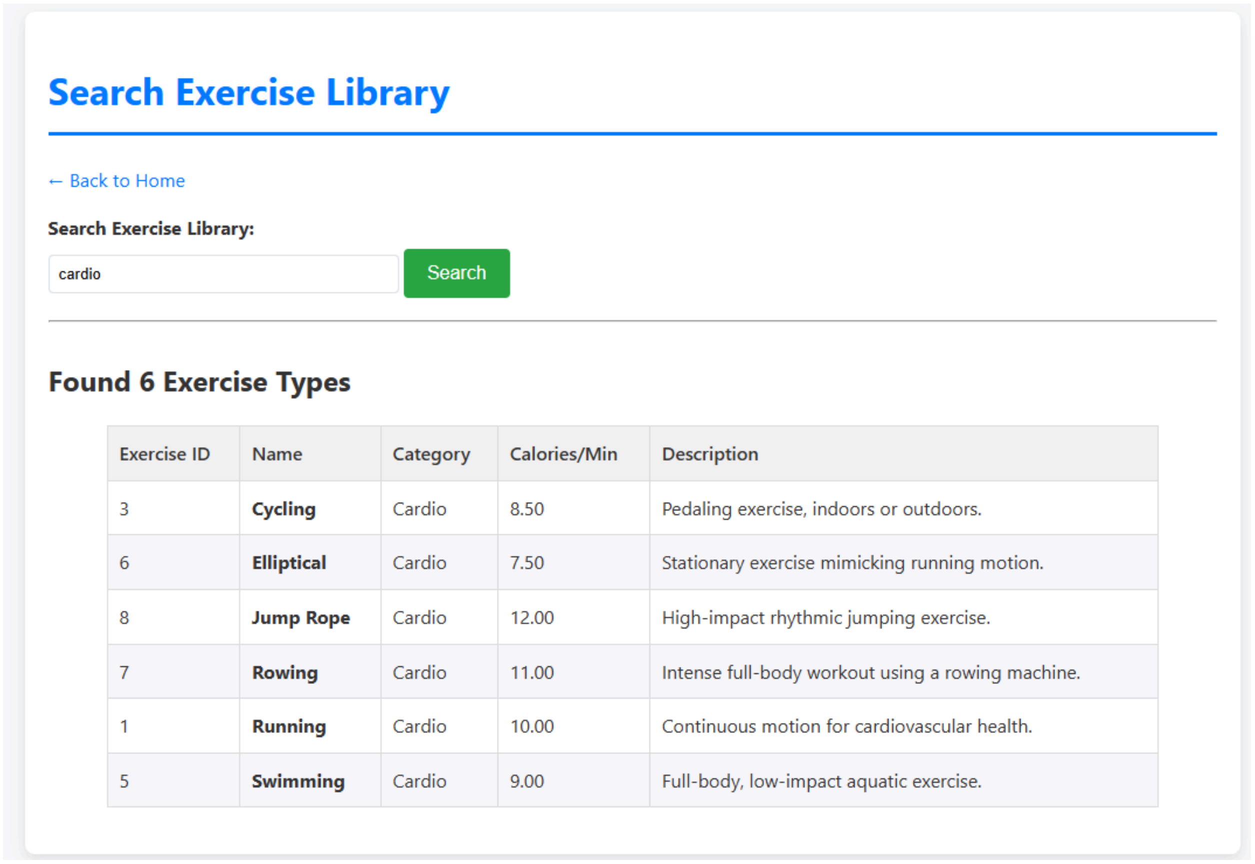
Search Exercise Library:

e.g., Running, Cardio, Strength

Search

Found 20 Exercise Types

Exercise ID	Name	Category	Calories/Min	Description
9	**Bodyweight Training**	Strength	4.50	Using the body's own weight for resistance (e.g., Push-ups, Squats).
18	**Boxing/Kickboxing**	Sport	9.50	Full-body combat sport movements.
3	**Cycling**	Cardio	8.50	Pedaling exercise, indoors or outdoors.
6	**Elliptical**	Cardio	7.50	Stationary exercise mimicking running motion.
20	**Functional Training**	Hybrid	7.00	Exercises that mimic real-life movements.
17	**HIIT**	Hybrid	13.00	High-Intensity Interval Training: alternating max effort with short recovery.
16	**Hiking**	Hybrid	5.00	Walking outdoors, often on trails with elevation changes.
8	**Jump Rope**	Cardio	12.00	High-impact rhythmic jumping exercise.
11	**Kettlebell Swing**	Strength	10.50	Dynamic, full-body exercise using a kettlebell.



Search Exercise Library

← Back to Home

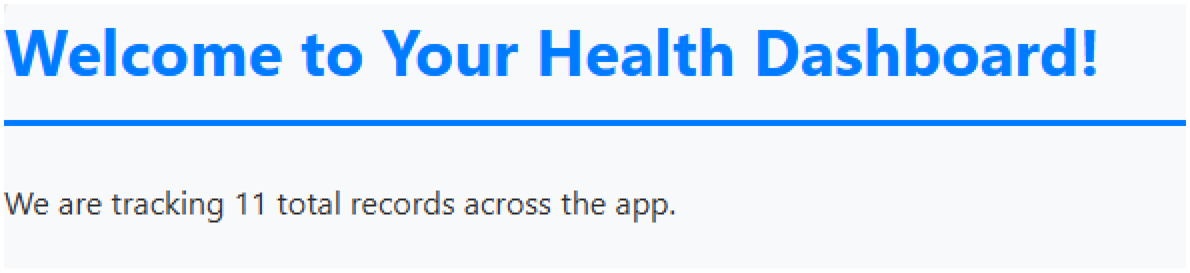
Search Exercise Library:

cardio

Found 6 Exercise Types

Exercise ID	Name	Category	Calories/Min	Description
3	**Cycling**	Cardio	8.50	Pedaling exercise, indoors or outdoors.
6	**Elliptical**	Cardio	7.50	Stationary exercise mimicking running motion.
8	**Jump Rope**	Cardio	12.00	High-impact rhythmic jumping exercise.
7	**Rowing**	Cardio	11.00	Intense full-body workout using a rowing machine.
1	**Running**	Cardio	10.00	Continuous motion for cardiovascular health.
5	**Swimming**	Cardio	9.00	Full-body, low-impact aquatic exercise.

6. Global incrementation



Welcome to Your Health Dashboard!

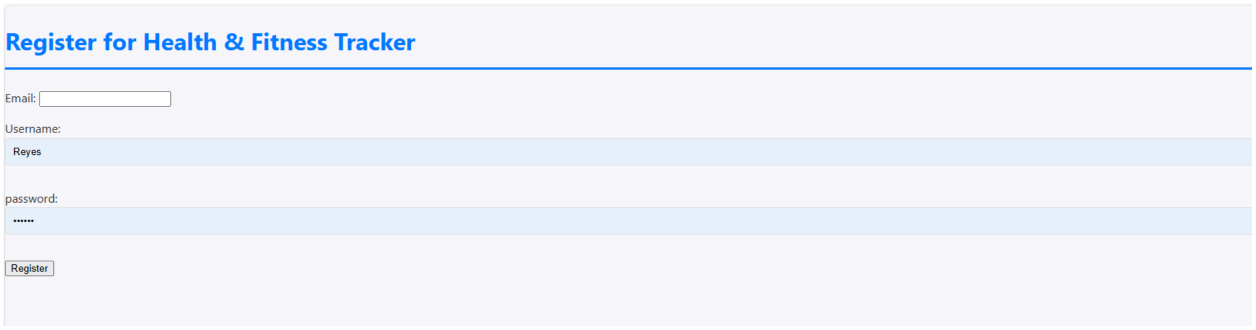
We are tracking 11 total records across the app.

Shows how many workouts have been logged across all users on the app

7. Registration Module

The registration module allows new users to create an account to access the secure features of the Health & Fitness Tracker. This process is highly focused on security and data integrity. When a user navigates to the /register route, they are presented with a simple form to collect their sign-up information.

- **Display**: The register.ejs template is rendered, passing the title "Register for Health & Fitness Tracker."
- **Required Fields**: The form requires input for Username, Email, and Password. (Note: While the HTML form collects First Name and Last Name, the server-side logic is configured to securely ignore them to match the database schema.)



Register for Health & Fitness Tracker

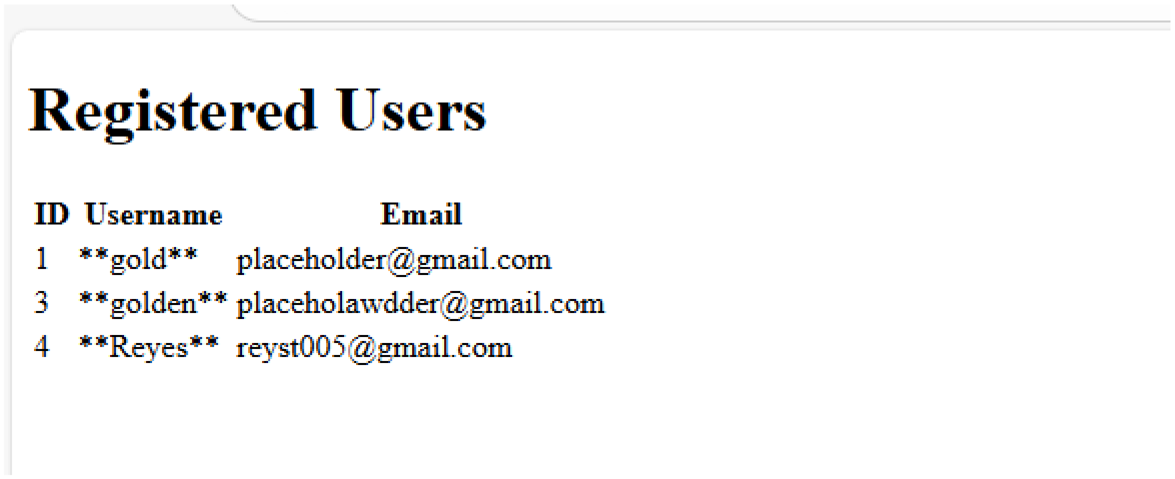
Email:

Username:
Reyes

password:
••••••

8. User list

Display a list of all registered users



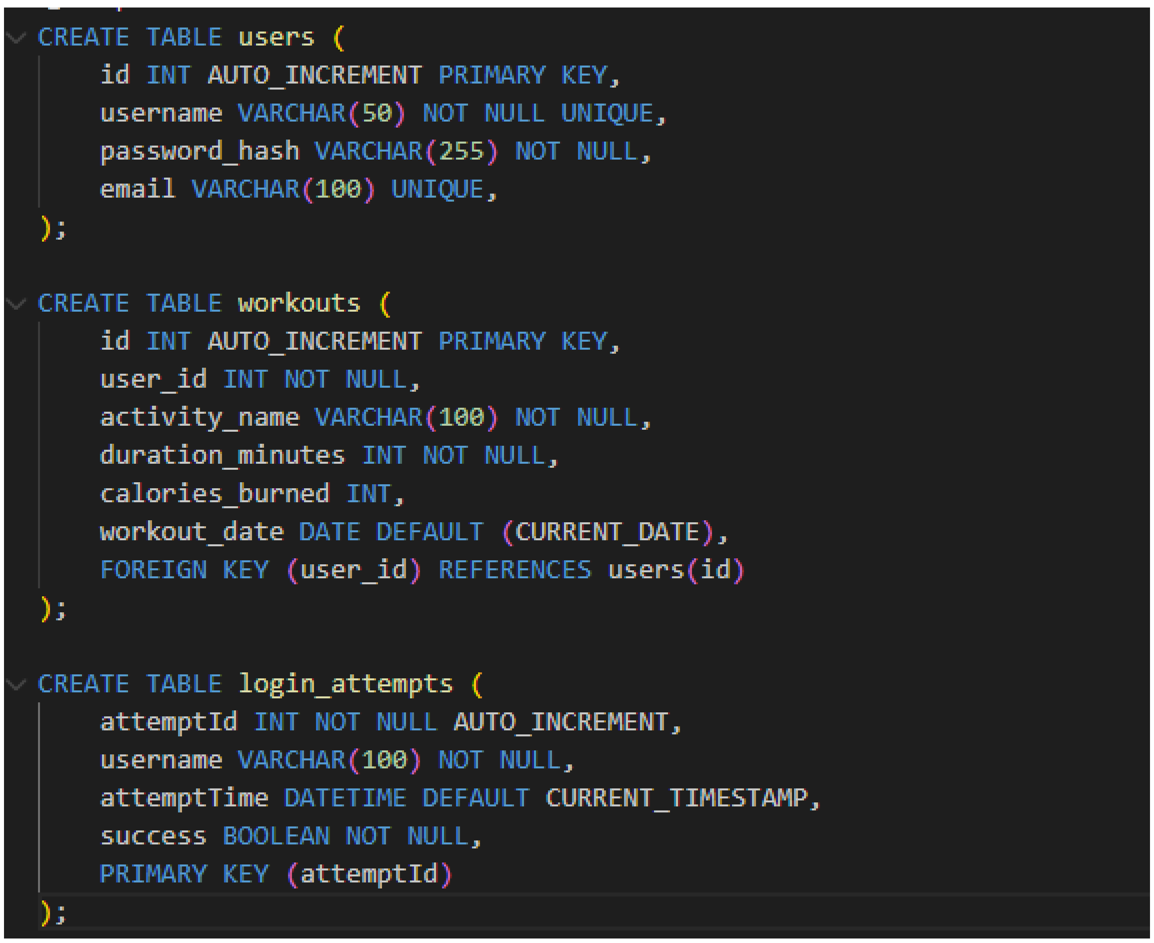
Registered Users

ID	Username	Email
1	**gold**	placeholder@gmail.com
3	**golden**	placeholawdder@gmail.com
4	**Reyes**	reyst005@gmail.com

– Advanced Techniques –

1. Database Normalization and Foreign Key Integrity

To prevent data redundancy and ensure logical consistency, the application employs database normalization by linking the static list of activities to the user's specific workout logs using a Foreign Key.



```sql
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    email VARCHAR(100) UNIQUE,
);

CREATE TABLE workouts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    activity_name VARCHAR(100) NOT NULL,
    duration_minutes INT NOT NULL,
    calories_burned INT,
    workout_date DATE DEFAULT (CURRENT_DATE),
    FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE login_attempts (
    attemptId INT NOT NULL AUTO_INCREMENT,
    username VARCHAR(100) NOT NULL,
    attemptTime DATETIME DEFAULT CURRENT_TIMESTAMP,
    success BOOLEAN NOT NULL,
    PRIMARY KEY (attemptId)
);
```

create_db.sql

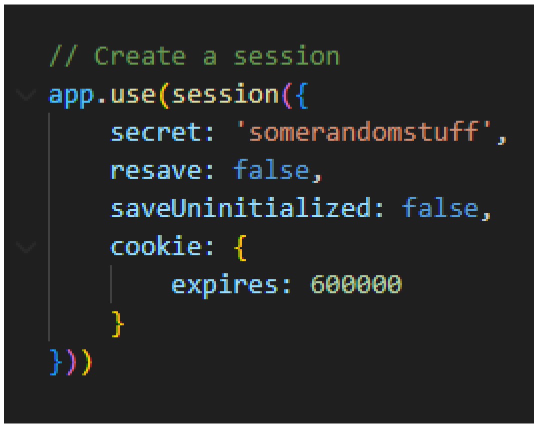
2. Secure Asynchronous Session Management

To prevent blocking the event loop and ensure session data is handled safely, the login process utilizes asynchronous password comparison bcrypt.compare and a session audit log with nested asynchronous callbacks.

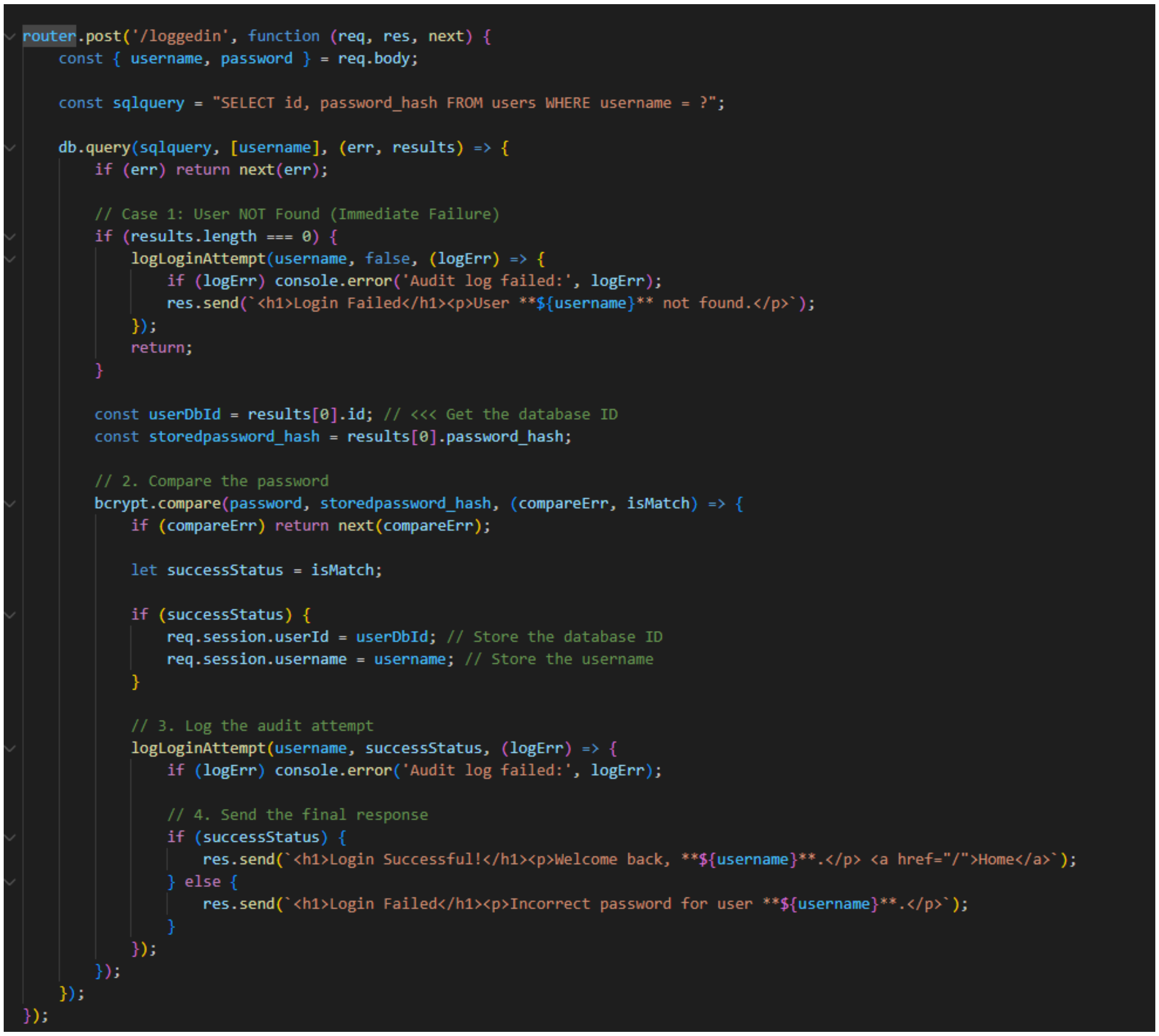
Demonstration:

The`router.post('/loggedin', function (req, res, next)`function handles sequential asynchronous tasks:

1. Async database query db.query to retrieve the hashed password.
2. Async bcrypt comparison bcrypt.comapre to verify the password.
3. Async database insert loginattempt to record the audit log *before* sending the final response.



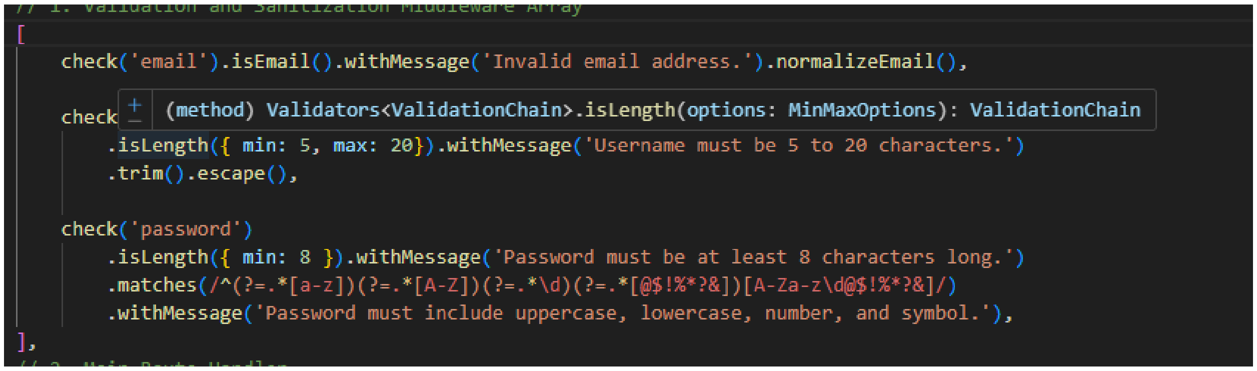
```
// Create a session
app.use(session({
    secret: 'somerandomstuff',
    resave: false,
    saveUninitialized: false,
    cookie: {
        expires: 600000
    }
}))
```



```
router.post('/loggedin', function (req, res, next) {
    const { username, password } = req.body;

    const sqlquery = "SELECT id, password_hash FROM users WHERE username = ?";

    db.query(sqlquery, [username], (err, results) => {
        if (err) return next(err);

        // Case 1: User NOT Found (Immediate Failure)
        if (results.length === 0) {
            logLoginAttempt(username, false, (logErr) => {
                if (logErr) console.error('Audit log failed:', logErr);
                res.send(`<h1>Login Failed</h1><p>User **${username}** not found.</p>`);
            });
            return;
        }

        const userDbId = results[0].id; // <<< Get the database ID
        const storedpassword_hash = results[0].password_hash;

        // 2. Compare the password
        bcrypt.compare(password, storedpassword_hash, (compareErr, isMatch) => {
            if (compareErr) return next(compareErr);

            let successStatus = isMatch;

            if (successStatus) {
                req.session.userId = userDbId; // Store the database ID
                req.session.username = username; // Store the username
            }

            // 3. Log the audit attempt
            logLoginAttempt(username, successStatus, (logErr) => {
                if (logErr) console.error('Audit log failed:', logErr);

                // 4. Send the final response
                if (successStatus) {
                    res.send(`<h1>Login Successful!</h1><p>Welcome back, **${username}**.</p> <a href="/">Home</a>`);
                } else {
                    res.send(`<h1>Login Failed</h1><p>Incorrect password for user **${username}**.</p>`);
                }
            });
        });
    });
});
```

3. Advanced Sanitization and validation:

This technique involves checking, cleaning, and restricting all user input at multiple points.Specifically in the Express middleware before it ever touches sensitive components like the database or the user session. This prevents numerous common web vulnerabilities.Validation ensures the input conforms to your business logic and, more importantly, the constraints of your database schema.By placing the checks in the middleware array, you ensure the business logic only executes once the data has passed every security and integrity check. This reduces the risk of errors and vulnerabilities deep within your code.



```
// 1. Validation and Sanitization Middleware Array
[
    check('email').isEmail().withMessage('Invalid email address.').normalizeEmail(),

    check (method) Validators<ValidationChain>.isLength(options: MinMaxOptions): ValidationChain
        .isLength({ min: 5, max: 20}).withMessage('Username must be 5 to 20 characters.')
        .trim().escape(),

    check('password')
        .isLength({ min: 8 }).withMessage('Password must be at least 8 characters long.')
        .matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]/)
        .withMessage('Password must include uppercase, lowercase, number, and symbol.'),
],
```

– AI Déclaration –

I acknowledge the use of chatgpt(ChatGP)to generate ideas. The prompts used include show me an example website about health or fitness. The output from these prompts was used to get a general idea of what a health and fitness app looks like.