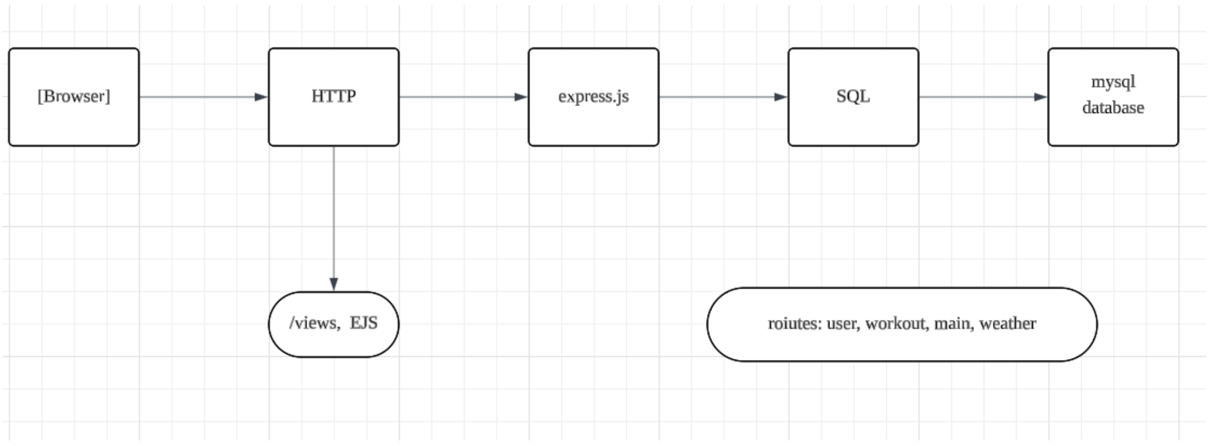
Outline:

Using Node.js, Express, MySQL, and EJS templating, the Health App is a web-based fitness tracking program. Users can add, amend, remove, and search exercises by kind or calories burned in addition to registering, logging in, and managing their workouts. An audit table records user login attempts, giving administrators information on both successful and unsuccessful logins. Bcrypt is used to hash passwords, express-sanitiser is used to sanitise input, and prepared SQL statements are used to stop injection attempts. Sessions control authentication and properly reroute users who are not authorised. To show local weather conditions, the system incorporates weather capabilities to see if conditions are suitable to run.

Architecture:

The app has a **three-tier architecture**:

- Presentation Layer (Client): The client renders HTML using EJS (Embedded JavaScript) templates, HTML5, and CSS. The client operates on the end user's side, requesting information from the server via HTTP.
- **Application tier**: Node.js with Express handles routing, session management, authentication, business logic, and dynamic HTML rendering via EJS. Middleware ensures input validation and access control.
- **Data tier**: MySQL stores user credentials, workouts, and audit logs. The app communicates with the database using the mysql2 library with prepared statements.



Data Model:

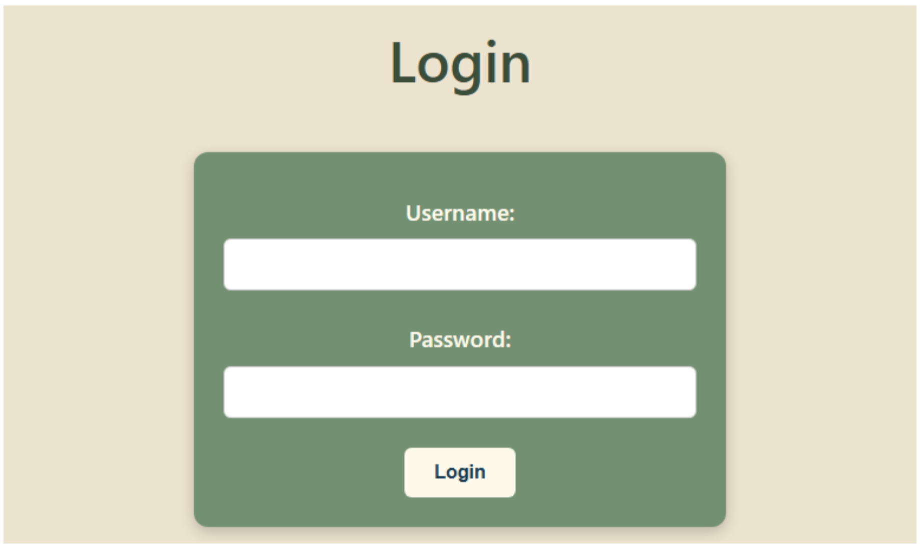
The database contains three main tables:

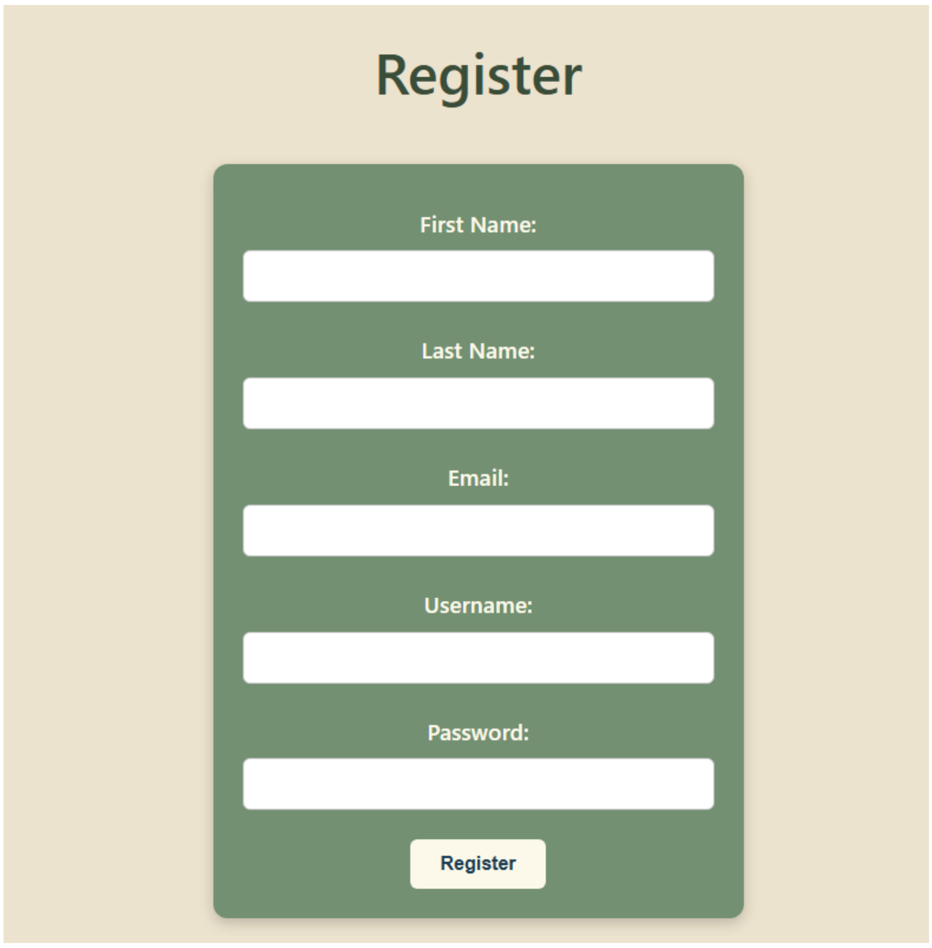
- **users**: Stores user credentials and personal details.
- **workouts**: Stores user workouts with type, duration, calories, and date.
- **audit_log**: Records login attempts with timestamp, IP address, and user agent.

User Facing Functionality:

Registration and Login

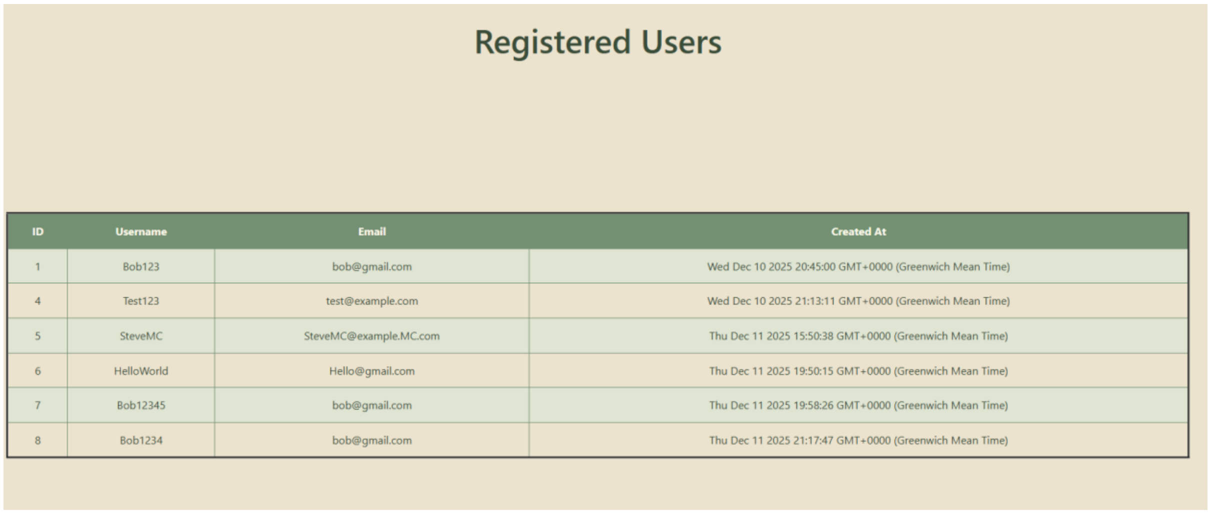
Users register via /users/register, entering username, email, first name, last name, and password. Passwords are hashed using bcrypt. Login occurs at /users/login. The session stores the username upon successful login. Failed attempts are logged in audit_log.



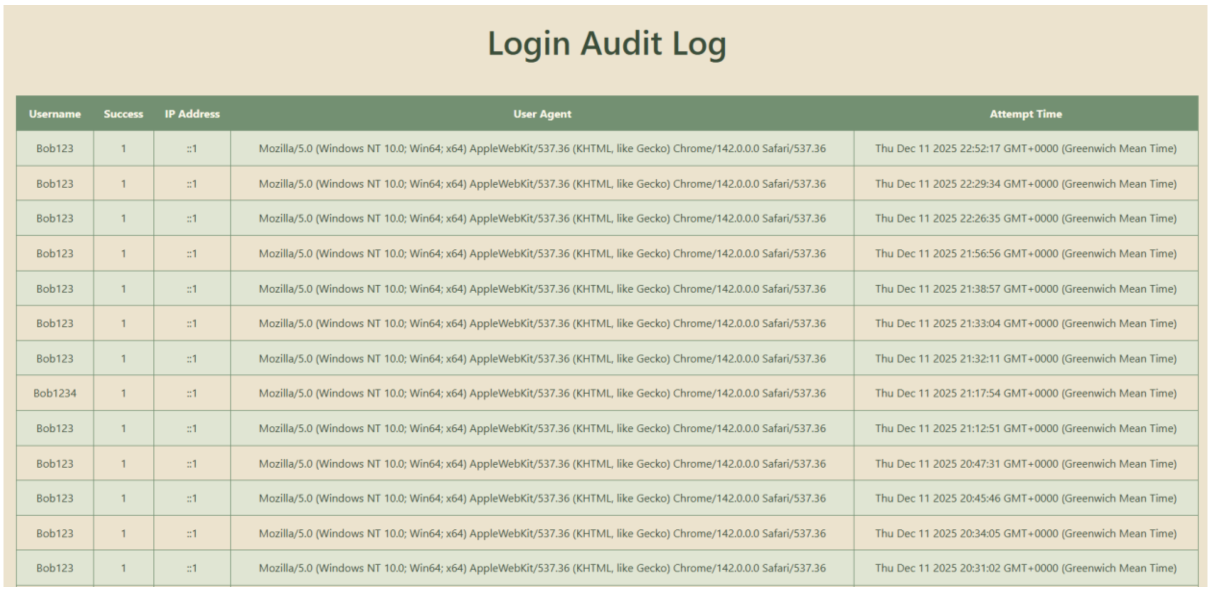


User List and Audit Log

Authenticated admins can access /users/list to see registered users and /users/audit to view login attempts, including IP addresses and success/failure status.


Registered Users

ID	Username	Email	Created At
1	Bob123	bob@gmail.com	Wed Dec 10 2025 20:45:00 GMT+0000 (Greenwich Mean Time)
4	Test123	test@example.com	Wed Dec 10 2025 21:13:11 GMT+0000 (Greenwich Mean Time)
5	SteveMC	SteveMC@example.MC.com	Thu Dec 11 2025 15:50:38 GMT+0000 (Greenwich Mean Time)
6	HelloWorld	Hello@gmail.com	Thu Dec 11 2025 19:50:15 GMT+0000 (Greenwich Mean Time)
7	Bob12345	bob@gmail.com	Thu Dec 11 2025 19:58:26 GMT+0000 (Greenwich Mean Time)
8	Bob1234	bob@gmail.com	Thu Dec 11 2025 21:17:47 GMT+0000 (Greenwich Mean Time)



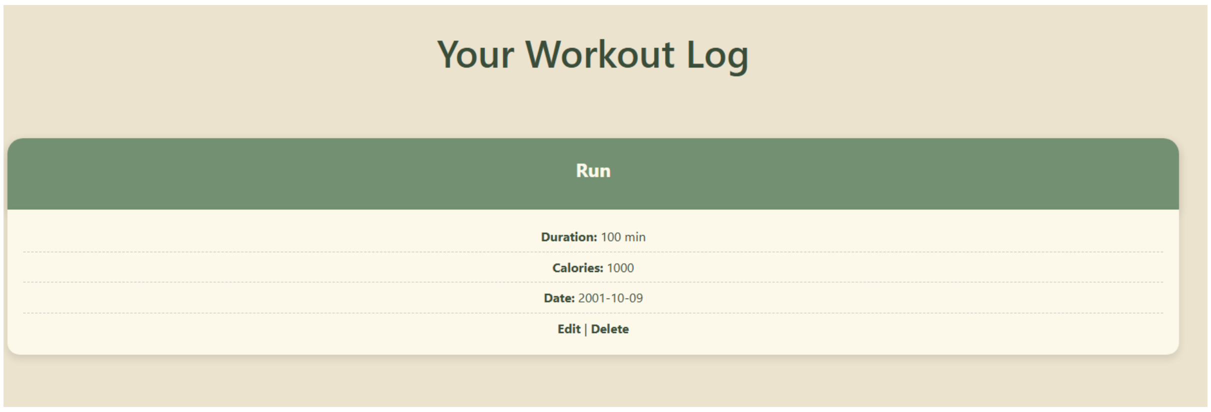
Login Audit Log

Username	Success	IP Address	User Agent	Attempt Time
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 22:52:17 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 22:29:34 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 22:26:35 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 21:56:56 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 21:38:57 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 21:33:04 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 21:32:11 GMT+0000 (Greenwich Mean Time)
Bob1234	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 21:17:54 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 21:12:51 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 20:47:31 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 20:45:46 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 20:34:05 GMT+0000 (Greenwich Mean Time)
Bob123	1	::1	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36	Thu Dec 11 2025 20:31:02 GMT+0000 (Greenwich Mean Time)

Workout Management

Authenticated users can:

1. **View Workout Log**: /workout/log displays workouts in descending order by date.



Your Workout Log

Run

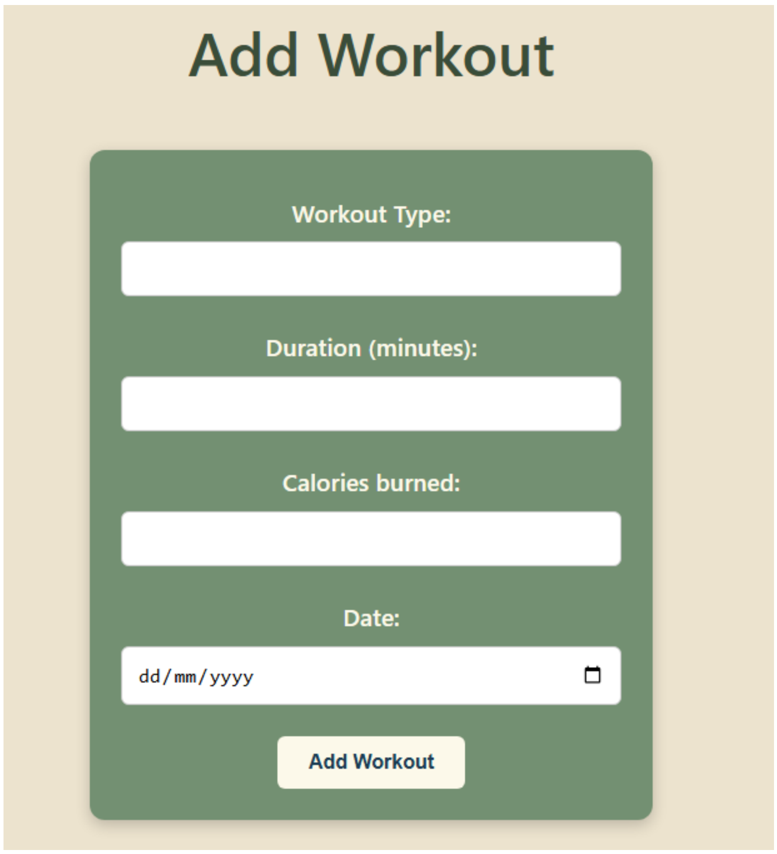
Duration: 100 min

Calories: 1000

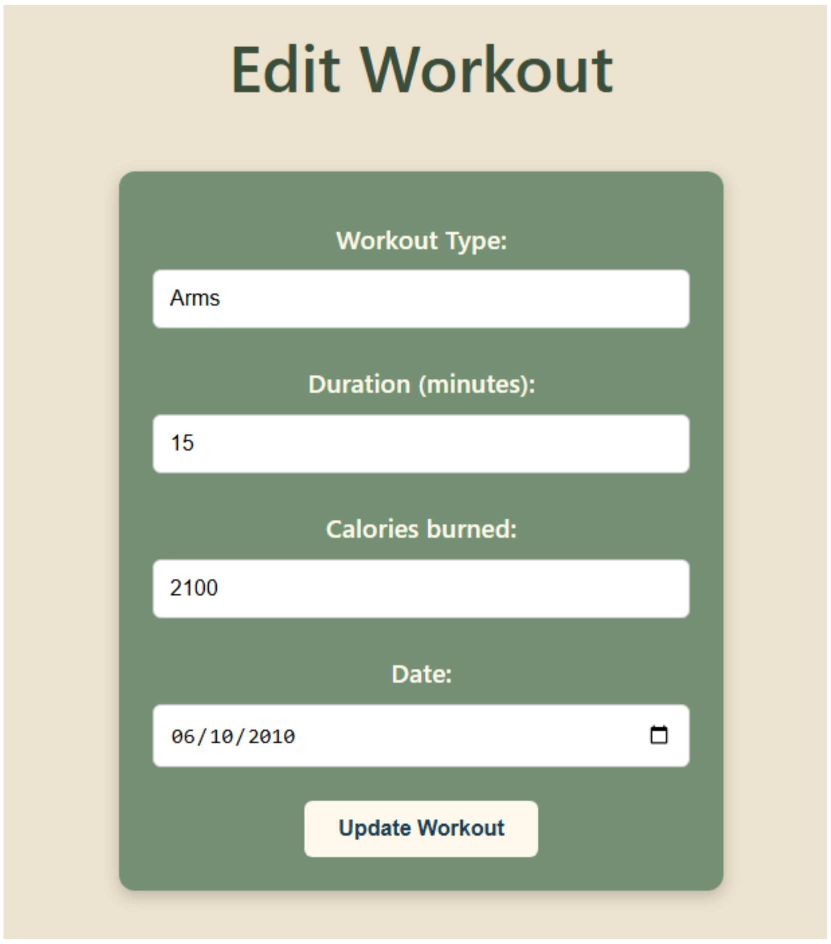
Date: 2001-10-09

Edit | **Delete**

2. **Add Workout**: /workout/add provides a form to enter type, duration, calories, and date. Inputs are validated.



3. **Edit Workout**: /workout/edit/:id allows users to modify workouts they own.



Edit Workout

Workout Type:

Arms

Duration (minutes):

15

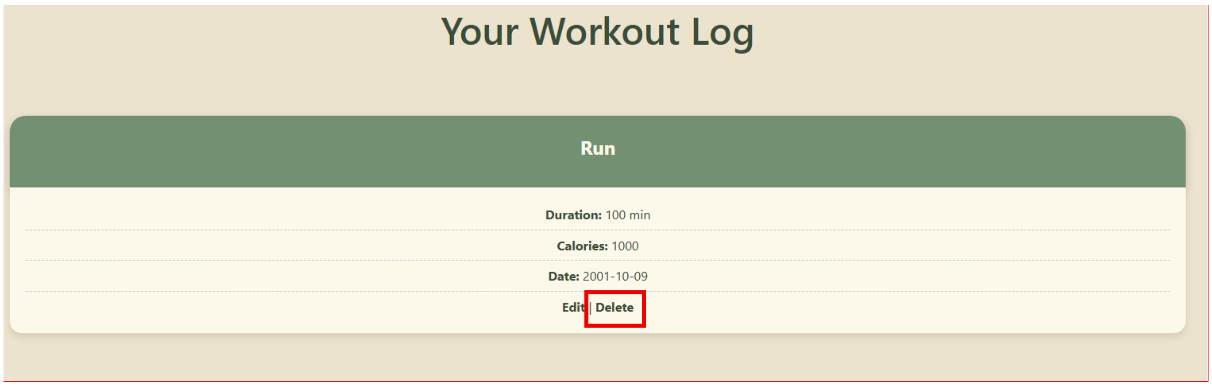
Calories burned:

2100

Date:

06/10/2010

4. **Delete Workout**: /workout/delete/:id removes a workout. (red box shows delete button)



Your Workout Log

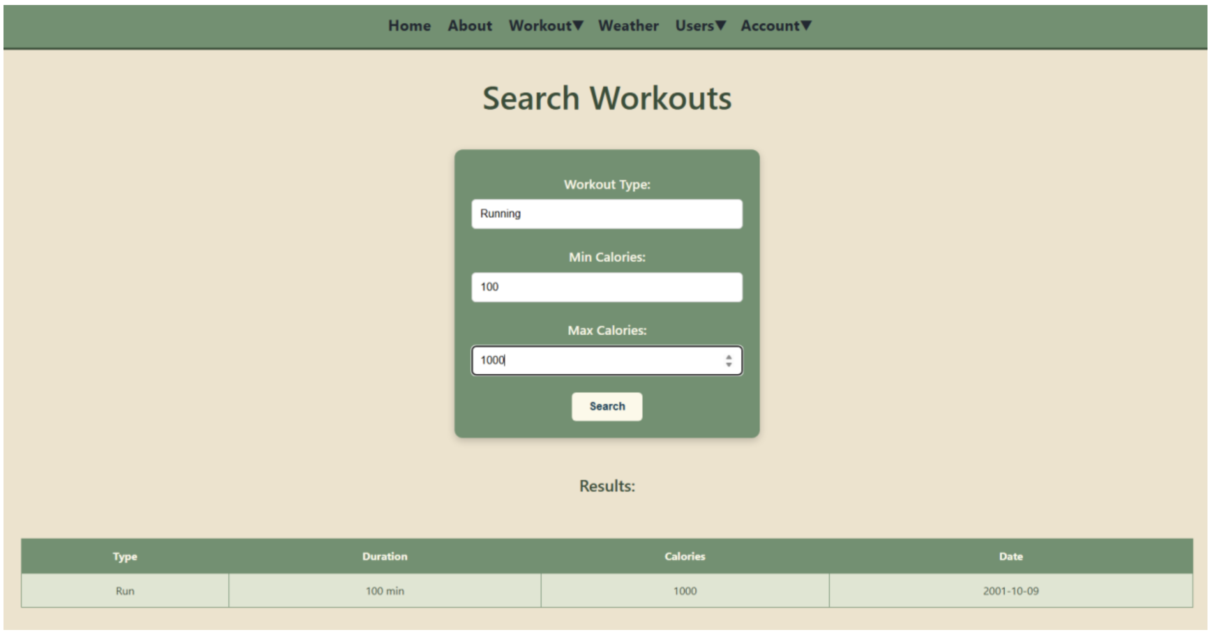
Run

Duration: 100 min

Calories: 1000

Date: 2001-10-09

5. **Search Workouts**: /workout/search filters workouts by type and calories.



Search Workouts

Workout Type: Running

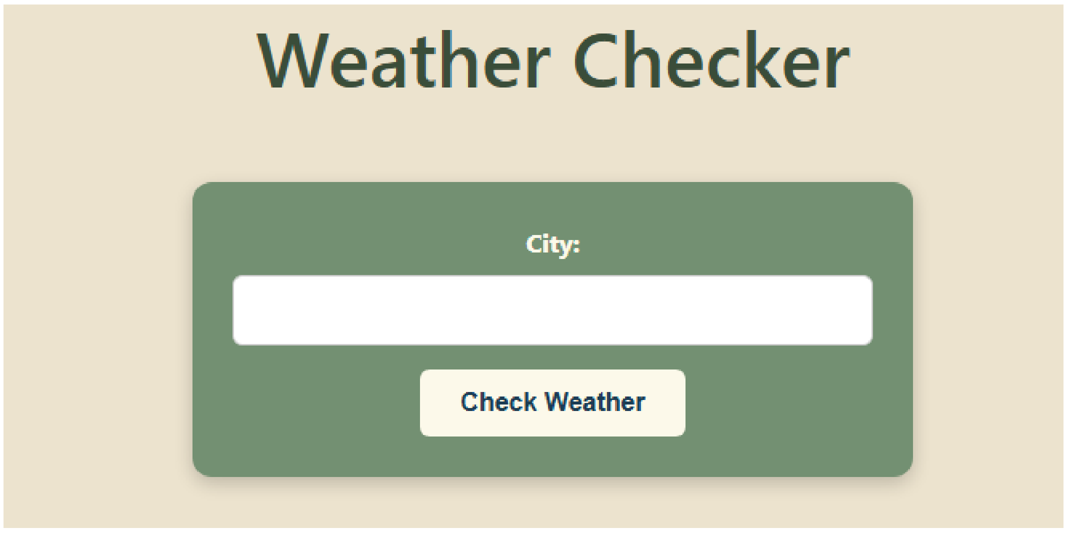
Min Calories: 100

Max Calories: 1000

Results:

Type	Duration	Calories	Date
Run	100 min	1000	2001-10-09

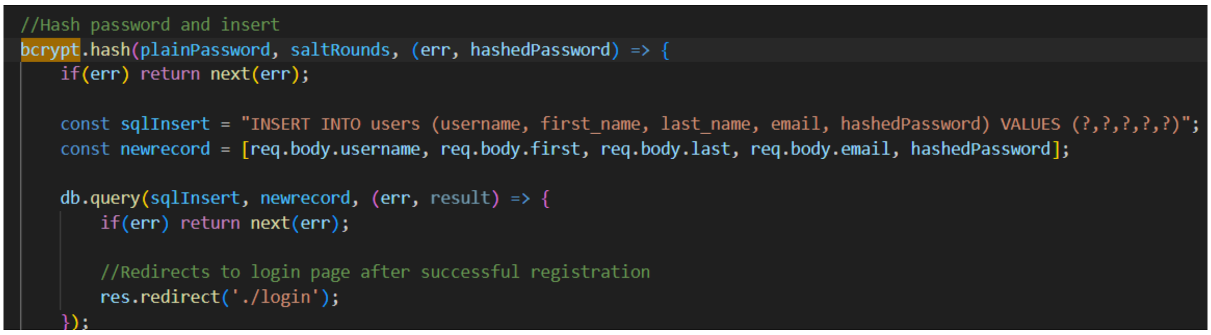
6. **Weather Conditions:** /weather shows weather conditions are perfect for running



Advanced Techniques:

1. Input Validation and Security

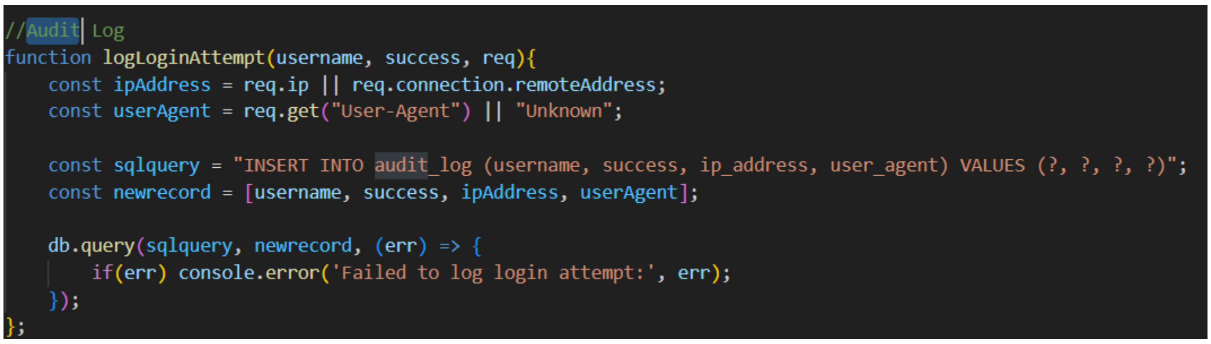
All user input is sanitised with express-sanitiser and validated using express-validator:



```
//Hash password and insert
bcrypt.hash(plainPassword, saltRounds, (err, hashedPassword) => {
    if(err) return next(err);

    const sqlInsert = "INSERT INTO users (username, first_name, last_name, email, hashedPassword) VALUES (?,?,?,?,?)";
    const newrecord = [req.body.username, req.body.first, req.body.last, req.body.email, hashedPassword];

    db.query(sqlInsert, newrecord, (err, result) => {
        if(err) return next(err);

        //Redirects to login page after successful registration
        res.redirect('./login');
    });
```

2. Audit logging

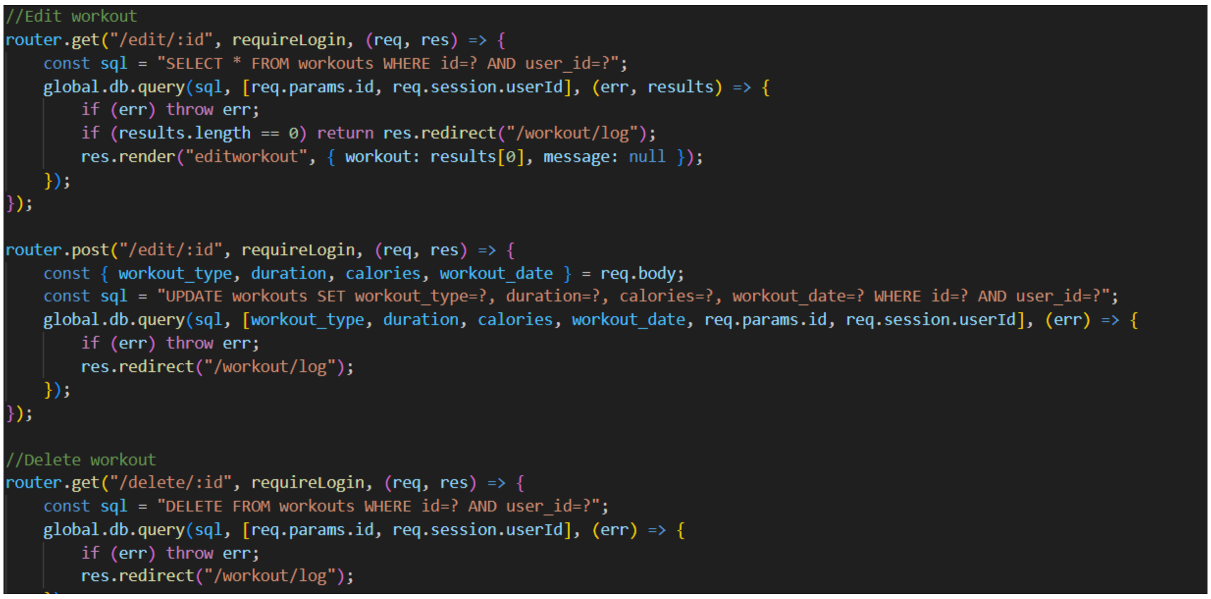
Automatically records users login



```
//Audit Log
function logLoginAttempt(username, success, req){
    const ipAddress = req.ip || req.connection.remoteAddress;
    const userAgent = req.get("User-Agent") || "Unknown";

    const sqlquery = "INSERT INTO audit_log (username, success, ip_address, user_agent) VALUES (?, ?, ?, ?)";
    const newrecord = [username, success, ipAddress, userAgent];

    db.query(sqlquery, newrecord, (err) => {
        if(err) console.error('Failed to log login attempt:', err);
    });
};
```

3. Delete/Edit Workout:

The application allows users to edit and delete workouts directly from their workout log. When a user selects Edit for a specific workout, the app fetches the workout data from the database using a prepared



```
//Edit workout
router.get("/edit/:id", requireLogin, (req, res) => {
    const sql = "SELECT * FROM workouts WHERE id=? AND user_id=?";
    global.db.query(sql, [req.params.id, req.session.userId], (err, results) => {
        if (err) throw err;
        if (results.length == 0) return res.redirect("/workout/log");
        res.render("editworkout", { workout: results[0], message: null });
    });
});

router.post("/edit/:id", requireLogin, (req, res) => {
    const { workout_type, duration, calories, workout_date } = req.body;
    const sql = "UPDATE workouts SET workout_type=?, duration=?, calories=?, workout_date=? WHERE id=? AND user_id=?";
    global.db.query(sql, [workout_type, duration, calories, workout_date, req.params.id, req.session.userId], (err) => {
        if (err) throw err;
        res.redirect("/workout/log");
    });
});

//Delete workout
router.get("/delete/:id", requireLogin, (req, res) => {
    const sql = "DELETE FROM workouts WHERE id=? AND user_id=?";
    global.db.query(sql, [req.params.id, req.session.userId], (err) => {
        if (err) throw err;
        res.redirect("/workout/log");
```

After successfully completing both edit and delete procedures, users are redirected back to the exercise log, offering prompt feedback and preserving a seamless user experience. These capabilities, which are implemented in routes/workout.js and backed by the associated EJS templates (editworkout.ejs), show dynamic route handling, safe database transactions, and user-specific access control.