MoveMore Application Documentation

Student ID: 33791129
Module: Dynamic Web Applications

Outline

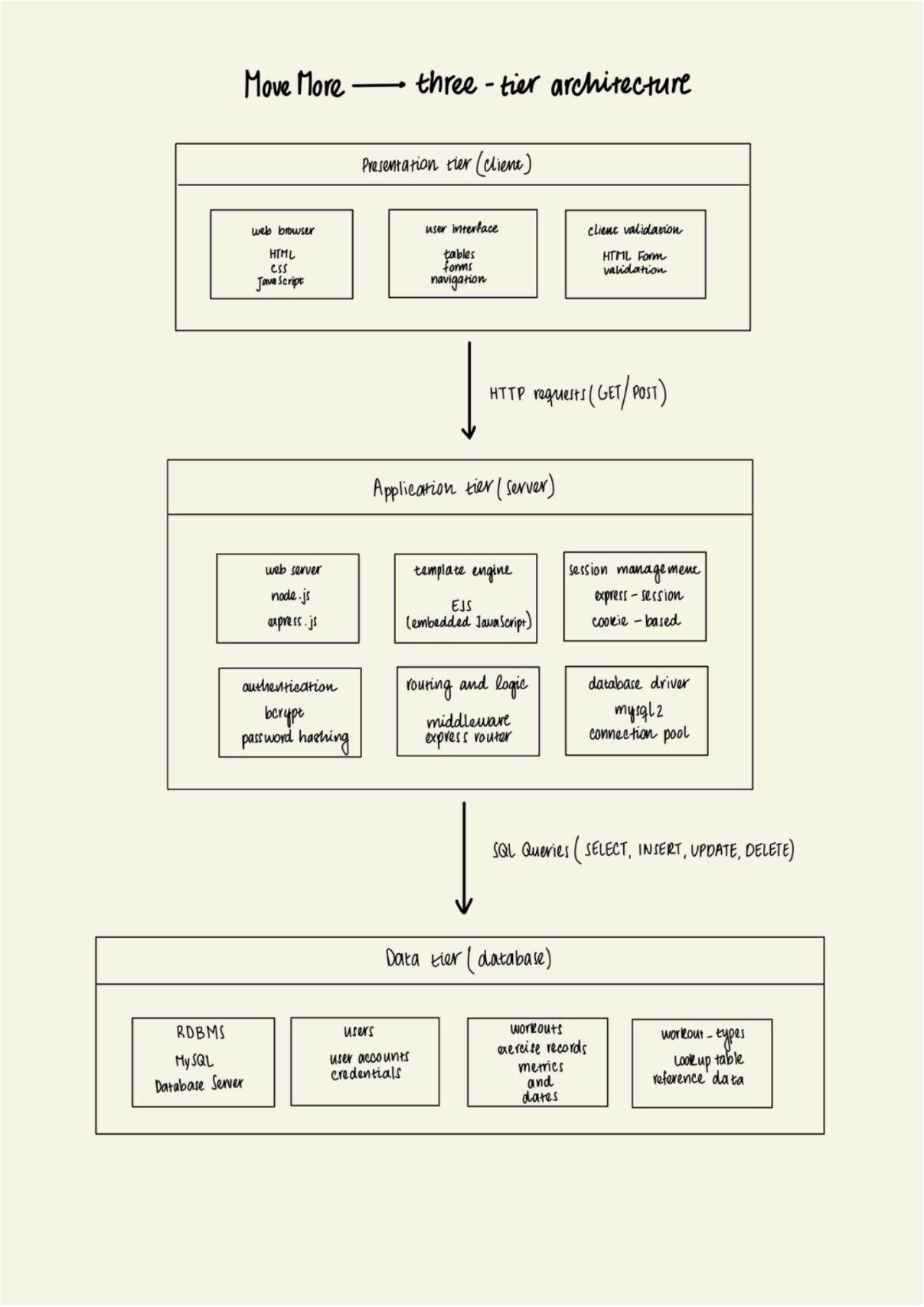
MoveMore is a web-based fitness tracking application that helps users monitor and analyze their workout activities. It enables individuals to create an account and log different types of exercises, including duration, distance, calories burned, and personal notes. By storing this information, the application gives users a clear and organized way to track their fitness habits.

Users can review their workout history, search through past entries, and access detailed statistics displayed on their dashboard. These insights help users understand their progress over time, identify trends, and stay motivated in pursuing their health and fitness goals.

The application addresses the common challenge of maintaining consistent fitness records by offering a simple, centralized platform for logging exercise data. With a focus on usability and reliable data persistence, MoveMore supports users in managing their fitness journey and achieving long-term personal improvement.

Architecture

Diagram



Description

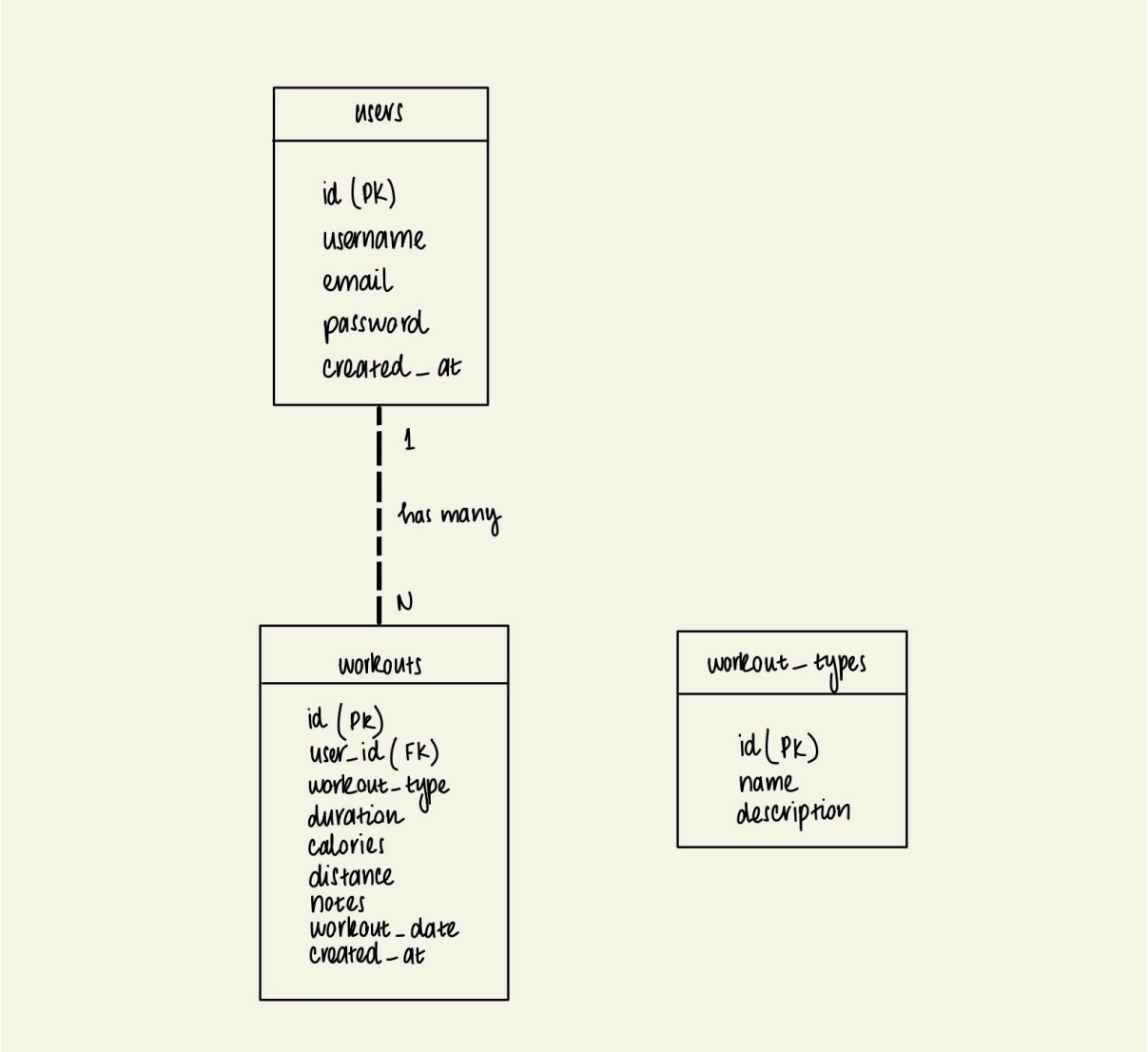
MoveMore uses a basic three-tier setup: the client, the server, and the database. The client side is just the user's browser, which loads the HTML, CSS, and JavaScript that make up the pages. This is where people fill in forms, click buttons, and record their workouts or check their stats.

The server side runs on Node.js with Express handling the routes and general logic. EJS is used to generate the pages that need dynamic content. I used express-session to keep track of logged-in users, and bcrypt for hashing passwords. The server deals with things like logging in, checking form input, adding and editing data, keeping sessions active, and running searches. Middleware is used to stop users from accessing certain pages unless they're signed in.

The database is a MySQL setup with three tables: users, workouts, and workout_types. These hold login details, workout entries, and the list of activity categories. The server sends SQL queries to insert data, look things up, run searches, and get totals for stats. All three parts work together: the browser sends requests to the server, and the server passes information to and from the database.

Data Model

Diagram

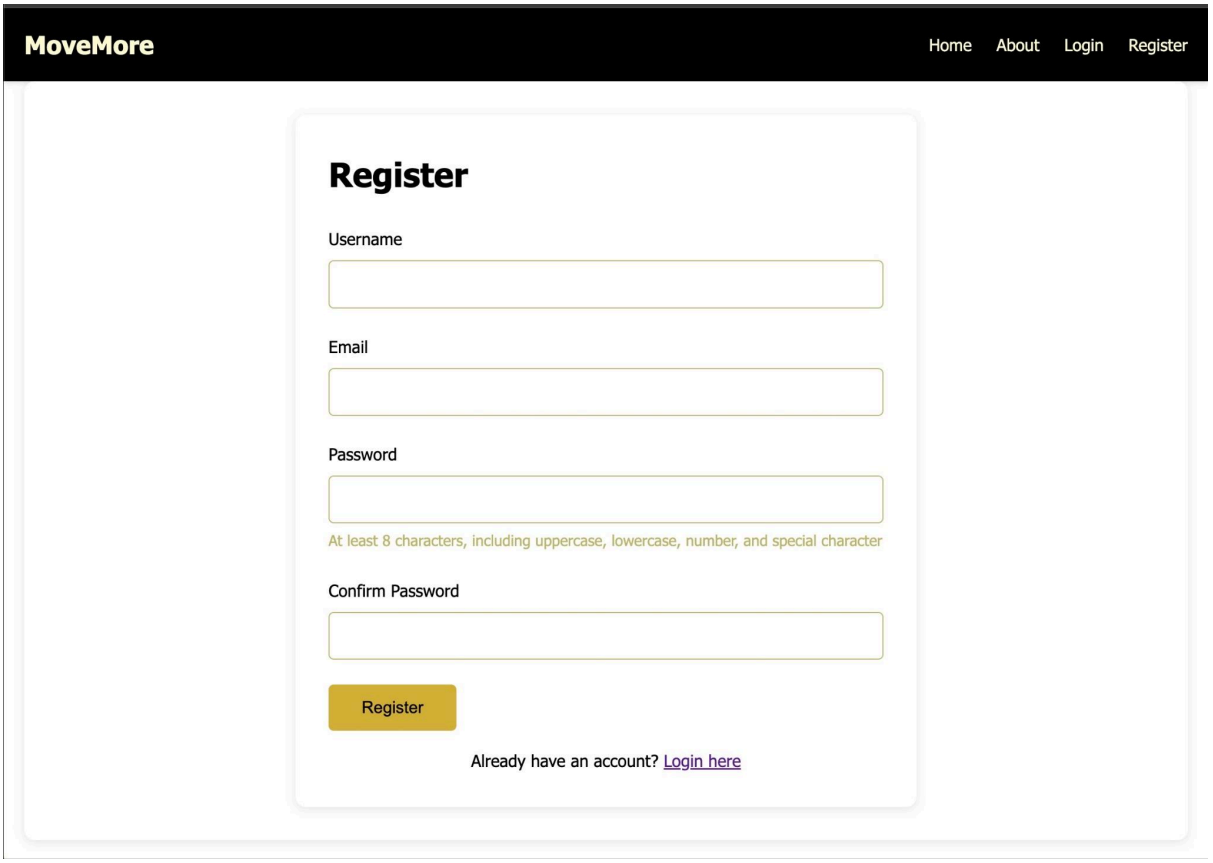


Description

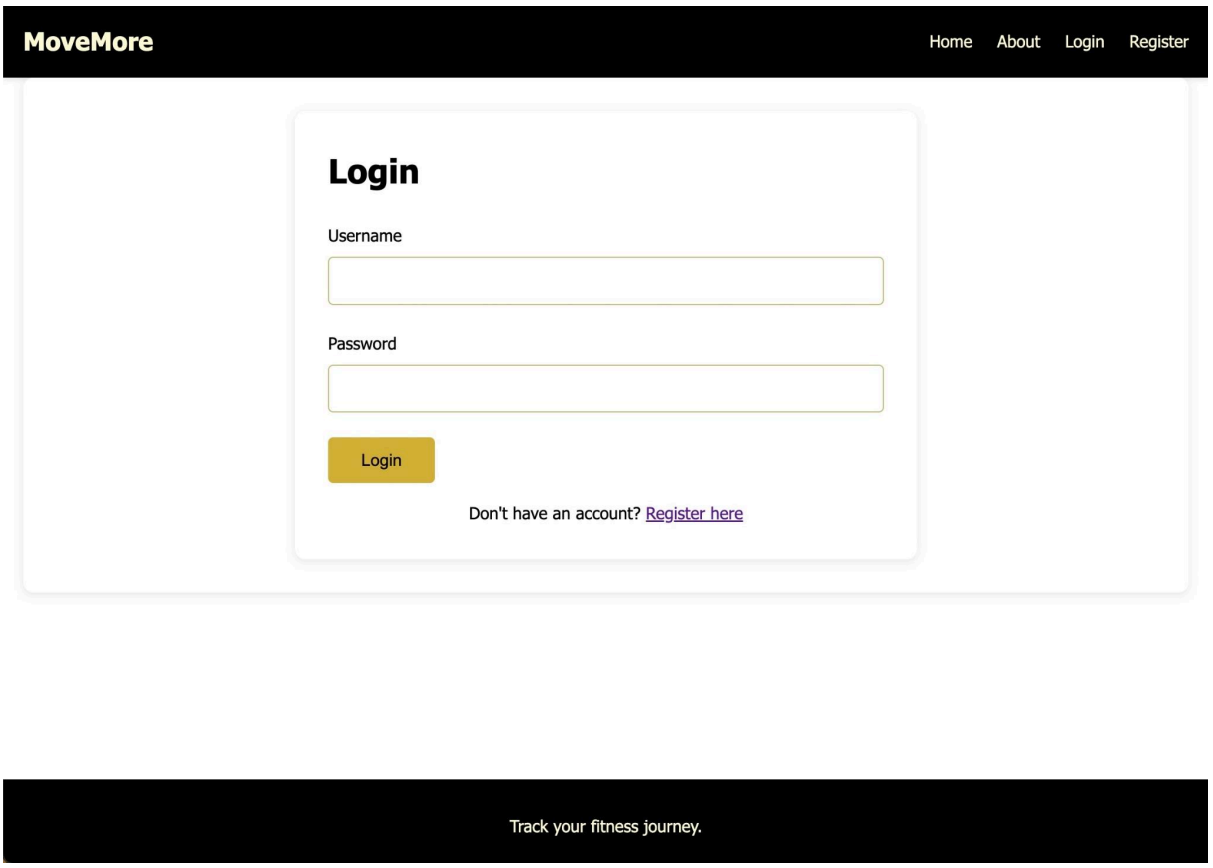
The database schema comprises three normalized tables. The users table stores account credentials with auto-incrementing primary key, unique constraints on username and email, and bcrypt-hashed passwords. The workouts table maintains exercise records with a foreign key (user_id) referencing users, establishing a one-to-many relationship where each user owns multiple workouts. It includes metrics for duration, distance, calories, notes, and workout date. CASCADE deletion ensures removing a user deletes associated workouts. The workout_types table provides a reference lookup for valid workout categories. All primary keys are auto-incrementing integers. Timestamps track record creation for audit purposes.

User Functionality

User Registration and Authentication

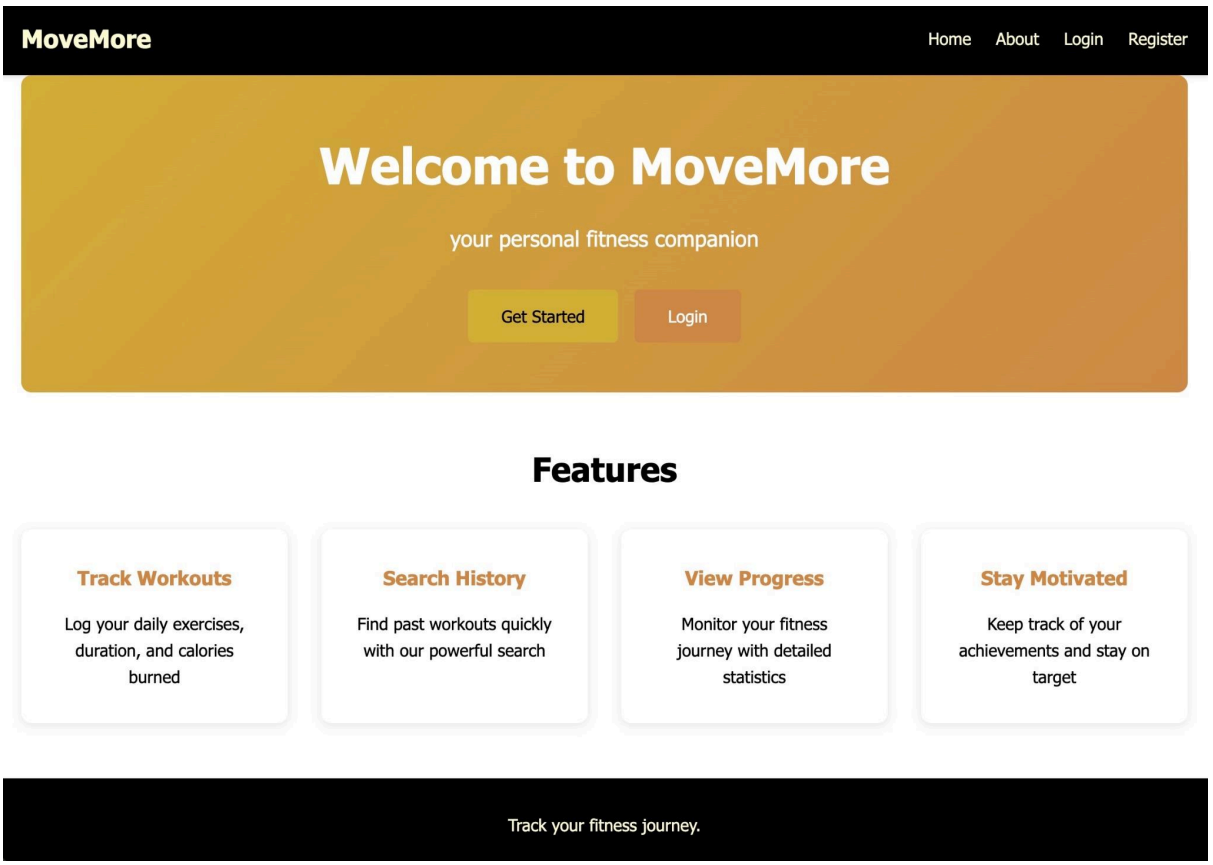


Users can create accounts through the registration page by providing a username, email, and password. Passwords must meet security requirements, including a minimum of eight characters with at least one uppercase letter, one lowercase letter, one number, and one special character. For added security, passwords are hashed using bcrypt before being stored in the database.

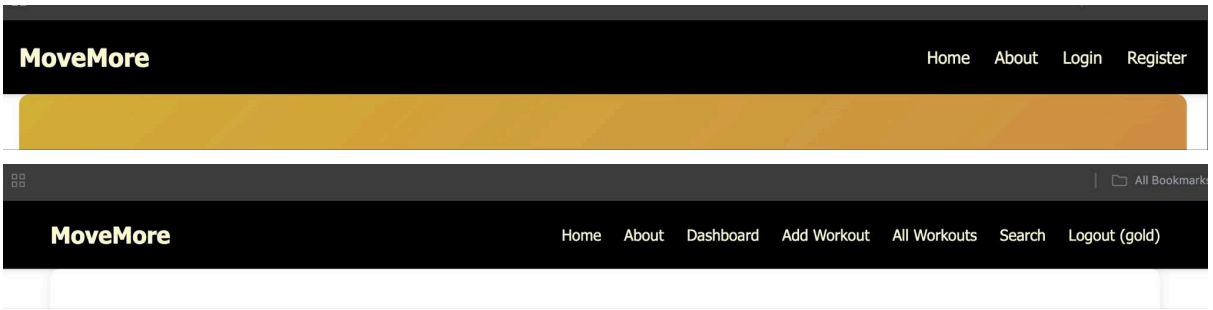


The login system authenticates users by verifying credentials against the database and establishes secure sessions for authorized access. Any attempt to access protected pages without authentication automatically redirects users to the login page.

Home Page and Navigation

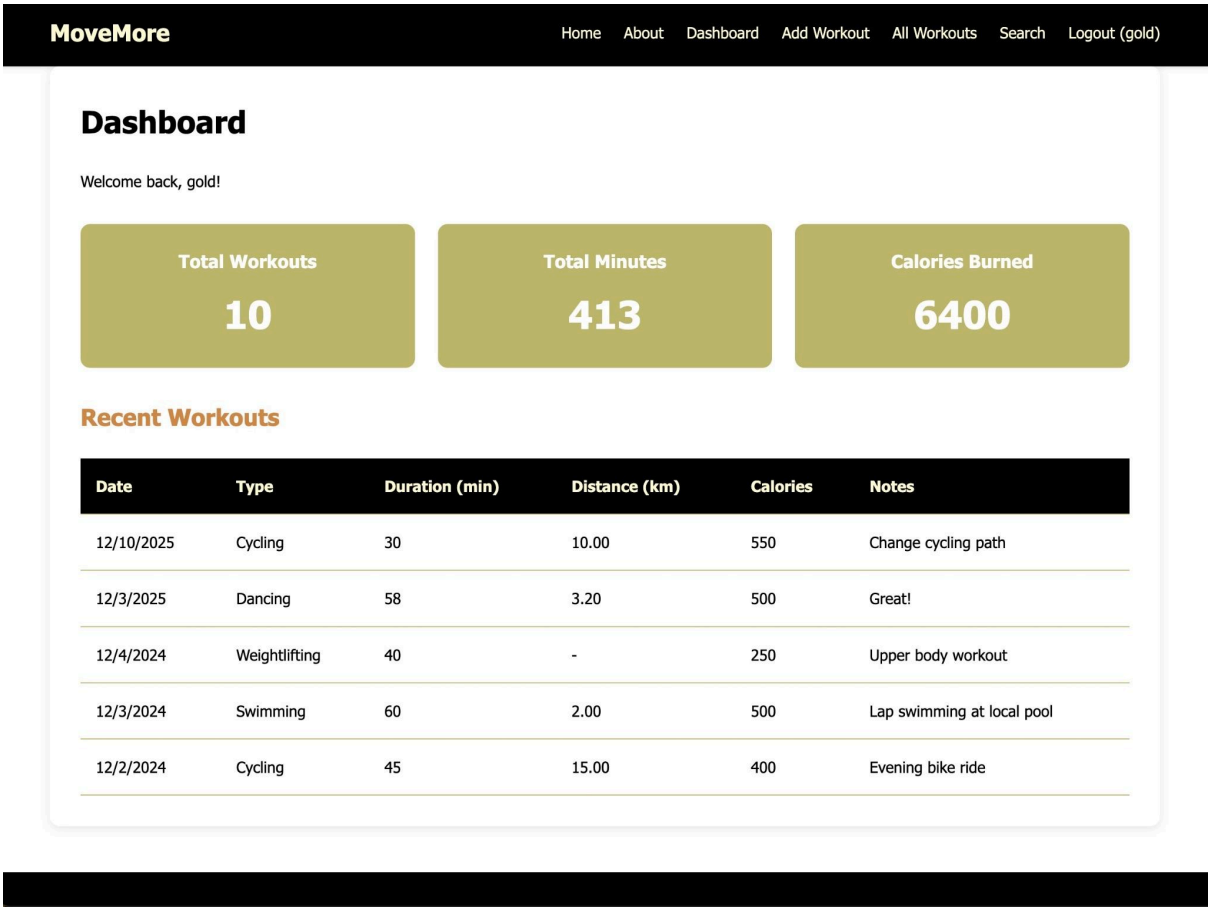


The home page offers an overview of the application's features and includes clear call-to-action buttons that allow users to register or log in.



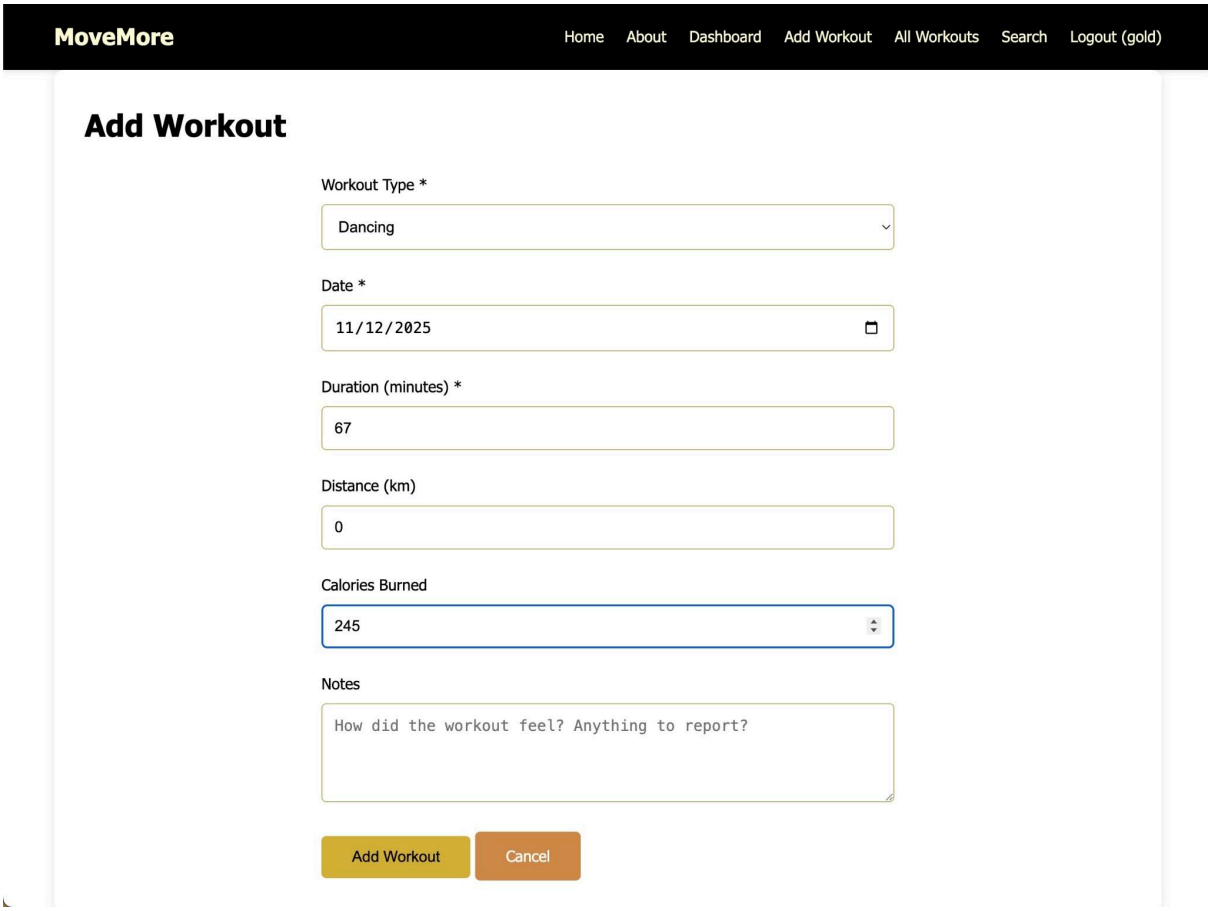
The navigation menu changes depending on whether the user is logged in. Guests see options for Login and Register, while logged-in users see links for Dashboard, Add Workout, All Workouts, Search, and Logout.

Dashboard



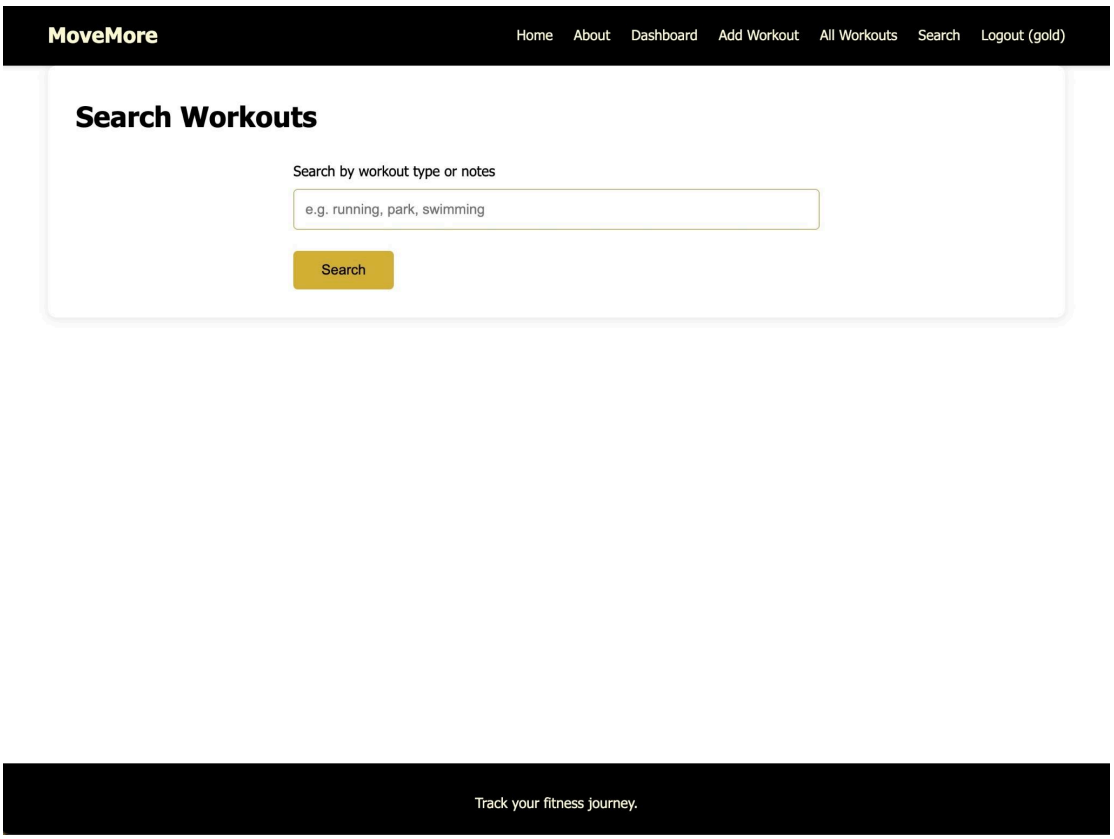
After logging in, the dashboard is the main page users see. It shows three basic stats: total workouts, total minutes exercised, and total calories burned, which are calculated using SQL queries. Below the stats, there's a table listing the five most recent workouts, including the date, type, duration, distance, calories, and any notes. This lets users quickly see their recent activity and how they're doing overall.

Adding Workouts

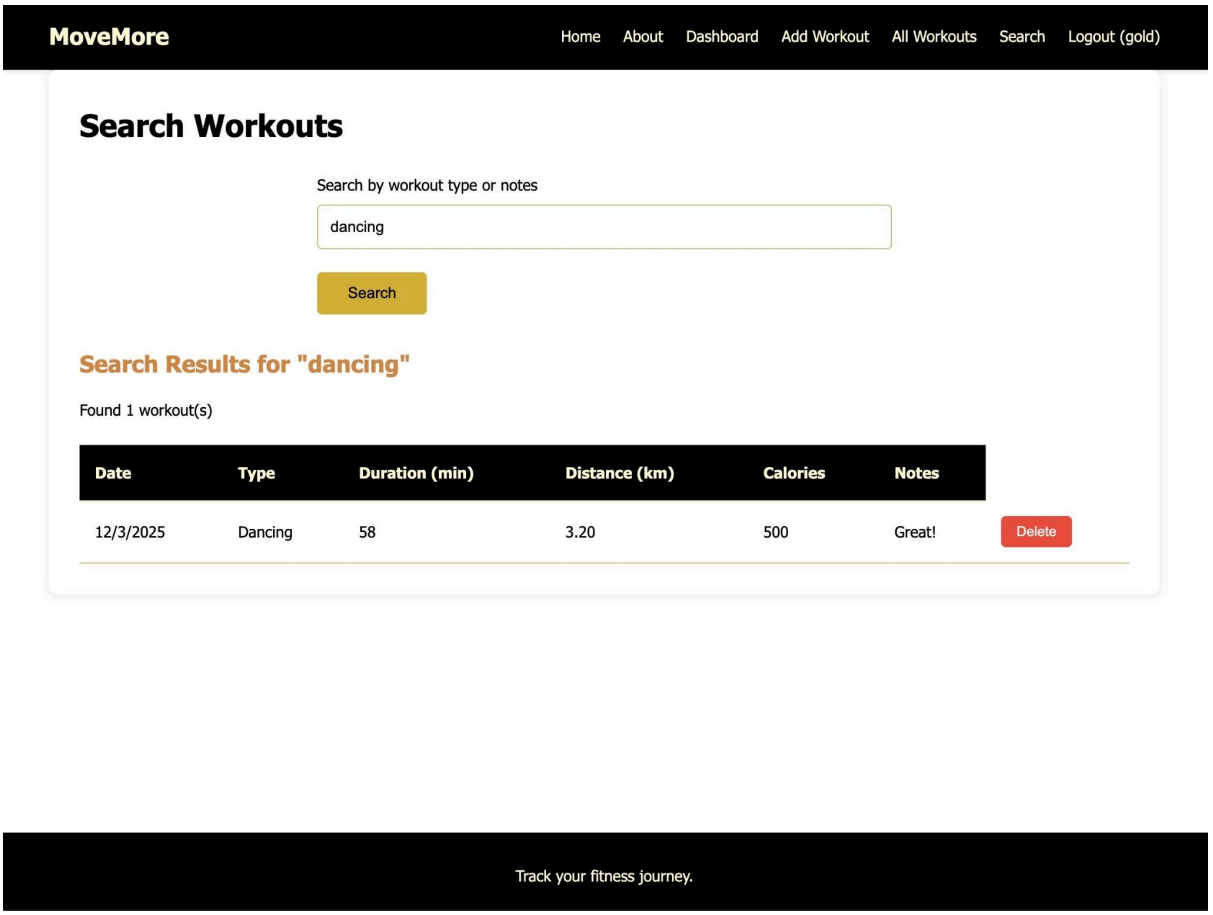


The add workout form lets users log new exercises. Users must select a workout type from a list, pick a date (future dates aren't allowed), and enter the duration in minutes. They can also add distance, calories burned, and notes if they want. The form checks the data both in the browser and on the server. Once submitted, the workout is saved in the database and users are taken back to their updated dashboard.

Search Functionality

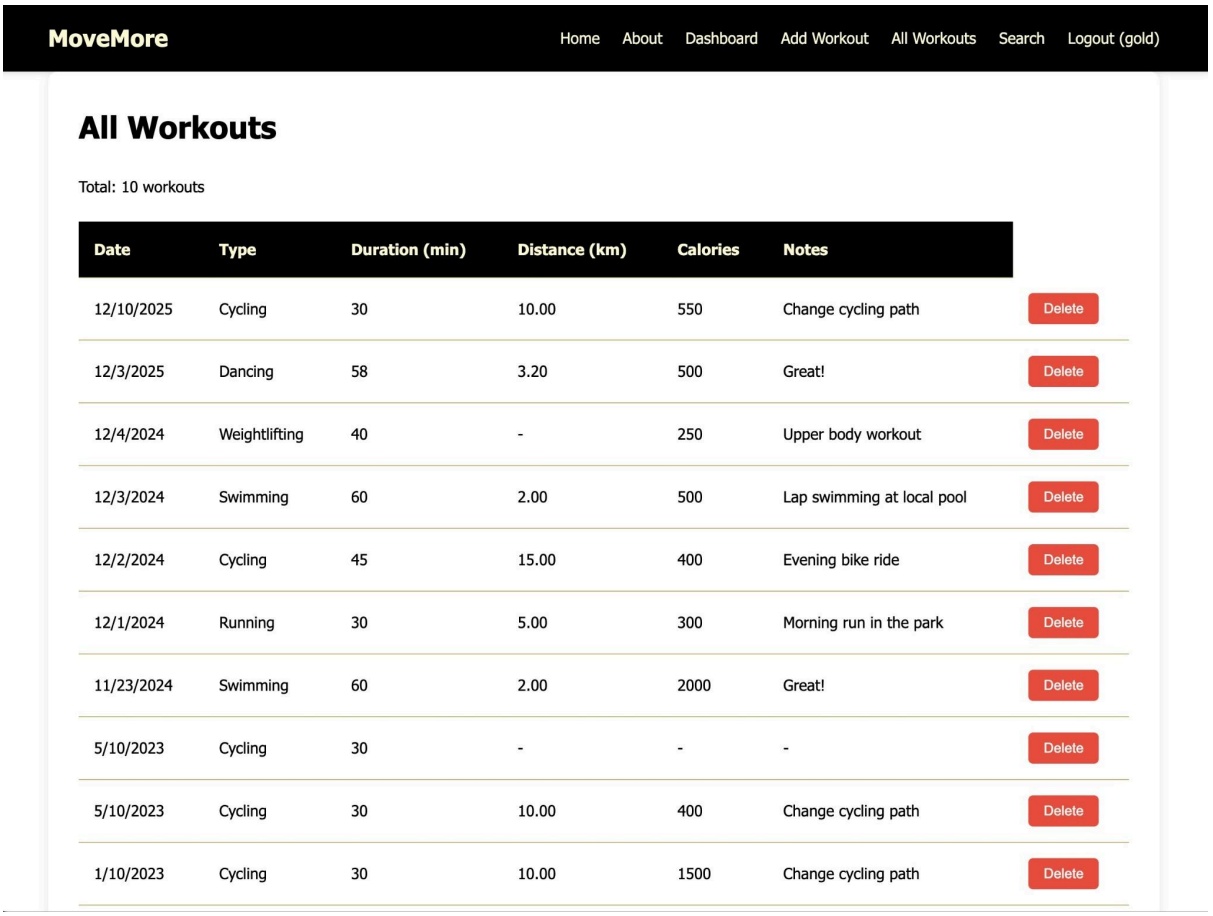


The search feature enables users to find specific workouts by entering keywords.



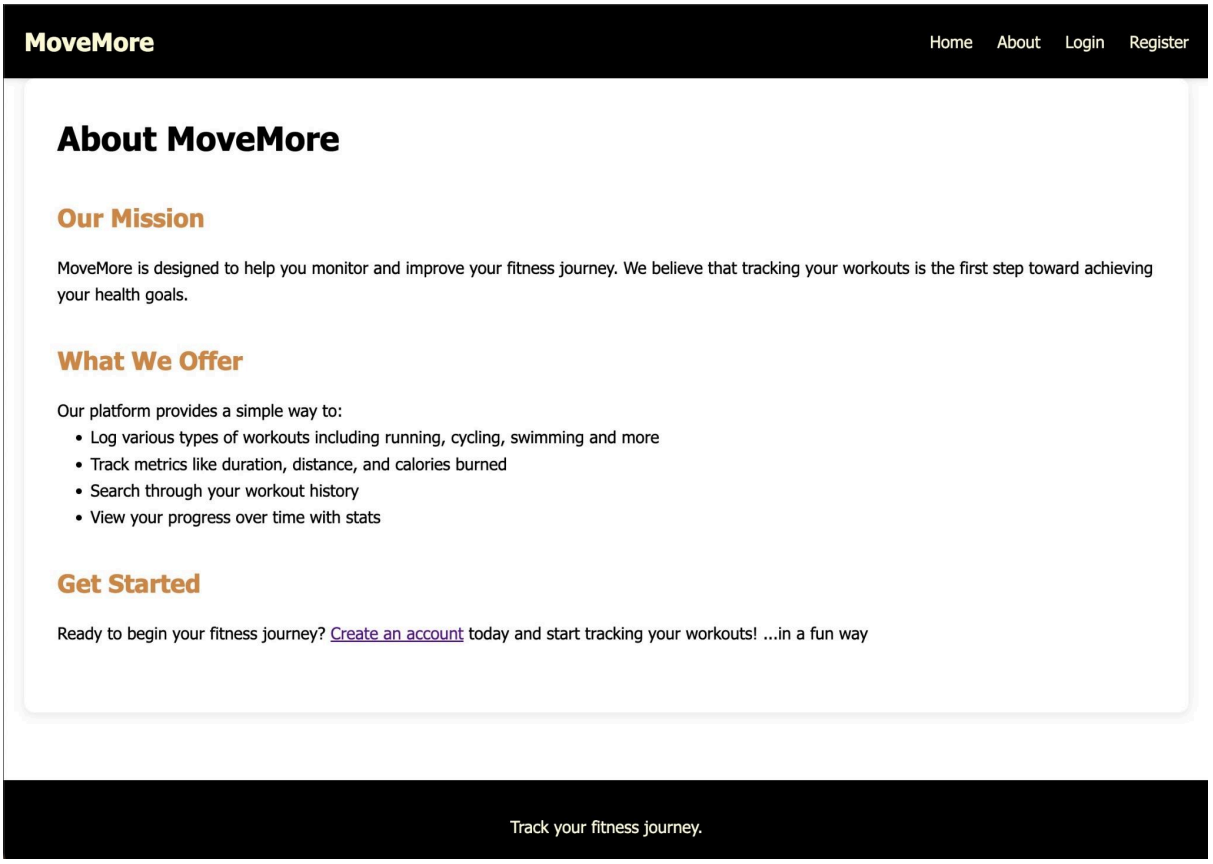
Matching workouts are shown in a table like the one on the dashboard, with the most recent first. The page also shows how many results were found and gives a message if there are no matches.

All Workouts



The All Workouts page shows every workout the user has added, and there is also an option to delete workouts from this page.

About Page



The About page explains what the app does, its main features, and its goal of helping users track their fitness. It gives new visitors some context and includes links to sign up for an account.

Advanced Techniques

Security Implementation

Passwords are secured using bcrypt with a salt round of 10, which hashes them so they stay safe even if the database is accessed. The code also correctly uses async/await when handling these operations.

```
try {
    const hashedPassword = await bcrypt.hash(password, 10);

    db.query(
        'INSERT INTO users (username, email, password) VALUES (?, ?, ?)',
        [username, email, hashedPassword],
```

Session Management and Authentication

The application implements an authentication middleware to protect routes:

```
//authentication middleware
const requireAuth = (req, res, next) => {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/login');
    }
};
```

This middleware is used on all protected pages (dashboard, add workout, all workouts, and search) to make sure users are logged in before they can access them. Sessions are set up with proper timeouts and security settings.

Database Query Optimization

The dashboard uses nested queries to get both the workout records and the summary statistics at the same time.

```
    //get recent workouts
    db.query(
        'SELECT * FROM workouts WHERE user_id = ? ORDER BY workout_date DESC LIMIT 5',
        [userId],
        (err, workouts) => {
            if (err) {
                return res.render('dashboard', { title: 'Dashboard', workouts: [], stats: {} });
            }

            //get stats
            db.query(
                'SELECT COUNT(*) as total_workouts, SUM(duration) as total_minutes, SUM(calories_burned) as total_calories FROM workouts WHERE user_id = ?',
                [userId],
                (err, stats) => {
                    res.render('dashboard', {
                        title: 'Dashboard',
                        workouts,
                        stats: stats[0] || {}
                    });
                }
            );
        }
    );
});
```

This reduces the number of database calls while still giving all the data needed for the page.

Input Validation

The application implements both client-side and server-side validation. Password requirements use regex pattern matching:

```
//password req: 8 chars - 1 lowercase - 1 uppercase - 1 numbe - 1 special
const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/;
if (!passwordRegex.test(password)) {
    return res.render('register', {
        title: 'Register',
        error: 'Password must be 8+ characters with uppercase, lowercase, number, and special character'
    });
}
```

Responsive Design

The CSS implements responsive design using media queries and flexible grid layouts:

```
/* responsive */
@media (max-width: 768px) {
    .nav-links {
        flex-direction: column;
        gap: 0.5rem;
    }

    .hero h1 {
        font-size: 2rem;
    }

    .cta-buttons {
        flex-direction: column;
    }

    .workout-table {
        font-size: 0.9rem;
    }

    .workout-table th,
    .workout-table td {
        padding: 0.5rem;
    }

}
```

This ensures the application functions effectively across different devices.

AI Declaration

I used AI tools mainly to help with small wording tasks, such as writing some of the generic text on my site (for example, parts of the features section and the about page), and for simple UI wording like button labels. I also got feedback on the structure and wording of this report.

On the technical side, I used AI only to help me understand and debug a few MySQL issues when I was having trouble entering queries.