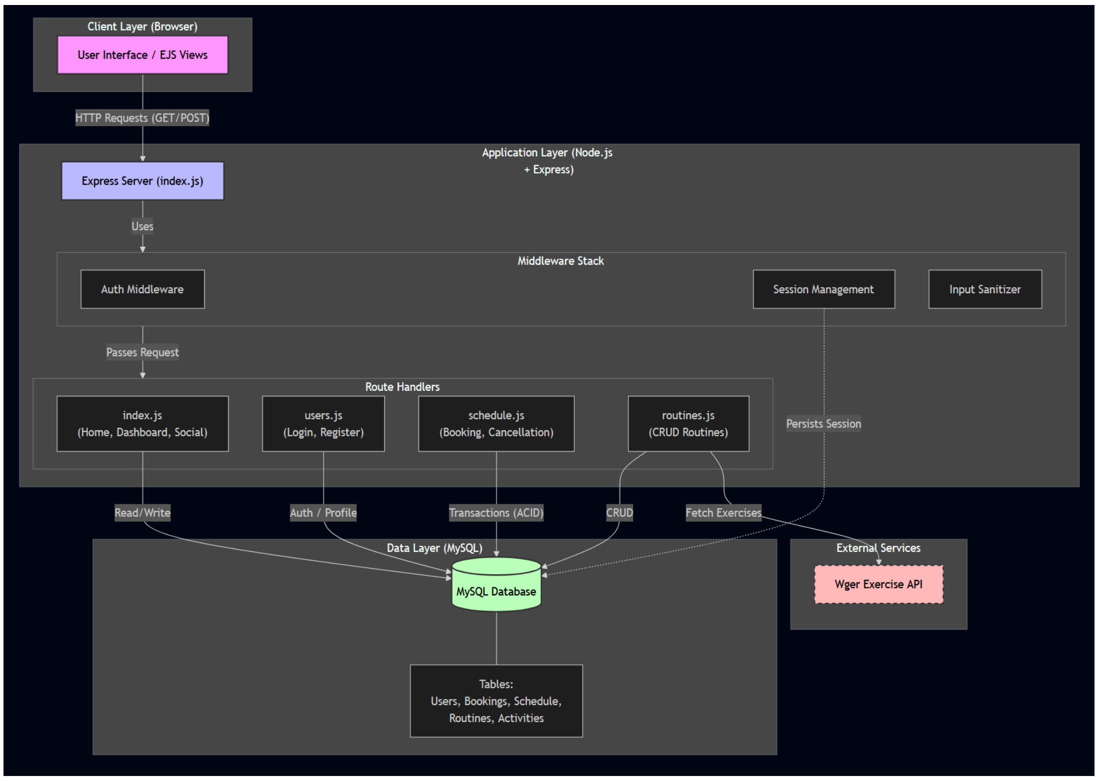
DOCUMENTATION

Outline

Gym&Gain is a full-stack web application designed to simplify gym management and workout tracking. Built on a Node.js and Express framework with a MySQL database, the system provides a responsive environment for managing user interactions. The application implements a tiered membership model (Guest, Silver, Gold) that integrates with a token system for class access.

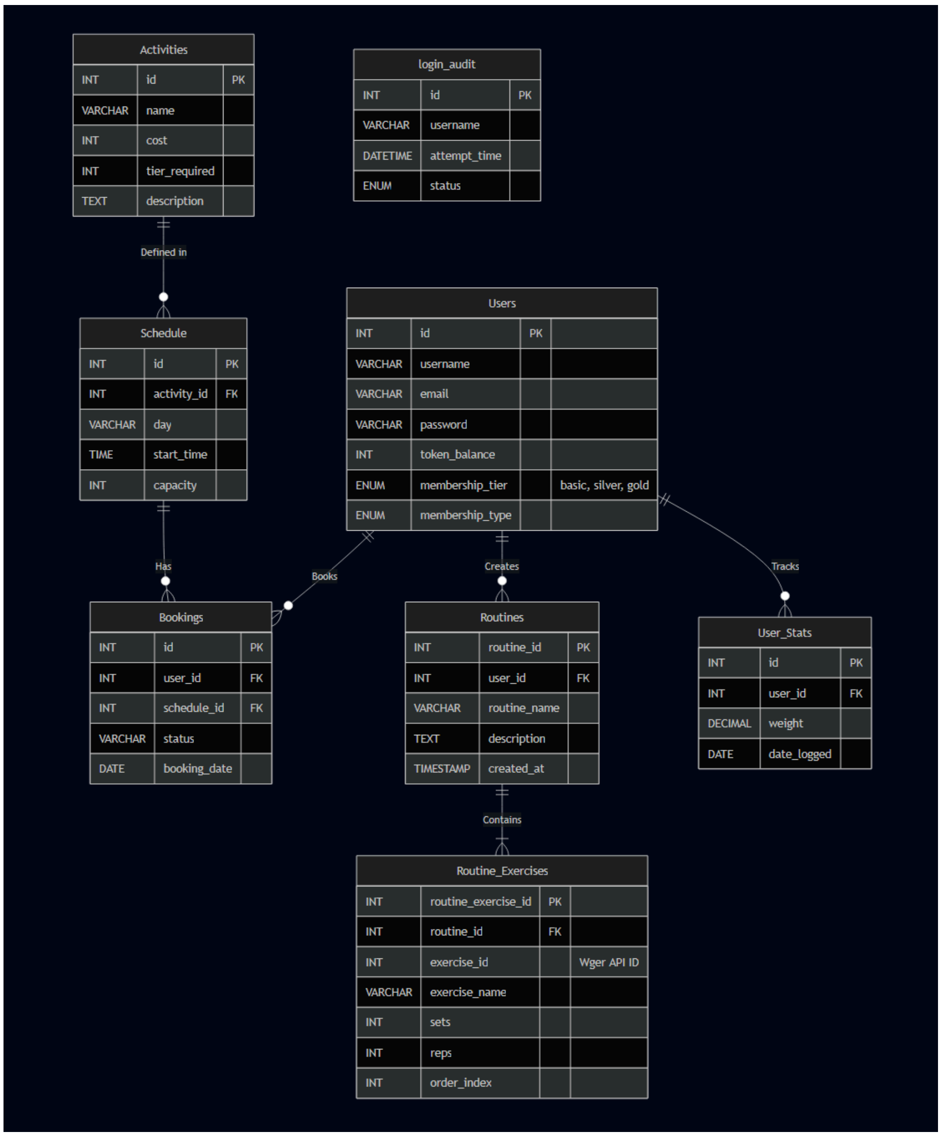
From a technical perspective, the core functionality relies on a scheduling engine that uses database transactions to ensure booking integrity and enforce class capacity limits. Beyond administrative features, the application enables users to define and store personalised exercise routines. The architecture prioritizes security through standard practices, including bcrypt for password hashing, input sanitization, and secure session management to protect user data.

Architecture



The Application Tier uses Node.js with Express.js for routing, middleware, and business logic. Security and session management are handled via express-mysql-session, while express-validator ensures input sanitization. Axios integrates external data from the Wger API. The Data Tier consists of a MySQL database accessed through a mysql2 connection pool, supporting ACID transactions for critical operations like class bookings. The Presentation Layer employs EJS for server-side HTML rendering, with client-side JavaScript and SweetAlert2 providing dynamic user feedback and asynchronous interactions

Data Model



Activities		
INT	id	PK
VARCHAR	name	
INT	cost	
INT	tier_required	
TEXT	description	

login_audit		
INT	id	PK
VARCHAR	username	
DATETIME	attempt_time	
ENUM	status	

Schedule		
INT	id	PK
INT	activity_id	FK
VARCHAR	day	
TIME	start_time	
INT	capacity	

Users			
INT	id	PK	
VARCHAR	username		
VARCHAR	email		
VARCHAR	password		
INT	token_balance		
ENUM	membership_tier		basic, silver, gold
ENUM	membership_type		

Bookings		
INT	id	PK
INT	user_id	FK
INT	schedule_id	FK
VARCHAR	status	
DATE	booking_date	

Routines		
INT	routine_id	PK
INT	user_id	FK
VARCHAR	routine_name	
TEXT	description	
TIMESTAMP	created_at	

User_Stats		
INT	id	PK
INT	user_id	FK
DECIMAL	weight	
DATE	date_logged	

Routine_Exercises			
INT	routine_exercise_id	PK	
INT	routine_id	FK	
INT	exercise_id		Wger API ID
VARCHAR	exercise_name		
INT	sets		
INT	reps		
INT	order_index		

The database uses a clean structure centred on the Users table, which holds login details, membership levels, and token balances. We separated the scheduling logic into Activities (the class types) and Schedule (the specific time slots) to keep timetable management flexible. The Bookings table links users to specific class times and acts as the main record for reservations. Finally, the Routines and Routine_Exercises tables allow users to build and save their own workout plans directly to their account.

User Functionality

Authentication and Onboarding. New users can securely register for an account and are automatically assigned a "Guest" status with a starting balance of tokens. The login system uses secure sessions to keep users signed in as they move through the app, so they don't have to constantly re-login.

Class Discovery and Scheduling The main hub of the application is the Weekly Schedule view. Here, users can browse classes sorted by day and time. Each listing clearly shows what the class is, how much it costs in tokens, and how many spots are left (e.g., "3/20 slots"). There is also a search bar that lets users filter the schedule instantly by activity name, like "Yoga" or "HIIT".

Booking System When a user clicks to book, the system runs immediate checks to ensure they have the right membership tier, enough tokens, and that the class isn't full. If everything looks good, they get a success popup and their tokens are deducted automatically. If there’s an issue, like insufficient funds, they get a clear error message explaining why.

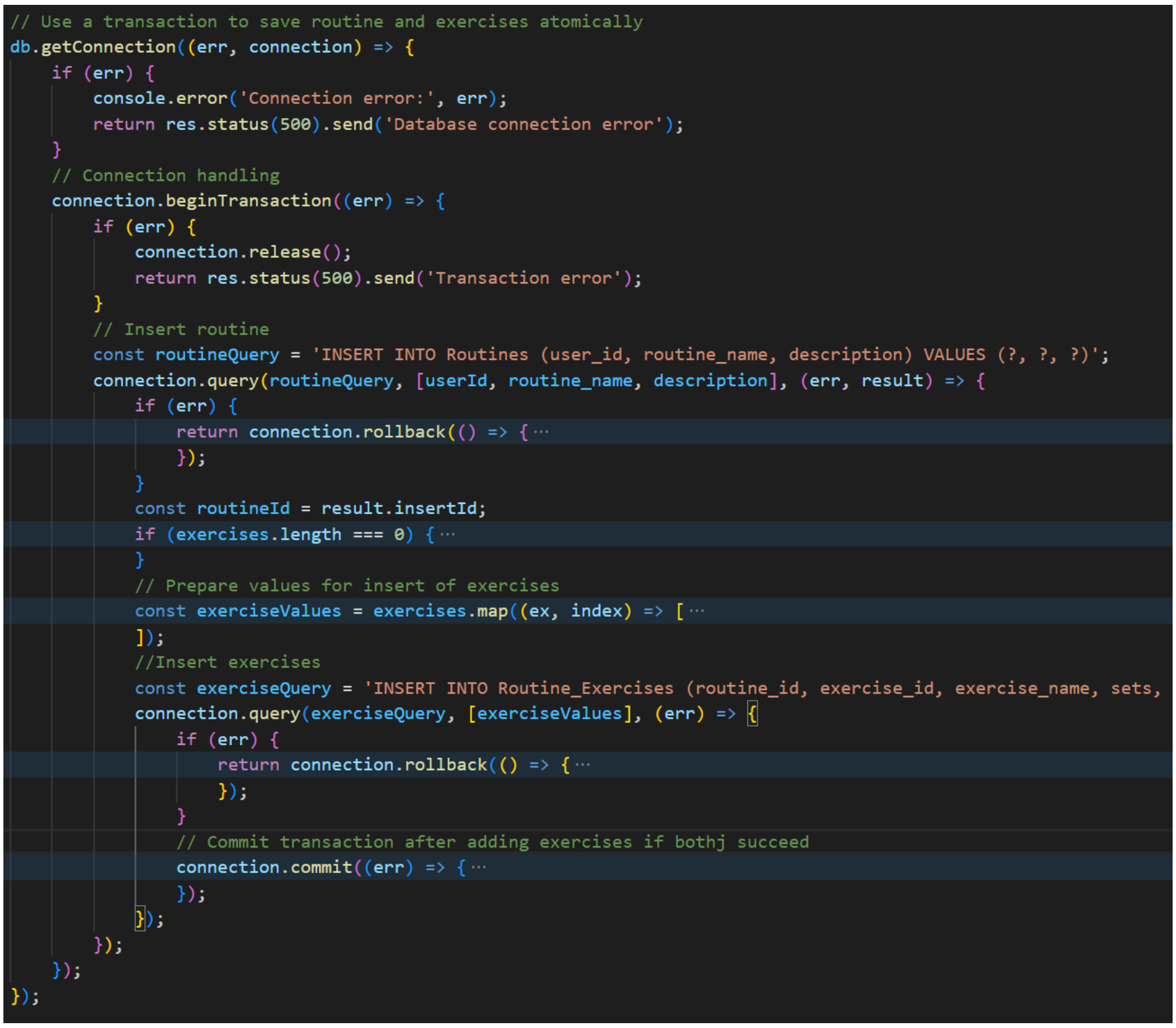
User Dashboard The Dashboard acts as the user's personal control centre. It lists all their upcoming confirmed bookings in date order. If plans change, users can cancel a class directly from here. The system handles cancellations by immediately refunding the tokens to the user's balance, making the process risk-free.

Routine Builder Beyond just booking classes, the app lets users manage their own training. The Routine Builder allows users to search for specific exercises (pulled from the Wger API) and add them to a custom plan. They can define the sets and reps for each movement and then save the finalised routine to their profile for future workouts.

Community and Social Feed To gain a sense of community, the “Social” tab allows registered members to view a live feed of bookings made by other users. It respects privacy by only showing necessary details, creating a motivating environment where users can see what classes are trending and who else is working out

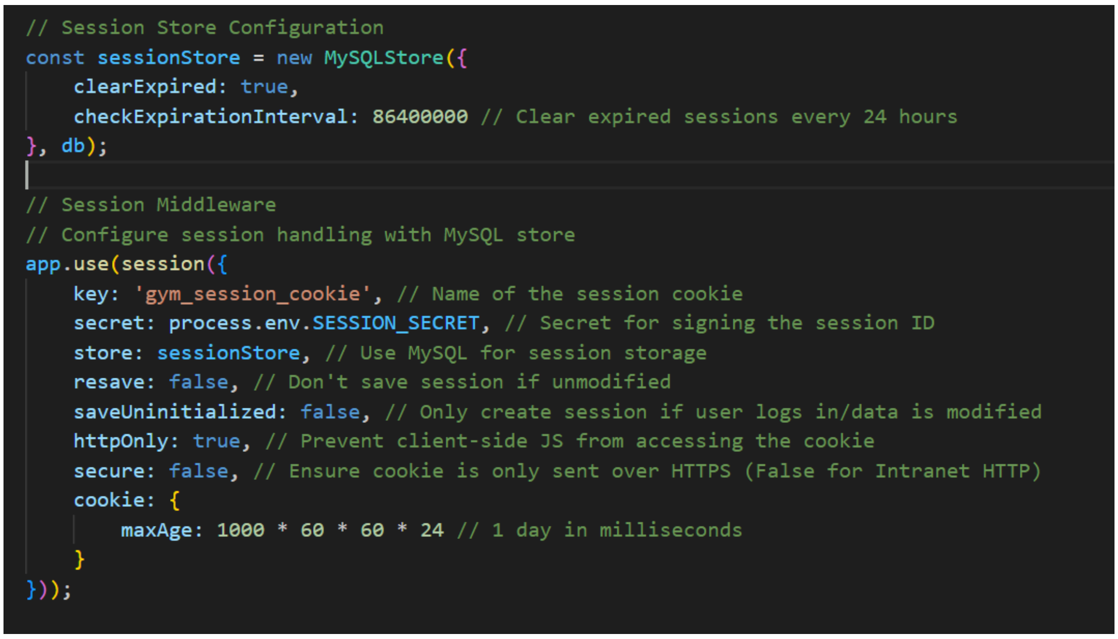
Advanced Techniques

1. Data Integrity with ACID Transactions: To ensure reliability during complex database operations such as saving a new workout routine with multiple exercises, the application utilises SQL transactions within the routine creation logic. This ACID approach guarantees atomicity: if any part of the process fails, such as an error adding a specific exercise, the entire operation is rolled back. This prevents partial data writes and keeps the database clean of residual records.



```javascript
// Use a transaction to save routine and exercises atomically
db.getConnection((err, connection) => {
    if (err) {
        console.error('Connection error:', err);
        return res.status(500).send('Database connection error');
    }
    // Connection handling
    connection.beginTransaction((err) => {
        if (err) {
            connection.release();
            return res.status(500).send('Transaction error');
        }
        // Insert routine
        const routineQuery = 'INSERT INTO Routines (user_id, routine_name, description) VALUES (?, ?, ?)';
        connection.query(routineQuery, [userId, routine_name, description], (err, result) => {
            if (err) {
                return connection.rollback(() => {⋯
                });
            }
            const routineId = result.insertId;
            if (exercises.length === 0) {⋯
            }
            // Prepare values for insert of exercises
            const exerciseValues = exercises.map((ex, index) => [⋯
            ]);
            //Insert exercises
            const exerciseQuery = 'INSERT INTO Routine_Exercises (routine_id, exercise_id, exercise_name, sets,
            connection.query(exerciseQuery, [exerciseValues], (err) => {
                if (err) {
                    return connection.rollback(() => {⋯
                    });
                }
                // Commit transaction after adding exercises if bothj succeed
                connection.commit((err) => {⋯
                });
            });
        });
    });
});
```

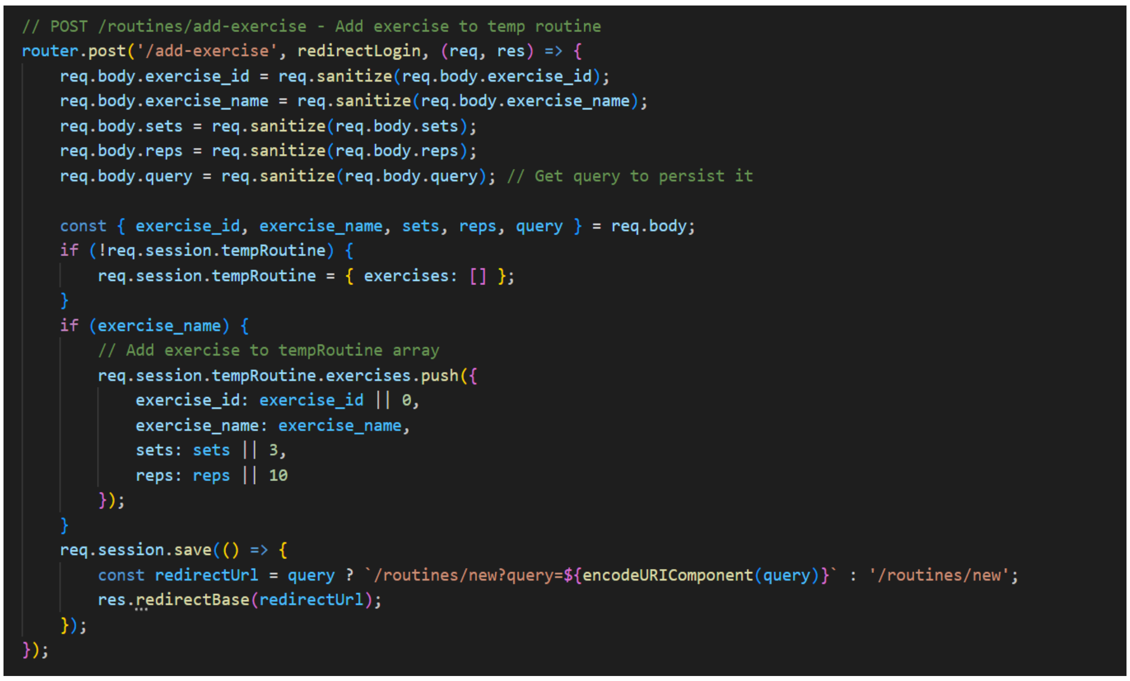
2. Persistent Session Management: Rather than relying on the default MemoryStore, which is prone to memory leaks and resets on server restarts, the application implements express-mysql-session. This solution stores active user sessions directly in the MySQL database. This approach ensures that users remain logged in even if the server reboots and provides a scalable foundation that could support load balancing across multiple server instances in the future.



```javascript
// Session Store Configuration
const sessionStore = new MySQLStore({
    clearExpired: true,
    checkExpirationInterval: 86400000 // Clear expired sessions every 24 hours
}, db);

// Session Middleware
// Configure session handling with MySQL store
app.use(session({
    key: 'gym_session_cookie', // Name of the session cookie
    secret: process.env.SESSION_SECRET, // Secret for signing the session ID
    store: sessionStore, // Use MySQL for session storage
    resave: false, // Don't save session if unmodified
    saveUninitialized: false, // Only create session if user logs in/data is modified
    httpOnly: true, // Prevent client-side JS from accessing the cookie
    secure: false, // Ensure cookie is only sent over HTTPS (False for Intranet HTTP)
    cookie: {
        maxAge: 1000 * 60 * 60 * 24 // 1 day in milliseconds
    }
}));
```

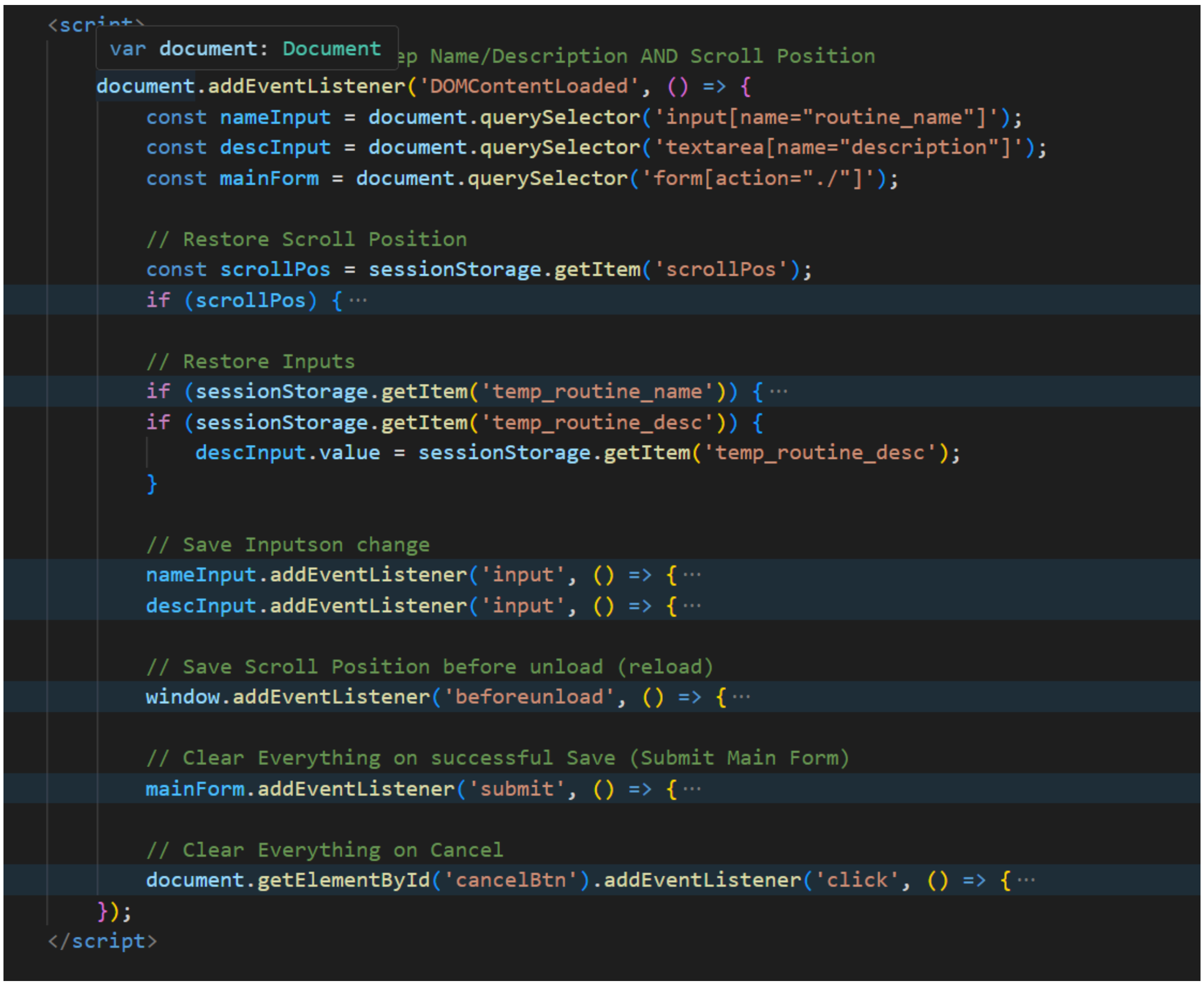
3. Session-Based State Management: To create a smoother user experience when building workout plans, the app uses a "temporary state" pattern. When a user creates a routine, their changes like adding or removing exercises are stored in a temporary session object (req.session.tempRoutine) rather than being written to the database immediately. This reduces unnecessary database load and gives users the freedom to experiment with their routine setup before "committing" the final version to permanent storage.



```javascript
// POST /routines/add-exercise - Add exercise to temp routine
router.post('/add-exercise', redirectLogin, (req, res) => {
    req.body.exercise_id = req.sanitize(req.body.exercise_id);
    req.body.exercise_name = req.sanitize(req.body.exercise_name);
    req.body.sets = req.sanitize(req.body.sets);
    req.body.reps = req.sanitize(req.body.reps);
    req.body.query = req.sanitize(req.body.query); // Get query to persist it

    const { exercise_id, exercise_name, sets, reps, query } = req.body;
    if (!req.session.tempRoutine) {
        req.session.tempRoutine = { exercises: [] };
    }
    if (exercise_name) {
        // Add exercise to tempRoutine array
        req.session.tempRoutine.exercises.push({
            exercise_id: exercise_id || 0,
            exercise_name: exercise_name,
            sets: sets || 3,
            reps: reps || 10
        });
    }
    req.session.save(() => {
        const redirectUrl = query ? `/routines/new?query=${encodeURIComponent(query)}` : '/routines/new';
        res.redirectBase(redirectUrl);
    });
});
```

4. Hybrid UX Persistence "No Reload" Feel: Traditional server-side apps can often feel clunky because the page resets every time you do something. I wanted the smooth feel of a modern SPA (Single Page App) without the added complexity of a frontend framework.
To achieve this, I implemented a hybrid system: the server manages the core data (the exercise list), while a custom script uses the browser's sessionStorage to store the user's scroll position and text inputs. So you can search, add, and edit exercises, and the page stays exactly where you left it, no jumping and no data loss.



```
<script>
    var document: Document ep Name/Description AND Scroll Position
    document.addEventListener('DOMContentLoaded', () => {
        const nameInput = document.querySelector('input[name="routine_name"]');
        const descInput = document.querySelector('textarea[name="description"]');
        const mainForm = document.querySelector('form[action="./"]');

        // Restore Scroll Position
        const scrollPos = sessionStorage.getItem('scrollPos');
        if (scrollPos) {···

        // Restore Inputs
        if (sessionStorage.getItem('temp_routine_name')) {···
        if (sessionStorage.getItem('temp_routine_desc')) {
            descInput.value = sessionStorage.getItem('temp_routine_desc');
        }

        // Save Inputson change
        nameInput.addEventListener('input', () => {···
        descInput.addEventListener('input', () => {···

        // Save Scroll Position before unload (reload)
        window.addEventListener('beforeunload', () => {···

        // Clear Everything on successful Save (Submit Main Form)
        mainForm.addEventListener('submit', () => {···

        // Clear Everything on Cancel
        document.getElementById('cancelBtn').addEventListener('click', () => {···
    });
</script>
```

AI declaration

Throughout the development of Gym&Gain, I utilised Gemini 3 as a sort of digital 'pair programmer.' My goal wasn't to have it write the app for me, but to use it to translate my architectural ideas and the complex code required to execute them particularly regarding state management and server configuration.
A major help was solving Environment-Agnostic Routing. I needed the app to work seamlessly on both my local machine and the university's subdirectory-based intranet. I explained the constraint to the AI, and it helped me draft the logic for centralized helper functions like redirectBase and url. I then integrated and tested these across the app to ensure links remained valid in any environment.
I also leaned on the assistant for the Hybrid Persistence Pattern. I knew I wanted that smooth "no-reload" user experience, but syncing the secure server-side session with the browser's client-side storage was technically tricky. The AI provided the JavaScript and EJS boilerplate to link these two layers, which I then refined to fit my specific UI requirements.
finally, the AI was a massive help with Debugging. When I got stuck on things like circular redirect loops in my auth middleware or syntax errors in nested routes, the AI analyzed the stack traces and pointed out the fix immediately. This allowed me to focus on the high-level logic rather than getting stuck in syntax. Importantly, I manually reviewed and tested every suggestion to ensure I fully understood the code and that it met the project specs.