LAB 10- HEALTH AND FITNESS APPLICATION

Dynamic Web Applications

AISHA BHUDYE
33734353

Contents

Outline

The Clinic Appointment Manager is a web application that allows patients to book, view, and search for appointments at a clinic. It also includes a user authentication system for staff members, tracking login attempts in an audit log. The application is built using Node.js with Express as the web framework, MySQL for the database, and EJS for server-side templates. Users can submit appointment requests via forms, while staff can manage users and view logs of login activity.

Architecture

Technologies & Components:

- Application tier: Node.js, Express, EJS templates, bcrypt for password hashing
- Data tier: MySQL database with tables for `appointments`, `users`, and `login_audit`

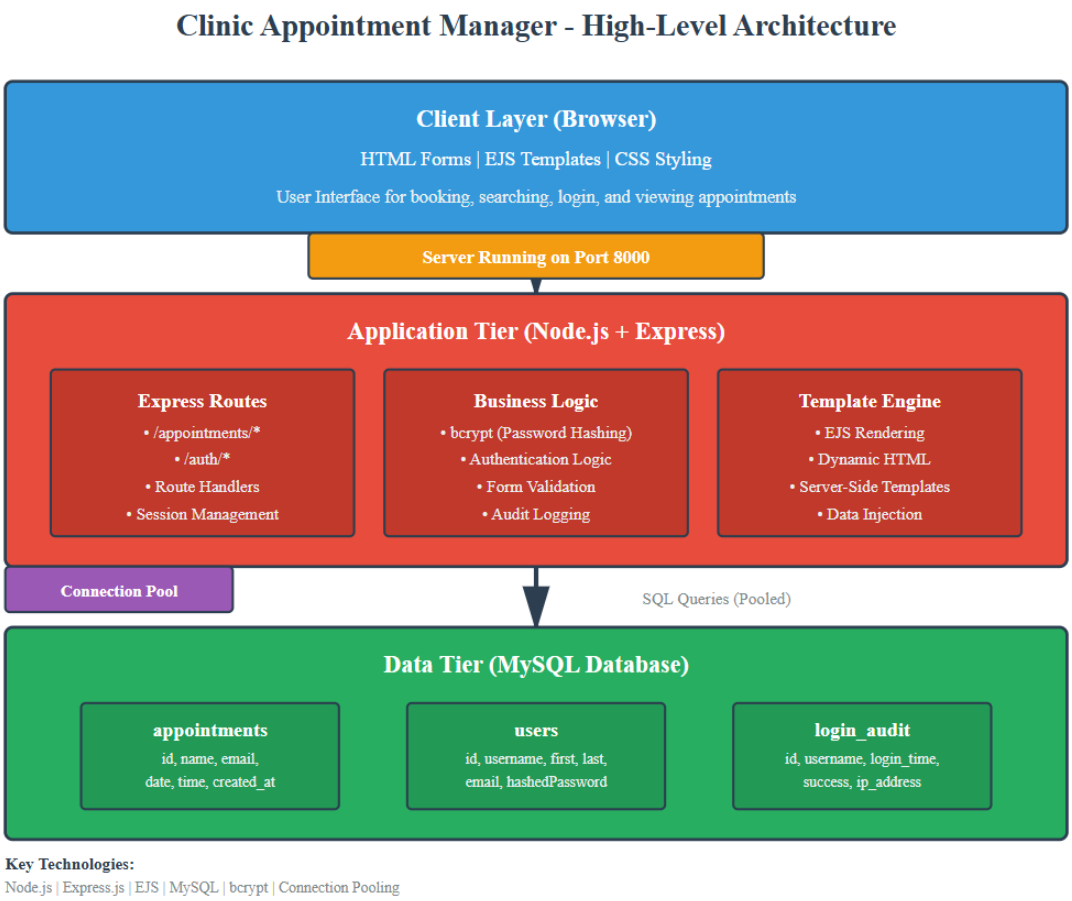
Deployment & Environment Configuration

The application is designed to be fully installable on the marker's machine. The environment variables required for database connection use the recommended configuration:

- HEALTH_HOST='localhost'
- HEALTH_USER='health_app'
- HEALTH_PASSWORD='qwertyuiop'
- HEALTH_DATABASE='health'

Running `npm install` installs all dependencies, and running `node index.js` starts the application on port eight thousand.

High-level Architecture Diagram:

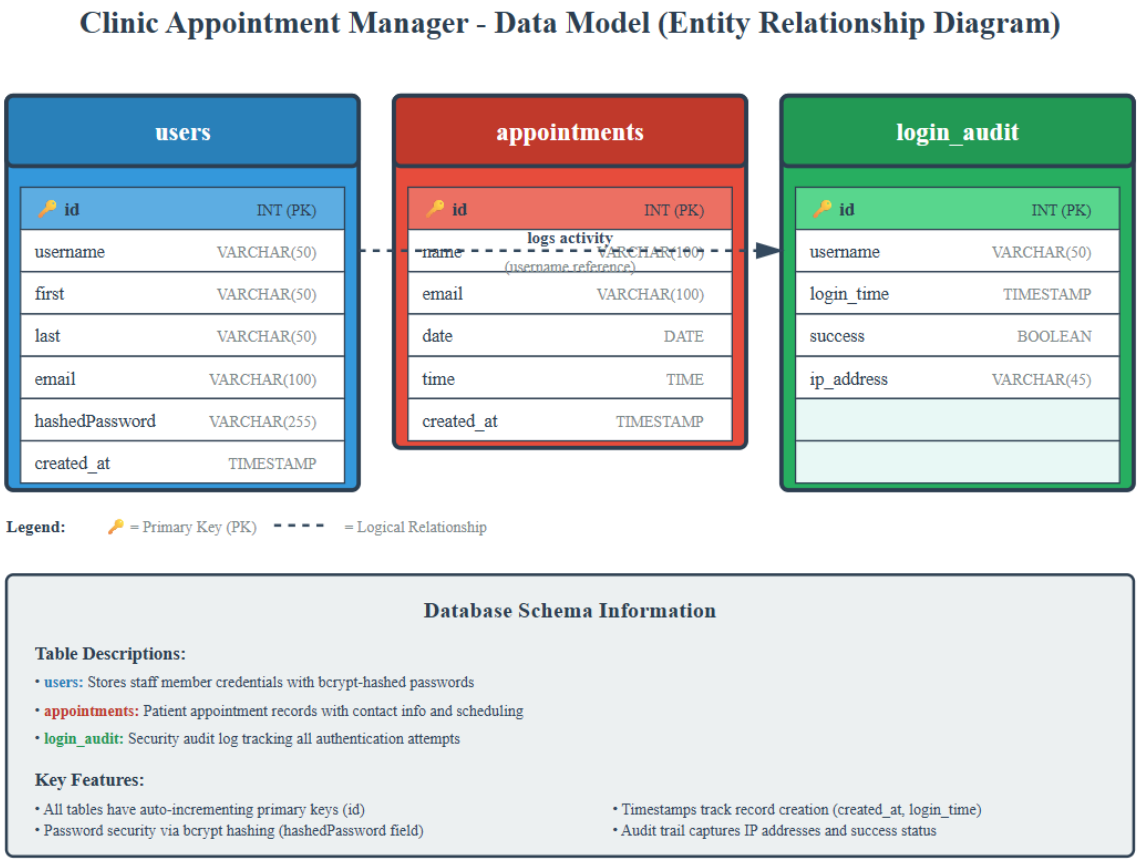


Data Model

The database has three main tables:

1. appointments – Stores patient appointments (id, name, email, date, time, created_at)
2. users – Stores staff user accounts (id, username, first, last, email, hashedPassword, created_at)
3. login_audit – Logs login attempts (id, username, login_time, success, ip_address)

Data Model Diagram:



Database Installation Scripts

Two SQL scripts are included as required:

- **create_db.sql** – Creates the health database and defines all tables (appointments, users, login_audit).
- **insert_test_data.sql** – Inserts initial data including the default staff user (**gold/smiths**) and optional example appointments.

These scripts ensure the database can be recreated from scratch during marking.

User Functionality

1. Book Appointment – Users fill a form (`/appointments/book`) with name, email, date, and time. On submission, the appointment is stored in the database.
2. List Appointments – Users or staff can view all appointments (`/appointments/list`) in a table sorted by date and time.
3. Search Appointments – Users can search for appointments by patient name (`/appointments/search`).
4. User Authentication – Staff can register (`/auth/register`) and log in (`/auth/login`). Passwords are hashed with bcrypt.
5. Audit Log – All login attempts are stored in `login_audit`, accessible via `/auth/audit`. This logs username, IP address, success status, and timestamp.

6. API Functionality – allows external applications to access clinic appointment data in JSON format. This enables integration with other systems and provides machine-readable access to appointment information

The application includes a Home page that provides navigation to all major features of the system, and an About page describing the purpose of the clinic appointment manager and the technologies used.

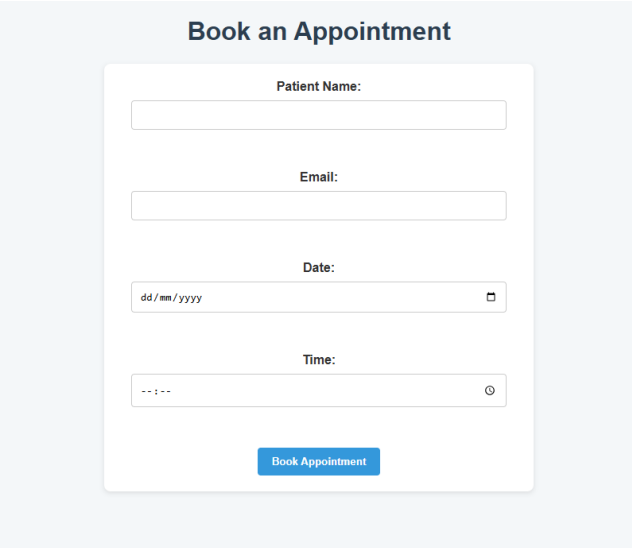
A default user account has been created as required by the brief:

- Username: gold
- Password: smiths

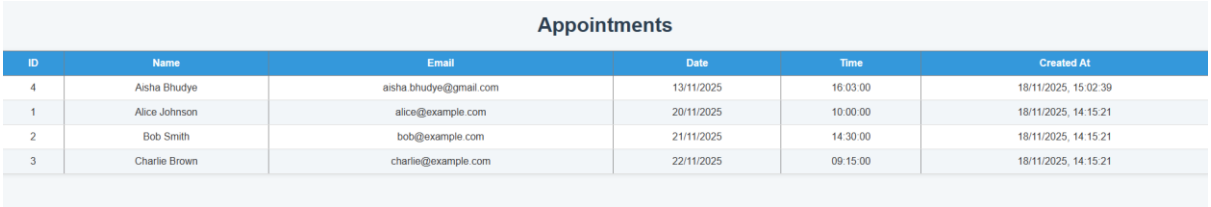
This account is inserted automatically through the insert_test_data.sql script when the application is deployed. It allows the marker to log in and access staff-only features such as viewing users and audit logs.

Example Screenshot Descriptions:

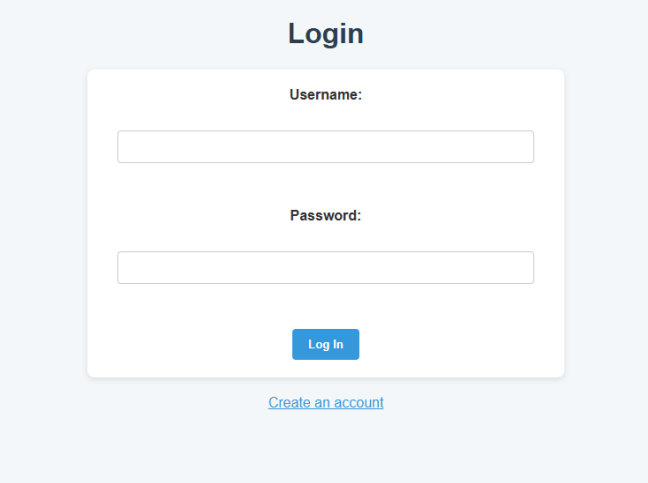
Booking form submission



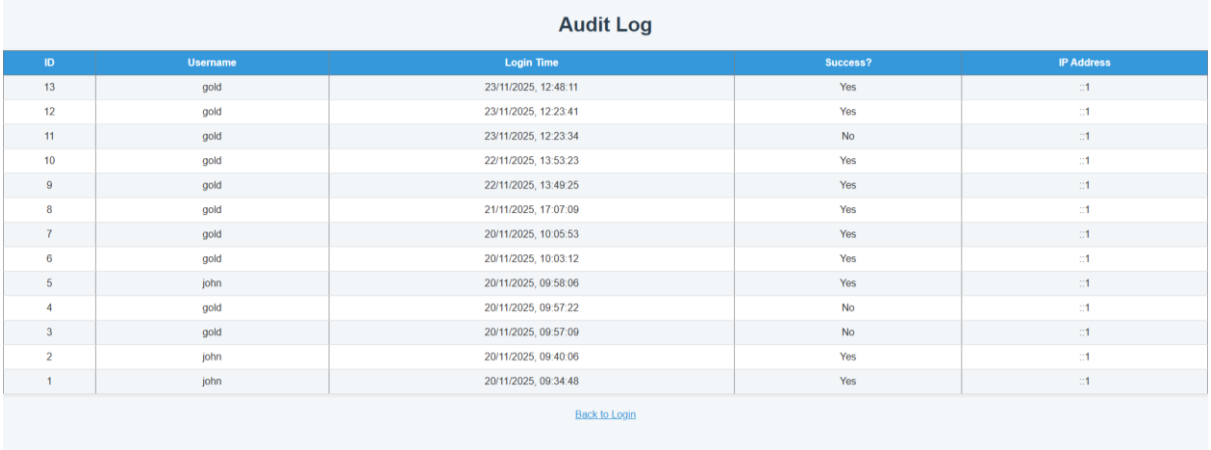
Appointment list table



Login page



Audit log showing successful and failed login attempts.



Basic Techniques

1. Password Hashing (bcrypt)

The application uses bcrypt with 10 salt rounds to securely hash user passwords before storing them in the database. During login, the entered password is compared with the stored hash using bcrypt.compare(). Successful and failed login attempts are also recorded in the login_audit table for security monitoring.

2. Input Validation (express-validator)

All registration fields are validated using express-validator to ensure data quality. Checks include proper email formatting, username length (5–20 chars), password complexity (minimum 8 chars), and non-empty first/last names. If validation fails, the form is re-rendered with specific error messages.

3. Input Sanitisation

To protect against XSS attacks, all inputs are sanitised using express-sanitizer before being processed or stored. This removes unsafe HTML/JavaScript and prevents users from injecting harmful scripts into form fields.

4. Session-Based Authentication

A custom redirectLogin middleware ensures only authenticated staff can access protected pages such as the user list and audit log. Sessions store the logged-in username, and logging out destroys the session entirely for security.

5. Database Connection Pooling

The application uses mysql2.createPool() to manage database connections efficiently. Pooling reduces overhead, supports concurrent queries, handles reconnections automatically, and prevents the server from opening excessive connections.

6. Audit Logging

Every login attempt—successful or failed—is logged in the login_audit table, including timestamp, username, success status, and IP address. This supports security analysis, detection of brute-force attempts, and system monitoring.

7. Dynamic Templating (EJS)

EJS templates dynamically render server-side data for the appointment lists, audit logs, forms, and validation errors. Loops and conditional blocks allow clean, reusable UI components that react to backend data.

8. RESTful API

A RESTful API provides JSON access to appointment data with optional search, date filtering, and sorting through query parameters. SQL queries are built dynamically using parameterised inputs to prevent injection. Additional endpoints include /api/stats, which returns aggregate statistics such as total appointments and unique patient count.

Advanced Techniques

In addition to the core functionality taught in the module, I implemented an advanced security enhancement to protect the authentication system from brute-force attacks. This was achieved through the use of rate limiting, a technique that restricts the number of login attempts a user can make within a specific time window.

Login Rate Limiting

To strengthen the security of the login process, I used the *express-rate-limit* middleware to limit how many times a user can attempt to log in per minute. This prevents automated scripts or malicious users from repeatedly guessing passwords. The limiter was configured to allow a maximum of five login attempts per minute from the same IP address:

```
const rateLimit = require("express-rate-limit");


const loginLimiter = rateLimit({
  windowMs: 60 * 1000,
  max: 5,
  message: "Too many login attempts. Please try again later."
});
```

The limiter is applied directly to the POST /login route so that each login request passes through the rate-limit protection before any authentication logic is executed:

```
router.post("/login", loginLimiter, (req, res, next) => {
  // login processing
});
```

If a user exceeds the limit, they receive a friendly error message instead of the standard login processing. This technique helps to prevent brute-force attacks while ensuring that genuine users can still retry their password after a short cooldown.

Integrating the rate limiter with my existing audit logging and session-based authentication demonstrates an enhancement beyond the lab material and improves the overall robustness of the application.

Password Strength Indicator

The password strength feature provides real-time feedback to users on how strong their password is as they type. The implementation involves JavaScript interacting with the DOM.

1. Selecting Elements

```
const passwordField = document.getElementById("password");
const strengthText = document.querySelector("#strengthOutput span");
```

- passwordField references the password input field in the form.
- strengthText references the inside the paragraph that displays the password strength.

2. Adding an Event Listener

```
passwordField.addEventListener("input", function () { ... });
```

- The input event fires every time the user types or deletes a character.
- This allows the script to update the password strength in real time.

3. Checking Password Strength

```
let strength = "Weak";
```

let color = "#c0392b"; // red

- Every time the event fires, the default strength is set to Weak (red color).
- The script will then test the password against certain rules to possibly upgrade its strength.

4. Medium Strength Rule

```
if (password.length >= 8) {
  strength = "Medium";
  color = "#f39c12"; // orange
}
```

- If the password has 8 or more characters, the strength is considered Medium.
- Color changes to orange for visual feedback.

5. Checking for Numbers and Symbols

```
const hasNumber = /\d/.test(password);
const hasSymbol = /[!@#$%^&*]/.test(password);
```

- \d checks if the password contains at least one digit (0–9).
- [!@#$%^&*] checks if the password contains at least one special character.
- Using regular expressions makes these checks concise and reliable.

6. Strong Strength Rule

```
if (password.length >= 10 && hasNumber && hasSymbol) {
  strength = "Strong";
  color = "#27ae60"; // green
}
```

- If the password is 10+ characters and contains at least one number and one symbol, it is considered Strong.
- Color changes to green for clear positive feedback.

7. Updating the UI

```
strengthText.innerText = strength;
strengthText.style.color = color;
```

- Updates the text inside the to show Weak, Medium, or Strong.
- Updates the text color to match the strength visually.

Summary

- This script provides real-time feedback on password strength.
- It uses length, numbers, and symbols as metrics for strength.
- DOM manipulation and event listeners allow the interface to update dynamically.
- Regular expressions make the checks efficient and readable.

AI Declaration

I acknowledge the use of ChatGPT (OpenAI, 2025) and Grammarly (Grammarly Inc., 2025) to assist with proofreading, grammar checking, and improving sentence structure and clarity in this assessment. Both tools were used solely for language refinement; no generated text, ideas, or analytical content were included in the submitted work. ChatGPT was also used to help create the diagrams in this report. ChatGPT was accessed via https://chat.openai.com/ in November 2025 using the prompt:

“Proofread and improve the grammar and flow of my report.”

Grammarly was used for grammar correction and sentence structure review via https://www.grammarly.com/.

Reference List

OpenAI (2025) ChatGPT [Generative AI model]. Available at: https://chat.openai.com/ (Accessed: 10 November 2025).

Grammarly Inc. (2025) Grammarly [AI writing assistance tool]. Available at: https://www.grammarly.com/ (Accessed: November 2025).