Outline

The **Clinic Appointment Booking app** is a web application that allows patients to book,view and search appointments. Users can register, log in, and interact with forms and a user authentication system.. The app provides search functionality to find patients or health records quickly.A registered user can manage all appointments and view patient details as well as see the list of registered users and an audit log.The app focuses on usability, responsive design, and data persistence. By integrating Node.js with Express for the server, MySQL for the database, and EJS for server-side templating, the app consolidates concepts learned in labs, including routing,database connectivity, and session management. Security measures include password validation, session-based login, and the use of environment variables to store sensitive information. The application is fully installable from GitHub, ensuring that users can run it locally or on the VM.

Architecture

The app uses a **two-tier architecture**:

- **Application Tier:** Node.js with Express handles routing,session management, and form validation. EJS templates render dynamic HTML.bcrypt for password hashing

- **Data Tier:** MySQL stores tables of users, appointments and audit logins. Environment variables manage database credentials securely.

Deployment & Configuration:

The application will be fully installable on the marker's machine.Sensitive data like database credentials are stored in a `.env` file.

HEALTH_HOST=localhost

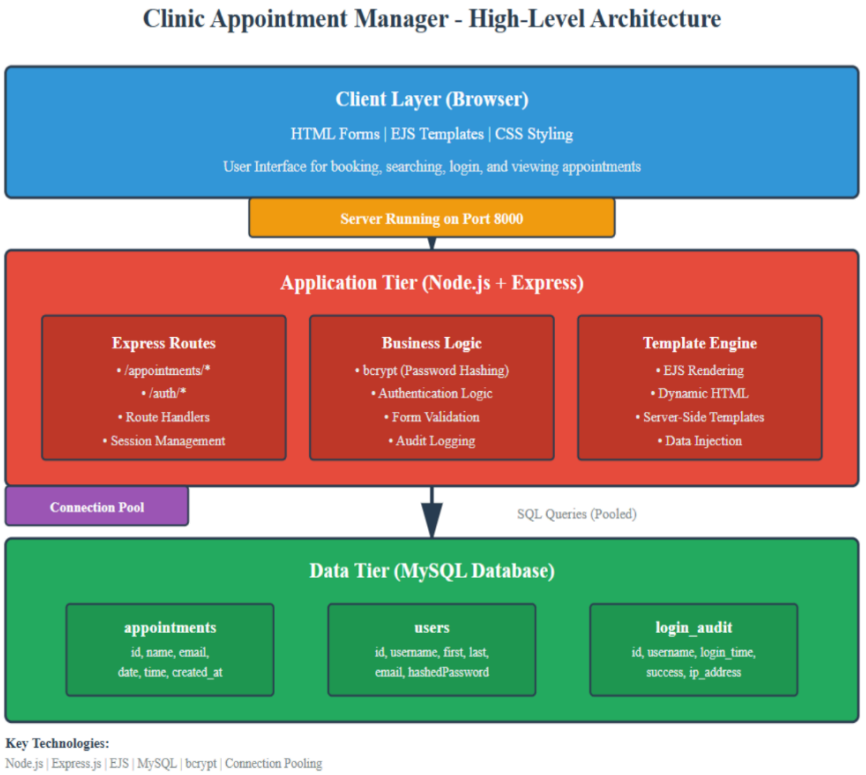
HEALTH_USER=health_app

HEALTH_PASSWORD=qwertyuiop

HEALTH_DATABASE=health

We run npm install to install the required dependencies then we run node index.js to run the application which can be accessed via the localhost 8000 link.

High level architecture diagram

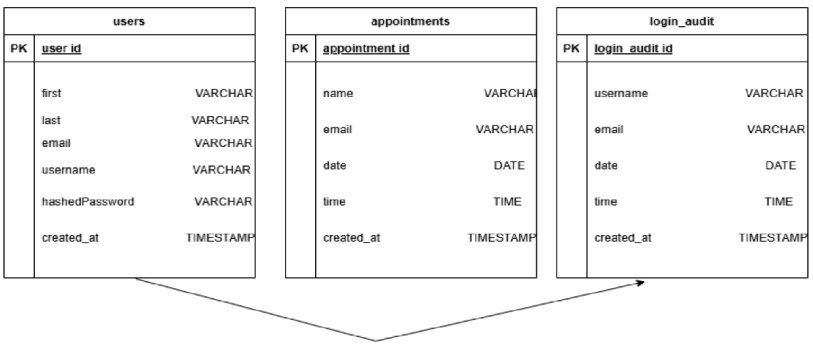


Data Model

The database includes three main tables:

users: stores staff user accounts(id,username,first,last,email,hashedPasword)
appointments: records patient name, email, date, and time.
login_audit: logs username, success status, timestamp, and IP address of each login attempt.

Data Model Diagram



Database Installation Scripts:

Two SQL Scripts are created as requested:

1.**create_db.sql:**Creates the health database and defines all tables(appointments,users,login_audit).

2.**insert_test_data.sql**:Inserts data into the database tables such as the default Gold user and example appointments.

User Functionality

The application provides a clean interface that staff members can use after authentication. Upon logging in, the user dashboard allows access to the booking system and appointment views.

1. Login System

Users enter a username and password. The system validates credentials using bcrypt hashing and logs the attempt in the audit log. Failed attempts display clear error messages.

2. Book Appointment

Users can schedule appointments by entering a patient's name, email, date, and time. Form validation ensures no empty fields. After submission, the appointment is saved to MySQL and the user is redirected to the appointment list.

3. Search Appointments

The search page allows staff to find appointments by entering a patient name. Results are displayed in a formatted table and support partial matches using SQL LIKE queries.

4. View All Appointments

A protected route displays all appointments sorted by date and time. The table view is styled for readability and shows relevant appointment details.

5. Audit Log

All login attempts—successful and failed—are stored. This allows monitoring suspicious activity, repeated failures, or incorrect usernames.

6. Session & Access Control

Pages like booking, searching, and listing appointments require a valid session. Unauthorized users are redirected to the login page.

7.API Functionality

Allows external applicants to access clinic appointment data in JSON format.

The application also includes a Home page that provides navigation to all other features of the system and an About page which describes the purpose of the app.

A default user has been created as instructed:

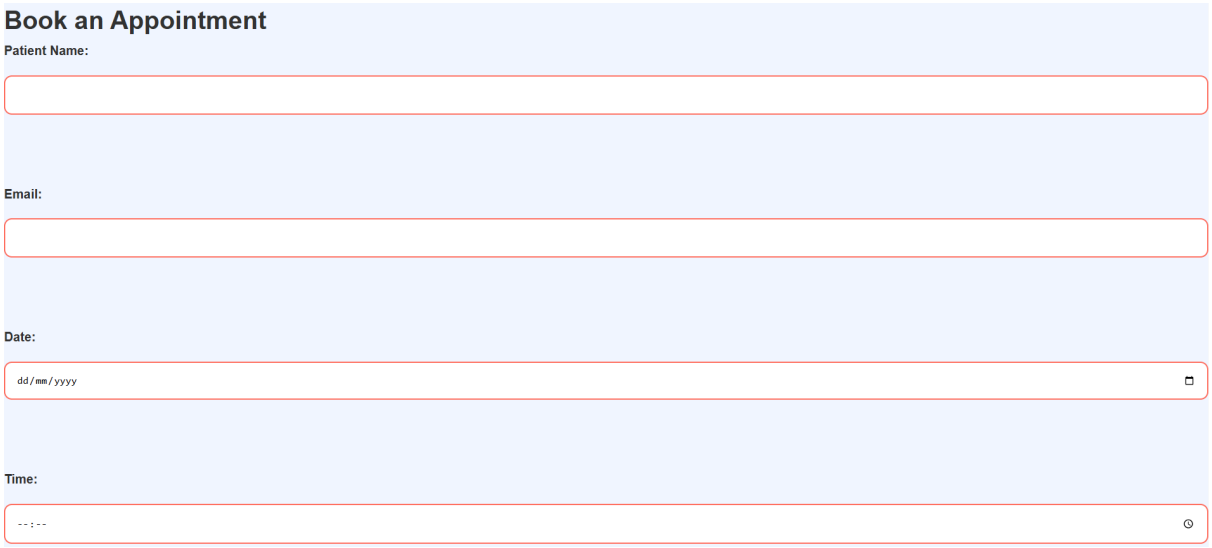
Username: gold

Password: smiths

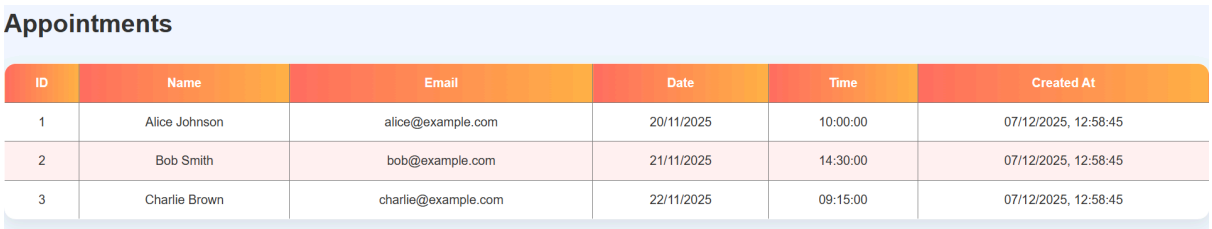
The account is inserted through the insert_test_data.sql script into the users when the application is deployed.

Functionality Screenshots

Book Appointment Form:



Appointments list table:



ID	Name	Email	Date	Time	Created At
1	Alice Johnson	alice@example.com	20/11/2025	10:00:00	07/12/2025, 12:58:45
2	Bob Smith	bob@example.com	21/11/2025	14:30:00	07/12/2025, 12:58:45
3	Charlie Brown	charlie@example.com	22/11/2025	09:15:00	07/12/2025, 12:58:45

Search Appointments:

Search Appointments

Patient Name:

Search

Registration Form:

User Registration

Username:

First Name:

Last Name:

Email:

Password:

Password Strength: Weak

Register

Login Page:

Login

Username:

Password:

Log In

Create an account

Audit Log:

Audit Log

ID	Username	Login Time	Success?	IP Address
4	gold	08/12/2025, 12:34:49	Yes	90.196.17.75
3	gold	08/12/2025, 11:44:43	Yes	90.196.17.75
2	gold	07/12/2025, 16:57:55	Yes	90.196.17.75
1	gold	07/12/2025, 13:06:58	Yes	90.196.17.75

Back to Login

API Appointments:

Pretty-print ☑


```
[
  {
    "id": 3,
    "name": "Charlie Brown",
    "email": "charlie@example.com",
    "date": "2025-11-22T00:00:00.000Z",
    "time": "09:15:00",
    "created_at": "2025-12-07T12:58:45.000Z"
  },
  {
    "id": 2,
    "name": "Bob Smith",
    "email": "bob@example.com",
    "date": "2025-11-21T00:00:00.000Z",
    "time": "14:30:00",
    "created_at": "2025-12-07T12:58:45.000Z"
  },
  {
    "id": 1,
    "name": "Alice Johnson",
    "email": "alice@example.com",
    "date": "2025-11-20T00:00:00.000Z",
    "time": "10:00:00",
    "created_at": "2025-12-07T12:58:45.000Z"
  }
]
```


API Stats:

Pretty-print ☑


```
{
  "total_appointments": 3,
  "unique_patients": 3,
  "earliest_appointment": "2025-11-20T00:00:00.000Z",
  "latest_appointment": "2025-11-22T00:00:00.000Z"
}
```


Advanced Techniques

This project includes several security and performance techniques demonstrating professional-level development skills.

1. Session-Based Access Control (Authentication Middleware)

You implemented a custom middleware (`redirectLogin`) to protect routes and ensure only logged in users can view these restricted pages:

auth/audit

auth/list

appointments/search

appointments/book

Why it's advanced:
It demonstrates understanding of authentication flows, session management, and secure route protection.

Code Example (appointments.js):

```
const redirectLogin = (req, res, next) => {

    if (!req.session.userId) {

        res.redirect("./login");

    } else {

        next();

    }

};
```

---

2. Secure Password Hashing with bcrypt

Passwords are never stored in plain text.
You hash them using bcrypt, which adds salting and prevents credential theft.

Why it's advanced:
Shows security awareness and proper cryptographic handling.

Code Example(appointments.js):

```
bcrypt.compare(password, hashedPassword, (err, match) => {

    if (match) {

        req.session.userId = username;
```

```
    }
});
```

---

3. Login Audit Logging (Security Monitoring)

Every login attempt (success or failure) is automatically logged with:

- username
- timestamp
- success/failure
- IP address

Why it's advanced:
This is a real-world security feature used in enterprise systems.

Code Example(appointments.js):

```
global.db.query(
  "INSERT INTO login_audit (username, success, ip_address) VALUES (?, ?, ?)",
  [username, false, ipAddress]
);
```

---

4. Rate Limiting on Login Attempts (Brute-Force Protection)

You added rate limiting to prevent repeated password guessing attacks.

Why it's advanced:
Shows knowledge of defending against brute-force attacks.

Code Example (appointments.js):

```
const loginLimiter = rateLimit({
```

```
    windowMs: 60 * 1000,
    max: 5,
    message: "Too many login attempts. Please try again later."
});
```

---

5. Prepared Statements for SQL Injection Prevention

All SQL queries use parameter placeholders (?), which protects your app from SQL injection.

Why it's advanced:
Modern secure coding practice.

Example(appointments.js):

```
const sql = "SELECT * FROM appointments WHERE name LIKE ?";
global.db.query(sql, [`%${name}%`], ...);
```

---

6. Connection Pooling (mysql2 built-in pool)

Your database uses a connection pool, meaning the app reuses DB connections instead of opening new ones.

Code example(index.js):

```
const db = mysql.createPool({
  host: process.env.HEALTH_HOST,
  user: process.env.HEALTH_USER,
  password: process.env.HEALTH_PASSWORD,
  database: process.env.HEALTH_DATABASE,
  connectionLimit: process.env.DB_CONN_LIMIT || 10
});
```

Why it's advanced:
Improves scalability, performance, and reduces resource usage.

---

7. Environment Variable Management (nano.env)

You moved database credentials into an `.env` file, using `dotenv` to securely load them.

Why it's advanced:
Used in professional deployments to prevent credential exposure which enhances the security of applications.

AI Declaration

I acknowledge the use of [1] ChatGPT (https://chat.openai.com/) and Grammarly()[2] assist with proofreading,grammar structure,clarifying programming concepts, and improving clarity in the assessment.. I entered the following prompts on Accessed: 16 November 2025 - 4 December 2025:

[3] Example prompts used:

“Explain why this error message appears in my code”

“What can I add to this diagram”

“Proofread my report and assess its content against the assessment criteria”

[4]Grammarly was used for improving sentence grammar and punctuation.

Reference

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