Outline

SmartClinic is a web-based clinic appointment booking system designed for a small medical practice. The application allows patients to register, log in, and book appointments with clinic staff, while staff users can manage appointments and patient information. The primary aim of the application is to demonstrate core Dynamic Web Application concepts such as server-side rendering, database driven functionality, authentication, authorisation, and search.

Patients can create accounts, view their upcoming appointments, and book new appointments by selecting a staff member, date, and reason. Staff users can log in to view all their appointments, update appointment statuses, and add clinical notes to patient profiles.

The application is built using Node.js, Express, EJS, and MySQL. The focus of the project is on backend logic, data modelling, and secure handling of user data rather than complex front-end frameworks.

The system supports role-based access control, ensuring that patients and staff only see functionality relevant to their role. Overall, SmartClinic provides a realistic and complete example of a health themed dynamic web application.

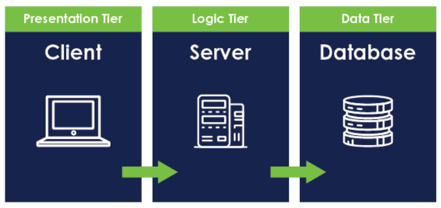
Architecture

The application follows a classic three tier architecture.

The presentation tier is responsible for the user interface and runs in the web browser. It uses EJS templates rendered on the server, along with HTML and CSS, to display pages and to collect user input through forms.

The application tier is built using Node.js and Express. It handles routing, business logic, session management, authentication, authorisation, and validation. Express routes process incoming requests, interact with the database, and pass data to the presentation tier for rendering.

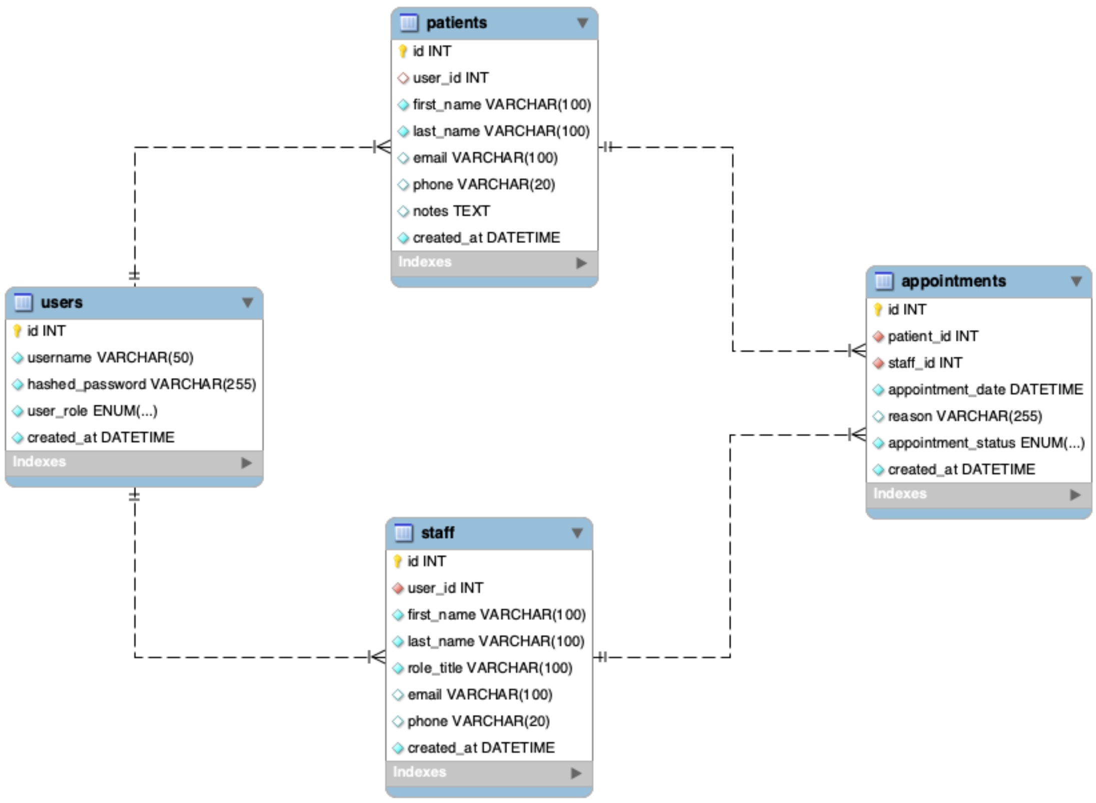
The data tier uses a MySQL database accessed via mysql2 with promise-based queries. It stores all persistent data, including users, patients, staff, and appointments, and enforces relationships using foreign keys. Environment variables are used to securely configure database connections and secrets.



Data Model

The data model consists of four main tables: *users*, *patients*, *staff*, and *appointments*. The *users* table stores authentication details and user roles. *Patients* and *staff* tables store role specific profile information and reference the *users* table via foreign keys. The *appointments* table links patients and staff together with appointment details such as date, reason, and status.

Foreign key constraints are used to maintain referential integrity. Cascading and null on delete rules ensure that deleting users does not leave orphaned records and that medical data remains consistent.



User Functionality

The home page provides an introduction to SmartClinic and navigation links.

An about page explains the purpose of the system.

Users can register for an account as a patient and then log in using their credentials. A default patient and staff account with username *gold_patient* and *gold_doctor* respectively, and password *smiths* is provided for testing.

Once logged in as a patient, users can access the patient dashboard. This page displays all the user's appointments and an option to cancel upcoming appointments. The book appointment page provides a form to book new appointments. Patients select a staff member, appointment date and time, and enter a reason for the visit. Search appointments allows users to search and filter through any of their past or upcoming appointments such as filtering by date, staff member, appointment status, or keywords in the reason.

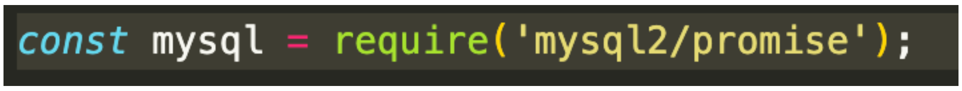
Staff users are redirected to the staff dashboard after logging in. The view appointments page displays all appointments belonging to that staff member and an option to cancel appointments. The patient search page allows staff to search for and view patient profiles and add or update clinical notes, which are stored securely in the database.

Validation ensures that required fields are completed and that users cannot access staff only routes.

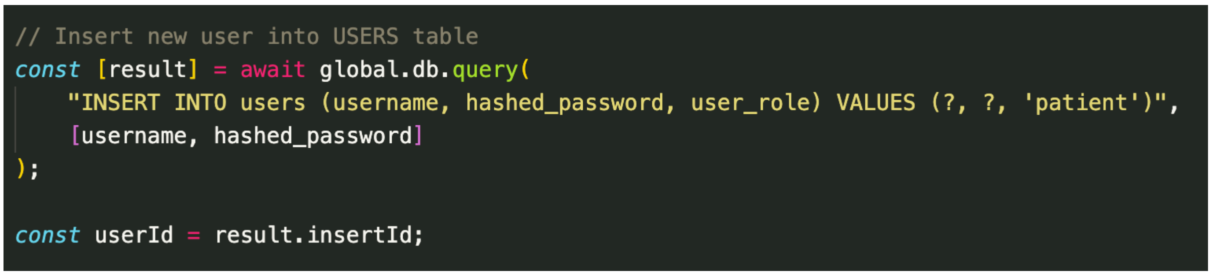
Navigation links update dynamically depending on whether a user is logged in and their role. Logout functionality is provided to securely end sessions.

Advanced Techniques

The application uses the mysql2/promise wrapper instead of the default callback based mysql2 interface. This allows database queries to return Promises, resulting in cleaner and more readable code. It avoids deeply nested callbacks and reduces repetitive error handling, making database logic easier to understand and maintain



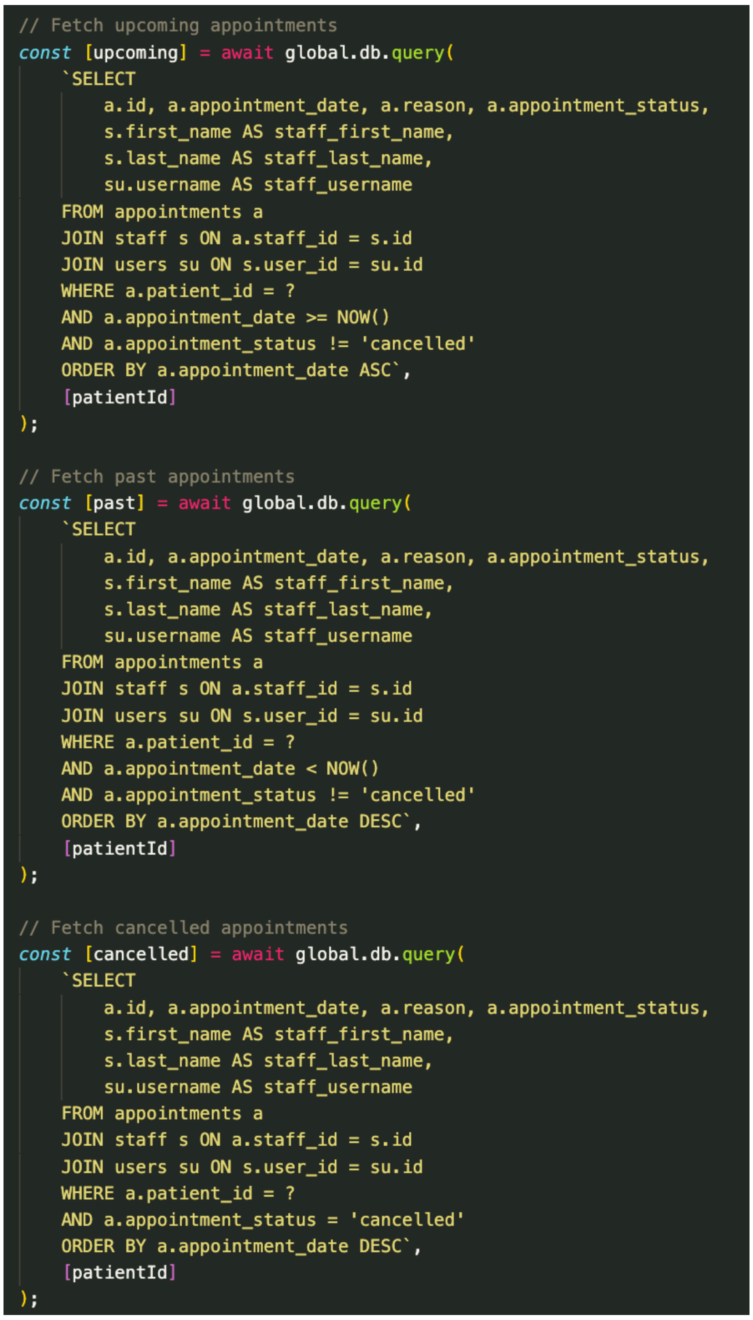
```javascript
const mysql = require('mysql2/promise');
```



```javascript
// Insert new user into USERS table
const [result] = await global.db.query(
    "INSERT INTO users (username, hashed_password, user_role) VALUES (?, ?, 'patient')",
    [username, hashed_password]
);

const userId = result.insertId;
```

Using the promise-based database interface allows route handlers to be written with async and await. Authentication, registration, and booking logic use await to handle database queries and password hashing in a linear way. This makes asynchronous code behave like sequential code, reducing errors and improving readability.

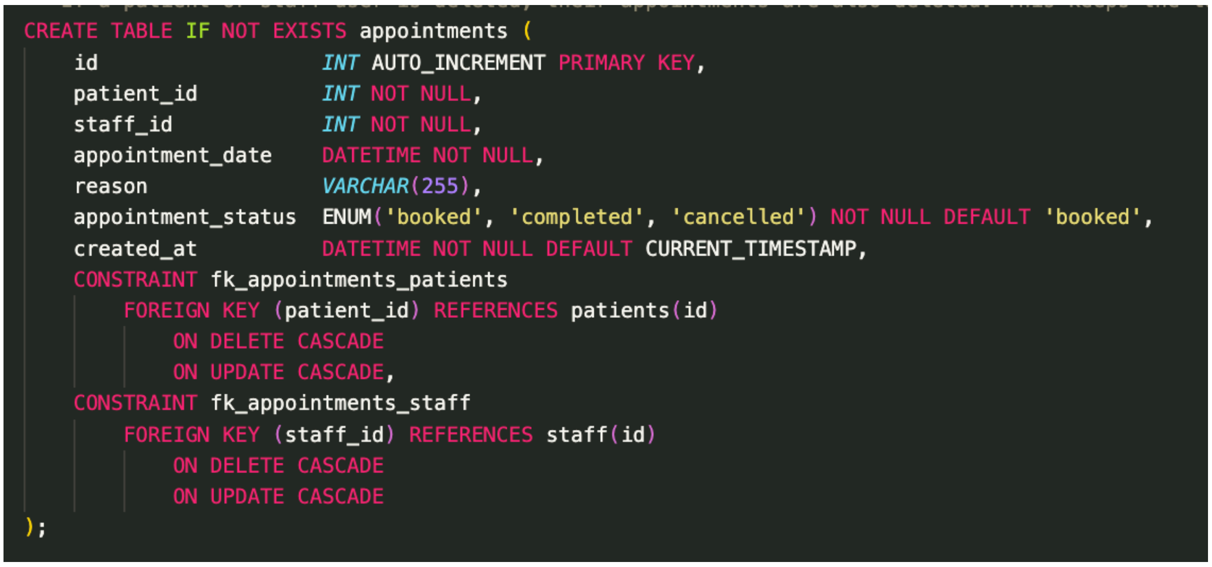


```
// Fetch upcoming appointments
const [upcoming] = await global.db.query(
    `SELECT
        a.id, a.appointment_date, a.reason, a.appointment_status,
        s.first_name AS staff_first_name,
        s.last_name AS staff_last_name,
        su.username AS staff_username
    FROM appointments a
    JOIN staff s ON a.staff_id = s.id
    JOIN users su ON s.user_id = su.id
    WHERE a.patient_id = ?
    AND a.appointment_date >= NOW()
    AND a.appointment_status != 'cancelled'
    ORDER BY a.appointment_date ASC`,
    [patientId]
);

// Fetch past appointments
const [past] = await global.db.query(
    `SELECT
        a.id, a.appointment_date, a.reason, a.appointment_status,
        s.first_name AS staff_first_name,
        s.last_name AS staff_last_name,
        su.username AS staff_username
    FROM appointments a
    JOIN staff s ON a.staff_id = s.id
    JOIN users su ON s.user_id = su.id
    WHERE a.patient_id = ?
    AND a.appointment_date < NOW()
    AND a.appointment_status != 'cancelled'
    ORDER BY a.appointment_date DESC`,
    [patientId]
);

// Fetch cancelled appointments
const [cancelled] = await global.db.query(
    `SELECT
        a.id, a.appointment_date, a.reason, a.appointment_status,
        s.first_name AS staff_first_name,
        s.last_name AS staff_last_name,
        su.username AS staff_username
    FROM appointments a
    JOIN staff s ON a.staff_id = s.id
    JOIN users su ON s.user_id = su.id
    WHERE a.patient_id = ?
    AND a.appointment_status = 'cancelled'
    ORDER BY a.appointment_date DESC`,
    [patientId]
);
```

A fully relational MySQL database is implemented with foreign keys linking users, patients, staff, and appointments. Cascading rules automatically remove related records, such as appointments when a staff member is deleted. This prevents orphaned data and ensures database consistency without additional application logic.



```sql
CREATE TABLE IF NOT EXISTS appointments (
    id                  INT AUTO_INCREMENT PRIMARY KEY,
    patient_id          INT NOT NULL,
    staff_id            INT NOT NULL,
    appointment_date    DATETIME NOT NULL,
    reason              VARCHAR(255),
    appointment_status  ENUM('booked', 'completed', 'cancelled') NOT NULL DEFAULT 'booked',
    created_at          DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_appointments_patients
        FOREIGN KEY (patient_id) REFERENCES patients(id)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT fk_appointments_staff
        FOREIGN KEY (staff_id) REFERENCES staff(id)
            ON DELETE CASCADE
            ON UPDATE CASCADE
);
```

AI Declaration

AI tools were used during this assignment to support learning and development. Specifically, AI was used to help explain concepts and generate test data for the SQL database. All code was written, understood, and integrated by the student, and no AI generated content was copied without review or modification. The final design decisions, implementation, and testing were carried out independently.