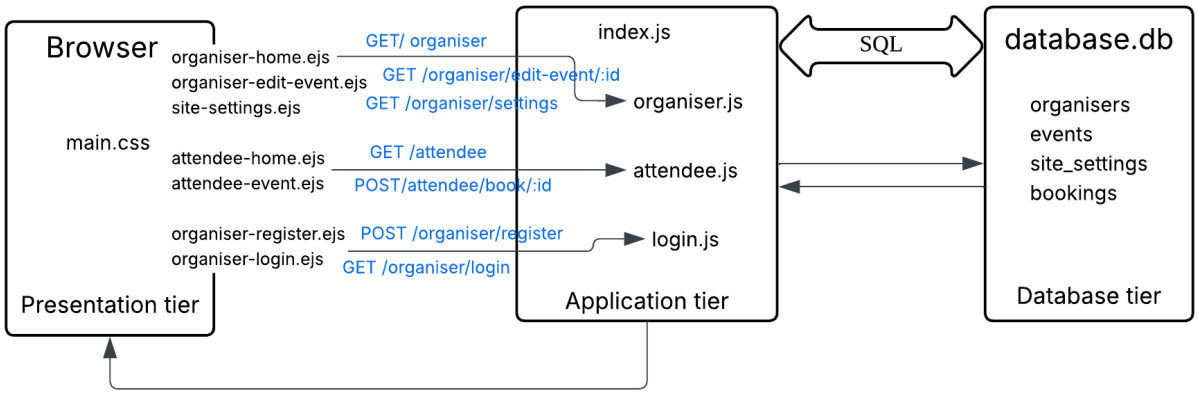
Website Architecture



The presentation tier includes a client-side interface which displays EJS views (.ejs templates). They are dynamically generated and sent to the browser and also include static assets such as main.css for styling. Users interact via GET/POST requests and form submissions.

The application tier contains the core logic in my node.js files. The index.js file serves as the main entry point and wires together all the other route handlers:

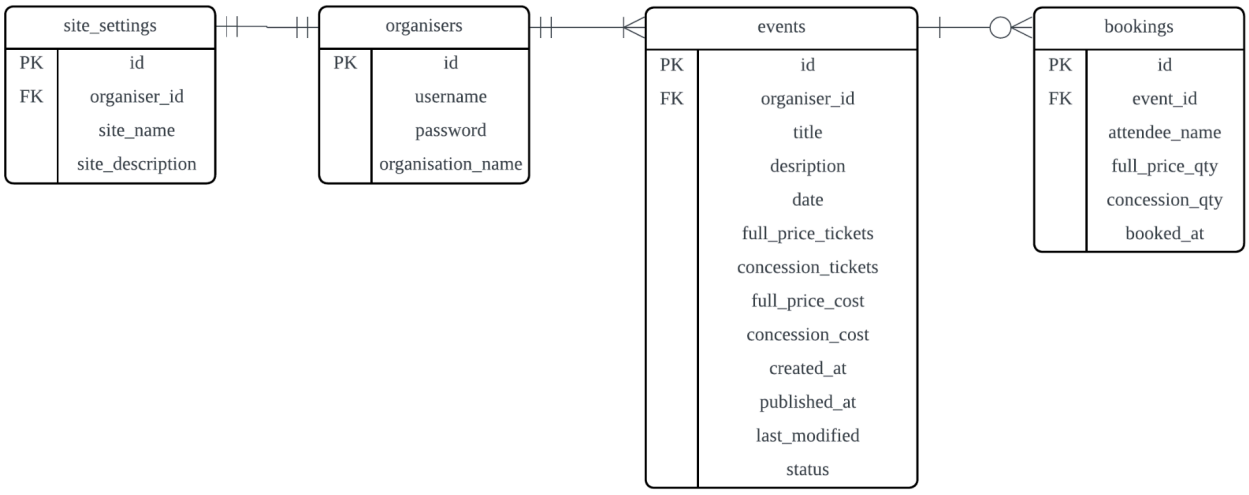
- organiser.js handles logic related to event creation, publishing, site settings, and organiser-specific pages.
- attendee.js manages ticket booking and public event views.
- login.js handles organiser registration, login, session creation, and authentication-related middleware.

These modules respond to HTTP requests from the presentation tier (e.g., GET /organiser, POST /organiser/login) and interact with the database using SQL queries.

The database tier stores a SQLite database which is structured into 4 main tables: organisers, events, site_settings and bookings. These tables are queried and updated by the application tier in response to user actions.

The login extension added new views (organiser-login.ejs and organiser-register.ejs) to allow users to input credentials. The login.js route handler was also implemented to manage POST requests for authentication and session control. Although the database schema remained unchanged, the organisers table's password field was activated for real-world authentication. Hashed passwords are stored securely during registration, and verified during login.

Data Model (ER diagram)



The organisers table stores the credentials and organisation names of registered event organisers. This table is linked to multiple other entities:

- Each organiser is associated with exactly one record in the site_settings table. This mandatory one-to-one relationship ensures that every organiser has their own customised site title and description.
- The events table is linked to organisers via a one-to-many (1:M) relationship. Each event is created by a single organiser (organiser_id), but an organiser can create multiple events.
- The bookings table links to events via one-to-many (1:M) relationship. A single event can receive multiple bookings (or may not have any bookings, hence, optional), but each booking is tied to one event only.

Although the login system from my extension doesn't add new tables or attributes into the schema, it adds security and better control of the existing organisers table. The password field is used with secure hashing (bcrypt) to authenticate organiser logins. Once they are authenticated, the organiser_id and organisation_name are stored in session variables, allowing access to events they created and their site_settings. Middleware also only allows organisers with valid sessions to access /organiser routes, depending on session state.

Extension

Overview

I implemented a password authentication system backed by session management. This ensures that only authenticated organisers can access pages such as the organiser home page, create event tools, and site settings.

Implementation

First, I created a registration and login system using a combination of bcrypt for secure password hashing, express-session for session handling, and custom route middleware to restrict access to authenticated users only. When a new organiser registers via the /organiser/register route, their password is hashed using bcrypt with a defined number of salt rounds (SALT_ROUNDS=10). Upon successful registration, the organiser is added to the organisers table, and default site_settings are inserted for that organiser which they can later customise. A flash message then prompts the organiser to proceed to the login page.

```
28
29          const hashedPassword = await bcrypt.hash(password, SALT_ROUNDS);
30
31          // Insert new organiser
32          const insertOrgQuery = `
33              INSERT INTO organisers (username, password, organisation_name)
34              VALUES (?, ?, ?)
```

```
61              //redirect user to login page to login again
62              req.flash("success", "Registration successful! You may now log in.");
63              res.redirect("/organiser/login");
```

The login route (/organiser/login) authenticates users by comparing the submitted password to the hashed one stored in the database using bcrypt.compare. If the credentials match, an Express session is initiated, storing key organiser metadata in req.session such as isOrganiser, organiserId, and organisationName.

```
144         req.session.isOrganiser = true;
145         req.session.organiserId = organiser.id;
146         req.session.organisationName = organiser.organisation_name;
147         res.redirect("/organiser");
```

To enforce route protection, a custom middleware is placed after all login/registration routes. This middleware checks whether req.session.isOrganiser is true before granting access to any subsequent /organiser routes. If not, it redirects the user back to the login screen. This ensures

that session-less users (e.g., unauthenticated or after logout) cannot access organiser-only content.

```
154  router.use((req, res, next) => {
155    if (req.session.isOrganiser || req.path === '/login') {
156      return next();
157    }
158    return res.redirect("/organiser/login");
159  });
```

However, if an organiser logs in and does not explicitly log out, their session remains active. This means the browser retains access to organiser pages unless the session expires or is cleared. Thus, the login page isn't shown again until the organiser logs out manually via /organiser/logout, which destroys the session.

Design decisions

To enhance user experience, I also integrated the connect-flash package, which enables contextual feedback messages to appear on actions such as registration success, login failure, or ticket booking issues. These messages are passed through the session and rendered via the EJS views.

Overall, this extension improves the security and integrity of the platform by ensuring only authorised organisers can manage their events and settings. It also offers a relatively professional and user-friendly authentication experience, and the structure makes it easy to scale for future multi-user enhancements (e.g password resets or role-based permissions).

Bibliography

Bcrypt. npm. (n.d.). https://www.npmjs.com/package/bcrypt

Express. npm. (n.d.). https://www.npmjs.com/package/express

Express-session. npm. (n.d.). https://www.npmjs.com/package/express-session

Connect-flash. npm. (n.d.). http://npmjs.com/package/connect-flash

EJS. npm. (n.d.). https://www.npmjs.com/package/ejs