Databases, Network and the Web

Midterm Coursework Report

Blogging Tool

Introduction

The purpose of this project was to develop a comprehensive blogging tool that allows users to create, edit, and publish articles, while also enabling readers to view and comment on these articles. The major aim of this project is to provide a user-friendly platform for authors to share their thoughts and ideas with a wider audience and for readers to engage with content through comments and likes.

The blogging tool was built using a modern web development stack that includes Node.js for the backend, Express.js as the web framework, SQLite as the database, and EJS as the templating engine. Bootstrap and custom CSS styles were used for the frontend design to ensure a responsive and visually appealing interface.

Structure of the Blogging Tool

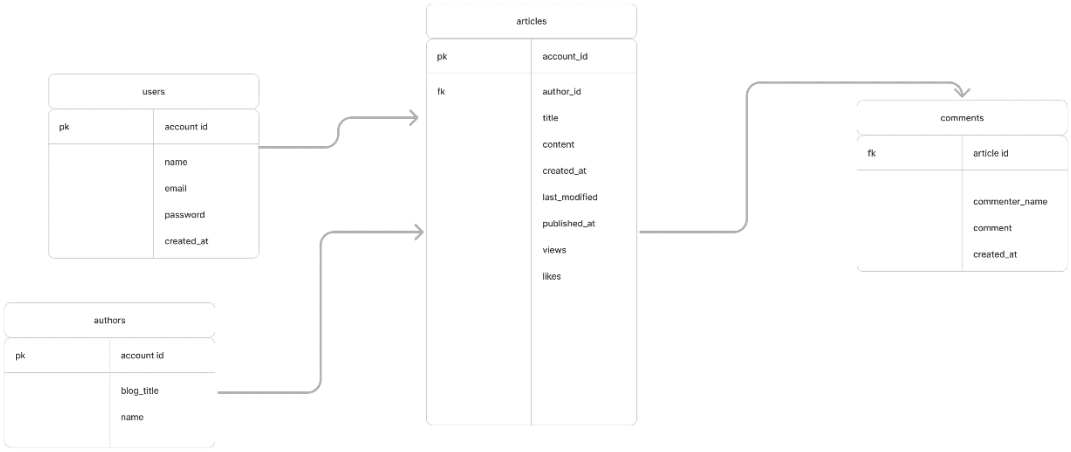
The blogging tool is structured using a three-tier architecture, which includes the client-side(frontend), server-side(backend), and the database. This modular approach ensures separation of concerns, making the system easier to manage and maintain

The Client side of the blogging tool is responsible for the user interface and experience which includes the browser which is the platform where users interact with the application, EJS templates used to dynamically render HTML content based on the data provided by the server, CSS styling frameworks and custom styles used to ensure a responsive and visually appealing interface and Client-side Javascript that are scripts added interactivity and enhance the user experience.
The server-side handles the logic, processing requests and interacting with the database which includes express.js server, middleware which are additional layers that process requests before they reach the route handlers.

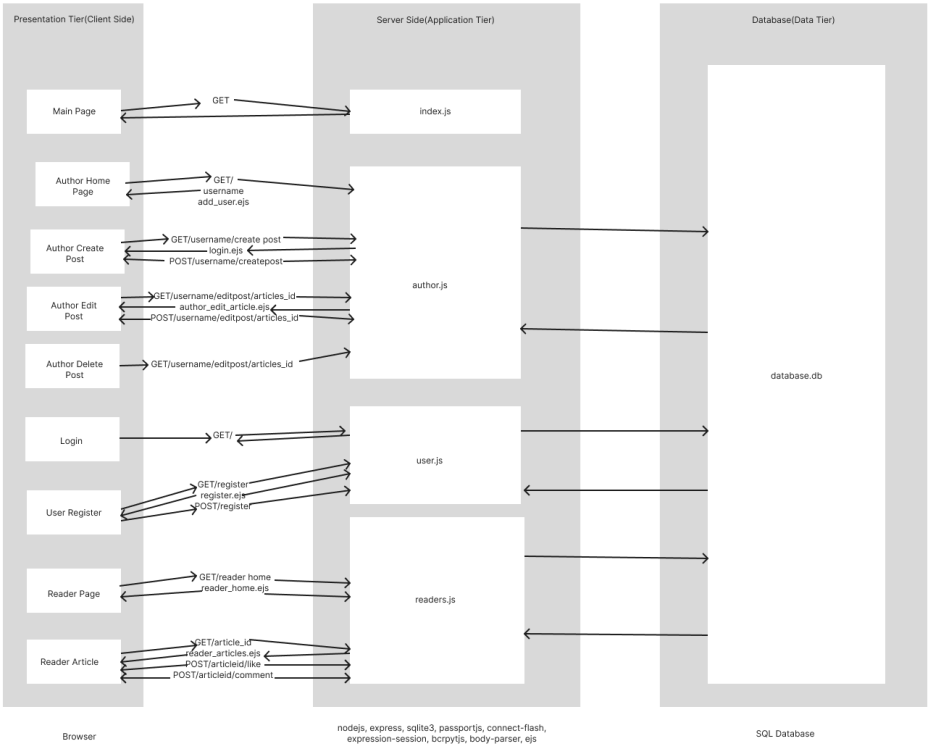
Finaly the database layer is responsible for data storage and retrieval. It includes SQLite Database, which is a lightweight, file-based database used to store application data and database schema which defines the structure of the database, including tables and relationships.

You can find the necessary diagrams below.

Entity-Relationship Diagram



System Architecture Diagram



Extension Description: Password Access for Author Pages and Routes

The extension implemented for this project is password access for author pages and routes. This ensures that only authenticated authors can access the author-specific functionalities for the blogging too. It includes creating an author login page, authenticating authors against a securely stored password, and using sessions to maintain secure access throughout the session.

To secure the author pages and routes, middleware was implemented to check for authentication status. This middleware prevents unauthorized access to author endpoints.

File ‘config/auth.js’

Line 1-7

module.exports = {

```
    ensureAuthenticated: function(req, res, next) {
        if (req.isAuthenticated()) {
            return next();
        }
        req.flash('error_msg', 'Please log in to view that resource');
        res.redirect('/users/login');
    }
};
```

This middleware checks if the user is authenticated using ‘req.isAuthenticated()’. If the user is authenticated, the request proceeds to the next middleware or route handler. If not, the user is redirected to the login page with an error message

Passport configuration, Passport.js was used to handle the authentication process. A local strategy was configured to authenticate authors using an email and password.

File ‘config/passport.js’

Line 1-33

```
const LocalStrategy = require('passport-local').Strategy;
const bcrypt = require('bcryptjs');
const sqlite3 = require('sqlite3').verbose();
const db = new sqlite3.Database('./database.db');

module.exports = function(passport) {
    passport.use(
        new LocalStrategy({ usernameField: 'email' }, (email, password, done) => {
            db.get('SELECT * FROM users WHERE email = ?', [email], (err, user) => {
                if (err) throw err;
                if (!user) {
                    return done(null, false, { message: 'No user found' });
                }

                bcrypt.compare(password, user.password, (err, isMatch) => {
```

```
                if (err) throw err;
                if (isMatch) {
                    return done(null, user);
                } else {
                    return done(null, false, { message: 'Password incorrect' });
                }
            });
        });
    })
);

passport.serializeUser((user, done) => {
    done(null, user.id);
});

passport.deserializeUser((id, done) => {
    db.get('SELECT * FROM users WHERE id = ?', [id], (err, user) => {
        done(err, user);
    });
});
};
```

This configuration sets up a local strategy for Passport.js to authenticate users by checking the provided email and password against the database records. The passwords are securely hashed using ‘bcryptjs’.

An Author login page was then created to allow authors to log in and access the author-specific routes

File ‘views/login.ejs’

Line 1-50

```
<!DOCTYPE html>
```

```
<html>
<head>
    <title>Login</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
    <link rel="stylesheet" href="/main.css">
    <style>
        body, html {
            height: 100%;
            display: flex;
            justify-content: center;
            align-items: center;
            background-color: #f8f9fa;
        }
        .container {
            max-width: 800px;
        }
        .row {
            display: flex;
            flex-direction: row;
            align-items: center;
            justify-content: center;
        }
        .welcome-message {
            text-align: center;
            color: #007bff;
        }
        .subtext {
            color: #343a40;
```

```
            text-align: center;
        }
        .gradient-button {
            background: linear-gradient(to right, #007bff, #00c6ff);
            border: none;
            color: white;
        }
        .gradient-button:hover {
            background: linear-gradient(to left, #007bff, #00c6ff);
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="row">
            <div class="col-md-6 welcome-message">
                <h1>Welcome Author</h1>
                <p class="subtext">Please log in to access your author dashboard and manage your articles.</p>
            </div>
            <div class="col-md-6">
                <h1 class="mb-4">Login</h1>
                <form action="/users/login" method="POST">
                    <% if (error_msg) { %>
                        <div class="alert alert-danger">
                            <p><%= error_msg %></p>
                        </div>
                    <% } %>
                    <div class="form-group">
```

```
                    <label for="email">Email:</label>
                    <input type="email" id="email" name="email" class="form-control"
required>
                </div>
                <div class="form-group">
                    <label for="password">Password:</label>
                    <input type="password" id="password" name="password"
class="form-control" required>
                </div>
                <button type="submit" class="btn gradient-button">Login</button>
            </form>
            <p class="mt-2">Don't have an account? <a
href="/users/register">Register here</a></p>
        </div>
    </div>
</div>
</body>
</html>
```

This login page provides a form for authors to enter their email and password. Upon submission, the form sends a POST request to the ‘user/login’ route to authenticate the author.

The ‘express-session’ package was used to manage user sessions, ensuring that authenticated authors remain logged in as they navigate through the author -specific pages

File ‘index.js’

Line 13-21

```
const session = require('express-session');


// Express session
app.use(
    session({
```

```
        secret: 'secret',
        resave: true,
        saveUninitialized: true
    })
);

// Passport middleware
app.use(passport.initialize());
app.use(passport.session());
```

This configuration sets up session management using ‘express-session’. The session secret is used to sign the session ID cookie, ensuring its integrity and security.

In conclusion, the middleware ensures that only authenticated authors can access author-specific routes, enhancing security. Passwords are securely hashed using ‘bcryptjs’, preventing plain text password storage. Sessions are managed using ‘express-session’, maintaining secure access throughout the user session and all these implementations provide a secure mechanism for protecting author-specific functionalities, ensuring that only authorized users can access and manage the content.