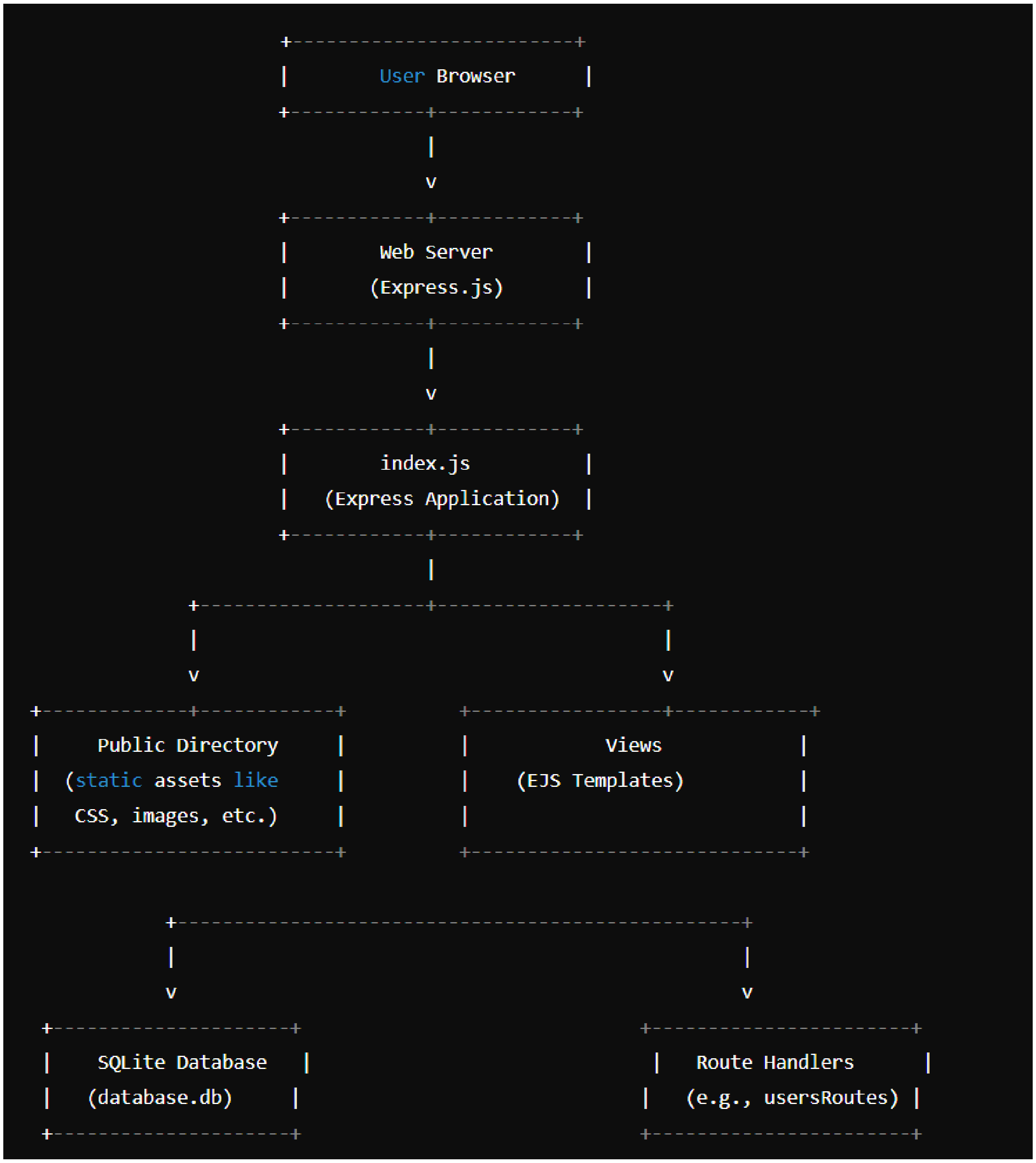
MIDTERM PROJECT REPORT

1. ARCHITECTURE DIAGRAM



```
                      +-------------------------+
                      |        User Browser      |
                      +------------+------------+
                                   |
                                   v
                      +------------+------------+
                      |        Web Server        |
                      |       (Express.js)       |
                      +------------+------------+
                                   |
                                   v
                      +------------+------------+
                      |        index.js          |
                      |   (Express Application)  |
                      +------------+------------+
                                   |
              +--------------------+--------------------+
              |                                         |
              v                                         v
+-------------+------------+          +-----------------+------------+
|     Public Directory     |          |            Views            |
|  (static assets like     |          |    (EJS Templates)          |
|   CSS, images, etc.)     |          |                             |
+--------------------------+          +-----------------------------+

            +--------------------------------------------------+
            |                                                  |
            v                                                  v
 +---------------------+                              +-----------------------+
 |    SQLite Database   |                              |   Route Handlers      |
 |   (database.db)     |                              |   (e.g., usersRoutes) |
 +---------------------+                              +-----------------------+
```

1. **User Browser**: Represents the end-user accessing your web application through a web browser.
2. **Web Server (Express.js)**: Handles incoming HTTP requests from clients and routes them to appropriate handlers.
3. **index.js (Express Application)**: Main entry point of your Node.js application using Express. Sets up middleware, routes, and starts the server.
4. **Public Directory**: Stores static assets like CSS, images, and client-side JavaScript files. Served directly to clients by Express.
5. **Views (EJS Templates)**: Contains EJS templates (e.g., author-home.ejs) rendered by Express to generate HTML pages dynamically based on data from the server.
6. **SQLite Database (database.db)**: Local database used to persist application data, such as user information, blog posts, and drafts.
7. **Route Handlers**: Modules that define and handle application routes (e.g., /, /about, /users/add-user). These modules interact with the database as needed and render EJS templates to generate dynamic content.

Key Components:

- **Express.js**: Provides a framework for building web applications in Node.js, handling routing, middleware, and rendering.
- **SQLite**: Embedded SQL database engine used for local storage of structured data.
- **EJS**: Templating language for generating HTML markup with plain JavaScript.

This architecture diagram outlines the flow of data and interactions between components in your web application, illustrating how requests from users are processed, data is retrieved from the database, and dynamic web pages are generated and served back to clients.

2. DATABASE MODEL DIAGRAM

```
+---------------------+       +---------------------+
|       users         |       |       drafts        |
+---------------------+       +---------------------+
| PK user_id          |       | PK draft_id         |
| username            | 1----<| FK user_id          |
| password            |       | title               |
| blog_title          |       | body                |
+---------------------+       | created             |
                              | last_modified       |
                              +---------------------+
                                           ^
                                           |
                                           |
                              +---------------------+
                              |      blogpost       |
                              +---------------------+
                              | PK blogpost_id      |
                         1----| FK user_id          |
                              | title               |
                              | body                |
                              | date_published      |
                              | last_modified       |
                              | likes               |
                              | views               |
                              +---------------------+
                                           ^
                                           |
                                           |
                              +---------------------+
                              |      comments       |
                              +---------------------+
                              | PK comment_id       |
                         1----| FK blogpost_id      |
                              | username            |
                              | body                |
                              | date                |
                              +---------------------+
```

- **users**: Contains information about users who can create drafts and blog posts. Each user is identified by a unique user_id.
- **drafts**: Represents draft blog posts created by users. Each draft has a user_id referencing the user who created it.
- **blogpost**: Stores published blog posts. Each blog post has a user_id indicating the author.
- **comments**: Holds comments made on blog posts. Each comment is associated with a specific blogpost_id.

Relationships:

- **One-to-Many (1**

):

  - A user can have multiple drafts (users to drafts).
  - A user can publish multiple blog posts (users to blogpost).
  - A blog post can have multiple comments (blogpost to comments).

Cardinality Notation:

- **One-to-Many** relationships are denoted by a line with a crow's foot (|----<) indicating "many" and a straight line (1----) indicating "one".

This diagram illustrates how the tables relate to each other through their primary and foreign keys.

3. EXTENSION DESCRIPTION:

   a. **express-session Implementation**
      What was implemented: The express-session middleware was integrated to manage user sessions, allowing persistent login state across requests.

      Implementation Details:
      Middleware Configuration:

```
const session = require('express-session'); const router = express.Router();
router.use(session({ secret: 'your-secret-key', // Replace with a strong secret
key resave: false, saveUninitialized: true, cookie: { secure: false } // Set to true
if using HTTPS }));
```

- **secret:** A secret key used to sign the session ID cookie. It should be kept confidential and strong.
- **resave and saveUninitialized:** These options configure session behavior. Setting resave to false prevents session data from being saved on every request. saveUninitialized ensures uninitialized sessions (new and not modified) get saved.
- **cookie:** Configures the session cookie settings. Here, secure is set to false for non-HTTPS environments.

Usage in routes:

Sessions are utilized to store user information (req.session.user) after successful login, allowing subsequent requests to identify the user.

b. **bcrypt Implementation**

What was implemented: The bcrypt library was used for secure password hashing and comparison during user authentication.

Implementation Details:
Hashing Passwords:

```
const bcrypt = require('bcrypt'); bcrypt.hash(password, 10, (err, hash) => { //
Store 'hash' in the database for the user's password });
```

- bcrypt.hash() salts and hashes the plaintext password (password) with a salt round of 10, generating a secure hash.
- The resulting hash is then stored in the database alongside the user's other information.

Password Comparison (Login):

```
bcrypt.compare(password, user.password, (err, isMatch) => { if (isMatch) { //
User password is correct } else { // Invalid password } });
```

 bcrypt.compare() compares the plaintext password entered during login with the stored hashed password (user.password) retrieved from the database.
 If isMatch is true, the user is authenticated successfully; otherwise, the login attempt fails.

Key Aspects to Highlight:

- **Security:** Emphasize the use of bcrypt for secure password handling, mitigating risks associated with storing plaintext passwords.
- **Session Management:** Discuss how express-session facilitates persistent sessions, ensuring a seamless user experience across pages.
- **Code Organization:** Ensure clarity by indicating where each extension is implemented (e.g., filenames and line numbers), crucial for code review and maintenance.

This approach ensures robust user authentication and session management within the application, enhancing security and user experience simultaneously.