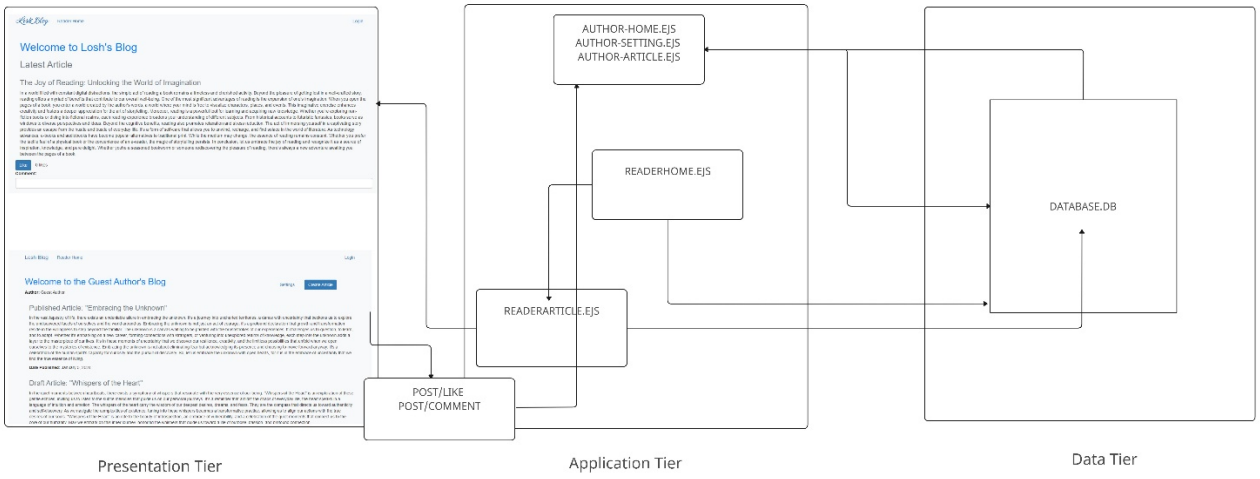
Losh Blog Website Report

High Level Schematic Diagram:



Implementation Overview:

Changes in author.js:

I added a new route for handling the creation of a new draft article.

Modified the existing route for handling article creation and update to differentiate between creating a new draft and updating an existing article.

```
    // New route for creating a new draft article
router.get('/article/new', (req, res) => {
    res.render('author/article-edit', {
        title: 'Create New Draft Article',
        article: new Article(),
    });
});

});
```

The new route /article/new is responsible for rendering the author/article-edit view, providing a title for the page and creating a new Article object. This object is then passed to the view, allowing the author to start fresh when creating a new draft.

1. Separation of Article Creation and Update Logic:

To enhance clarity and maintainability, the code has been modified to distinguish between creating a new article, updating an existing article, and creating a new draft article.

This ensures that the logic for each operation is well-defined and easier to follow.

2. New Draft Article View:

The introduction of a new route for creating a draft article also involved creating a dedicated view (author/article-edit.ejs).

This view provides the author with a form to input the title and content of the new draft article.

3. Reusable Article Object:

The Article object is used to represent an article, and creating a new instance of it for the draft article ensures consistency in data structure throughout the application.

This contributes to maintainability and avoids redundancy in the code.

4. Enhanced User Experience:

The new functionality aims to enhance the user experience for authors by providing a straightforward way to initiate the creation of a draft article.

The separation of routes and views makes the process intuitive and aligns with common expectations of content creation workflows.

Changes in reader.js:

Added a new route to handle AJAX requests for liking an article.

Modified the existing route for rendering article pages to include information about the reader's liking status for each article.

Changes in reader.ejs:

Integrated the "Like" button into the article view to allow readers to like or unlike an article.

Dynamically updated the "Like" button based on the reader's liking status for the article.

```
// Handle AJAX request for liking an article
router.post('/article/:id/like', async (req, res) => {
    const articleId = req.params.id;
    const userId = req.session.user.id;

    try {
        // Check if the user has already liked the article
        const likeCheckSql = 'SELECT * FROM Likes WHERE ArticleId = ? AND UserId = ?';
        const likeCheckResult = await db.get(likeCheckSql, [articleId, userId]);

        if (likeCheckResult) {
            // User has already liked the article, unlike it
            const unlikeSql = 'DELETE FROM Likes WHERE ArticleId = ? AND UserId = ?';
            await db.run(unlikeSql, [articleId, userId]);
            res.json({ liked: false });
        } else {
            // User has not liked the article, like it
            const likeSql = 'INSERT INTO Likes (ArticleId, UserId) VALUES (?, ?)';
            await db.run(likeSql, [articleId, userId]);
            res.json({ liked: true });
        }
    } catch (error) {
        console.error(error.message);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});
```

```
// Reader Article
router.get('/article/:id', checkUserAccess, (req, res) => {
    const _articleId = req.params.id;
    const sqlArticle = 'SELECT * FROM Articles WHERE id = ?';
    const sqlComments = `SELECT Comments.*, userLoginInfo.user_name AS username FROM Comments JOIN userLoginInfo ON Comments.UserId = userLoginInfo.user_id WHERE ArticleId = ? ORDER BY createdAt DESC`;

    let _userId = null;
    if(req.session.user) {
        _userId = req.session.user.id;
    }

    const sqlLike = _userId ? `SELECT * FROM Likes WHERE ArticleId = ? AND UserId = ?` : null;

    db.get(sqlArticle, [_articleId], (err, row) => {
        if (err) {
            return console.error(err.message);
        }

        if (row) {
            row.articleCreation = `This article was created on ${new Date(row.createdAt).toLocaleDateString()}`;

            db.all(sqlComments, [_articleId], (err, comments) => {
                if (err) {
                    console.log('error9');
                    return console.error(err.message);
                }

                if(_userId && sqlLike) {
                    db.get(sqlLike, [_articleId, _userId], function(err, like) {
                        if (err) {
                            console.log('error99');
                            return console.error(err.message);
                        }
                        res.render('readerarticle', {
                            article: row,
                            comments,
                            userHasLiked: like != null
                        });
                    });
                } else {
                    res.render('readerarticle', {
                        article: row,
                        comments,
                        userHasLiked: false
                    });
                }
            });
        } else {
            res.redirect('/reader');
        }
    });
});
```

1. Enhanced User Interaction:

The "Like" feature introduces a new layer of interaction for readers, allowing them to express their appreciation for articles with a simple click.

This engagement feature enhances the overall user experience, making the platform more dynamic and responsive.

2. AJAX Handling for Like/Unlike:

The implementation utilizes AJAX to handle asynchronous requests for liking or unliking an article without requiring a full page reload.

This contributes to a smoother and more seamless interaction for readers.

3. Dynamic "Like" Button Appearance:

The "Like" button dynamically updates its appearance based on the reader's liking status for the article.

This visual feedback ensures that readers are aware of their liking status and provides a more intuitive interface.

4. Server-Side Handling of Like Status:

The server-side logic includes checking whether the currently logged-in user has already liked the article.

This information is then passed to the reader.ejs view, allowing for the dynamic adjustment of the "Like" button.

5. Error Handling:

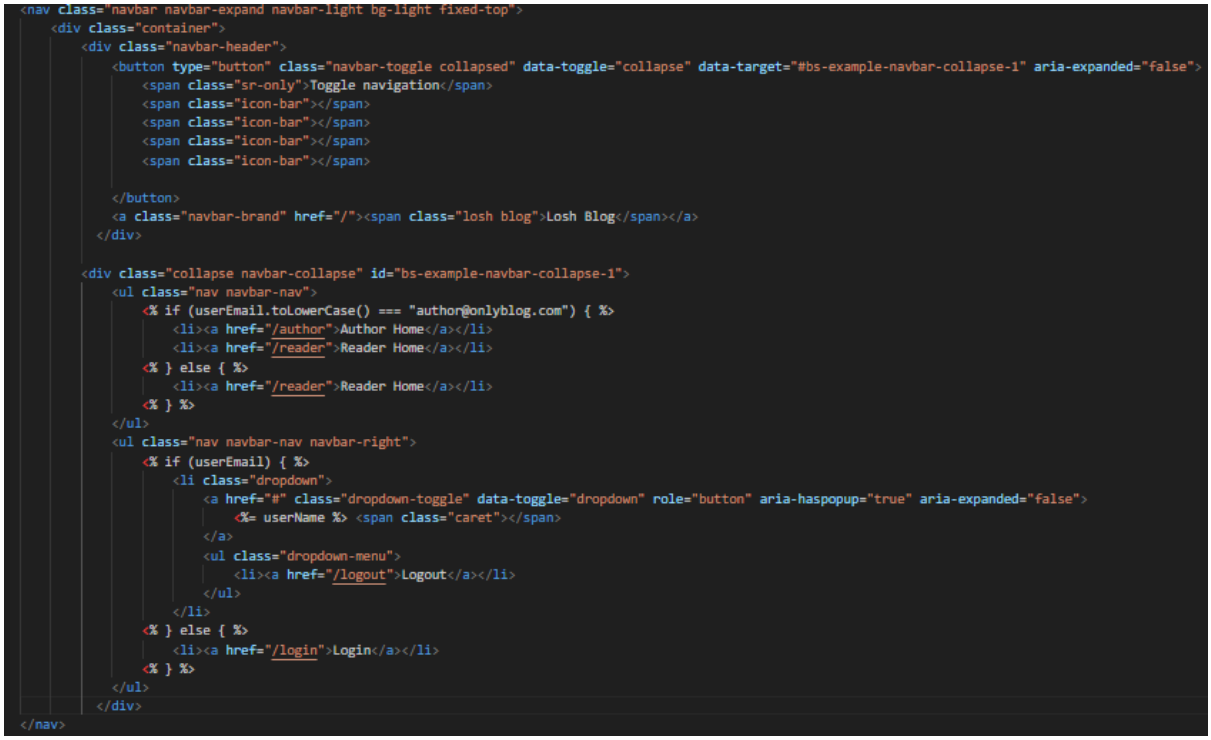
Appropriate error handling has been implemented to address potential issues, such as server errors or failures in database operations.

This ensures a robust and reliable experience for readers interacting with the "Like" feature.

In the above excerpt, the navigation bar includes links to the main sections of the website, such as "Author" and "Reader." The visibility of these links is dependent on whether a user is logged in (userLoggedIn variable).

Navigation Bar Structure (navbar.ejs):

The navigation bar is a shared component across multiple pages and is implemented in a modular fashion using a partial template (navbar.ejs). This approach ensures consistency in the layout and functionality of the navigation bar. The navbar.ejs file encapsulates the structure and functionality of the navigation bar. It employs the Bootstrap framework for styling and responsiveness, ensuring a visually appealing and user-friendly experience. The navigation bar includes a brand link, general navigation links, and dynamic links that are conditionally displayed based on the user's authentication status.



```
<nav class="navbar navbar-expand navbar-light bg-light fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1" aria-expanded="false">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>

            </button>
            <a class="navbar-brand" href="/"><span class="losh blog">Losh Blog</span></a>
          </div>

        <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
            <ul class="nav navbar-nav">
                <% if (userEmail.toLowerCase() === "author@onlyblog.com") { %>
                    <li><a href="/author">Author Home</a></li>
                    <li><a href="/reader">Reader Home</a></li>
                <% } else { %>
                    <li><a href="/reader">Reader Home</a></li>
                <% } %>
            </ul>
            <ul class="nav navbar-nav navbar-right">
                <% if (userEmail) { %>
                    <li class="dropdown">
                        <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">
                            <%= userName %> <span class="caret"></span>
                        </a>
                        <ul class="dropdown-menu">
                            <li><a href="/logout">Logout</a></li>
                        </ul>
                    </li>
                <% } else { %>
                    <li><a href="/login">Login</a></li>
                <% } %>
            </ul>
          </div>
</nav>
```

Index Page (index.ejs):

The index page serves as the entry point to the Losh Blog website. It displays a list of articles, introducing readers to the content available on the platform. The navigation bar allows users to seamlessly navigate to other sections. The index.ejs page represents the main landing page of the Losh Blog. It incorporates the navigation bar by including the navbar.ejs partial. This inclusion is achieved using the <% include navbar %> syntax, emphasizing modularity and code reusability.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <%- include ('base/head'); %>
  <title>Author Setting</title>
  <style>
      body {
          background-color: #f8f9fa;
      }
      .card {
          background-color: #ffffff;
      }
  </style>
</head>
    <%- include ('base/navibar'); %>
    <div class="container mt-4">
    <a href="/author" class="btn btn-secondary" style="margin-bottom: 20px;">&lt; Back to Author Home</a>
    <h1 class="my-4" style="color: #007bff;">Author Settings</h1>
</div>

<div class="container thinner-container">
    <form action="/author/setting/update" method="POST">
        <div class="form-group">
            <label for="blogTitle">Blog Title:</label>
            <input type="text" class="form-control" id="blogTitle" name="blogTitle" value="<%- blogTitle %>" required>
        </div>
        <div class="form-group">
            <label for="blogSubtitle">Blog Subtitle:</label>
            <input type="text" class="form-control" id="blogSubtitle" name="blogSubtitle" value="<%- blogSubtitle %>" required>
        </div>
        <button type="submit" class="btn btn-primary">Save Settings</button>
    </form>

    <!-- Add the following code to display a form for blog settings -->
    <h2>Settings</h2>
    <form action="/author/setting/update" method="post">
        <div class="form-group">
            <label for="blogTitle">Blog Title:</label>
            <input type="text" id="blogTitle" name="blogTitle" value="<%- blogTitle %>" required>
        </div>
        <div class="form-group">
            <label for="authorName">Author Name:</label>
            <input type="text" id="authorName" name="authorName" value="<%- authorName %>" required>
        </div>
        <div class="form-group">
            <button type="submit">Update Settings</button>
        </div>
    </form>
    <a href="/author">Back to Home Page</a>
</div>

</body>
</html>
```

Linking and Navigation Logic:

The navigation links are implemented using standard HTML anchor (<a>) tags. Each link corresponds to a specific route in the Express.js application, defined in the server-side JavaScript code.

Code Example :

html

Copy code

<!-- Anchor tag for navigating to the Author page -->

Author

The href attribute of the anchor tag specifies the path to the corresp onding route, allowing users to navigate seamlessly between pages.

The Express.js routes for these pages, such as /author and /reader, are handled on the server side, ensuring that the appropriate content is rendered based on user requests.

Dynamic Navigation Bar:

The visibility of certain navigation links, such as "Author" and "Reader," is dynamically controlled based on whether a user is logged in. This dynamic behavior is achieved by conditionally rendering these links in the navbar.ejs partial.

html

Copy code

<!-- Conditional rendering of "Author" and "Reader" links in navbar.ejs -->

<% if (userLoggedIn) { %>

<!-- ... (Render "Author" and "Reader" links) -->

<% } %>

This ensures that users see relevant navigation options based on their authentication status.