index.js

```
const express = require('express');
// Start
const session = require('express-session');
const helmet = require('helmet');
// End
const bodyParser = require('body-parser');
const compression = require('compression');
// Start
const path = require('path');
const dotenv = require('dotenv');
// End
const sqlite3 = require('sqlite3').verbose();
// Start
dotenv.config();
// End
const app = express();
const port = 3000;

// Set up SQLite
global.db = new sqlite3.Database('./database.db', function(err) {
    if (err) {
        console.error(err);
        process.exit(1); // bail out we can't connect to the DB
    } else {
        console.log("Database connected");
        global.db.run("PRAGMA foreign_keys=ON"); // tell SQLite to pay attention to foreign key constraints
    }
});

// Start
// Security Best Practices
app.use(helmet());
app.disable('x-powered-by');

// Middleware setup
app.use(compression()); // Compress responses
app.use(bodyParser.urlencoded({ extended: true })); // Parse URL-encoded bodies

app.set('view engine', 'ejs');

// Static file caching
app.use(express.static(path.join(__dirname, 'public'), { maxAge: '1y' }));

// Session management with secure cookies
app.use(session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: true,
    cookie: {
        httpOnly: true,
        secure: process.env.NODE_ENV === 'production', // Ensure cookies are secure in production
        sameSite: 'lax',
        maxAge: 60 * 60 * 1000 // 1 hour
    }
}));

// Prevent caching for dynamic responses
app.use((req, res, next) => {
    res.setHeader('Cache-Control', 'no-store');
    next();
});

// Middleware and Routes setup
const authMiddleware = require('./middleware/auth');
const setUserMiddleware = require('./middleware/setUser');

const authRoutes = require('./routes/auth');
const authorRoutes = require('./routes/author');
const readerRoutes = require('./routes/reader');
const settingsRoutes = require('./routes/settings');

app.use(setUserMiddleware); // Ensure setUserMiddleware runs before routes

app.use('/auth', authRoutes); // Public route
app.use('/author', authMiddleware, authorRoutes); // Protected route
app.use('/reader', readerRoutes); // Public route
app.use('/settings', authMiddleware, settingsRoutes); // Protected route

// Route for getting user data
app.get('/api/user', (req, res) => {
    if (req.session.isAuthenticated) {
        res.json({ user: req.session.user });
    } else {
        res.status(401).json({ error: 'Unauthorized' });
    }
    res.setHeader('Cache-Control', 'no-store');
});

// Route for serving the main home page
app.get('/', (req, res) => {
    res.render('mainHome');
});

// Start the server
app.listen(port, () => {
    console.log(`Blogging tool listening at http://localhost:${port}`);
});
// End
```

db_schema.sql

```sql
PRAGMA foreign_keys=ON;

-- Start
CREATE TABLE IF NOT EXISTS users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password TEXT
);

CREATE TABLE IF NOT EXISTS articles (
    article_id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    content TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    published_at DATETIME,
    author_id INTEGER,
    views INTEGER DEFAULT 0,
    FOREIGN KEY (author_id) REFERENCES users(user_id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS comments (
    comment_id INTEGER PRIMARY KEY AUTOINCREMENT,
    article_id INTEGER,
    commenter_name TEXT NOT NULL,
    comment TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (article_id) REFERENCES articles(article_id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS likes (
    like_id INTEGER PRIMARY KEY AUTOINCREMENT,
    article_id INTEGER,
    user_id INTEGER,
    FOREIGN KEY (article_id) REFERENCES articles(article_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    UNIQUE(article_id, user_id)
);

CREATE TABLE IF NOT EXISTS settings (
    setting_id INTEGER PRIMARY KEY AUTOINCREMENT,
    blog_title TEXT NOT NULL,
    author_name TEXT NOT NULL,
    author_id INTEGER NOT NULL UNIQUE,
    FOREIGN KEY (author_id) REFERENCES users(user_id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS views (
    view_id INTEGER PRIMARY KEY AUTOINCREMENT,
    article_id INTEGER,
    user_id INTEGER,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (article_id) REFERENCES articles(article_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);

-- Insert a default user if not exists
INSERT INTO users (user_name, email, password)
SELECT 'Default Author', 'default@example.com', NULL
WHERE NOT EXISTS (SELECT 1 FROM users);

-- Insert initial settings if not exists
INSERT INTO settings (blog_title, author_name, author_id)
SELECT 'My Blog', 'Author Name', user_id
FROM users
WHERE NOT EXISTS (SELECT 1 FROM settings);
-- End
```

.env

```
# Start
SESSION_SECRET=

GOOGLE_CLIENT_ID=
GOOGLE_CLIENT_SECRET=
# End
```

views\settingsPage.ejs

```
<!-- Start -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Settings</title>

  <!-- Add a favicon for the website to avoid 404 error message -->
  <link rel="shortcut icon" href="#">

  <!-- Loading CSS -->
  <link rel="stylesheet" type="text/css" href="/main.css">
</head>
<body>
  <!-- ----------------------------------navbar------------------------------------------ -->
  <nav class="navbar">
    <h1>Settings Page</h1>
    <!-- ----------navbar Links----------- -->
    <ul>
      <li><a href="/author">Back to Author Home</a></li>
      <li><a href="/auth/logout">Logout</a></li>
    </ul>
  </nav>
  <!-- ----------------------------------Blog------------------------------------------ -->
  <div class="blog-section">
    <form action="/settings" method="post">
      <label>Blog Title:</label>
      <input type="text" name="blog_title" value="<%= settings.blog_title %>">
      <label>Author Name:</label>
      <input type="text" name="author_name" value="<%= settings.author_name %>">
      <button type="submit">Save Settings</button>
    </form>
  </div>
</body>
</html>
<!-- End -->
```

views\register.ejs

```html
<!-- Start -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Register</title>

    <!-- Add a favicon for the website to avoid 404 error message -->
    <link rel="shortcut icon" href="#">

    <!-- Loading CSS -->
    <link rel="stylesheet"  type="text/css" href="/main.css" />
</head>
<body>
  <!-- ----------------------------------navbar------------------------------------------ -->
  <nav class="navbar">
    <h1>Register</h1>
  </nav>
  <!-- ----------------------------------register form------------------------------------------ -->
    <div class="register-form">
      <h2>Register and then login</h2>
    <form action="/auth/register" method="post">
        <label>User Name:</label>
        <input type="text" name="user_name" required>
        <label>Email:</label>
        <input type="email" name="email" required>
        <label>Password:</label>
        <input type="password" name="password" required>
        <button type="submit">Register</button>
    </form>
    <!-- ----------Register error----------- -->
    <% if (errors && errors.length > 0) { %>
      <ul id="registerError-text">
        <% errors.forEach(error => { %>
          <li><%= error.msg %></li>
        <% }) %>
      </ul>
    <% } %>
    <!-- ----------Link to back to login---------- -->
    <p>Back to <a href="/auth/login">Login</a></p>
  </div>
</body>
</html>
<!-- End -->
```

views\readerHome.ejs

```
<!-- Start -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title><%= blogTitle %> - Reader Home</title>

    <!-- Add a favicon for the website to avoid 404 error message -->
    <link rel="shortcut icon" href="#">

    <!-- Loading CSS -->
    <link rel="stylesheet" type="text/css" href="/main.css">
</head>
<body>
    <!-- ----------------------------------navbar------------------------------------------ -
->
    <nav class="navbar">
        <h1>Reader Home Page</h1>
        <!-- ----------navbar Links----------- -->
        <ul>
            <li><a href="/">Main Home</a></li>
            <li class="active">Reader Home</li>
        </ul>
    </nav>
    <!-- ----------------------------------Blog------------------------------------------ -->
        <% if (articles.length > 0) { %>
            <% articles.forEach(article => { %>
                <div class="article-item">
                    <h3><a href="/reader/article/<%= article.article_id %>"><%= article.title
%></a></h3>
                    <p class="publication-date">
                        Article by: <%= article.author_name %>
                        Published on: <%= article.published_at %> PKT
                    </p>
                </div>
            <% }) %>
        <% } else { %>
            <p class="no-article-item">No articles to display.</p>
        <% } %>
</body>
</html>
<!-- End -->
```

views\mainHome.ejs

```
<!-- Start -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Main Home</title>

  <!-- Add a favicon for the website to avoid 404 error message -->
  <link rel="shortcut icon" href="#">

  <!-- Loading CSS -->
  <link rel="stylesheet" type="text/css" href="/main.css">
</head>
 <body>
  <!-- ----------------------------------navbar------------------------------------------ -->
  <nav class="navbar">
    <h1>Blogging Tool</h1>
    <!-- ----------navbar Links----------- -->
    <ul>
      <li class="active">Home</li>
      <li><a href="/author">Author Home Page</a></li>
      <li><a href="/reader">Reader Home Page</a></li>
    </ul>
    <!-- ----------------------------------Blog------------------------------------------ -->
    <% if (user && user.user_name !== 'Guest') { %>
      <div class="welcome-message">
        Welcome, <%= user.user_name %>! <a href="/auth/logout">Logout</a>
      </div>
    <% } else { %>
      <div class="welcome-message">
        Welcome, Guest! <a href="/auth/login">Login</a>
      </div>
    <% } %>
  </nav>
  <div class="main-blog-articles">
    <div class="blog-articles">
      <h4>Welcome</h4>
      <p>to the Blogging Tool</p>
    </div>
  </div>
 </body>
</html>
<!-- End -->
```

views\login.ejs

```
<!-- Start -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>

    <!-- Add a favicon for the website to avoid 404 error message -->
    <link rel="shortcut icon" href="#">

    <!-- Loading CSS -->
    <link rel="stylesheet"  type="text/css" href="/main.css" />
</head>
 <body>
  <!-- ---------------------------------navbar----------------------------------------- -->
  <nav class="navbar">
    <h1>Login</h1>
    <!-- ----------navbar Links----------- -->
    <ul>
      <li><a href="/">Main Home</a></li>
      <li class="active">Author Home Page</li>
    </ul>
  </nav>
    <!-- ---------------------------------login form------------------------------------------ -->
    <div class="login-form">
      <form action="/auth/login" method="post">
          <label>Email:</label>
          <input type="email" name="email" required>
          <label>Password:</label>
          <input type="password" name="password" required>
          <button type="submit">Login</button>
      </form>
    <!-- ----------Login error----------- -->
    <% if (errors && errors.length > 0) { %>
      <ul id="loginError-text">
        <% errors.forEach(error => { %>
          <li><%= error.msg %></li>
        <% }) %>
      </ul>
    <% } %>
    <!-- ----------Other Login options----------- -->
   <p>Don't have an account? <a href="/auth/register">Register here</a> </p>
   <p>Or</p>
    <a href="/auth/google">Login with Google</a>
    </div>
 </body>
</html>
<!-- End -->
```

views\editArticle.ejs

```
<!-- Start -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Edit Article</title>

  <!-- Add a favicon for the website to avoid 404 error message -->
  <link rel="shortcut icon" href="#">

  <!-- Loading CSS -->
  <link rel="stylesheet" type="text/css" href="/main.css">
</head>
<body>
  <!-- ----------------------------------navbar------------------------------------------ -->
  <nav class="navbar">
    <h1>Edit Article</h1>
    <!-- ----------navbar Links----------- -->
    <ul>
      <li><a href="/author">Back to Author Home</a></li>
    </ul>
  </nav>
  <!-- ----------------------------------Blog------------------------------------------ -->
  <div class="blog-section">
    <div class="article-container">
      <h5>Edit Article</h5>
      <form class="editArticle-field" action="/author/edit/<%= article.article_id %>"
method="post">
        <label>Title:</label>
        <input type="text" name="title" value="<%= article.title %>" required>
        <label>Content:</label>
        <textarea name="content" required><%= article.content %></textarea>
        <button type="submit">Save Changes</button>
      </form>
    </div>
  </div>
</body>
</html>
<!-- End -->
```

views\authorHome.ejs

```html
<!-- Start -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Author Home</title>

    <!-- Add a favicon for the website to avoid 404 error message -->
    <link rel="shortcut icon" href="#">

    <!-- Loading CSS -->
    <link rel="stylesheet" type="text/css" href="/main.css">
    <!-- Loading Javascript -->
    <script src="/confirm.js" defer></script>
</head>
<body>
    <!-- ----------------------------------navbar------------------------------------------ -->
    <nav class="navbar">
        <h1><%= blogTitle %></h1>
        <!-- ----------navbar Links----------- -->
        <ul>
            <li><a href="/">Main Home</a></li>
            <li class="active">Author Home</li>
            <li><a href="/settings">Settings</a></li>
            <li><a href="/auth/logout">Logout</a></li>
        </ul>
    </nav>
    <!-- ----------------------------------Blog------------------------------------------ -->
    <div class="main-blog-articles">
        <div class="blog-articles">
            <h4>Published Articles</h4>
            <ul>
                <% publishedArticles.forEach(article => { %>
                <li class="published-articles">
                    <a href="/reader/article/<%= article.article_id %>">View <%= article.title %></a> by <%= authorName %>
                    <p>Created at: <%= moment.utc(article.created_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss') %> PKT</p>
                    <p>Last modified at: <%= moment.utc(article.updated_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss') %> PKT</p>
                    <p>Published at: <%= moment.utc(article.published_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss') %> PKT</p>
                    <form id="delete-published-<%= article.article_id %>" action="/author/delete/<%= article.article_id %>" method="post">
                        <button type="button" class="delete-published-button" data-form-id="delete-published-<%= article.article_id %>">Delete</button>
                    </form>
                </li>
                <% }) %>
            </ul>
        </div>
        <div class="blog-articles line-between-publish-and-draft">
            <h4>Draft Articles</h4>
            <ul>
                <% draftArticles.forEach(article => { %>
                <li class="draft-articles">
                    <a href="/author/edit/<%= article.article_id %>">Click to Edit <%= article.title %></a> by <%= authorName %>
                    <p>Created at: <%= moment.utc(article.created_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss') %> PKT</p>
                    <p>Last modified at: <%= moment.utc(article.updated_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss') %> PKT</p>
                    <form id="publish-draft-<%= article.article_id %>" action="/author/publish/<%= article.article_id %>" method="post">
                        <button type="button" class="publish-button" data-form-id="publish-draft-<%= article.article_id %>">Publish</button>
                    </form>
                    <% if (publishMessage) { %>
                    <p class="publish-message"><%= publishMessage %></p>
                    <% }; %>
                    <form id="delete-draft-<%= article.article_id %>" action="/author/delete/<%= article.article_id %>" method="post">
                        <button type="button" class="delete-draft-button" data-form-id="delete-draft-<%= article.article_id %>">Delete</button>
                    </form>
                </li>
                <% }) %>
            </ul>
            <form action="/author/create-draft" method="post">
                <button type="submit">Create new draft</button>
            </form>
        </div>
    </div>

    <!-- Modal for confirmation dialog -->
    <div id="confirmationModal" class="modal">
        <div class="modal-content">
            <p id="confirmationMessage"></p>
            <button id="confirmYes">Yes</button>
            <button id="confirmNo">No</button>
        </div>
    </div>
    <style>
        .modal /* Styles for the modal container */
        {
            display: none; /* Initially hide the modal */
            position: fixed; /* Fixed position so it stays in place regardless of scrolling */
            z-index: 1; /* Ensure modal is on top of other content */
            left: 0;
            top: 0;
            width: 100%; /* Full width of the viewport */
            height: 100%; /* Full height of the viewport */
            overflow: auto; /* Enable scrolling if content exceeds viewport */
            background-color: rgb(0,0,0); /* Fallback color for older browsers */
            background-color: rgba(0,0,0,0.4); /* Transparent black overlay */
        }

        .modal-content /* Styles for the modal content */
        {
            background-color: #fefefe; /* White background for the modal content */
            margin: 20% auto; /* Center the modal vertically & horizontally */
            padding: 30px; /* Padding inside the modal content */
            border: 1px solid #4d388b;
            width: 30%;
            text-align: center;
        }

        #confirmYes, #confirmNo /* Styles for 'Yes' and 'No' buttons */
        {
            margin-top: 10px; /* Add some space above the buttons */
        }
    </style>
</body>
</html>
<!-- End -->
```

views\articlePage.ejs

```
<!-- Start -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title><%= article.title %></title>

  <!-- Add a favicon for the website to avoid 404 error message -->
  <link rel="shortcut icon" href="#">

  <!-- Loading CSS -->
  <link rel="stylesheet" type="text/css" href="/main.css">
</head>
<body>
  <!-- ---------------------------------navbar------------------------------------------ -->
  <nav class="navbar">
    <h1><%= article.title %></h1>
    <!-- ----------navbar Links----------- -->
    <ul>
      <li><a href="/reader">Back to Reader Home</a></li>
      <li><a href="/author">Back to Author Home</a></li>
    </ul>
  </nav>
  <!-- ---------------------------------Blog----------------------------------------- -->
  <div class="blog-section">
    <div class="article-container">
      <h5><%= article.title %></h5>
      <% if (article.showAuthorName) { %>
        <p>By <%= article.author %></p>
      <% } %>
      <p class="p-article"><%= article.content %></p>
      <hr>
      <p class="article-info">Published on: <%=
moment.utc(article.published_at).tz('Asia/Karachi').format('YYYY-MM-DD HH:mm:ss') %> PKT</p>
      <p class="article-info">Views: <%= article.views %> | Likes: <%= likeCount %></p>
      <% if (likeMessage) { %>
        <p class="like-message"><%= likeMessage %></p>
      <% } %>
      <form action="/reader/article/<%= article.article_id %>/like" method="post">
        <button type="submit">Like 👍</button>
      </form>
      <form action="/reader/article/<%= article.article_id %>/comment" method="post">
        <label>Name:</label>
        <input type="text" name="commenter_name" required>
        <label>Comment:</label>
        <textarea name="comment" required></textarea>
        <button type="submit">Add Comment</button>
      </form>
      <ul>
        <% comments.forEach(comment => { %>
          <li>
```

```
            <strong><%= comment.commenter_name %>:</strong> <%= comment.comment %> - <%=
moment.utc(comment.created_at).tz('Asia/Karachi').format('YYYY-MM-DD HH:mm:ss') %> PKT
          </li>
        <% }); %>
      </ul>
    </div>
  </div>
</body>
</html>
<!-- End -->
```

routes\settings.js

```
// Start
const express = require('express');

const router = express.Router();

/**
 * Route serving the settings page.
 * @name get/settings
 * @function
 * @memberof module:routers/settings~settingsRouter
 * @inner
 * @param {string} path - Express path
 */
router.get('/', (req, res) => {
    const query = "SELECT * FROM settings WHERE author_id = ?";
    // Database interaction: Fetching the settings of the current user
    // Inputs: Author ID from the session
    // Outputs: Settings of the current user if found, else null
    global.db.get(query, [req.session.user.user_id], (err, settings) => {
        if (err)
        {
            console.error("Error retrieving settings:", err);
            res.status(500).send("Error retrieving settings");
            return;
        }
        if (!settings)
        {
            settings = { blog_title: 'Default Blog Title', author_name: 'Set your name' };
        }
        res.render('settingsPage', { settings: settings });
    });
});

/**
 * Route for updating the settings.
 * @name post/settings
 * @function
 * @memberof module:routers/settings~settingsRouter
 * @inner
 * @param {string} path - Express path
 */
router.post('/', (req, res) => {
    const blogTitle = req.body.blog_title;
    const authorName = req.body.author_name;
    const authorId = req.session.user.user_id;
    console.log("Received data:", { blogTitle, authorName, authorId });
    if (!blogTitle || !authorName || !authorId)
    {
        console.error("Missing required fields");
        return res.status(400).send("Missing required fields");
    }
    const query = `
        INSERT INTO settings (blog_title, author_name, author_id)
        VALUES (?, ?, ?)
        ON CONFLICT(author_id) DO UPDATE SET
        blog_title = excluded.blog_title,
        author_name = excluded.author_name;`;
    const params = [blogTitle, authorName, authorId];
    // Database interaction: Inserting or updating the settings in the database
    // Inputs: Blog title, author name, and author ID from the request body and session
    // Outputs: None
    global.db.run(query, params, function(err)
    {
        if (err)
        {
            console.error("Error updating settings:", err);
            res.status(500).send("Error updating settings");
            return;
        }
        console.log("Settings updated successfully");
        res.redirect('/author');
    });
});

module.exports = router;
// End
```

routes\reader.js

```
// Start
const express = require('express');
const moment = require('moment-timezone');
const TIMEZONE = 'Asia/Karachi';

const router = express.Router();

/**
 * Route serving the reader's home page.
 * @name get/reader
 * @function
 * @memberof module:routers/reader~readerRouter
 * @inner
 * @param {string} path - Express path
 */
router.get('/', (req, res) => {
    const articlesQuery = `
        SELECT articles.*, IFNULL(settings.author_name, users.user_name) AS author_name
        FROM articles
        LEFT JOIN settings ON articles.author_id = settings.author_id
        LEFT JOIN users ON articles.author_id = users.user_id
        WHERE published_at IS NOT NULL
        ORDER BY published_at DESC`;
    // Database interaction: Fetching all published articles
    // Inputs: None
    // Outputs: All published articles
    global.db.all(articlesQuery, [], (err, articles) => {
        if (err)
        {
            console.error(err);
            return res.status(500).send("Error retrieving articles");
        }
        articles.forEach(article => {
            article.published_at =
moment.utc(article.published_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss');
        });
        res.render('readerHome', {
            blogTitle: 'Blog',
            articles: articles,
            moment: moment
        });
    });
});

/**
 * Route for viewing an article.
 * @name get/reader/article/:id
 * @function
 * @memberof module:routers/reader~readerRouter
 * @inner
 * @param {string} path - Express path
 */
```

```
router.get('/article/:id', (req, res) => {
    const articleQuery = "SELECT articles.*, users.user_name FROM articles JOIN users ON
articles.author_id = users.user_id WHERE article_id = ?";
    const commentsQuery = "SELECT * FROM comments WHERE article_id = ? ORDER BY created_at
DESC";
    const likesQuery = "SELECT COUNT(*) AS like_count FROM likes WHERE article_id = ?";
    const viewsInsertQuery = "INSERT INTO views (article_id, user_id) VALUES (?, ?)";
    const viewsUpdateQuery = "UPDATE articles SET views = views + 1 WHERE article_id = ?";
    // Database interaction: Fetching the article
    // Inputs: Article ID from the URL parameters
    // Outputs: Article data if found, else null
    global.db.get(articleQuery, [req.params.id], (err, article) => {
        if (err)
        {
            console.error(err);
            res.status(500).send("Error retrieving article");
            return;
        }
        if (!req.session.user)
        {
            req.session.user = { user_id: null };
        }
        if (!req.session.viewedArticles)
        {
            req.session.viewedArticles = [];
        }
        if (!req.session.viewedArticles.includes(req.params.id))
        {
            // Database interaction: Inserting a view into the database
            // Inputs: Article ID from the URL parameters and user ID from the session
            // Outputs: None
            global.db.run(viewsInsertQuery, [req.params.id, req.session.user ?
req.session.user.user_id : null], (err) => {
                if (err)
                {
                    console.error(err);
                }
            });
            // Database interaction: Updating the views count of the article in the database
            // Inputs: Article ID from the URL parameters
            // Outputs: None
            global.db.run(viewsUpdateQuery, [req.params.id], (err) => {
                if (err)
                {
                    console.error(err);
                }
            });
            req.session.viewedArticles.push(req.params.id);
        }
        // Database interaction: Fetching the comments of the article
        // Inputs: Article ID from the URL parameters
        // Outputs: Comments of the article if found, else null
        global.db.all(commentsQuery, [req.params.id], (err, comments) => {
            if (err)
            {
                console.error(err);
                res.status(500).send("Error retrieving comments");
                return;
            }
            // Database interaction: Fetching the likes count of the article
            // Inputs: Article ID from the URL parameters
            // Outputs: Likes count of the article
            global.db.get(likesQuery, [req.params.id], (err, likeData) => {
                if (err)
                {
                    console.error(err);
                    res.status(500).send("Error retrieving likes");
                    return;
                }
                article.published_at = 
moment(article.published_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss');
                comments.forEach(comment => {
                    comment.created_at = 
moment(comment.created_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss');
                });
                const likeMessage = req.session.likeMessage;
                req.session.likeMessage = null;
                res.render('articlePage', {
                    article: article,
                    comments: comments,
                    likeCount: likeData.like_count,
                    moment: moment,
                    likeMessage: likeMessage
                });
            });
        });
    });
});

/**
 * Route for liking an article.
 * @name post/reader/article/:id/like
 * @function
 * @memberof module:routers/reader~readerRouter
 * @inner
 * @param {string} path - Express path
 */
router.post('/article/:id/like', (req, res) => {
    const userId = req.session.user ? req.session.user.user_id : null;
    if (!userId)
    {
        req.session.likeMessage = 'Please log in to like articles.';
        return res.redirect(`/reader/article/${req.params.id}`);
    }
    const checkLikeQuery = "SELECT COUNT(*) AS count FROM likes WHERE article_id = ? AND 
user_id = ?";
    const insertLikeQuery = "INSERT INTO likes (article_id, user_id) VALUES (?, ?)";
    // Database interaction: Checking if the user has already liked the article and inserting a like if not
    // Inputs: Article ID from the URL parameters and user ID from the session
    // Outputs: None
    global.db.get(checkLikeQuery, [req.params.id, userId], (err, row) => {
        if (err)
        {
            console.error(err);
            res.status(500).send("Error checking like status");
            return;
        }
        if (row.count > 0)
        {
            return res.redirect(`/reader/article/${req.params.id}`);
        }
        // Database interaction: Inserting a like into the database
        // Inputs: Article ID from the URL parameters and user ID from the session
        // Outputs: None
        global.db.run(insertLikeQuery, [req.params.id, userId], (err) => {
            if (err)
            {
                console.error(err);
                res.status(500).send("Error liking article");
                return;
            }
            res.redirect(`/reader/article/${req.params.id}`);
        });
    });
});

/**
 * Route for commenting on an article.
 * @name post/reader/article/:id/comment
 * @function
 * @memberof module:routers/reader~readerRouter
 * @inner
 * @param {string} path - Express path
 */
router.post('/article/:id/comment', (req, res) => {
    const query = "INSERT INTO comments (article_id, commenter_name, comment) VALUES (?, ?, ?)";
    const params = [req.params.id, req.body.commenter_name, req.body.comment];
    // Database interaction: Inserting a new comment into the database
    // Inputs: Article ID from the URL parameters, commenter name, and comment from the request body
    // Outputs: None
    global.db.run(query, params, (err) => {
        if (err)
        {
            console.error(err);
            res.status(500).send("Error adding comment");
            return;
        }
        res.redirect(`/reader/article/${req.params.id}`);
    });
});

module.exports = router;
// End
```

routes\author.js

```
// Start
const express = require('express');
const moment = require('moment-timezone');
const { body, validationResult } = require('express-validator');
const TIMEZONE = 'Asia/Karachi';

const router = express.Router();

/**
 * Route serving the author's home page.
 * @name get/author
 * @function
 * @memberof module:routers/author~authorRouter
 * @inner
 * @param {string} path - Express path
 */
router.get('/', (req, res) => {
    const queryPublished = `
        SELECT articles.*, users.user_name
        FROM articles
        JOIN users ON articles.author_id = users.user_id
        WHERE published_at IS NOT NULL AND articles.author_id = ?`;
    const queryDrafts = `
        SELECT articles.*, users.user_name
        FROM articles
        JOIN users ON articles.author_id = users.user_id
        WHERE published_at IS NULL AND articles.author_id = ?`;
    const querySettings = "SELECT * FROM settings WHERE author_id = ?";
    // Database interaction: Fetching published articles, drafts & any settings of the current user
    // Inputs: User ID from the session
    // Outputs: Published articles, drafts & any setting of the current user
    global.db.all(queryPublished, [req.session.user.user_id], (err, publishedArticles) => {
        if (err) throw err;
        global.db.all(queryDrafts, [req.session.user.user_id], (err, draftArticles) => {
            if (err) throw err;
            global.db.get(querySettings, [req.session.user.user_id], (err, settings) => {
                if (err) throw err;
                publishedArticles.forEach(article => {
                    article.created_at = moment(article.created_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss');
                    article.updated_at = moment(article.updated_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss');
                    article.published_at = moment(article.published_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss');
                });
                draftArticles.forEach(article => {
                    article.created_at = moment(article.created_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss');
                    article.updated_at = moment(article.updated_at).tz(TIMEZONE).format('YYYY-MM-DD HH:mm:ss');
                });
```

```javascript
                res.render('authorHome', {
                    publishedArticles: publishedArticles,
                    draftArticles: draftArticles,
                    TIMEZONE: TIMEZONE,
                    moment: moment,
                    blogTitle: settings ? settings.blog_title : 'Set your Blog Title in Settings',
                    authorName: settings ? settings.author_name : 'Default Author Name',
                    publishMessage: res.locals.publishMessage
                });
            });
        });
    });
});

/**
 * Route for creating a new draft.
 * @name post/author/create-draft
 * @function
 * @memberof module:routers/author~authorRouter
 * @inner
 * @param {string} path - Express path
 */
router.post('/create-draft', (req, res) => {
    if (!req.session.user || !req.session.user.user_id)
    {
        res.status(401).send("Unauthorized");
        return;
    }
    const query = "INSERT INTO articles (title, content, author_id) VALUES (?, ?, ?)";
    const params = ['New Draft', '', req.session.user.user_id];
    // Database interaction: Inserting a new draft into the database
    // Inputs: Title, content, and author ID
    // Outputs: None
    global.db.run(query, params, function(err)
    {
        if (err)
        {
            console.error(err);
            res.status(500).send("Error creating draft");
            return;
        }
        res.redirect(`/author/edit/${this.lastID}`);
    });
});

/**
 * Route for editing an article.
 * @name get/author/edit/:id
 * @function
 * @memberof module:routers/author~authorRouter
 * @inner
 * @param {string} path - Express path
 */
```

```
router.get('/edit/:id', (req, res) => {
    const query = "SELECT * FROM articles WHERE article_id = ? AND author_id = ?";
    // Database interaction: Fetching the article to be edited
    // Inputs: Article ID from the URL parameters and author ID from the session
    // Outputs: Article data if found, else null
    global.db.get(query, [req.params.id, req.session.user.user_id], (err, article) => {
        if (err)
        {
            console.error(err);
            res.status(500).send("Error retrieving article");
            return;
        }
        res.render('editArticle', { article: article, errors: [] });
    });
});

/**
 * Route for updating an article.
 * @name post/author/edit/:id
 * @function
 * @memberof module:routers/author~authorRouter
 * @inner
 * @param {string} path - Express path
 * @param {function} middleware - Express middleware
 */
router.post('/edit/:id', [
    body('title').notEmpty().withMessage('Title is required'),
    body('content').notEmpty().withMessage('Content is required')
],
    (req, res) => {
        const errors = validationResult(req);
        if (!errors.isEmpty())
        {
            const query = "SELECT * FROM articles WHERE article_id = ? AND author_id = ?";
            // Database interaction: Fetching the article to be updated
            // Inputs: Article ID from the URL parameters and author ID from the session
            // Outputs: Article data if found, else null
            global.db.get(query, [req.params.id, req.session.user.user_id], (err, article)
=> {
                if (err)
                {
                    console.error(err);
                    res.status(500).send("Error retrieving article");
                    return;
                }
                res.render('editArticle', {
                    article: article,
                    errors: errors.array()
                });
            });
            return;
        }
        const query = "UPDATE articles SET title = ?, content = ?, updated_at =
CURRENT_TIMESTAMP WHERE article_id = ? AND author_id = ?";
```

```
        const params = [req.body.title, req.body.content, req.params.id, req.session.user.user_id];
        // Database interaction: Updating the article in the database
        // Inputs: Title, content, article ID from the URL parameters, and author ID from the session
        // Outputs: None
        global.db.run(query, params, (err) => {
            if (err)
            {
                console.error(err);
                res.status(500).send("Error updating article");
                return;
            }
            res.redirect('/author');
        });
    });

/**
 * Route for publishing an article.
 * @name post/author/publish/:id
 * @function
 * @memberof module:routers/author~authorRouter
 * @inner
 * @param {string} path - Express path
 */
router.post('/publish/:id', (req, res) => {
    const query = "SELECT title, content FROM articles WHERE article_id = ? AND author_id = ?";
    // Database interaction: Fetching the title and content of the article to be published
    // Inputs: Article ID from the URL parameters and author ID from the session
    // Outputs: Title and content of the article if found, else null
    global.db.get(query, [req.params.id, req.session.user.user_id], (err, article) => {
        if (err)
        {
            console.error(err);
            res.status(500).send("Error retrieving article");
            return;
        }
        if (!article.title || !article.content) {
            req.session.publishMessage = 'Title and content are required to publish the article.';
            res.redirect('/author');
            return;
        }
        const updateQuery = "UPDATE articles SET published_at = CURRENT_TIMESTAMP WHERE article_id = ? AND author_id = ?";
        // Database interaction: Updating the published_at timestamp of the article in the database
        // Inputs: Article ID from the URL parameters and author ID from the session
        // Outputs: None
        global.db.run(updateQuery, [req.params.id, req.session.user.user_id], (err) => {
            if (err)
            {
                console.error(err);
                res.status(500).send("Error publishing article");
                return;
            }
            res.redirect('/author');
        });
    });
});

/**
 * Route for deleting an article.
 * @name post/author/delete/:id
 * @function
 * @memberof module:routers/author~authorRouter
 * @inner
 * @param {string} path - Express path
 */
router.post('/delete/:id', (req, res) => {
    const query = "DELETE FROM articles WHERE article_id = ? AND author_id = ?";
    // Database interaction: Deleting the article from the database
    // Inputs: Article ID from the URL parameters and author ID from the session
    // Outputs: None
    global.db.run(query, [req.params.id, req.session.user.user_id], (err) => {
        if (err)
        {
            console.error(err);
            res.status(500).send("Error deleting article");
            return;
        }
        res.redirect('/author');
    });
});

module.exports = router;
// End
```

routes\auth.js

```javascript
// Start
const express = require('express');
const bcrypt = require('bcrypt');
const passport = require('passport');
const { body, validationResult } = require('express-validator');
require('dotenv').config();

const router = express.Router();

/**
 * Route serving login form.
 * @name get/auth/login
 * @function
 * @memberof module:routers/auth~authRouter
 * @inner
 * @param {string} path - Express path
 */
router.get('/login', (req, res) => {
    res.render('login', { errors: [] });
});

/**
 * Route processing login form.
 * @name post/auth/login
 * @function
 * @memberof module:routers/auth~authRouter
 * @inner
 * @param {string} path - Express path
 * @param {function} middleware - Express middleware
 */
router.post('/login',
    body('email').isEmail().withMessage('Enter a valid email').normalizeEmail(),
    body('password').notEmpty().withMessage('Password is required').trim().escape(),
    (req, res) => {
        const errors = validationResult(req);
        if (!errors.isEmpty())
        {
            return res.status(400).render('login', { errors: errors.array() });
        }
        const { email, password } = req.body;
        const query = "SELECT * FROM users WHERE email = ?";
        // Database interaction: Fetching user with the provided email
        // Inputs: Email provided by the user
        // Outputs: User data if found, else null
        global.db.get(query, [email], async (err, user) => {
            if (err || !user) {
                return res.status(401).render('login', { errors: [{ msg: 'Invalid
credentials' }] });
            }
            // Compare hashed password for regular users
            const passwordMatch = await bcrypt.compare(password, user.password);
            if (passwordMatch)
            {
                req.session.isAuthenticated = true;
                req.session.user = user;
                return res.redirect('/author');
            }
            else
            {
                return res.status(401).render('login', { errors: [{ msg: 'Invalid
credentials' }] });
            }
        });
    });

/**
 * Route serving register form.
 * @name get/auth/register
 * @function
 * @memberof module:routers/auth~authRouter
 * @inner
 * @param {string} path - Express path
 */
router.get('/register', (req, res) => {
    res.render('register', { errors: [] });
});

/**
 * Route processing register form.
 * @name post/auth/register
 * @function
 * @memberof module:routers/auth~authRouter
 * @inner
 * @param {string} path - Express path
 * @param {function} middleware - Express middleware
 */
router.post('/register',
    body('user_name').notEmpty().withMessage('User name is required').trim().escape(),
    body('email').isEmail().withMessage('Enter a valid email').normalizeEmail(),
    body('password').isLength({ min: 6 }).withMessage('Password must be at least 6
characters long').trim().escape(),
    async (req, res) => {
        const errors = validationResult(req);
        if (!errors.isEmpty())
        {
            return res.status(400).render('register', { errors: errors.array() });
        }
        const { user_name, email, password } = req.body;
        const hashedPassword = await bcrypt.hash(password, 10);
        const query = "INSERT INTO users (user_name, email, password) VALUES (?, ?, ?)";
        // Database interaction: Inserting new user into the database
        // Inputs: User name, email, and hashed password provided by the user
        // Outputs: None
        global.db.run(query, [user_name, email, hashedPassword], function(err)
        {
            if (err)
```

```javascript
            {
                return res.status(500).render('register', { errors: [{ msg: 'Error registering user' }] });
            }
            res.redirect('/auth/login');
        });
    });

const GoogleStrategy = require('passport-google-oauth20').Strategy;
passport.use(new GoogleStrategy({
    clientID: process.env.GOOGLE_CLIENT_ID,
    clientSecret: process.env.GOOGLE_CLIENT_SECRET,
    callbackURL: "http://localhost:3000/auth/google/callback" },
    (accessToken, refreshToken, profile, done) => {
    const email = profile.emails[0].value;
    const query = "SELECT * FROM users WHERE email = ?";
    // Database interaction: Fetching user with the provided email
    // Inputs: Email provided by Google OAuth
    // Outputs: User data if found, else null
    global.db.get(query, [email], (err, user) => {
        if (user)
        {
            return done(null, user);
        }
        else
        {
            const newUser = {
                user_name: profile.displayName,
                email: email,
                password: null
            };
            const insertQuery = "INSERT INTO users (user_name, email, password) VALUES (?, ?, ?)";
            // Database interaction: Inserting new user into the database
            // Inputs: User name and email provided by Google OAuth
            // Outputs: None
            global.db.run(insertQuery, [newUser.user_name, newUser.email, newUser.password], function(err)
            {
                if (err)
                {
                    return done(err, null);
                }
                newUser.user_id = this.lastID;
                return done(null, newUser);
            });
        }
    });
}));

/**
 * Route for Google OAuth.
 * @name get/auth/google
 * @function
 * @memberof module:routers/auth~authRouter
 * @inner
 * @param {string} path - Express path
 * @param {function} middleware - Express middleware
 */
router.get('/google',
    passport.authenticate('google', { scope: ['profile', 'email'] }));

/**
 * Route for Google OAuth callback.
 * @name get/auth/google/callback
 * @function
 * @memberof module:routers/auth~authRouter
 * @inner
 * @param {string} path - Express path
 * @param {function} middleware - Express middleware
 */
router.get('/google/callback',
    passport.authenticate('google', { failureRedirect: '/auth/login' }),
    (req, res) => {
        req.session.isAuthenticated = true;
        req.session.user = req.user;
        res.redirect('/author');
    });

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

passport.deserializeUser((id, done) => {
    const query = "SELECT * FROM users WHERE user_id = ?";
    // Database interaction: Fetching user with the provided user ID
    // Inputs: User ID provided by Passport.js
    // Outputs: User data if found, else null
    global.db.get(query, [id], (err, user) => {
        done(err, user);
    });
});

/**
 * Route for logout.
 * @name get/auth/logout
 * @function
 * @memberof module:routers/auth~authRouter
 * @inner
 * @param {string} path - Express path
 */
router.get('/logout', (req, res) => {
    req.session.destroy((err) => {
        if (err)
        {
            console.log(err);
        }
        else
        {
            res.clearCookie('connect.sid'); // Clear the session cookie
            res.redirect('/auth/login');
        }
    });
});

module.exports = router;
// End
```

public\main.css

```css
// Start
/*
  Note: for styling, I have reused most contents from my Final Assignment of previous module
  named CM1040 Web Development
  Another thing to note is <h1>, <h2> and so on might look disordered in ejs templates & it is
  because I have used them in order in CSS instead
*/

/* ==========================================================================
   - Global (like body, navbar, default settings etc) CSS style for all pages
   ========================================================================== */

body, html
{
  font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman', serif;
  margin: 0; /* Remove default margin */
  padding: 0; /* Remove default padding */
  overflow-x: hidden; /*to prevent any possible horizontal scroll bar*/
  background-color: lightcyan
  position: relative; /*stay Relative to its positon*/
}

/* Making & Adjusting default settings-------------------------------------------------------*/

/* Paragraphs */
p
{
  padding: 0px; /* Remove default padding & margin */
  margin: 0;
}

/* Links */
li
{
  list-style: none; /* Removes default dot on links*/
}

li a:hover /*when mouse is dragged on the link*/
{
  color: red
  text-decoration: underline;
}

a
{
  text-decoration: none; /* Removes default underline on links*/
  color: blue
}

a:hover /*when mouse is dragged on the link*/
```

```css
{
  color: red
  text-decoration: underline;
}

/* Button */
button
{
  font-weight: bolder; /*weight of texts on buttons*/
  background-color: white /*BG color of buttons*/
  color: #3db8c0; /*color of texts on buttons*/
  border: 1px solid #4d388b; /*border size & color*/
  cursor: pointer; /*To make buttons look clickable*/
}

button:hover /*when mouse is dragged on the button*/
{
  background-color: rgb(221, 221, 221); /*BG color of buttons*/
}

button:active /*when mouse is clicked on the button*/
{
  background-color: #ffe7c2; /*BG color of buttons*/
}

/* NAVBAR-------------------------------------------------------------------*/
.navbar
{
  position: fixed; /*sticky navbar*/
  z-index: 9999; /*To make the navbar always appear first*/
  top: 0;
  width: 100%;
  background-color: #3db8c0;
  padding: 30px;
  text-align: center;
  box-shadow: 5px 4px 20px grey; /* shadow effect from below*/
}

h1
{
  margin: 0;
  font-size: 36px;
  color: white
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5); /* shadow effect to the text */
  transition: color 0.3s ease; /* smooth transition effect to the text color on mouse drag*/
}

h1:hover
{
  color: #ff9900; /* Changes the text color when mouse is dragged over h1*/
}

/* NAVBAR links */
nav ul li
```

```css
{
  display: inline; /*Horizontal display of text*/
  margin-right: 20px; /*Space between links*/
}

nav ul li a
{
  color: white /*Non-active links color*/
  font-size: 20px; /*Non-active links size*/
  transition: color 0.3s ease;
  display: inline-block; /* Ensure links are displayed */
}

nav ul li a:hover /*when mouse is dragged over links*/
{
  color: #ff9900; /* Changes the text color of the navbar links from blue */
  text-decoration: none; /*remove default underline decoration*/
  font-weight: bolder; /*Make text bolder*/
}

nav ul li.active /*active links size & weight*/
{
  color: white
  font-size: 25px;
  font-weight: bolder;
}

/* ==========================================================================
   - articlePage.ejs CSS style
   ========================================================================== */

.p-article /*article blog content*/
{
  font-weight: bolder;
  font-size: 20px;
  color: rgb(43, 0, 160);
  line-height: 1.4;
}

.article-info /*when was article published, how many likes & views it have*/
{
  font-size: 20px;
  font-weight: bolder;
  color: #000000;
  line-height: 1.5;
}

/* ==========================================================================
   - authorHome.ejs CSS style
   ========================================================================== */

.published-articles, .draft-articles /* 2 contents (published, draft)*/
{
  border: 1px solid #4d388b; /*border size & color*/
  border-radius: 10px; /*rounded corners of border*/
  margin: 3% 5% auto; /*3% sets the top margin, 5% sets the right & left margins, & auto
sets the bottom margin*/
  padding: 10px; /* Optional: Add some spacing between articles */
}

.line-between-publish-and-draft /*vertical line separating 2 articles content*/
{
  border-left: 1px solid #007e92;
}

/*Display a message if publish button is clicked with empty title and/or content*/
.blog-articles .publish-message
{
  font-weight: normal;
  color: ■ #000;
  margin-right: 25%;
  font-size: 15px;
}

/* ==========================================================================
   - editArticle.ejs CSS style
   ========================================================================== */

.editArticle-field /*text fields & save button*/
{
  position: relative;
  width: 10%;
  display: grid;
  grid-gap: 5px; /*Vertical gap*/
  text-align: left;
  color: ■ #000;
}

/* ==========================================================================
   - login.ejs CSS style
   - register.ejs CSS style
   ========================================================================== */

.login-form
{
  position: relative;
  margin: 13% 30% 1% 41%; /* To adjust it in center */
  width: 10%;
  display: grid;
  grid-gap: 10px; /*Vertical gap*/
  text-align: center;
  padding: 70px;
  color: ■ #000;
  border: 1px solid #4d388b; /*border size & color*/
  border-radius: 8px; /*rounded corners*/
}

.login-form button, .register-form button
```

```css
{
  font-weight: bolder;
  background-color: #3db8c0;
  color: white
  border: none;
  cursor: pointer;
  width: 80%; /*width size*/
  font-size: 100%;
  padding: 10px 10px; /*overall size*/
  margin: 8px 0; /*a top & bottom margin of 8px, a left & right margin of 0 */
  border: none; /*remove default border*/
  border-radius: 10px; /*corner roundness*/
}

/* Styles for the submit button when mouse is dragged over it*/
.login-form button:hover, .register-form button:hover
{
  background-color: rgb(1, 98, 113); /*buttons BG color*/
}

/* Styles for the submit button when clicked */
.login-form button:active, .register-form button:active
{
  background-color: #ff9900; /*buttons BG color*/
}

h2 /*register title*/
{
  color: #007e92;
  font-size: 20px;
  margin-top: 0;
}

.register-form
{
  position: relative;
  margin: 11% 30% 1% 43%; /* To adjust it in center */
  width: 10%;
  display: grid;
  grid-gap: 10px; /*Vertical gap*/
  text-align: center;
  padding: 50px;
  color: #000;
  border: 1px solid #4d388b; /*border size & color*/
  border-radius: 8px; /*rounded corners*/
}

#loginError-text, #registerError-text
{
  width: 80%; /*border width*/
  margin: 20px auto 0; /*a top margin of 20px, horizontally center it & has a bottom margin of 0 */
  font-weight: bolder; /*text weight & color*/
  color: red
  border: 1px solid #4d388b; /*border size & color*/
  border-radius: 10px; /*rounded corners of border*/
  padding: 10px 10px; /*overall space added for text*/
}

/* ==========================================================================
   - readerHome.ejs CSS style
   ========================================================================== */

h3 /*titles of every published articles in article-item*/
{
  font-size: 20px;
  margin-bottom: 10px;
}

.article-item /*Display list of articles if published*/
{
  display: inline-grid;
  grid-template-rows: 1fr;
  grid-auto-flow: column;
  grid-gap: 10px;
  position: relative;
  margin: 13% auto auto 0.3%; /*13% sets the top margin, auto sets the right & bottom margins & 0.3% sets the left margin.*/
  border: 1px solid #4d388b; /*border size & color*/
  border-radius: 10px; /*rounded corners of border*/
  padding: 5px;
}

.no-article-item /*otherwise display a message saying No articles to display */
{
  position: relative;
  top: 100%; /* position 100% below its original position */
  margin: 13% auto auto 3%; /*13% sets the top margin, auto sets the right & bottom margins, & 3% sets the left margin*/
}

/* ==========================================================================
   - authorHome.ejs CSS style
   - mainHome.ejs CSS style
   ========================================================================== */

/*display welcome message in mainHome & article (published, draft) contents in authorHome in columns*/
.main-blog-articles
{
  display: grid;
  grid-auto-flow: column;
  margin-top: 200px;
}

/*Overall space for welcome message in mainHome & article (published, draft) contents in authorHome */
.blog-articles
```

```css
{
  padding: 8px 32px 8px 32px; /*to adjust it in center*/

}

h4 /*title for welcome message in mainHome & article (published, draft) in authorHome */
{
  font-size: 32px;
  text-align: center;
  font-weight: bolder;
  color: rgb(127, 85, 243);
  margin: 0px;
}

.blog-articles p /*p below welcome message in mainHome &, in publish & draft article contents in authorHome*/
{
  font-weight: bolder;
  font-size: 20px;
  color: rgb(127, 85, 243);
  margin: 0;
  text-align: center;
}

/* ==========================================================================
  - articlePage.ejs CSS style
  - editArticle.ejs CSS style
  ========================================================================== */

.article-container /*1 container in articlePage & other in editArticle*/
{
  margin-bottom: 40px;
  padding: 20px;
  border: 1px solid #4d388b;
  border-radius: 8px;
  width: auto;
}

h5 /* 'Edit Article' title displayed in editArticle & display titles set by authors in articlePage */
{
  color: ☐ white
  font-size: 30px;
  font-weight: bolder;
  text-align: center;
  background-color: rgb(0,126,146);
  padding: 5px;
  margin: -20px -20px 15px;
  border-radius: 8px 8px 0 0;
}

/* ==========================================================================
   - articlePage.ejs CSS style
   - editArticle.ejs CSS style
   - settingPage.ejs CSS style
   ========================================================================== */

.blog-section /*main container for blog article read, edit & settings*/
{
  position: relative;
  margin: 15% 12% 0 12%;
}
// End
```

public\confirm.js

```
// Start
/*
    Content-Security-Policy: The page’s settings blocked an event handler (script-src-attr) from being executed because it violates the following directive: “script-src-attr 'none'”
    Source: confirmAction(event, 'Are you sure you w…

    To avoid the above error messages, I will implement event listeners using JavaScript instead of
    using inline event handlers (such as onclick in the HTML)
*/

// Wait for the DOM content to be fully loaded before executing the script
document.addEventListener('DOMContentLoaded', function ()
{
    // Select all elements with class 'publish-button' and store them in publishButtons array
    const publishButtons = document.querySelectorAll('.publish-button');

    // Select all elements with class 'delete-draft-button' and store them in deleteDraftButtons array
    const deleteDraftButtons = document.querySelectorAll('.delete-draft-button');

    // Select all elements with class 'delete-published-button' and store them in deletePublishedButtons array
    const deletePublishedButtons = document.querySelectorAll('.delete-published-button');

    // Get the confirmation modal element by its ID
    const confirmationModal = document.getElementById('confirmationModal');

    // Get the message element inside the confirmation modal
    const confirmationMessage = document.getElementById('confirmationMessage');

    // Get the 'Yes' button inside the confirmation modal
    const confirmYes = document.getElementById('confirmYes');

    // Get the 'No' button inside the confirmation modal
    const confirmNo = document.getElementById('confirmNo');

    // Function to show the confirmation modal with a specific message and a callback function
    function showConfirmationModal(message, callback)
    {
        // Set the text content of the confirmation message
        confirmationMessage.textContent = message;

        // Display the confirmation modal (make it visible)
        confirmationModal.style.display = 'block';

        // Event listener for the 'Yes' button click
        confirmYes.onclick = function ()
        {
            // Invoke the callback with true when 'Yes' is clicked
```

```javascript
            callback(true);
            // Hide the confirmation modal
            confirmationModal.style.display = 'none';
        };

        // Event listener for the 'No' button click
        confirmNo.onclick = function ()
        {
            // Invoke the callback with false when 'No' is clicked
            callback(false);
            // Hide the confirmation modal
            confirmationModal.style.display = 'none';
        };
    }

    // Loop through each publish button in the publishButtons array
    publishButtons.forEach(button => {
        // Add click event listener to each publish button
        button.addEventListener('click', function (event)
        {
            // Prevent default form submission behavior
            event.preventDefault();

            // Get the form ID from the dataset attribute of the clicked button
            const formId = event.target.dataset.formId;

            // Display confirmation modal with message and define callback function
            showConfirmationModal('Are you sure you want to publish this draft?', function (confirmed)
            {
                // If user confirms (callback returns true), submit the form with the corresponding ID
                if (confirmed)
                {
                    document.getElementById(formId).submit();
                }
            });
        });
    });

    // Similar logic for delete draft buttons
    deleteDraftButtons.forEach(button => {
        button.addEventListener('click', function (event)
        {
            event.preventDefault();
            const formId = event.target.dataset.formId;
            showConfirmationModal('Are you sure you want to delete this draft?', function (confirmed)
            {
                if (confirmed)
                {
                    document.getElementById(formId).submit();
                }
            });
        });
    });

    // Similar logic for delete published buttons
    deletePublishedButtons.forEach(button => {
        button.addEventListener('click', function (event)
        {
            event.preventDefault();
            const formId = event.target.dataset.formId;
            showConfirmationModal('Are you sure you want to delete this article?', function
(confirmed)
            {
                if (confirmed)
                {
                    document.getElementById(formId).submit();
                }
            });
        });
    });
});
// End
```

middleware\setUser.js

```
// Start
/**
 * Middleware to set the user object in the response locals.
 * If the user is authenticated, it sets the user object in the response locals.
 * If the user is not authenticated, it sets the user object in the response locals to null.
 * If there is a publishMessage in the session, it sets the publishMessage in the response locals and clears it from the session.
 *
 * @function
 * @param {Object} req - Express request object
 * @param {Object} res - Express response object
 * @param {function} next - Express next middleware function
 * @property {boolean} req.session.isAuthenticated - Indicates if the user is authenticated
 * @property {Object} req.session.user - The authenticated user's session data
 * @property {string} req.session.publishMessage - The message to be published
 * @property {Object} res.locals.user - The authenticated user's local data
 * @property {string} res.locals.publishMessage - The message to be published
 */
module.exports = function(req, res, next)
{
    if (req.session.isAuthenticated)
    {
        // Purpose: To set the user's session data in the response locals
        // Inputs: User's session data
        // Outputs: User's local data
        res.locals.user = req.session.user || { user_name: 'Guest' };
    }
    else
    {
        // Purpose: To clear the user's local data if not authenticated
        // Inputs: None
        // Outputs: User's local data set to null
        res.locals.user = null;
    }
    if (req.session.publishMessage) {
        // Purpose: To set the publish message in the response locals and clear it from the session
        // Inputs: Publish message from the session
        // Outputs: Publish message in the response locals
        res.locals.publishMessage = req.session.publishMessage;
        req.session.publishMessage = null;
    }
    next();
};
// End
```

middleware\auth.js

```
// Start
/**
 * Middleware to check if the user is authenticated.
 * If the user is authenticated, it sets the user object in the response locals and calls the next middleware.
 * If the user is not authenticated, it redirects the user to the login page.
 *
 * @function
 * @param {Object} req - Express request object
 * @param {Object} res - Express response object
 * @param {function} next - Express next middleware function
 * @property {boolean} req.session.isAuthenticated - Indicates if the user is authenticated
 * @property {Object} req.session.user - The authenticated user's session data
 * @property {Object} res.locals.user - The authenticated user's local data
 */
module.exports = function(req, res, next)
{
    if (req.session.isAuthenticated)
    {
        // Purpose: To set the user's session data in the response locals
        // Inputs: User's session data
        // Outputs: User's local data
        res.locals.user = req.session.user;
        next();
    }
    else
    {
        // Purpose: To redirect unauthenticated users to the login page
        // Inputs: None
        // Outputs: Redirection to '/auth/login'
        res.redirect('/auth/login');
    }
};
// End
```