```
================================================================================
                                INDEX.JS START
================================================================================
=
/**
   * index.js
   * This is your main app entry point
   */

   // Set up express, bodyparser and EJS
const express = require('express');
const session = require("express-session");
const app = express();
const port = 3000;
var bodyParser = require("body-parser");

app.use(bodyParser.urlencoded({ extended: true }));
app.set('view engine', 'ejs'); // set the app to use ejs for rendering
app.use(express.static(__dirname + '/public')); // set location of static files
********************************************************************************************************
                                MY CODE START
********************************************************************************************************
app.use(session({
   secret: "]c7)R5%!*bzZ|#jVNEoJ~3!yB|.r~~cDjP6@<ljUZ3rjnvRlg2!5'Qas,N5!yCJ",
   resave: true,
   saveUninitialized: true,
   cookie: {
      // maxAge is milloseconds, so here we do 60 seconds (60 * 1000)
      // and then we multiply by 60 again so the cookie lasts 1 hour
      maxAge: 60 * 60 * 1000,
   }
}));
********************************************************************************************************
                                MY CODE END
********************************************************************************************************

// Set up SQLite
// Items in the global namespace are accessible throughout the node application
const sqlite3 = require('sqlite3').verbose();
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
    }
});

// Handle requests to the home page
app.get('/', (req, res) => {
    res.render("home.ejs");
});

// Add all the route handlers in usersRoutes to the app under the path /users
const usersRoutes = require('./routes/users');
const authorRoutes = require('./routes/author');
const readerRoutes = require('./routes/reader');
app.use('/users', usersRoutes);
app.use('/author', authorRoutes);
app.use('/reader', readerRoutes);


// Make the web application listen for HTTP requests
app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

```
================================================================================
                                    INDEX.JS END
================================================================================

================================================================================
                                  ROUTES/USERS.JS
================================================================================
```

```
/**
 * users.js
 * Used for managing user login, banning, deleting etc, anything to do with
    * users
 *
 * NB. it's better NOT to use arrow functions for callbacks with the SQLite library
*
 */

const express = require("express");
```

```
**********************************************************************************************************
**********************************************************************************************************
**********************************************************************************************************
**********************************************************************************************************
                                            MY CODE START
**********************************************************************************************************
**********************************************************************************************************
**********************************************************************************************************
**********************************************************************************************************
const router = express.Router();
const bcrypt = require("bcrypt");


/**
 * @desc Everything to do with adding, removing or editing users is here.
    * These are all post requests as we have no user page
 */
//
router.get("/login", (req, res) => {
    res.render("login.ejs");
});

router.get("/register", (req, res) => {
    res.render("register.ejs");
});


/**
 * @desc clears the users session, logging them out
    *
 */
//
router.post("/logout", (req, res, next) => {
    req.session.destroy((err) => {
        if (err) {
            console.error(err);

        } else {
            res.redirect("/users/login");
        }
    });
});
/**
 * @desc Logs the user in if they provide the correct username and password
```

```
    * combination
 */
const genericLoginError = "Invalid username or password";
router.post("/login", (req, res, next) => {
    const username = req.body?.username;
    const password = req.body?.password;
    if (!username || !password) {
        return res.status(401).send(genericLoginError);
    }
    const query_parameters = [username];
    let query = "SELECT * FROM users WHERE username = ?;";
    global.db.get(query, query_parameters, function (err, user) {
        if (err) {
            next(err); //send the error on to the error handler

        } else {
            if (!user) {
                return res.status(401).send(genericLoginError)
            }
            const hash = user.password;
            bcrypt.compare(password, hash, function(err, result) {
                if (result) {
                    // give them a cookie to allow them to access each page
                    req.session.user = {
                        user_id: user.user_id,
                        username: user.username
                    };
                    res.redirect("/author");
                } else {
                    return res.status(401).send(genericLoginError);
                }
            });
        }
    });
});

/**
 * @desc Add a new user to the database based on data from the submitted form
 */
router.post("/add-user", (req, res, next) => {
    const blogTitle = req.body?.blog_title;
    const authorName = req.body?.author_name;
    const username = req.body?.username;
    const password = req.body?.password;
```

```
    const confirmPassword = req.body?.confirm_password;

    // ensure all fields are filled in
    if (!username || !password || !authorName || !blogTitle || !confirmPassword) {
        return res.status(401).send("Please fill out all fields");
    };

    if (password !== confirmPassword) {
        return res.status(401).send("Your passwords do not match, please try again.");
    }

    const saltRounds = 10;
    bcrypt.hash(password, saltRounds, function(err, hash) {
        let query = "SELECT * FROM USERS WHERE username = ?;"
        query_parameters = [username];
        // check to ensure that the username isn't taken;
        global.db.all(query, query_parameters,
            function (err, rows) {
                if (err) next(err); //send the error on to the error handler
                else if(rows.length != 0) return res.status(401).send("Username has already been
taken");
            });

        query = "INSERT INTO users (username, password, author_name) VALUES (?, ?, ?);";
        query_parameters = [username, hash, authorName];

        // Execute the query and send a confirmation message
        global.db.run(query, query_parameters, function (err) {
            if (err) next(err); //send the error on to the error handler
            else {
                // add the users blog settings
                query = "INSERT INTO blogs (user_id, blog_title, author_name) VALUES (?, ?, ?);";
                query_parameters = [this.lastID, blogTitle, authorName];
                // Execute the query and send a confirmation message
                global.db.run(query, query_parameters,
                function (err) {
                    if (err) next(err);
                });
                next();
                res.redirect("/users/login");
            }
        });
    })
});
```

```
// Export the router object so index.js can access it
module.exports = router;
```

==========

USERS.JS END

==========

==========

ROUTES/AUTHOR.JS START

==========

```
/**
 * author.js
 * Used for managing authors adding, removing, publishing, editing, articles *
*
 */

const express = require("express");
const router = express.Router();

// redirect the user to the login page if they aren't authorized
function unauthorized(req, res) {
    if (!req.session?.user) {
        // redirect them to the login page if they aren't authenticated
        res.redirect("/users/login");
        return true;  // and ensure that we return true, since this user
                    // is unauthorized
    }
    return false;
}

/**
    * @Display this authors home page with their articles
    */
router.get("/", (req, res, next) => {
    if (unauthorized(req, res)) return;
    let query = "SELECT * FROM articles WHERE user_id = ? ORDER BY last_modified DESC;";
    const userId = req.session.user.user_id;
    // ensure this user is authorized to come here
    // execute sql query
    global.db.all(query, [userId], function(err, articles) {
        if (err) {
            next(err);
```

```
        } else {
            query = "SELECT * FROM blogs WHERE user_id = ?;";
            global.db.get(query, [userId], function(err, blog) {
                if (err) {
                    next(err);
                }
                res.render("author-home", {
                    articles: articles,
                    blog: blog,
                    user: req.session.user,
                });
            });
        }
    })
});


/**
 * @The author settings page where the author can change the blogs title and author
* name and other settings
* */
router.get("/settings", (req, res, next) => {
    // ensure this user is authorized to come here
    if (unauthorized(req, res)) return;
    const userId = req.session.user.user_id;
    const query = "SELECT * FROM blogs WHERE user_id = ?;";
    global.db.get(query, [userId], function(err, blog) {
        if (err) {
            next(err);
        } else {
            res.render("author-settings.ejs", { blog: blog });
        }
    })
});

/**
 * @The author settings page where the author can change the blog title and author
* name and other settings. Posting to this actually changes the settings
* */
router.post("/settings", (req, res, next) => {
    // ensure this user is authorized to come here
    if (unauthorized(req, res)) return;

    const query = "UPDATE blogs SET blog_title = ?, author_name = ? WHERE user_id = ?"
```

```
    const blogTitle = req.body.blog_title;
    const authorName = req.body.author_name;
    const queryParams = [blogTitle, authorName, req.session.user.user_id];

    global.db.run(query, queryParams, function(err) {
        if (err) {
            next(err);
        } else {
            // after we save the settings we redirect the user to the author
            // home
            res.redirect("/author");
        }
    });
});

/**
 * @The author edit page where the author can amend and publish individual
    * articles
* */
router.get("/edit/:articleId", (req, res, next) => {
    // ensure this user is authorized to come here
    if (unauthorized(req, res)) return;
    const articleId = req.params.articleId;
    if (isNaN(parseInt(articleId))) {
        res.redirect("/author");
    }
    global.db.get(`SELECT * FROM articles WHERE article_id = ${articleId}`, function(err, row) {
        if (err) {
            next(err);
        } else if (row == undefined) {
            // REDIRECT TO SOME SORT OF ERROR PAGE HERE
            // REDIRECT TO SOME SORT OF ERROR PAGE HERE
            res.redirect("/author");
            next();

        } else {
            res.render("author-edit.ejs", { article: row });
            next();
        }
    });
});

/**
 * @ When first creating a new draft, we create a empty article post
```

```
   *
* */
router.post("/add-draft", (req, res, next) => {
    // ensure this user is authorized to come here
    if (unauthorized(req, res)) return;
    global.db.get("SELECT datetime('now', 'localtime') AS currDateTime", function(err, time) {
        if (err) {
            next(err); //send the error on to the error handler
        } else {
            global.db.get("SELECT author_name FROM users WHERE user_id = ?",
                [req.session.user.user_id],
                function(err, userData) {

                const query = "INSERT INTO articles \
                (user_id, author_name, created, last_modified, title, body, published) \
                VALUES (?, ?, ?, ?, ?, ?, ?)";

                const user_id = req.session.user.user_id
                const author_name = userData.author_name;
                const created = time.currDateTime;
                const lastModified = time.currDateTime;
                const articleTitle = req.body.article_title;
                const articleBody = req.body.article_body;
                const published = "";

                const query_parameters = [
                    user_id,
                    author_name,
                    created,
                    lastModified,
                    articleTitle,
                    articleBody,
                    published
                ];
                global.db.run(query, query_parameters,
                function (err) {
                    if (err) {
                        next(err); //send the error on to the error handler
                    } else {
                        res.redirect(`/author/edit/${this.lastID}`);
                        next();
                    }
                });
            });
```

```
        }
    })
})

/**
 * @ Deletes a post at the given article id
    *
* */

router.post("/delete/:articleId", (req, res, next) => {
    // ensure this user is authorized to come here
    if (unauthorized(req, res)) return;
    const articleId = req.params.articleId;
    if (isNaN(parseInt(articleId))) {
        res.redirect("/author");
    }
    global.db.run("DELETE FROM articles WHERE article_id = ?", [articleId], function(err) {
        if (err) {
            next(err);
        } else {
            res.redirect("/author");
        }
    });
});

/**
 * @ Edits the contents of a given articleID
    *
* */
router.post("/edit/:articleId", (req, res, next) => {
    // ensure this user is authorized to come here
    if (unauthorized(req, res)) return;
    const articleId = req.params.articleId;
    //ensure the article id is a valid number
    if (isNaN(parseInt(articleId))) {
        res.redirect("/author");
    }
    global.db.get("SELECT datetime('now', 'localtime') AS currDateTime", function(err, time) {
        global.db.get(`SELECT * FROM articles WHERE article_id = ?`,
            [articleId],
            function(err, row) {
            if (err) {
                next(err);
            } else if (row == undefined) {
```

```
            res.redirect("/author");
            next();

        } else {
            const query = "UPDATE articles SET \
            last_modified = ?,\
            title = ?,\
            body = ?,\
            published = ?\
            WHERE article_id = ?"

            const lastModified = time.currDateTime;
            const articleTitle = req.body.article_title;
            const articleBody = req.body.article_body;
            const published = "";

            const query_parameters = [
                lastModified,
                articleTitle,
                articleBody,
                published,
                articleId];

            global.db.run(query, query_parameters, function (err) {
                if (err) {
                    next(err); //send the error on to the error handler
                } else {
                    res.redirect("/author"); next();
                }
            });
        }
    });
  });
});

/**
 * @desc when the user publishes an article we call this function. It changes
   * the article from draft to published, giving it a publish date. Once
   * it is published it cannot be unpublished, or edited only deleted
   *
 */
router.post("/publish/:articleId", (req, res, next) => {
   // ensure the user is authorized to come here
   if (unauthorized(req, res)) return;
```

```
    const articleId = req.params.articleId;
    //ensure the article id is a valid number
    if (isNaN(parseInt(articleId))) {
        res.redirect("/author");
    }
    global.db.get("SELECT datetime('now', 'localtime') AS currDateTime", function(err, time) {
        global.db.get(`SELECT * FROM articles WHERE article_id = ?`,
        [articleId],
        function(err, row) {
            if (err) {
                next(err);
            } else if (row == undefined) {
                res.redirect("/author");
                next();

            } else {
                const currTime = time.currDateTime;
                global.db.run( `UPDATE articles SET published = ? WHERE article_id = ?`,
                    [currTime, articleId],
                    function (err) {
                    if (err) {
                        next(err); //send the error on to the error handler
                    } else {
                        res.redirect("/author"); next();
                    }
                });
            }
        });
    });
});
// Export the router object so index.js can access it
module.exports = router;


================================================================================
                                ROUTES/AUTHOR.JS END
================================================================================
================================================================================
                                ROUTES/READER.JS START
================================================================================

/**
 * author.js
 * Used for managing reader comments, likes, views, and all the reader pages
```

```
*
 */

const express = require("express");
const router = express.Router();

/**
 * @desc The home page for the reader, it shows all the different blogs
    * from all the different authors of the website
*/
router.get("/", (req, res, next) => {
    const query = "SELECT * FROM blogs;"
    global.db.all(query, function(err, blogs) {
        if (err) {
            next(err);
        } else {
            res.render("reader-home", { blogs: blogs });
        }
    });
});

/**
    * @desc Shows all articles for the given blog owner. We use the blog
    * owners user id to keep track of the owners blog
*/
router.get("/blog/:blog_owner_user_id", (req, res, next) => {
    const blogOwnerUserId = req.params.blog_owner_user_id;
    // select all the articles that the blog owner has published and we sort
    // them so that the most recently published article is on the top
    let query = "SELECT * FROM blogs WHERE user_id = ?;";
    global.db.get(query, [blogOwnerUserId], function(err, blog) {
        if (err) {
            next(err);
        }
        query = "SELECT * FROM articles WHERE user_id = ? AND published <> " ORDER BY
published DESC";
        // execute sql query
        global.db.all(query, [blogOwnerUserId], function(err, articles) {
            if (err) {
                // if we fail to load the articles we just keep the user on the
                // home page
                res.redirect("/reader-home");
            } else {
                res.render("reader-blog.ejs", {
```

```
                articles: articles,
                blog: blog
            });
        }
    })
  })
});

/**
   * @desc Shows the given article that has a matching ":articleId"
   * it allows commenting, liking, and viewing of the blog
*/
router.get("/article/:articleId", (req, res, next) => {
   const articleId = req.params.articleId;
   const users_IP = req.ip;
   let userLikedPost = false;
   // check whether or not this user has liked the post
   let query = "SELECT ip_address FROM article_likes WHERE article_id = ?";
   global.db.all(query, [articleId], function(err, article_likes) {
      if (err) {
         next(err);
      } else {
         for (const like of article_likes) {
            if (like.ip_address == users_IP) {
               userLikedPost = true;
               break;
            }
         }
         query = "SELECT * FROM articles WHERE article_id = ?";
         // send the user the article information for the given link
         global.db.get(query, [articleId], function(err, article) {
            if (err) {
               next(err);
            }
            query = "SELECT * FROM article_comments WHERE article_id = ? ORDER BY
created DESC;";
            global.db.all(query, [articleId], function (err, comments) {
               if (err) {
                  next(err);
               } else {
                  // add one to the number of views for this article if their ip has not
                  // visited this page before
                  query = "SELECT ip_address FROM article_views WHERE article_id = ?;";
                  global.db.all(query, [articleId], function (err, article_views) {
```

```
                    if (err) {
                        next(err); //send the error on to the error handler
                    }
                    res.render("reader-article", {
                        article: article,
                        userLikedPost: userLikedPost,
                        comments: comments,
                    });

                    // if their ip address is here then we return to not allow another
                    // view to be counted. One view per ip address
                    for (const view of article_views) {
                        if (view.ip_address == users_IP) {
                            next();
                            return;
                        }
                    }
                    // if we reach here hten their ip address has not viewed the post, so
                    // we increase the views by one
                    query = "UPDATE articles SET number_of_views = number_of_views + 1
WHERE article_id = ?;";
                    global.db.run(query, [articleId], function (err) {
                        if (err) {
                            next(err); //send the error on to the error handler
                        }

                        // add their ip to the article views database
                        query = "INSERT INTO article_views (article_id, ip_address) VALUES (?,
?);";
                        global.db.run(query, [articleId, users_IP], function (err) {
                            if (err) {
                                next(err); //send the error on to the error handler
                            }
                        });
                    });
                });
            }
        });
    });
}
});

/*
```

```
* @desc Allows the user to like the article specified
*/
router.post("/article/like-article/:articleId", (req, res, next) => {
    const articleId = req.params.articleId;
    const users_IP = req.ip;
    // add one to the number of views for this article
    let query = "SELECT ip_address FROM article_likes WHERE article_id = ?";
    global.db.all(query, [articleId], function (err, article_likes) {
        if (err) {
            next(err); //send the error on to the error handler
        } else {
            query = "UPDATE articles SET number_of_likes = number_of_likes + 1 WHERE
article_id = ?;";
            global.db.run(query, [articleId], function (err) {
                if (err) {
                    next(err); /*send the error on to the error handler*/
                }
            });

            query = "INSERT INTO article_likes (article_id, ip_address) VALUES (?, ?);";
            global.db.run(query, [articleId, users_IP], function(err) {
                if (err) {
                    next(err); //send the error on to the error handler
                } else {
                    res.redirect("/reader/article/" + articleId);
                }
            });
        }
    });
});

/*
* @desc Allows the user to unlike the article specified
*/
router.post("/article/unlike-article/:articleId", (req, res, next) => {
    const articleId = req.params.articleId;
    const user_IP = req.ip;
    let query = "UPDATE articles SET number_of_likes = number_of_likes - 1 WHERE article_id
= ?;";
    // first we remove one like from the page since the user unliked
    global.db.run(query, [articleId], function (err) {
        if (err) {
            next(err); //send the error on to the error handler
        }
```

```
    });

    // then we remove the like from the article_likes table so that their ip
    // address isn't liking the post
    query = "DELETE FROM article_likes WHERE article_id = ? AND ip_address = ?";
    global.db.run(query, [articleId, user_IP], function(err) {
        if (err) {
            next(err);
        } else {
            res.redirect("/reader/article/" + articleId);
        }
    });
});

/*
    * @desc likes the article for the given articleId
*/
router.post("/article/submit-comment/:articleId" , (req, res, next) => {
    const articleId = req.params.articleId
    const commenter_name = req.body.commenter_name;
    const comment = req.body.comment;
    global.db.get("SELECT datetime('now', 'localtime') AS currDateTime", function(err, time) {
        const created = time.currDateTime;
        const query = "INSERT INTO article_comments (created, commenter_name, comment,
article_id) VALUES (?, ?, ?, ?);";
        const query_params = [created, commenter_name, comment, articleId];
        global.db.run(query, query_params, function(err) {
            if (err) {
                next(err);
            } else {
                res.redirect("/reader/article/" + articleId);
            }
        });
    });
});

// Export the router object so index.js can access it
module.exports = router;
```

==========

ROUTES/READER.JS END

==========

==========
VIEWS/AUTHOR-EDIT.EJS START
==========

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"  type="text/css" href="/main.css" />
   <title>Author edit</title>
</head>
<body>
   <h1>Author edit page</h1>
   <a href="/author"><button>Home</button></a>
   <form action="/author/edit/<%=article.article_id%>" method="post">
      <h3 id="articleCreated">Created: <%=article.created%></h3>
      <h3 id="articleLastModified">Last modified:<%=article.last_modified%></h3>
      <input id="article_title" type="text" name="article_title" placeholder="Article title"
value="<%=article.title%>"></input>
      <br>
      <textarea id="article_body" rows="15" cols="50" name="article_body" placeholder="Article
body..."><%=article.body%></textarea>
      <br>
      <button id="<%=article.article_id%>" type="submit">Submit</button>
   </form>
</body>
</html>
```

==========
VIEWS/AUTHOR-EDIT.EJS END
==========
==========
VIEWS/AUTHOR-HOME.EJS START
==========

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"  type="text/css" href="/main.css" />
```

```
    <title>Author Home Page</title>
</head>
<body>
    <a href="/"><button>Front page</button></a>
    <form action="/users/logout" method="post">
        <button>Logout</button>
    </form>
    <h1>Authors Home page</h1>
    <h2>Hello <%=blog.author_name%>!</h2>
    <h2>This is the <%=blog.blog_title%> blog</h2>
    <a href="/author/settings"><h3>Settings</h3></a>
    <br>
    <form action="/author/add-draft" method="post">
        <a href="/author/edit"><button>Create new draft</button></a>
    </form>

    <h1><u>Published articles:</u></h1>
        <ul>
            <% articles.forEach(function(article){ %>
                <% if (article.published) { %>
                    <li id="<%=article.article_id%>">
                        <h3><%="Article title: "  + article.title  %></h3>
                        <h3><%="Created: "  + article.created  %></h3>
                        <h3><%="Published: "  + article.published  %></h3>
                        <h3><%="Last modified: "  + article.last_modified  %></h3>
                        <h3>Number of likes: <%=article.number_of_likes%></h3>
                        <h3>Number of views: <%=article.number_of_views%></h3>
                        <h3>Sharing link:
                            <a href="/reader/article/<%=article.article_id%>">
                                <u>localhost:3000/reader/article/<%=article.article_id%></u>
                            </a>
                        </h3>
                        <form action="/author/delete/<%=article.article_id%>" method="post">
                            <button class="delete-article-button">Delete</button>
                        </form>
                    </li>
                <% } %>
            <% }) %>
        </ul>
        <h1><u>Drafts:</u></h1>
        <ul>
            <% articles.forEach(function(article){ %>
                <%if (!article.published) { %>
                    <li>
```

```
                <h3><%="Article title: "  + article.title  %></h3>
                <h3><%="Created: "  + article.created  %></h3>
                <h3>Published: never</h3>
                <h3><%="Last modified: "  + article.last_modified  %></h3>
                <form action="/author/publish/<%=article.article_id%>" method="post">
                  <button class="publish-article-button">Publish?</button>
                </form>
                <form action="/author/edit/<%=article.article_id%>" method="get">
                  <button class="edit-article-button">Edit</button>
                </form>
              </li>
            <% } %>
          <% }) %>
      </ul>
</body>
</html>

================================================================================
                              VIEWS/AUTHOR-HOME.EJS END
================================================================================
================================================================================
                              VIEWS/AUTHOR-SETINGS.EJS START
================================================================================
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet"  type="text/css" href="/main.css" />
  <title>Author Settings</title>
</head>
<body>
  <h1>Account Settings</h1>

  <a href="/author"><button>Home</button></a>
  <form id="settings-form" action="/author/settings" method="post">
    <h3>Blog name: </h3>
    <input id="blog_title" type="text" name="blog_title" value="<%=blog.blog_title%>">
    <h3>Author name: </h3>
    <input id="author_name" type="text" name="author_name"
value="<%=blog.author_name%>">
    <br>
    <button type="submit">Save changes</button>
  </form>
```

```
    <p id="error-message"></p>

    <script>
        // here we ensure that every field is filled in before submitting.
        // although client side verification could be easily bypassed by simply
        // removing this code, it is not a security risk as it is simply the author
        // name and blog title
        document.getElementById("settings-form").addEventListener("submit", function(event) {
            const fields = ["blog_title", "author_name"];
            for (const field of fields) {
                if (document.getElementById(field).value == "") {
                    event.preventDefault();
                    document.getElementById("error-message").innerText = "Please fill out all fields
before submitting";
                    break;
                }
            }
        });
    </script>
</body>
</html>
```

```
================================================================================
                                VIEWS/AUTHOR-SETTINGS.EJS END
================================================================================
================================================================================
                                VIEWS/HOME.EJS START
================================================================================
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"  type="text/css" href="/main.css" />
    <title>Home page</title>
</head>
<body>
    <h1>Home page</h1>
    <a href="/author"><h3>Author Home Page</h3></a>
    <a href="/reader"><h3>Reader Home Page</h3></a>
</body>
```

```
</html>

================================================================================
                                VIEWS/HOME.EJS END
================================================================================
================================================================================
                                VIEWS/LOGIN.EJS START
================================================================================
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"  type="text/css" href="/main.css" />
   <title>Register</title>
</head>
<body>
   <h1>Login below</h1>
   <form action="/users/login" method="post">
      <p>Username: <input id="user" type="text" name="username" /></p>
      <p>Password: <input id="password" type="password" name="password" /></p>
      <button type="submit">login</button>
   </form>
   <h3>Or register!</h3>
   <a href="/users/register">
      <button type="submit">Register</button>
   </a>
</body>
</html>

================================================================================
                                VIEWS/LOGIN.EJS END
================================================================================
================================================================================
                                VIEWS/READER-ARTICLE.EJS START
================================================================================
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"  type="text/css" href="/main.css" />
   <title>Reader article</title>
</head>
```

```
<body>
   <h1>Reader article</h1>
   <a href="/reader/blog/<%=article.user_id%>"><button>Go back</button></a>
   <h2 id="article_title" type="text" name="article_title" placeholder="Article title">Title:
<%=article.title%></h2>
   <h3 id="author_name">Author name: <%=article.author_name%></h3>
   <h3 id="articlePublished">Published: <%=article.published%></h3>
   <br>
   <p id="reader-article-body"><%=article.body%></p>
   <br>
   <h3 id="number_of_views">number of views: <%=article.number_of_views%></h3>
   <h3 id="number_of_likes">number of likes: <%=article.number_of_likes%></h3>
   <%if (userLikedPost) {%>
   <form action="/reader/article/unlike-article/<%=article.article_id%>" method="post">
      <button>Unlike the article</button>
   </form>
   <%} else {%>
   <form action="/reader/article/like-article/<%=article.article_id%>" method="post">
      <button>Like the article</button>
   </form>
   <%}%>

   <form action="/reader/article/submit-comment/<%=article.article_id%>" method="post">
      <input id="commenter_name" name="commenter_name" placeholder="Your
name"></input>
      <textarea id="comment" rows="15" cols="50" name="comment" placeholder="Your
comment..."></textarea>
      <button>Submit Comment</button>
   </form>

      <% comments.forEach(function(comment){ %>
      <h3><%=comment.commenter_name%></h3>
      <p><%=comment.comment%></p>
   <% }) %>

</body>
</html>
```

==========

VIEWS/READER-ARTICLE.EJS END

==========

==========

VIEWS/READER-BLOG.EJS START

```
================================================================================
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"  type="text/css" href="/main.css" />
   <title>Reader Blog</title>
</head>
<body>
   <h1>Reader Blog</h1>
   <h2>This is the <u><%=blog.blog_title%></u> blog</h2>
   <a href="/reader"><button>Reader Home</button></a>

   <ul>
   <% articles.forEach(function(article){ %>
      <li class="reader-article">
         <h3><%=article.title%><h3>
         <h4><%=article.author_name %>, <%=article.published%></h4>
         <a href="/reader/article/<%=article.article_id%>"><button>Visit article</button></a>
      </li>
   <% }) %>
   </ul>
</body>
</html>

================================================================================
                              VIEWS/READER-BLOG.EJS END
================================================================================
================================================================================
                              VIEWS/READER-HOME.EJS START
================================================================================
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"  type="text/css" href="/main.css" />
   <title>Reader Home</title>
</head>
<body>
   <h1>Reader Home</h1>
   <a href="/"><button>Front page</button></a>
```

```
    <ul>
        <% blogs.forEach(function(blog){ %>
        <li>
            <h4>Blog title: <%=blog.blog_title %></h4>
            <h4>Author name: <%=blog.author_name %></h4>
            <a href="/reader/blog/<%=blog.user_id%>"><button>Visit blog</button></a>
        </li>
        <% }) %>
    </ul>
</body>
</html>
```

==========

VIEWS/READER-HOME.EJS END

==========

==========

VIEWS/REGISTER.EJS START

==========

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"  type="text/css" href="/main.css" />
    <title>Register</title>
</head>
<body>
    <h1>Register below</h1>
    <form action="/users/add-user" method="post">
        <p>Blog title: <input id="blog_title" type="text" name="blog_title" /></p>
        <p>Author name: <input id="author_name" type="text" name="author_name" /></p>
        <p>Username: <input id="username" type="text" name="username" /></p>
        <p>Password: <input id="password" type="password" name="password" /></p>
        <p>Confirm Password: <input id="confirm_password" type="password"
name="confirm_password" /></p>
        <button type="submit">Register</button>
    </form>
</body>
</html>
```

==========

VIEWS/REGISTER.EJS END

==========

****

****

****

****

MY CODE END

****

****

****

****

==========

DB_SCHEMA.SQL START

==========

```
-- This makes sure that foreign_key constraints are observed and that errors will be thrown for violations
PRAGMA foreign_keys=ON;

BEGIN TRANSACTION;

-- Create your tables with SQL commands here (watch out for slight syntactical differences with SQLite vs MySQL)
```

****

****

****

****

MY CODE START

****

****

****

****

```
-- Here we store users with their passwords for login
CREATE TABLE IF NOT EXISTS users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL,
    password TEXT NOT NULL,
    author_name TEXT
);

-- Each user has one blog, and all articles they have are assosiated to that
-- blog
```

```
CREATE TABLE IF NOT EXISTS blogs (
    blog_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INT, --the user that the blog settings belongs to
    blog_title TEXT NOT NULL,
    author_name TEXT NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

-- These are the articles assossiated with the blog the user has
CREATE TABLE IF NOT EXISTS articles (
    article_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INT NOT NULL,  --the user that the articles belongs to
    author_name TEXT NOT NULL, -- the display name that the user has in the users table
    created TEXT NOT NULL,
    last_modified TEXT NOT NULL,
    title TEXT,
    body TEXT,
    published TEXT, -- the time at which it was published
    number_of_views INTEGER DEFAULT 0,
    number_of_likes INTEGER DEFAULT 0
);

-- Here we store the article likes for all articles
CREATE TABLE article_likes (
    like_id INTEGER PRIMARY KEY,
    article_id INTEGER NOT NULL, -- This must match the article from the articles table
    ip_address TEXT,
    FOREIGN KEY (article_id) REFERENCES articles(article_id) -- the articles id that it gets in
the articles table
);

-- Here we store the article views for all articles
CREATE TABLE article_views (
    view_id INTEGER PRIMARY KEY,
    article_id INTEGER NOT NULL, -- This must match the article from the articles table
    ip_address TEXT,
    FOREIGN KEY (article_id) REFERENCES articles(article_id) -- the articles id that it gets in
the articles table
);

-- Here we store the comments views for all articles
CREATE TABLE article_comments (
    comment_id INTEGER PRIMARY KEY,
    article_id INTEGER NOT NULL, -- This must match the article from the articles table
```

```
    created TEXT NOT NULL,
    commenter_name TEXT NOT NULL,
    comment TEXT NOT NULL,
    FOREIGN KEY (article_id) REFERENCES articles(article_id) -- the articles id that it gets in
the articles table
);
-- DEFAULT STUFF THAT WAS HERE BEFORE I STARTED EDITING IT
-- Set up three users

--=====================================================================
=========
-- BELOW, IS HERE FOR DEMONSTRATION PURPOSES TO HELP SHOW THE BLOGS
-- FUNCTIONALITY AND TO HELP YOU GRADE MY ASSIGNMENT. THIS WOULD BE
REMOVED IN
-- A FINAL IMPLAMENTATION IF THIS WAS A REAL BLOG OF COURSE.
--=====================================================================
=========

-- These users will be impossible to log into, even with the correct password
-- here As it hashes your input password when you log in and then compares it
-- to the password we are storing in the database, aka this password below.
-- These are just here for demonstration to help you grade my assignment, and
-- to show the functionality of the blog.
INSERT INTO users ('username', "author_name", "password") VALUES ('Spongebob',
"Spongebob", "SpongesAreAwesome");
INSERT INTO users ('username', "author_name", "password") VALUES ('Patrick', "Patrick Star",
"Starfish62");
INSERT INTO users ('username', "author_name", "password") VALUES ('Krusty Krabs', "Krusty
Krabs", "TheSecretRecipe");

-- These numbers and values are extremely specific, one mistake and a post
-- won't be shown or an entire blog. These are usually made by the backend but
-- as i mentioned above this is just for demonstration purposes
INSERT INTO blogs ("user_id", "blog_title", "author_name") VALUES (1, "Spongebobs
adventures", "Spongebob");
INSERT INTO blogs ("user_id", "blog_title", "author_name") VALUES (2, "Patricks lounge",
"Patrick Star");
INSERT INTO blogs ("user_id", "blog_title", "author_name") VALUES (3, "Money, Money,
Monnnnneeeeeyyyy!!!", "Krusty Krabs");

INSERT INTO articles ("user_id", "author_name", "created", "last_modified", "title", "body",
"published", "number_of_views", "number_of_likes") VALUES (1, "Spongebob", "2024-01-14
16:55:16", "2024-01-14 16:55:16", "The Cold Morning", "Hello everyone, and welcome to the
```

first Bikini Bottom Blog! I went outside of my pineapple today and it was freezing so bad my snail almost caught a cold!", "2024-01-14 16:55:16", 31, 10);
INSERT INTO articles ("user_id", "author_name", "created", "last_modified", "title", "body", "published", "number_of_views", "number_of_likes") VALUES (1, "Spongebob", "2023-01-14 16:55:16", "2023-01-14 16:55:16", "Planktons Evil Plan", "Today plankton tried to steal Mr.krabs's formula again! Luckily me and patrick caught him since i am a fantastic driver hahahahaha!", "2023-01-14 16:55:16", 91, 25);

INSERT INTO articles ("user_id", "author_name", "created", "last_modified", "title", "body", "published", "number_of_views", "number_of_likes") VALUES (2, "Patrick Star", "2024-01-14 16:55:16", "2024-01-14 16:55:16", "My morning routine", "I wake up, and then, i go back to bed, I just sleep all day until Spongebob wakes me up!", "2024-01-14 16:55:16", 21, 15);

INSERT INTO articles ("user_id", "author_name", "created", "last_modified", "title", "body", "published", "number_of_views", "number_of_likes") VALUES (3, "Krusty Krabs", "2020-01-14 16:55:16", "2020-01-14 16:55:16", "Simply, Money.", "When i wake up I think of money, when i go to bed, i dream of money, when i make money, i'm thinking of how amazing money is. Isn't money just amazing? Spongebob tells me i have a money hoarding problem, but i disagree.", "2024-01-14 16:55:16", 1000, 25);

INSERT INTO article_comments ("article_id", "created", "commenter_name", "comment") VALUES (1, "2024-01-14 16:55:16", "Plankton", "I will find that secret recipe! Why do i write this on every post...");
INSERT INTO article_comments ("article_id", "created", "commenter_name", "comment") VALUES (2, "2024-01-14 16:55:16", "Plankton", "I will find that secret recipe! Why do i write this on every post...");
INSERT INTO article_comments ("article_id", "created", "commenter_name", "comment") VALUES (3, "2024-01-14 16:55:16", "Plankton", "I will find that secret recipe! Why do i write this on every post...");
INSERT INTO article_comments ("article_id", "created", "commenter_name", "comment") VALUES (4, "2024-01-14 16:55:16", "Plankton", "I will find that secret recipe! Why do i write this on every post...");

COMMIT;

==========

DB_SCHEMA.SQL END

==========

****

****

****

****

MY CODE END

****

****

****

****