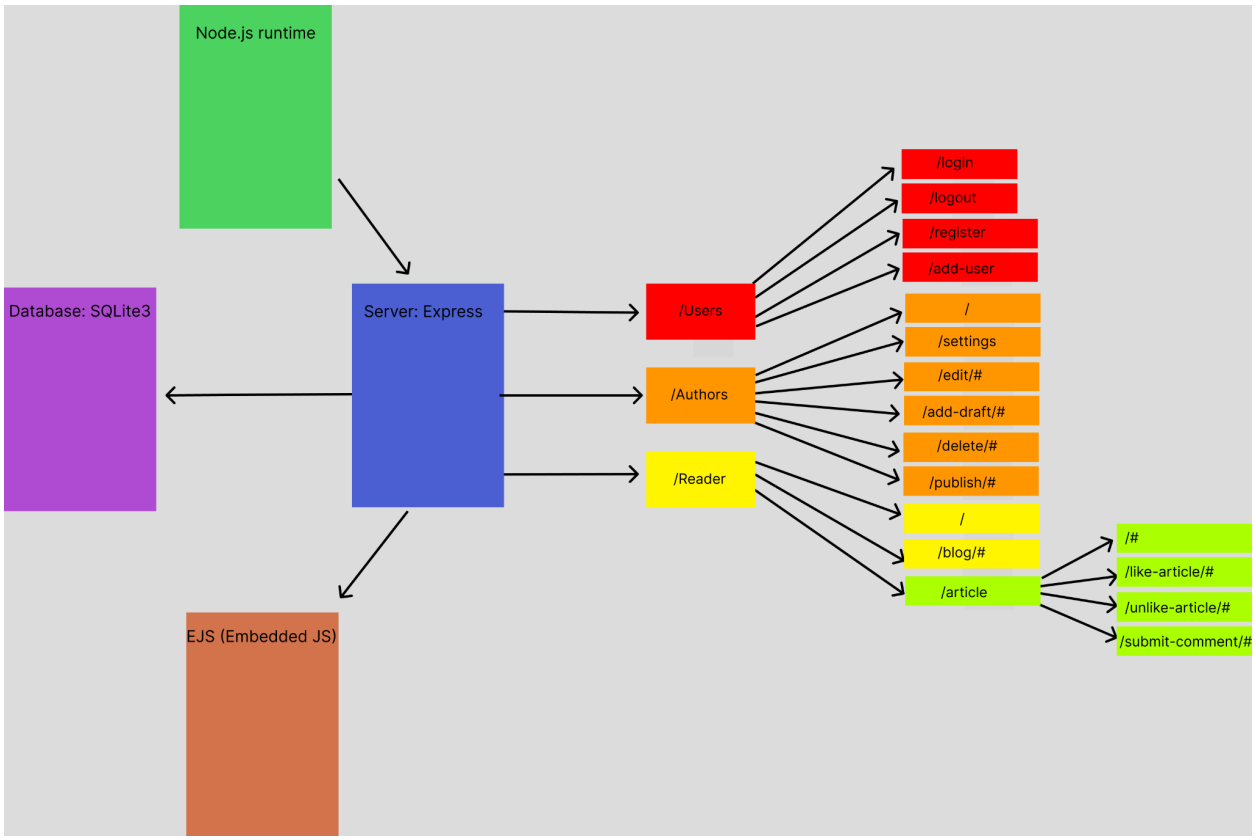
A high level schematic diagram of your website demonstrating all three tiers of your architecture, the end points that connect client to server



Specify which extension you implemented:

The extension I added was the Password access for author pages and routes extension. I wanted to challenge myself for this project so I decided that making a somewhat secure password access point would be a good idea since all the other extensions are things i've done before, or have lots of experience in.

Discuss what you did to implement it. Paste relevant parts of your code into the document to explain how you implemented and refer to the filenames and line numbers where the code can be found. Highlight aspects which you particularly want us to pay attention to.

I decided that rather than following the project's instructions of "*Naively stored server-side password (eg. in an environmental variable)"* I would store the passwords encrypted in the database. In doing this, i'm sure i have made a few flaws that would allow this authentication to be bypassed by a bad actor, for example, i send the plaintext password to the website, so if the packet got intercepted this would be bad news. I did however store it as a hash in the database and therefore the database itself is secure if it gets hacked.

To implement this I did the following:
THE SNIPS OF CODE BELOW ARE LOCATED IN ROUTES/USERS.js UNDER THE ADD-USER REQUEST HANDLER
First I got the required fields from the user using a simple form. Then i ensured they were all filled and that they typed their password the same time twice as shown below:

```
==========================================================================
   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.");
   }
==========================================================================
```

Then I used the Bcrypt library to encrypt the password given and to output a hashed value based on that given password. After encrypting the password I check to ensure that the username has not been taken, because if it has been taken then the user cannot use this username and we want to send them some sort of error message. This is shown below

```
==========================================================================
   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");
         });
==========================================================================
```

I then insert the username, the hashed password and the author name into the users database. I also create their blog in the blogs table as those fields are also filled upon registering this is all done in the snip of code below.

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

      console.log(hash);
      // 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");
         }
      });
   })
==========================================================================
```

Once the user has successfully registered we also redirect them to the /users/login page to allow them to login with their new account. This is shown at the end of the code snip above.

Registering was actually quite easy to implement once i figured out how to use the bcrypt library, but using session cookies was completely new to me, but i decided to use express-session as it was suggested in the mid-term requirements document.
To implement logging in i first ensure that they have put a username and password in otherwise i send them an error message.

==========

```
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);
   }
```

==========

If they have inputted a username and password, i look up their username in the database, if it does not exist, then i send them another error message as they cannot log into an account that does not exist.

==========

```
 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)
         }
```

==========

If they get past these steps then I get the users password from the database and use Bcrypt to compare their password provided to the hash we have in the database. We also give them a session cookie to allow them to stay logged in, this is created in index.js and has a 1 hour expiry date where it will deactivate.

==========

```
            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);
            }
          });
        }
    });
});
```

Since we are only graded for one extension I completed the base project, as specified exactly by the project description. The only difference is that when you are on the reader home page i show multiple authors blogs, and from there i show all the articles associated to that blog, or person. Since I allow different author accounts I can manage what blogs and posts are assigned to each author. This can be viewed if you navigate to /reader and follow any of the template blog posts i created.