COSC 360 Project Document

## Project Name: Vue Your Blog

### Team Members:

Shawn Mountenay - 62247606

Eloise Espel - 63799779

Shreyasi Chauhan - 31282924

Shaya Selincourt – 32399305

## Project Description

1. Team Members Selection

Shreyasi: Documentation

Shawn: Coding Main Features

Shaya: Accessibility Features + Unit Testing + Unit Testing Documentation

Eloise: Additional Website features + Unit Testing + Documentation

1. Summary of the Project

Vue Your Blog is a website that lets users create and view blogs on various topics. Users can sign in to post their thoughts and ideas on any topic they want and engage with the community in the comments sections. Our tech stack is Vue.js for the frontend framework, along with Vuetify which supplies a selection of prefab components such as buttons and app bars. For the backend, we are using a standard node.js server to consume API calls from the frontend and send it to the MySQL database. The API calls are created using Prisma, an object–relational mapping library to allow us to make database calls without writing SQL queries directly.

## Requirement List

From meeting on 03/15/2021.

1. Wireframe including planned layout, elements, sizes, placement

Has columns, header and footer.

1. Form validation with JavaScript

Does not let the user submit the form is the data entered does not meet the safety requirements (e.g., username is unique, password is safe and so on).

1. Data storage in MySQL and data queries with Prisma

Stores users and posts allows for searching and modifications. The database is updated asynchronously following AJAX principles.

1. Responsive design philosophy

The layout of the website changes depending on the user’s signed-in status or the action of the user (e.g., the user is writing a comment vs writing a post).

1. User images (thumbnail) and profile stored in database

The user profiles are stored and can be modified on the user’s profile page.

1. Error handling

Errors that occur during the operation of the website are caught in try/catch blocks and handled.

1. Accessibility

Ensure that text contrasts appropriately with the background and aria labels are

used across any component so those using screen readers can utilize the site.

1. Alert on page change

The user gets an alert when they switch page to make sure they are aware (not in the middle of writing a new post or a comment).

1. User Profile

The user can see their profile and modify their username, email, password and profile picture.

1. Search

Users can search for posts in the search bar, the search bar scans the database and returns the results to the user.

1. Admin

The admin can search for post and users (by post title, username and/or email) and disable or enable users. This functionality is only available if the user is signed in as an admin (in database isAdmin = 1).

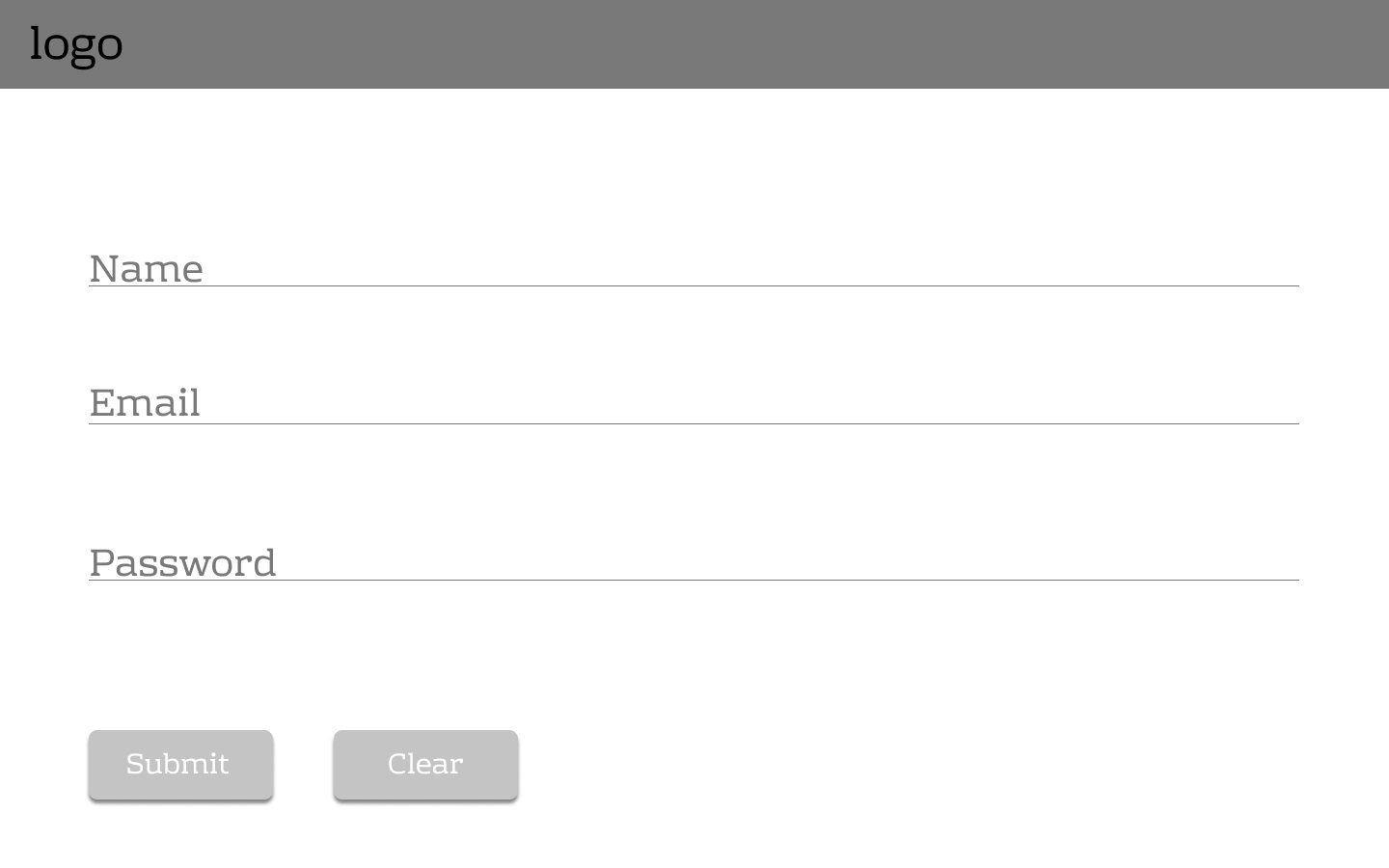
## Website Layout

From meeting on 03/15/2021.

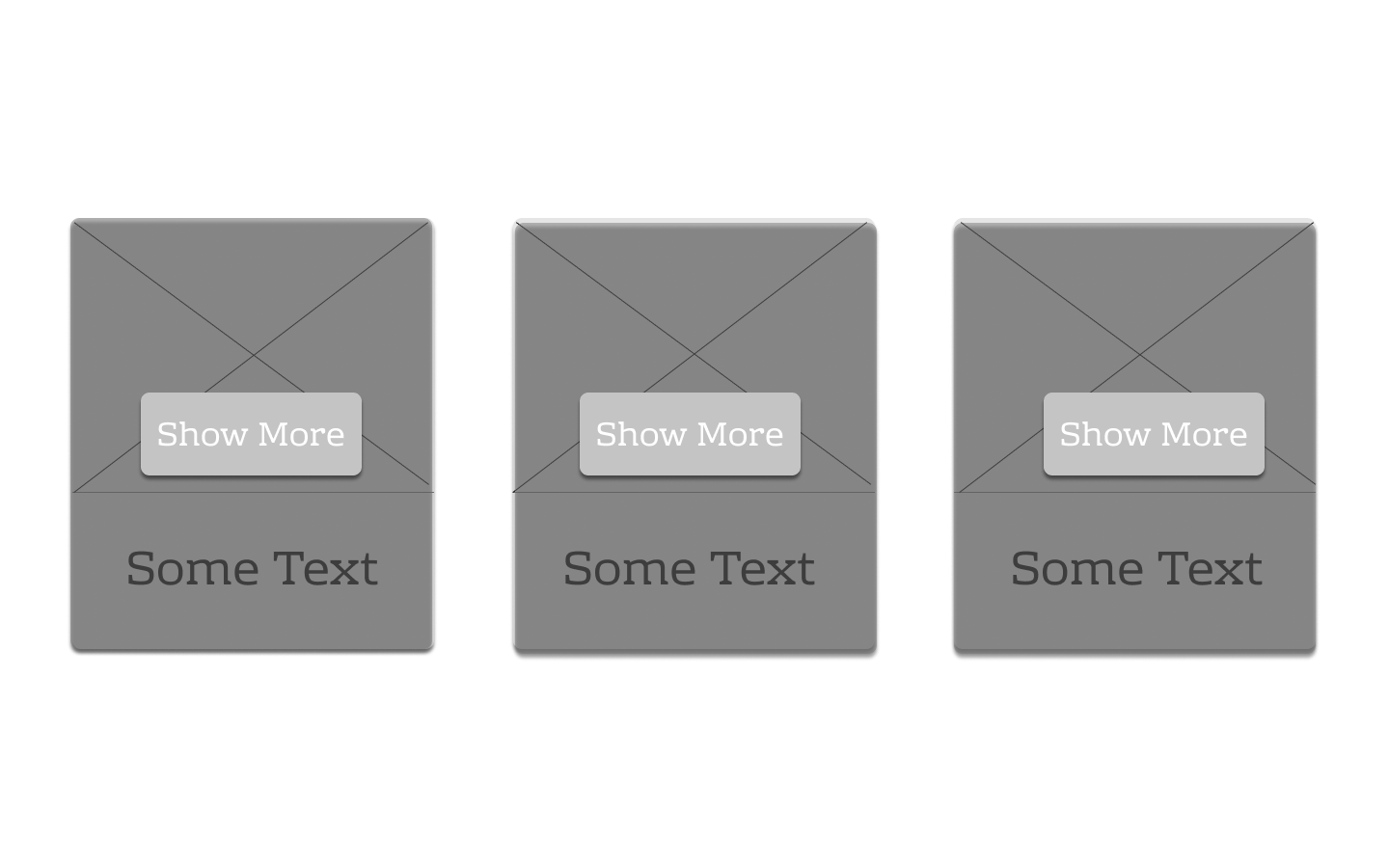
*Make the blog:*



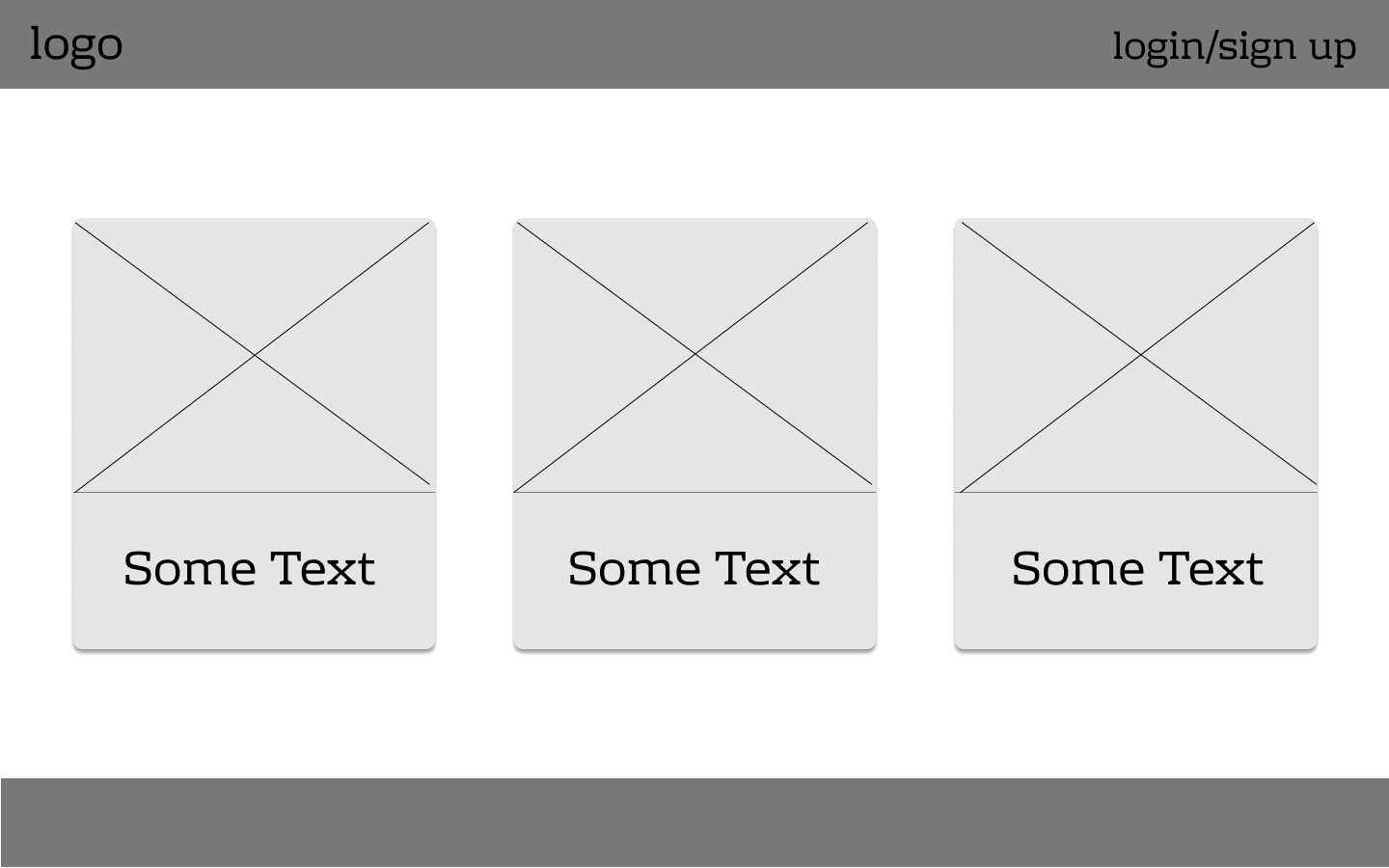
*Login page:*



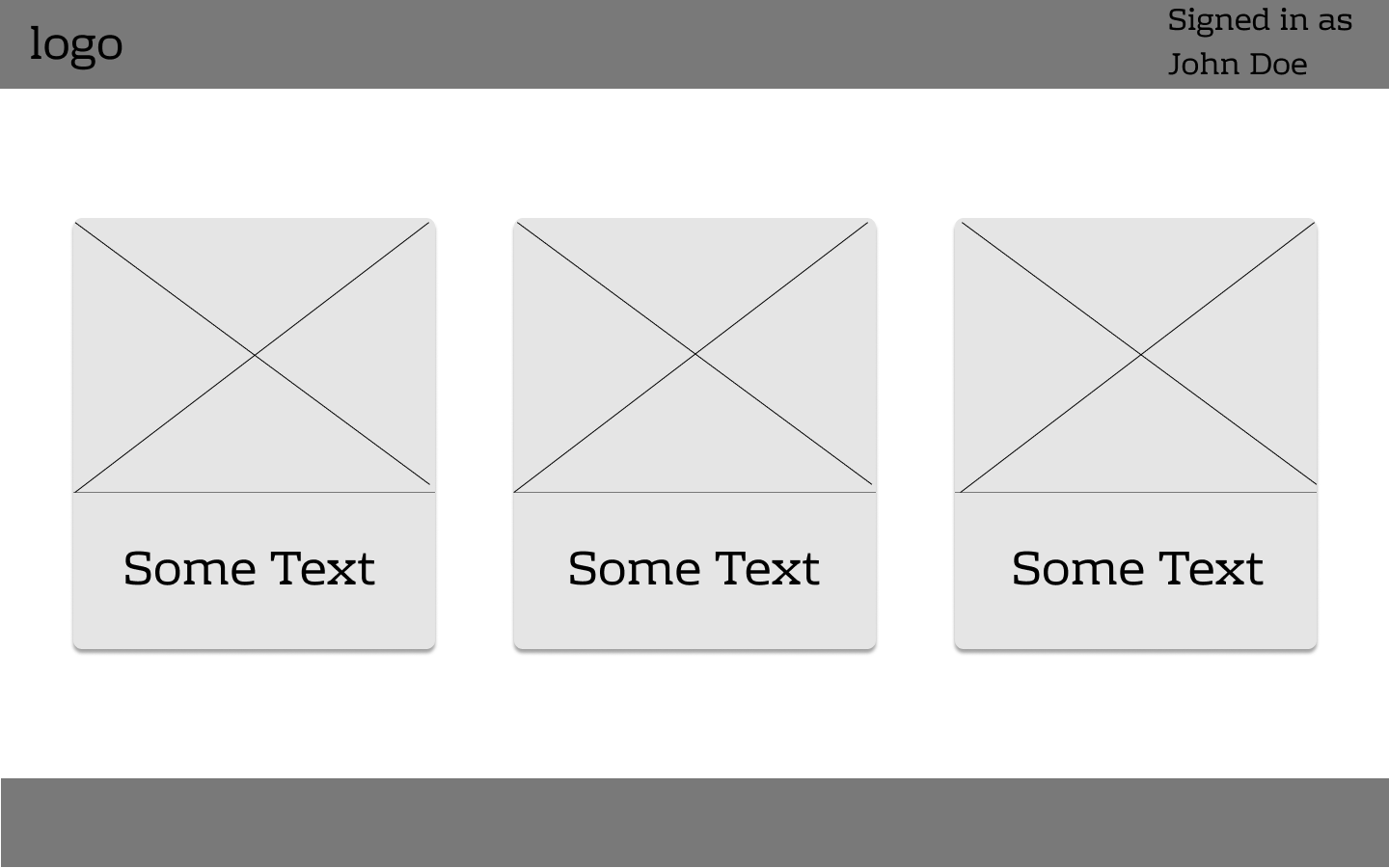
*Cards on hover:*



*Cards on homepage (status: User is Not Signed-In):*

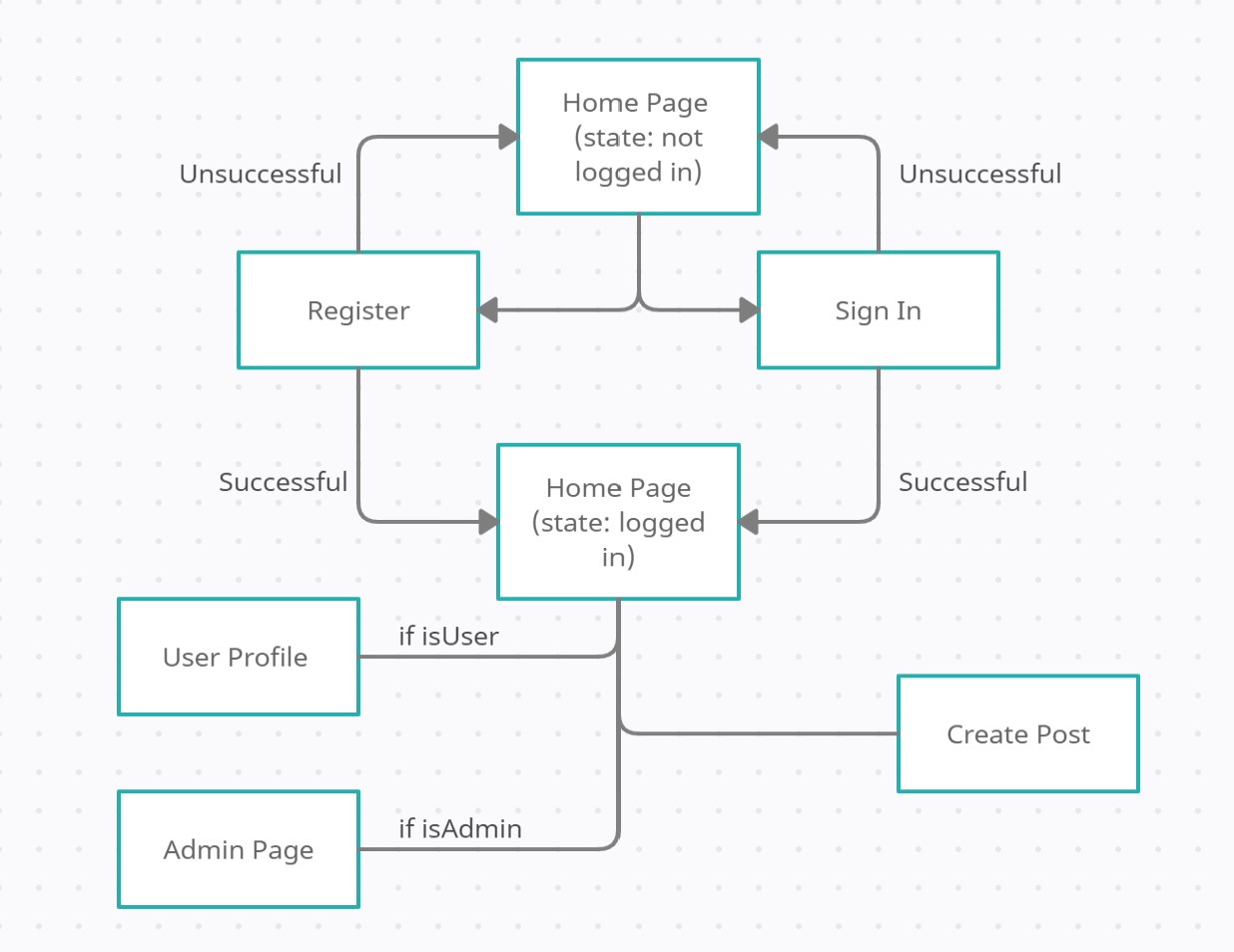


*Cards on homepage (status: User is Signed-In):*



## Site Map

Site map from 03/16/2021



**User**

When the user first opens the website, they are prompt to sign in or register (they can still view posts). Once the user is registered or logged in, they can access the home page to view blog posts from other users. The user can also access the “Make a Blog” page to write a new post that will be sent to the homepage for other users to read.

**Admin**

When the admin first opens the website, they are prompt to sign in. The database checks that the user with this username and password as an isAdmin value of 1 (0 = isAdmin = false, 1=isAdmin=true). The admin can now go into admin page search for users and posts and enable and disable users.

## Limitations

* Does not work on Internet Explorer due to Vue requiring polyfills
* The site doesn’t have built in testing protocols yet
* A remote server hasn’t been set up for public deployment
* Admin users cannot remove posts or comments
* Password recovery is currently not implemented, users will have to contact admins directly

## Security and Validation

All passwords are encrypted using the node backend library called bcrypt. We use this to hash a password once it arrives in the server, either to store it in the database, or to compare it to the value already in the database. Users who are not logged in or registered are also unable to access web pages such as a user’s profile or the admin menu. When a user registers, we ensure that the username and email hasn’t already been added to the database and they must agree to allow us to use their data for whatever we see fit.

Description of Files and Folders of the Code Base

This project uses a simple client-server architecture, in which a client carries out actions in the browser, which make requests to the server when data requires accessing, creating, mutating or deleting. There are three main units of our client-server architecture. On the client-side, we have the View, and on the server-side, we have the Model and the Controller (MVC). Our three main libraries are VueJS for the View, ExpressJS, and Prisma for the Controller, and MySQL for the Model. VueJS runs on the client's computer, ExpressJS handles the requests and queries the MySQL database with Prisma.

**Client Side**

On the client-side, the project has a component-based architecture. The files and folders are broken up into some main logical blocks. In the root of the project, there are various configuration files, for linting, formatting, git, and package management. There is a node\_modules folder that contains dependencies of the project. There is a public folder that holds the index.html file, that has the app HTML, CSS, and JavaScript injected into it. The Source folder (src) is where most of the project code lies.

The Source folder has an assets folder, which contains favicons that are referenced in the public folder’s index.html. The source folder has a main Javascript file, this is where we tell our project to run VueJS and add our plugins to it. The main plugins we have in the frontend are VueX, VueRouter, Vuetify, Vuelidate, TiptapVuetify, and Axios.

VueX is the global state management plugin, it allows variables to be set and accessed globally across the project. VueX has a folder in the source folder called “store” with a single JavaScript file. VueRouter allows routes to be changed dynamically with JavaScript in the Vue code. VueRouter has a folder in the source folder called “routes” with a single JavaScript file. Vuetify is a Google Material Design based component library that has attractive looking and feeling components such as tables, buttons, cards, and more. Vuetify has a configuration file inside the source in the plugins folder. Vuelidate is used to validate input fields dynamically. TipTapVuetify is a plugin for users to edit text. Vuelidate and TipTap Vuetify require no configuration files. Axios has a file in services that specifies how to make an HTTP request to the server (that is, which headers, protocols, and data formats to send and receive).

The Vue Code is split up into three main logical units of components. At the top level is the App component. Below that are view components, which are found in the “views” folder, and below that are reusable components, and are found in the “components” folder.

The App component, contains the NavBar, the Footer, and the Main section of the app. The NavBar and the Footer are mostly concerned with having links to URLs that switch the Main section of the App and change the route on click. However, the NavBar also has a search function on the home page.

The view components are tied to a route and go inside the main section of the app component. For example, the sign-in page, the registration page, or the home page.

The reusable components are components that are used either in the App component, the views components or in other reusable components. For example, the FormSubmitAndClear component is used in both the RegisterForm and the BlogForm components.

**Server Side**

The server-side implements an asynchronous functional programming style, with many functions waiting to handle HTTP requests with ExpressJS, query the database with Prisma (Prisma is our ORM (Object Relational Mapping library) and send back appropriate responses. On the top level, there are configuration files for package management, linting, Git and environment variables. There is a folder with user/blog images in uploads. There is a Prisma folder that has a config file that specifies the schema of our Database so we can make Prisma Database calls. There is a node\_modules folder that includes project dependencies. Furthermore, there is a source folder (src). Inside the source folder, we have a middleware folder that has a Multer configuration file (Multer is used to handle image uploads in the server directory). There is a JavaScript file in the source directory that has the bulk of our ExpressJS code. This is the file in which our code in which ExpressJS handles our HTTP requests, queries the DB with Prisma, and sends back responses to the client.

## **End to End example**

When a user is signing in, he goes to the /signin route in the browser. This is represented by a “SignIn.vue” view component which contains a “SignInForm.vue” component. The username and password are sent with a submit method which contains an axios HTTP request function that we have implemented in a JavaScript file called EventService.js in the services folder in the frontend. This Axios function specifies the HTTP headers to use, which responses to accept, which type of HTTP method to use, and takes the username and password as a JSON object. This Axios function specifies the server directory ‘/signin’, makes a Prisma query to MySQL sending the username and returning the user row with the hashed password if the username exists. We then compare the user entered password with the hashed password that we received from the database/Prisma. If there is a match, then we send as a response that the credentials were valid via JSON.