Authenticate Me:

In this multi-part project, you will learn how to put together an entire Express + React application with authentication.

- **Backend Phases**
  - **Phase 0** - Set Up
  - **Phase 1** - API Routes
  - **Phase 2** - Error Handling
  - **Phase 3** - User Authentication
  - **Phase 4** - User Auth Routes
  - **Phase 5** - Validating the Request Body
- **Frontend Phases**
  - **Phase 0** - Set Up
  - **Phase 1** - Login Form Page
  - **Phase 2** - Signup Form Page
  - **Phase 3** - Logout
  - **Bonus** - Make Login Form Page into a Modal
- **Deployment to Heroku**

Backend Dependencies

- **`bcryptjs`** - Password Hashing
- **`cookie-parser`** - parsing cookies from requests
- **`cors`** - CORS
- **`csurf`** - CSRF protection
- **`dotenv`** - load environment variables into Node.js from a `.env` file
- **`express`** - Express
- **`express-async-handler`** - handling `async` route handlers
- **`express-async-errors`** - handling `async` route handlers
- **`express-validator`** - used to validate the body of requests for routes
- **`helmet`** - security middleware
- **`jsonwebtoken`** - JWT
- **`morgan`** - logging information about server requests/responses
- **`per-env`** - use environment variables for starting app differently
- **`pg@">=8.4.1"`** - PostgreSQL greater or equal to version 8.4.1
- **`sequelize@6`** - Sequelize
- **`sequelize-cli@6`** - use `sequelize` in the command line

Backend Dev Dependencies

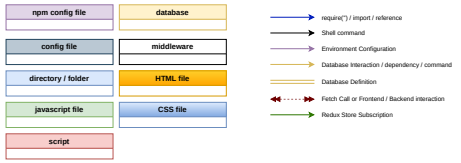
- **`dotenv-cli`** - use `dotenv` in the command line
- **`nodemon`** - hot reload server `backend` files

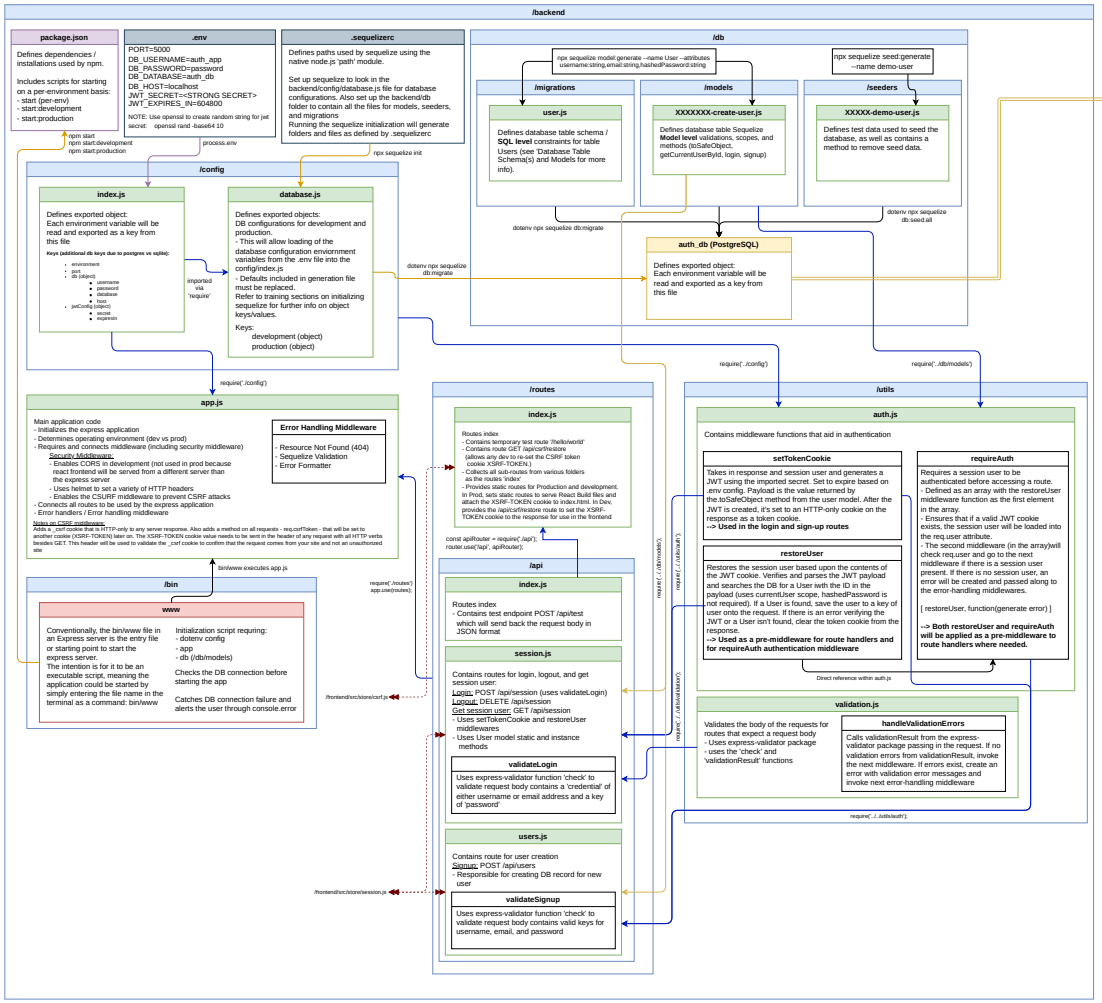
Frontend Dependencies

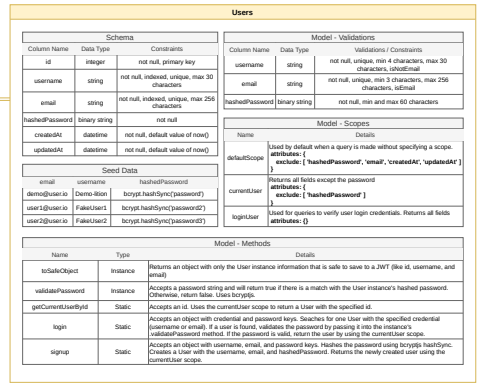
- **`js-cookie`** - extracts cookies
- **`react-redux`** - React components and hooks for Redux
- **`react-router-dom@^5`** - routing for React
- **`redux`** - Redux
- **`redux-thunk`** - add Redux thunk

Frontend Dev Dependencies

- **`redux-logger`** - log Redux actions in the browser's dev tools console







Authentication Login Flow

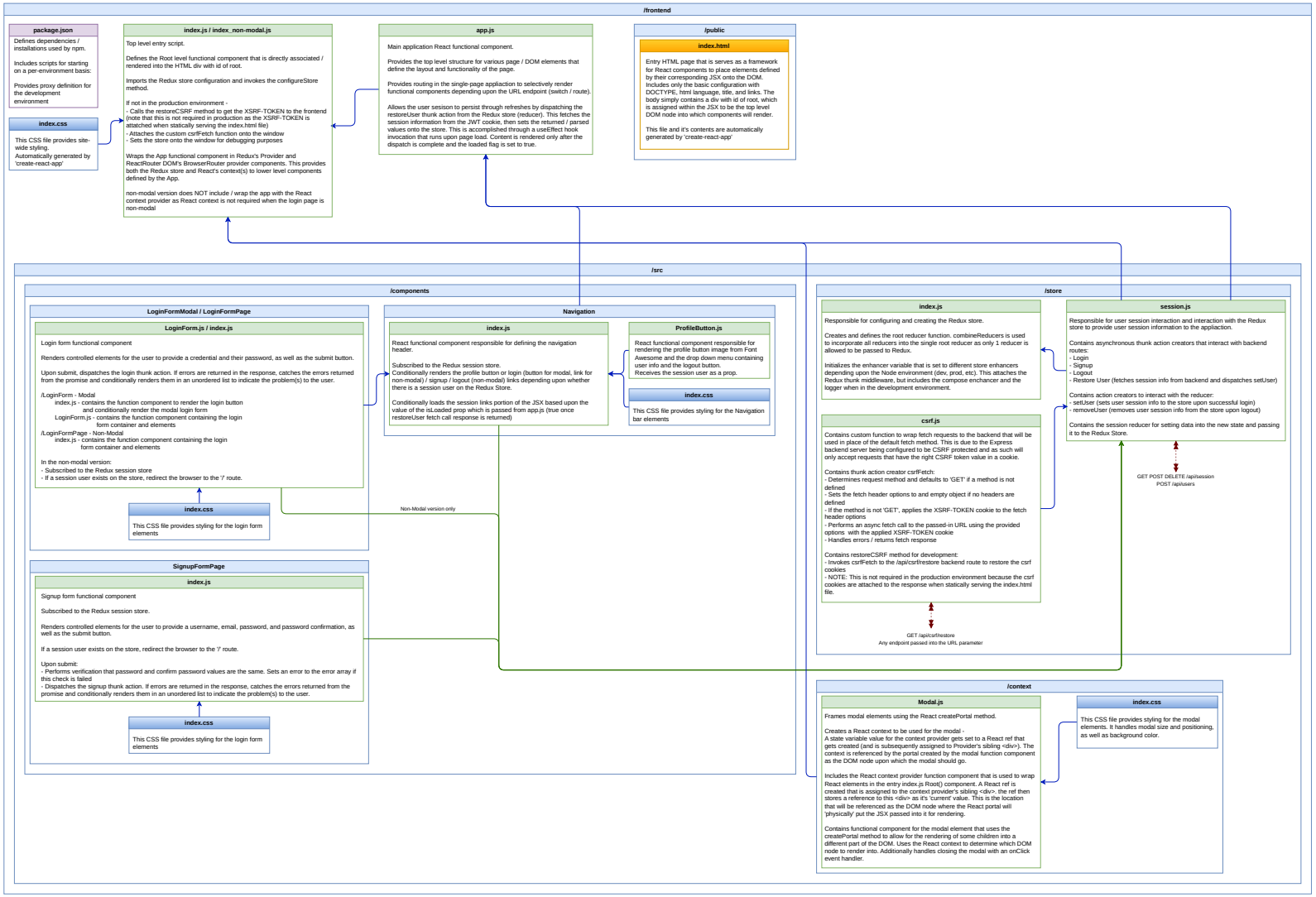
1. The API login route will be hit with a request body holding a valid credential (either username or email) and password combination.
2. The API login handler will look for a `User` with the input credential in either the `username` or `email` columns.
3. Then the `hashedPassword` for that found `User` will be compared with the input `password` for a match.
4. If there is a match, the API login route should send back a JWT in an HTTP-only cookie and a response body. The JWT and the body will hold the user's `id`, `username`, and `email`.

Sign-up Flow

1. The API signup route will be hit with a request body holding a `username`, `email`, and `password`.
2. The API signup handler will create a `User` with the `username`, an `email`, and a `hashedPassword` created from the input `password`.
3. If the creation is successful, the API signup route should send back a JWT in an HTTP-only cookie and a response body. The JWT and the body will hold the user's `id`, `username`, and `email`.

Logout Flow

1. The API logout route will be hit with a request.
2. The API logout handler will remove the JWT cookie set by the login or signup API routes and return a JSON success message.


/frontend
package.json
Defines dependencies / installations used by npm.
Includes scripts for starting on a per-environment basis:
Provides proxy definition for the development environment
index.css
This CSS file provides site-wide styling.
Automatically generated by 'create-react-app'
index.js / index_non-modal.js
Top level entry script.
Defines the Root level functional component that is directly associated / rendered into the HTML div with id of root.
Imports the Redux store configuration and invokes the configureStore method.
If not in the production environment -
- Calls the restoreCSRF method to get the XSRF-TOKEN to the frontend (note that this is not required in production as the XSRF-TOKEN is attatched when statically serving the index.html file)
- Attaches the custom csrfFetch function onto the window
- Sets the store onto the window for debugging purposes
Wraps the App functional component in Redux's Provider and ReactRouter DOM's BrowserRouter provider components. This provides both the Redux store and React's context(s) to lower level components defined by the App.
non-modal version does NOT include / wrap the app with the React context provider as React context is not required when the login page is non-modal
app.js
Main application React functional component.
Provides the top level structure for various page / DOM elements that define the layout and functionality of the page.
Provides routing in the single-page appliaction to selectively render functional components depending upon the URL endpoint (switch / route).
Allows the user sessison to persist through refreshes by dispatching the restoreUser thunk action from the Redux store (reducer). This fetches the session information from the JWT cookie, then sets the returned / parsed values onto the store. This is accomplished through a useEffect hook invocation that runs upon page load. Content is rendered only after the dispatch is complete and the loaded flag is set to true.
/public
index.html
Entry HTML page that is serves as a framework for React components to place elements defined by their corresponding JSX onto the DOM. Includes only the basic configuration with DOCTYPE, html language, title, and links. The body simply contains a div with id of root, which is assigned within the JSX to be the top level DOM node into which components will render.
This file and it's contents are automatically generated by 'create-react-app'
/src
/components
LoginFormModal / LoginFormPage
LoginForm.js / index.js
Login form functional component
Renders controlled elements for the user to provide a credential and their password, as well as the submit button.
Upon submit, dispatches the login thunk action. If errors are returned in the response, catches the errors returned from the promise and conditionally renders them in an unordered list to indicate the problem(s) to the user.
/LoginForm - Modal
index.js - contains the function component to render the login button and conditionally render the modal login form
LoginForm.js - contains the function component containing the login form container and elements
/LoginFormPage - Non-Modal
index.js - contains the function component containing the login form container and elements
In the non-modal version:
- Subscribed to the Redux session store
- If a session user exists on the store, redirect the browser to the '/' route.
index.css
This CSS file provides styling for the login form elements
Non-Modal version only
SignupFormPage
index.js
Signup form functional component
Subscribed to the Redux session store.
Renders controlled elements for the user to provide a username, email, password, and password confirmation, as well as the submit button.
If a session user exists on the store, redirect the browser to the '/' route.
Upon submit:
- Performs verification that password and confirm password values are the same. Sets an error to the error array if this check is failed
- Dispatches the signup thunk action. If errors are returned in the response, catches the errors returned from the promise and conditionally renders them in an unordered list to indicate the problem(s) to the user.
index.css
This CSS file provides styling for the login form elements
Navigation
index.js
React functional component responsible for defining the navigation header.
Subscribed to the Redux session store.
Conditionally renders the profile button or login (button for modal, link for non-modal) / signup / logout (non-modal) links depending upon whether there is a session user on the Redux Store.
Conditionally loads the session links portion of the JSX based upon the value of the isLoaded prop which is passed from app.js (true once restoreUser fetch call response is returned)
ProfileButton.js
React functional component responsible for rendering the profile button image from Font Awesome and the drop down menu containing user info and the logout button.
Receives the session user as a prop.
index.css
This CSS file provides styling for the Navigation bar elements
/store
index.js
Responsible for configuring and creating the Redux store.
Creates and defines the root reducer function. combineReducers is used to incorporate all reducers into the single root reducer as only 1 reducer is allowed to be passed to Redux.
Initializes the enhancer variable that is set to different store enhancers depending upon the Node environment (dev, prod, etc). This attaches the Redux thunk middleware, but includes the compose enchancer and the logger when in the development environment.
session.js
Responsible for user session interaction and interaction with the Redux store to provide user session information to the appliaction.
Contains asynchronous thunk action creators that interact with backend routes:
- Login
- Signup
- Logout
- Restore User (fetches session info from backend and dispatches setUser)
Contains action creators to interact with the reducer:
- setUser (sets user session info to the store upon successful login)
- removeUser (removes user session info from the store upon logout)
Contains the session reducer for setting data into the new state and passing it to the Redux Store.
GET POST DELETE /api/session
POST /api/users
csrf.js
Contains custom function to wrap fetch requests to the backend that will be used in place of the default fetch method. This is due to the Express backend server being configured to be CSRF protected and as such will only accept requests that have the right CSRF token value in a cookie.
Contains thunk action creator csrfFetch:
- Determines request method and defaults to 'GET' if a method is not defined
- Sets the fetch header options to and empty object if no headers are defined
- If the method is not 'GET', applies the XSRF-TOKEN cookie to the fetch header options
- Performs an async fetch call to the passed-in URL using the provided options with the applied XSRF-TOKEN cookie
- Handles errors / returns fetch response
Contains restoreCSRF method for development:
- Invokes csrfFetch to the /api/csrf/restore backend route to restore the csrf cookies
- NOTE: This is not required in the production environment because the csrf cookies are attached to the response when statically serving the index.html file.
GET /api/csrf/restore
Any endpoint passed into the URL parameter
/context
Modal.js
Frames modal elements using the React createPortal method.
Creates a React context to be used for the modal -
A state variable value for the context provider gets set to a React ref that gets created (and is subsequently assigned to Provider's sibling). The context is referenced by the portal created by the modal function component as the DOM node upon which the modal should go.
Includes the React context provider function component that is used to wrap React elements in the entry index.js Root() component. A React ref is created that is assigned to the context provider's sibling . the ref then stores a reference to this as it's 'current' value. This is the location that will be referenced as the DOM node where the React portal will 'physically' put the JSX passed into it for rendering.
Contains functional component for the modal element that uses the createPortal method to allow for the rendering of some children into a different part of the DOM. Uses the React context to determine which DOM node to render into. Additionally handles closing the modal with an onClick event handler.
index.css
This CSS file provides styling for the modal elements. It handles modal size and positioning, as well as background color.