LIBRARY EXAM

OVERVIEW

1. SETUP
2. WORKFLOW
3. SUBMISSION
4. API + VUE: CREATE BOOK FEATURE
5. API: AUTHENTICATION + AUTHORIZATION
6. VUE: AUTHENTICATION + AUTHORIZATION

1. SETUP

GITHUB

1. **Commit and push any commits** in your personal repository, if there are any
2. **Pull all commits** from the course repository
3. **Copy the library exam into your personal repository** from the course repository

MYSQL WORKBENCH

4. **Seed the database** by running `/library-exam/library-api/database/library_exam.sql` in MySQL Workbench

VS CODE: API PROJECT

5. Open the API project folder `/library-exam/library-api` in VS Code and open a terminal window inside VS Code
6. **Install project dependencies** by running `npm install`
7. **Update environment variables** by opening `.env`, inserting your actual MySQL database server username and password, and saving the file

VS CODE: VUE PROJECT

8. Open the Vue project folder `/library-exam/library-web/client/` in a new VS Code instance and open a terminal window inside this VS Code instance
9. **Install project dependencies** by running `npm install`
10. **Build the Vue project** by running `npm run build`, using Vite to transform the Vue project source code into the build output in `/library-exam/library-web/client/dist`

VS CODE: WEB PROJECT

11. Open the static web server project folder `/library-exam/library-web` in a new VS Code instance and open a terminal window inside this VS Code instance
12. **Install project dependencies** by running `npm install`

GITHUB

13. **Commit and push all of the initial changes immediately**, before starting to work

2. DEVELOPMENT WORKFLOW

- **Start the API server once** by running `npm run dev` inside the API project directory, `/library-exam/library-api`, and leave it running (Nodemon will restart the server automatically whenever you save any source code file)
- **Start a Vite development server once** by running inside the Vue client project directory, `/library-exam/library-web/client`, and leave it running (Vite will restart the server automatically whenever you save any source code file)
- **Launch the application** by opening http://localhost:5173 in a browser, and sign up or log in using credentials in `/library-exam/library-api/database/credentials.txt`
- **Develop and test** the library exam projects (in your personal repository, not the course repository)
- **Commit your changes regularly** to your personal repository
- **Reset the database at any time** by running `/library-exam/library-api/database/library_exam.sql` in MySQL Workbench

3. SUBMISSION

1. **Stop the Vite development server** by pressing CTRL + C in the terminal
2. **Build the Vue project** by running `npm run build`
3. **Start the static web server** by running `npm run dev`, in the web server project directory, `/library-exam/library-web`, to start serving the single-page application build output in `/library-exam/library-web/client/dist` to clients
4. **Launch the application** by opening http://localhost:8080 in a browser
5. **Test the application** to ensure that everything still works
6. **Commit and push all of your remaining changes** to your personal repository
7. **Make a .zip of your library exam, excluding the 3 node_modules directories** but including the rest of the exam, including the API project, static web server project, and the Vue application project
8. **Upload the .zip to Moodle**

4. API + VUE: CREATE BOOK FEATURE

API: BOOK REPOSITORY

1. **Implement the `createBook` function** in `/server/persistence/book-repository.js`
   - Define an `INSERT` SQL query to insert a new book in the `books` table using `?` parameter placeholders
   - Execute the query, passing the title, author, year, page count, and description as parameters to the database (do not insert them directly in the query, and ensure that you asynchronously execute the query by using the `await` keyword)

API: POST /BOOKS ROUTE

2. **Implement the `POST /books` route** in `/server/routes/v1/books.js`
   - Get the title, author, year, page count, and description parameters from the request body
   - Validate the parameters, and exit early by sending a 400 Bad Request response if they are invalid
   - Otherwise, asynchronously call the repository `createBook` function, passing the validated parameters, in order to create a new book in the database (ensure that you asynchronously call the function by using the `await` keyword)
   - Send the created book back to the client in the body of a 201 Created response

VUE: BOOK SERVICE

3. **Implement the `createBook` function** in `/src/services/book-service.js`
   - Validate the parameters, and return an object with the error if validation fails
   - Otherwise, asynchronously send an HTTP POST request to the API's `POST /books` route using Axios, passing the parameters in the request body (ensure that you asynchronously call the function by using the `await` keyword)
   - If the server returns a successful response, return a success message, but otherwise, handle the error and return an error message

VUE: ADD BOOK PAGE

4. **Implement the create book functionality** in `/src/pages/AddBookPage.vue`
   - Asynchronously call the book service function to send a request to the API using the form parameters whenever the "Add" button is clicked

5. API: AUTHENTICATION + AUTHORIZATION

ROUTES

1. **Add authentication and authorization to `POST /books` route** to ensure that the user is logged in and that their role is equal to `'librarian'`
2. **Add authentication and authorization to `PUT /books/:id` route** to ensure that the user is logged in and that their role is equal to `'librarian'`
3. **Add authentication and authorization to `PATCH /books/:id` route** to ensure that the user is logged in and that their role is equal to `'librarian'`
4. **Add authentication and authorization to `DELETE /books/:id` route** to ensure that the user is logged in and that their role is equal to `'librarian'`

- You are **not allowed to use the authenticator and authorizer middleware** provided in the demos, and instead must implement the logic according to the following instructions
- In each case, **add authentication and authorization** by doing the following:
    - **Ensure that `req.session` exists**
    - If not, then the user is not logged in, so exit early by sending a **401 Not Authorized** response (meaning not authenticated)
    - Otherwise, access `req.session.user` to check the user's role, and **ensure that their role is equal to `'librarian'`**
    - If not, then the user does not have the required role, so exit early by sending an error **403 Forbidden response** (meaning not authorized)
    - Otherwise, the user is authenticated and authorized to perform the given action, so **continue** to execute the main route handler code as normal
- Do all of the above manually, either:
    - Directly in each route handler (acceptable)
    - Extract duplicated code to a single function you call multiple times (better)
    - Create your own route-level middleware request handlers (authenticator and authorizer) and insert them before the main route handlers (best)

6. VUE: AUTHENTICATION + AUTHORIZATION

PAGE HEADER COMPONENT

1. **Only show the “Sign up” and “Log in” header links if the user is not authenticated**, meaning that the user is not logged in
    - In `PageHeader.vue`, use `authentication-service.js` to get the currently authenticated user, if any
2. **Only show the “Log out” header link if the user is authenticated**, meaning that the user is logged in
    - In `PageHeader.vue`, use `authentication-service.js` to get the currently authenticated user, if any
3. **Only show the “Add book” header link if the user is both authenticated and authorized**, meaning that the user is logged in and has the role `'librarian'`
    - In `PageHeader.vue`, use `authentication-service.js` to get the currently authenticated user, if any, and check their string `role` to ensure that it is equal to `'librarian'`

ADD BOOK PAGE COMPONENT

4. **Only show the form and allow the user to add a book if the user is both authenticated and authorized**, meaning that the user is logged in and has the role `'librarian'`
    - In `AddBookPage.vue`, use `authentication-service.js` to get the currently authenticated user, if any, and check their string `role` to ensure that it is equal to `'librarian'`

BOOK PAGE COMPONENT

5. **Only show the button/form and allow the user to edit a book if the user is both authenticated and authorized**, meaning that the user is logged in and has the role `'librarian'`
    - In `BookPage.vue`, use `authentication-service.js` to get the currently authenticated user, if any, and check their string `role` to ensure that it is equal to `'librarian'`