Project Instructions: SQL Library Manager

Open the project files in your favorite text editor and preview them in the browser. Also, it's a smart idea to test your project in multiple browsers!

Github

- Have a GitHub account and create a new repo for this project.
- Create a README.md file for your repo that explains what the project is and anything your fellow developers might need to know to use the project.

Download the project files. We've supplied several files for you to use:

- Use the library.db SQLite database file as the source of your data.
- HTML mockups and CSS files. These will be the basis of the use cases described in the project instructions.

Handle files and folders that shouldn't be stored in your repo

- Create a .gitignore file in your directory and save a reference to the node_modules folder so that your repo doesn't store or track that folder.

Initialize your project

- Open the command line, navigate to your project, and run the npm init command to set up your package.json file.
- Customize the package.json file if necessary so that running npm start launches the app.

Add your dependencies

- At a minimum, your project will need:
    - Sequelize
    - Sqlite3
    - Express
    - Pug
- Don't forget to use the --save flag if you're using a version of npm prior to 5.0, to ensure that references to the dependencies are stored in your package.json file. You can run npm -v to see what version you have installed.

Note: Use of the Sequelize CLI to initialize the configuration code, folders and helpers you will need for this project is allowed but is not required.

Initialize Sequelize and create your models

- The library.db SQLite database file should contain the tables you need.
- Create the following Sequelize model:
    - The Book model for the Books table should have the following properties with the following data types:
        - Title – string
        - Author – string
        - Genre – string
        - Year – integer
- In the Book Model, add the appropriate validation to ensure that the title and author properties will have values when the form is submitted. See the project resources for a link to the video that covers this.

Set up your server, middleware, and routes

- At the very least, you will need the following routes:
    - get / - Home route should redirect to the /books route.
    - get /books - Shows the full list of books.
    - get /books/new - Shows the create new book form.
    - post /books/new - Posts a new book to the database.
    - get /books/:id - Shows book detail form.
    - post /books/:id - Updates book info in the database.
    - post /books/:id/delete - Deletes a book. Careful, this can't be undone. It can be helpful to create a new "test" book to test deleting.
- Set up a custom error handler middleware function that logs the error to the console and renders an "Error" view with a friendly message for the user. This is useful if the server encounters an error, like trying to view the "Books Detail" page for a book :id that doesn't exist. See the error.html file in the example-markup folder to see what this would look like.
- Set up a middleware function that returns a 404 NOT FOUND HTTP status code and renders a "Page Not Found" view when the user navigates to a non-existent route, such as /error. See the page_found.html file in the example markup folder for an example of what this would look like.

Build your views

- Use the examples in the example-markup folder to see what your views should look like and what elements they should have. At the very least, you will need the following views:
    - layout.pug - for the boilerplate markup that should be on every page.
    - index.pug - for the main book listing page.

    - new-book.pug - for the new book form.
    - update-book.pug - for the update book form.
    - error.pug - for displaying a user friendly error message.
    - page-not-found.pug - for displaying a user friendly “Page Not Found” message.

NOTE: You should use Pug to render your views for this project. Avoid using a front end library or framework such as React, Angular, or Vue.

Required fields and forms

- If the required title and author fields are empty upon form submission, the user should be notified accordingly with a friendly error message on the page. See the form-error.html file for an example of what this will look like.
- Use Sequelize model validation for validating your form fields. Don't rely simply on HTML5 built in validation.
- When form labels are clicked, the corresponding input should be placed in the focus state. This is accomplished by matching the input’s id attribute to its label’s for attribute.

NOTE: When new book or book detail form is submitted, your app should redirect to the books listing page.

Styles and Layout

- You are provided with all the styles you will need for this project, in the public/stylesheets/styles.css file. This is the CSS file that you will need to link to your Pug templates.
- Feel encouraged to customize things like color, background color, fonts, borders, and shadows. But the layout and positioning of elements should generally match the example HTML files in the example-markup folder.

NOTE: The href value that you use in your layout.pug file to add the styles will not exactly match the ones in the example HTML files in the example-markup folder.

Add code comments!

EXTRA CREDIT

Search

- Include a search field for the books listing page. Search should work for all of the following fields:
    - Title
    - Author
    - Genre

    - Year

Note: Searching should be case insensitive and be good for partial matches for strings.

<u>Pagination</u>

- Include pagination for the books listing page.