Assignment 2

CS 7580 | Seattle | Spring 2021

Due date: 2/24/2021, 11:59pm

In this assignment, you will build a prototype note-taking app. Your system will use mock data to populate existing content and allow a user to add new content.

Objectives:

- Learn how to create a new React application with create-react-app
- Use state and props to handle dynamic styling and content
- Learn how to publish your app using GitHub Pages
- Practice applying user-centered UI principles

Setup

Clone your assignment 2 repo

Visit https://github.ccs.neu.edu/cs7580sp21-seattle, then look for a repo called `<your name>_Assignment2`. Your repo includes two folders: `data` and `docs`. `data` contains mock data that you can use in your assignment. The `docs` folder will be used to publish your application. Make sure these folders are preserved when you create your new application.

Create a new React application

These instructions assume you have installed all of the dependencies covered in Week 4. Revisit lab 4 if you are not sure and post on Piazza if you experience difficulties.

1. In terminal, `cd` to the directory that *contains* your local repo. For example, if your local repo is in a directory called `foo_assignment2`, `cd` to the directory that contains `foo_assignment2` rather than `foo_assignment2` itself.
2. Enter `npx create-react-app local_repo_name`, where `local_repo_name` is the name of your local repo. In the previous example, the command would be `npx create-react-app foo_assignment2`. Wait for setup to complete. This will automatically create a template for a React application.
3. Enter `cd local_repo_name`, where `local_repo_name` is the name of your local repo folder, which now contains the React application template.
4. Run `npm start`. This will start localhost and open the application in your browser.
5. Replace the `src` folder with the clean `src` folder available in the lecture-code repo.
6. Inside the `src` folder, add another folder called `components`.
7. Move the `data` folder that was included with your repo inside `src`.
8. Check that the changes are reflected in your browser.

Notes application

Your prototype application will enable users to store notes organized by project. You can think of the app as a very simplified version of OneNote, Evernote, or Google Keep.

The application will support two different types of note: Text and Link:

- Text notes have a title and content, where the content is a string of any length. Your application is *not* expected to handle formatting e.g. bold text, lists etc, but you should be able to render new lines (\n). To display the newline character (\n) as white space, set the display element's (CSS) `white-space` property to `pre-wrap`.
- Links have a URL and, optionally, display text. If display text is not provided, use the URL instead. When the Link is rendered, the user should be able to click on the text to open the URL in a new window/tab. Here are a couple of examples: (1) Northeastern homepage; (2) https://www.northeastern.edu/. In example 1, the display text is "Northeastern homepage" and the URL is "https://www.northeastern.edu/". In example 2, the URL is also used as the display text.

Notes are organized by project, where a project is a user-defined string. Take a moment to review `data.js`, the mock data file* provided with your Assignment 2 repo. The default export from the file is an object called `data`. The top level keys are project names and their values are arrays of individual note objects. The note objects have the following properties:

- `type` - indicating whether the note is Text or a Link
- `text` - the content of a Text note or the display text of a Link
- `title` - the title of a Text note
- `url` - the URL of a Link

You will see that one of the projects in the default object is called "no project", saved as a constant, `NO_PROJECT`. This project name can be used to store notes that a user adds without specifying a project.

**The mock data file is provided to help you get started. You may use it as is or edit it. See Q&A below for more guidance.*

The UI

The layout and style of the UI is up to you but your application must support the following:

- When the UI is loaded, all saved notes (i.e. the mock data) should be visible. You may choose to show the full contents of each note or just a preview.
- It should be easy for a user to tell which project a note belongs to.
- Users can select a project to view only the notes belonging to that project.
- Users can add a new note to any project (a named project or "no project"). The user will need to specify which type of note they are creating, Text or Link. Your UI will need to prompt them to complete the appropriate fields for the note type. New notes added

during a user's session should be visible in the UI for the duration of the session but you are not expected to permanently save new content (i.e. you don't have to write to the mock data file itself).
- Users can create a new project. Once a project has been created, the user should be able to add notes to that project. New projects added during a user's session should be visible in the UI for the duration of the session but you are not expected to permanently save new content.
- A new Link must have a URL in order to be saved. You do not need to verify that the text supplied by the user is a valid URL.
- For a new Text note, either the title or the content text can be empty but not both. You can decide how to handle empty fields.
- Provide appropriate feedback and constraints to guide the user e.g. if a Link note is missing a URL, notify the user and don't save the note.

You do not need to support deletion or editing of notes/projects, or moving notes from one project to another.

Component architecture

It is up to you to decide how to split the UI into components but here is some guidance:

- Each component should have a single responsibility—small components that do one thing are better than big components with a lot of conditional rendering.
- If you find yourself repeating JSX code within a component, split the repeated code into its own component.
- It is better to have lots of small, reusable components that can be composed into a range of complex views than one or two large components.

Q & A

How do I use the mock data?

Import the `data` object into `App` (or any component), just like you would a component. Because `data` is the default export, it can be imported using the file name without curly braces e.g. `import data from "../data/data";`

The constants can be imported using their name in curly braces e.g. `import {NOTE_TYPE} from "../data/data";`

To import the default object *and* constants, separate them with commas in the import statement e.g. `import data, {NOTE_TYPE} from "../data/data";`

How do I change the view to only render components from only one project?

You will need to use component state and conditional rendering. Consult the docs and Lab 4. Javascript's map() function can be used to render an array of data (see Lab 4). A common mistake made by new React developers is to try to use the map() function to iterate through a Javascript Object. See this Stack Overflow post for a description of the problem and a solution.

Suggested progression on this assignment:
In the first week:

- Get comfortable with creating a React application.
- Install and get familiar with Bootstrap 5.
- Plan the design of your application—visual and component structure.
- Create your core components.
- Use the mock data to dynamically render and populate your UI components.

In the second week:

- Add the form(s) that the user will use to create new notes/projects.
- Implement feedback and constraints on those forms.
- Test and refine your design.
- Practice building and publishing your application (see Submission below).

Requirements

Visual style

- It should be clear that your application has had some style applied - you're using defined fonts and colors, elements are neatly aligned, appropriate headings and text formatting is used.
- Make sure your application is fully responsive i.e. when viewed on a small screen, all content should fit in the viewport width and your visitors should not have to scroll left and right. You are not required to use Bootstrap but it's usually the most reliable approach.
- Add your own CSS to styles.css as appropriate.
- All clickable elements should have hover and selected styles, if applicable.
- Form inputs should be the appropriate type for the job e.g. radio buttons or a select for choosing a single option from pre-populated choices.

Code style

- **HTML**
    - All content should be properly enclosed in HTML tags (or components)--no text floating without a p, h1, etc…
    - All nested elements should be indented for readability.
    - All HTML tags should be lowercase.
    - All images must have alt text and the given alt text should be informative. Remember that the goal of the alt text is to describe the image contents for someone who can't see it.
- **CSS**
    - In your CSS files, keep document level styles (e.g. body, p, a) selectors at the top with more specific selectors (e.g. class names, hover states) at the bottom.
    - CSS class/id names should be meaningful.
- **React**

    - All component names should be TitleCase
    - All attribute names should be camelCase
    - Only define one component per file
    - All component JS files should be in your components folder.
- You do not need to document your code, although it's a good habit to foster.

Accessibility

- 10% of your score will be determined by the Chrome Lighthouse extension's accessibility audit. Make use of this tool during testing!

Submission

Step 1: Set your homepage URL

Set your project's Github Pages homepage URL in `package.json`. Your URL should be `https://pages.github.ccs.neu.edu/cs7580sp21-seattle/YOUR_REPO_NAME`. See the how-to on Canvas for more details.

Step 2: Build and publish your application

Your repo is set up to use GitHub Pages. Anything placed in the `docs` folder in the master branch of your repo will be accessible as a public web page. Carry out the following steps to build your React application to a static web site that can be hosted on GitHub Pages.

1. In terminal, from your application's root folder, enter `npm run build` and wait a few seconds for the process to finish. The deployable version of your application is now in a folder named `build`.
2. Copy and paste the contents of the `build` folder into the `docs` folder. Commit and push your changes to your master branch. Your application should now be available at your repo's GitHub Pages URL, `https://pages.github.ccs.neu.edu/cs7580sp21-seattle/YOUR_REPO_NAME`. You can add this link to your portfolio site (assignment 1) if you wish.

Step 3: Release your source code

Push your files to your GitHub repo and create a release. To create the release, go to your repo in your browser, click on the "releases" tab and click "create a new release". In the target menu, select your development branch. Enter "Assignment2" as the tag and release title, then click "Publish release".