# Project 1 : Basic ReactJS

## Description

Project 1 consist of a frontend ReactJS project. Make sure to have:

1. multiple components that interact via passing props and functions.
2. internal logic and functions that controls and manipulates the data of your application.
3. the useState() for both the state variable and the state set function.
4. Use of multiple views (pages), and the logic to toggle between multiple views.

# Project Steps

1. Write what each view on your app should show the user, what data it will need access to, and what actions the user can inact.
2. Diagram out the flow of data between each view. Is there certain components that don't need certain data? Make sure to only pass down data that is useful to children components. Is there any child component that needs to pass data up via function to a parent component?
3. Now that you have a feeling for what each view does you can start creating wireframes.Wireframes are typically done on graph paper, but even a blank or lined page will work. There are a lot of digital tools for facilitating wireframes and prototypes, but I find by hand first in pencil allows for a more direct tactile conneciton to your brain, and quick changes if there are concepts that aren't working. The art doesn't need to look good as long as you (and later other developers) can understand the meaning behind them.
4. Now that you have a good understanding of the flow of data, the components, and the views, you can create a list of tasks. My preferred tool for this is Trello, but there is a ton of tools for task tracking out there, and you can even do it by hand. Create a list for each view, and put the tasks under that section. Go through your list of tasks and move any that have dependency on other tasks to the end of the list.
5. Start coding! Now is the time to boot up that VSCode, write out your npx create-react-appmy-app command, and get started! Layout your file structure first, and then start creating components. As you code, if something doesn't feel right, it's ok to change it. Refactoring code and design is part of the development process.

## Rubric

| **Criteria** | **Beginning** | **Meeting** | **Exceeding** |
|---|---|---|---|
| **Design** | Fails to have a proper wireframe design, or any design. No documentation provided. | Some design documents are present, at the very least a wireframe. There is some documentation on project structure. | A well thought out and planned design is given wit ha wireframe as well as other design models. There is clear and precise documentation for each page of the application/website. |
| **Variables** | Fails to use proper syntax in declaring variables. Variable names are vague and hold little meaning. | Variables are declared using proper syntax. Variable names hold some meaning. | Variables are declared with proper syntax. Variables are declared only when needed. Variable names give a great amount of insight in their use. |
| **Functions** | Fails to use proper syntax in declaring functions. Function names are vague with little meaning. Functionality is focused on vague tasks, or else no task at all, and not a single specific task. | Functions are declared using `arrow function` syntax, and properly structured using the correct syntax. Functionality is relatively well focused on a single task. | Functions are declared using `arrow function` syntax, and properly structured using the correct syntax. Quick returns are placed where appropriate. Functionality is very focused on a single task. |
| **Components** | Fails to use proper syntax in declaring components. Components are not properly organized inside of a folder structure. | Components are properly declared. Components are properly organized in a folder structure. Componetized elements inforce the DRY method. | Components are properly declared in clear and concise syntax. Components are well organized in folder in properly organized in a folder structure. Componetized elements inforce the DRY method, and help assist in the overall structure of the application. |

| Criteria | Beginning | Meeting | Exceeding |
|---|---|---|---|
| **Props** | Fails to use any props in components, or fails to properly use props in components. | Props are properly declared, and used where appropriate in varies components. | Props are properly declared using well named keys and values, and used in places that assist in the overall memory footprint of the application. Props are properly validated using the `PropTypes` library, and default prop values are given for each prop. |
| **React Router** | Fails to use React Router in the application, or fails to properly use React Router in the correct way. | React Router implemented correctly and assists in some of the flow in the application. | React Router is properly implemented, with care to the URL names given to the router. The use of React Router helps the overall flow of the application. |
| **State** | Fails to use state in components, or fails to use state in components correctly. | State is used in components correctly and where appropriate. State names are clear and concise to their meaning. | State used in components correctly and where appropriate. State names are clear and concise to their meaning. State is declared in the components that keeps the memory footprint of the application at its lowest amount. |