

React JS Class

What Will I Need?



Installation

1. Install required software.
 - a. node js v12.13.1+
 - b. npm v6.13.4+
 - c. Visual Studio Code
 - d. (Optional) In VS Code, add the following extension:
 - i. **React Food Truck**Which includes, among other extensions:
 - **ESLint; Prettier; Simple React Snippets**

Q: What is the **ESLint** extension used for?

2. Create your first React APP.
Tip: Use [create-react-app](#)
3. Explore the created folder structure and analyze the file **package.json**.
4. Commands

Install dependencies

`npm install` OR `yarn install`

Start web application server

`npm start` OR `yarn start`

Note: Web application will be available at <http://localhost:3000/>

Warm Up Exercise

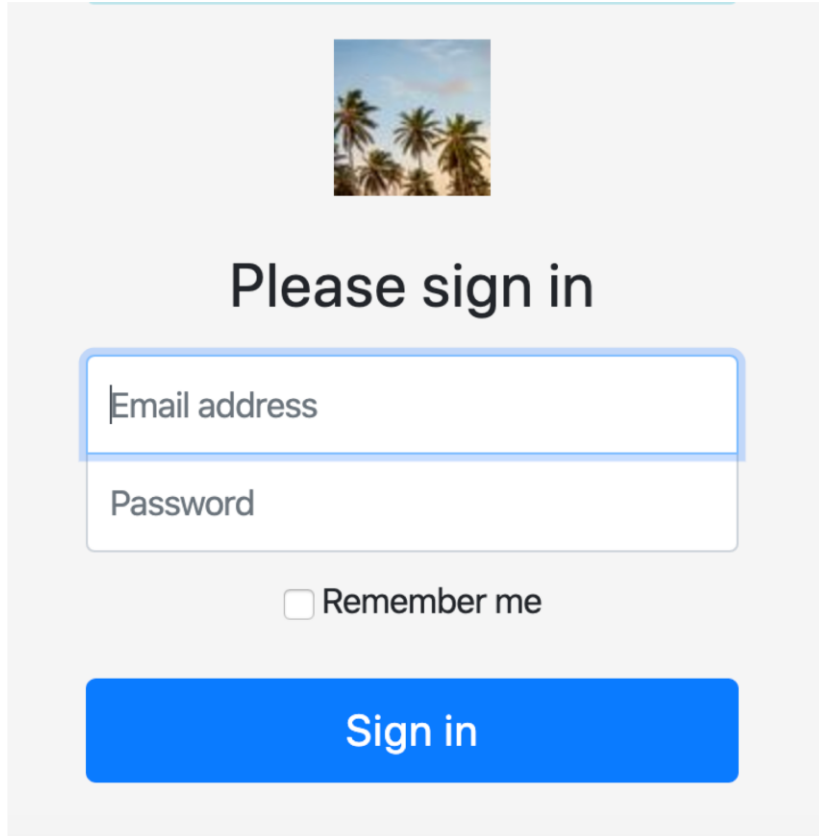
1. Create a simple react component called "HelloWorld" that receives a name as **prop** and displays a simple message:

"Hello World! My name is [your name] and this is my first React App!"
2. **Q:** What are Design Systems and what are their advantages? Examples:
 - a. [Bootstrap Design Systems](#)
 - b. [Material-UI](#)

Project

Build a Web application using **React**, with a **login form** and after that, a page to **search for GitHub Repositories**, using public GitHub API, with **pagination**.

1. Choose a Design System for the Project. In the following steps, you should use appropriate components for each scenario.
2. Create a stateful component for Sign in page. Example:



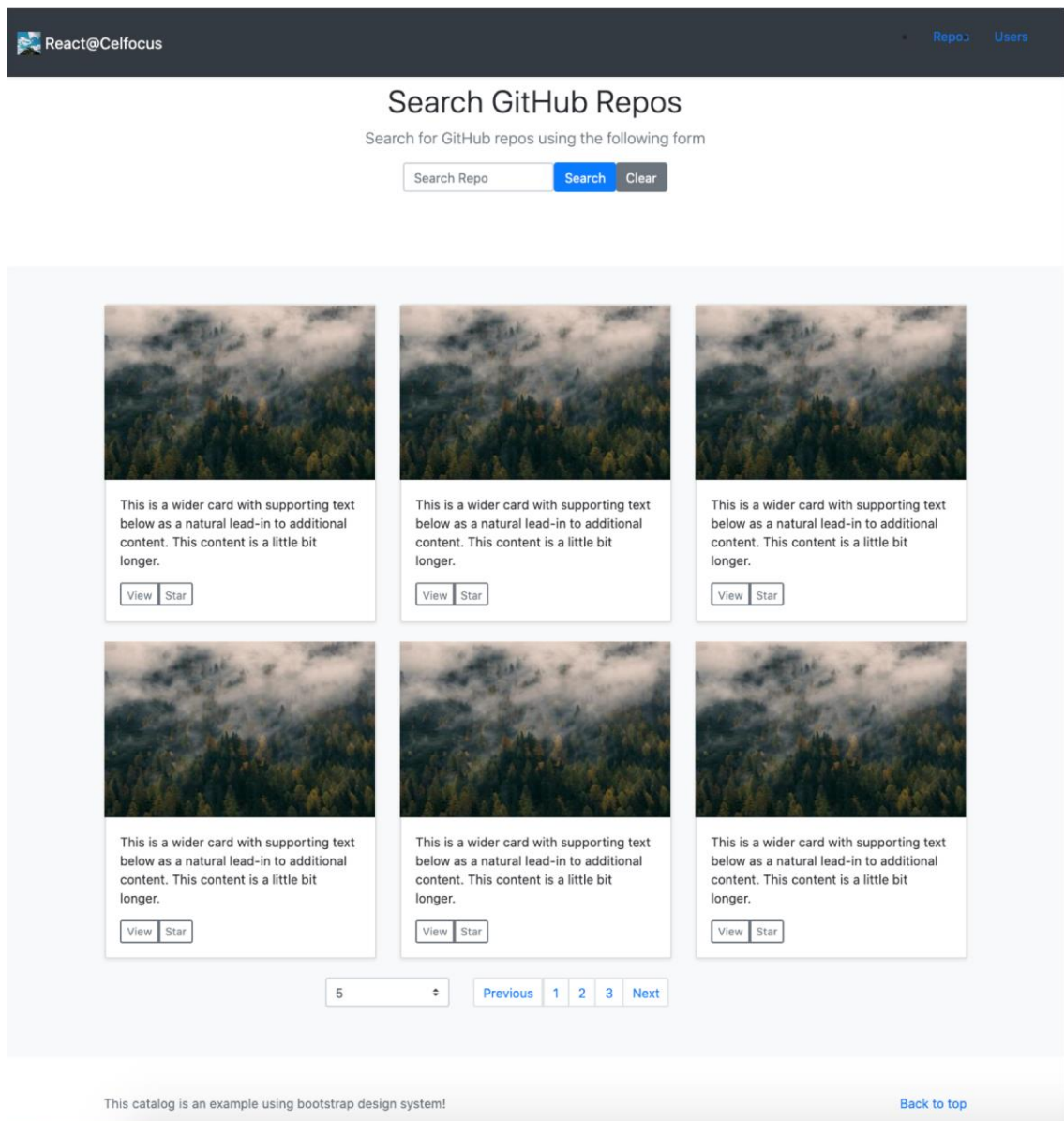
Tip: Choose between using **Class Components** or **Functional Components**.

Note: Make a decision on your folder structure.

3. **Q:** Class vs Functional Component. Which one did you choose and why?
4. Add validation to the Form. Could be a simple validation, like the number of characters of the email and password).

Note: You will not be implementing a real sign in. No need for backend communication. Just to simulate the session after sign in form being submitted, you can store user information at **local storage**.
5. Implement a **Loading** mechanism when submitting the form (simulate a delay of 2 seconds).
6. Show an **Alert** message, on top of Sign in form, when the form is invalid (use Design Systems)

7. Create a stateful component for the Github Repositories page. Example:



Tip: Skip the change page logic and change your entry component from the Sign In page to this newly created Repository Catalog page.

8. Use **GitHub Api** for Repository search:
 - a. Search Repositories API documentation [here](#);
 - b. Example of Repositories Search Endpoint:
https://api.github.com/search/repositories?q=tetris&per_page=9
 - c. Pagination API documentation [here](#);
 - d. Search Users API documentation [here](#);
 - e. Example of Users Search Endpoint:
https://api.github.com/search/users?q=joao&per_page=5;
9. Create search bar component.

10. Fetch repositories using the Github Api and show a list of repository names to see the results. Do not forget to use **key prop**.

Note: You could use **fetch** to consume GitHub API, or add another library (like **axios**) if you prefer.

11. Create a Card component to display the list of repositories. A Card should have:
- Profile Image
 - Title
 - Subtitle
 - Description
 - Two buttons: “View” and “Star”.



12. Create a Modal component, to show a single repository detail - it opens when you click on the “View” button. It contains the same information as the Card, but bigger and with more details (you choose).
13. Add the “Star” button behavior: should work as an “Add to Favorites” - Each card that is *starred* is marked as such.
14. Add pagination to the Repos Page.
15. So far, you probably have been changing our entry component to your Application manually. This cannot happen in a real project. The user needs a dynamic Application.

What we want to have is the following use cases:

- App -> User is logged in -> Repos Page

- App -> User is not logged in -> Sign In -> User successfully signs in -> Repos Page
- ➔ Try to implement this Authentication logic, making use of the **local storage** and **conditional rendering**.

16. Re-Implement the Authentication logic, using React's **Context Api**. Apply the Context Api to connect and share user session among different components that need it. The current page should update accordingly to application state in the Context.

Note: You could also use a React Router to help you change pages. The purpose of this lesson is to show you one use case for Context Api sharing data among different components.

After that you could try **react-router** to add a more sophisticated navigation system.

17. **Q: Context API vs Redux.**

- a. Which one should we use and when?
- b. What is **Redux** and what are its advantages?

18. Use [Jest](#) and Enzyme to Test some of your components:

- a. Start with a simple component, like a Button, or the Search Bar.
- b. Add tests to your Sign In page.