**Build a React application from scratch -** [Saumya Jain](https://medium.com/@jsaumya988?source=post_page-----6a52e4bebc1d--------------------------------) Mar 14, 2022

React Single Page Application

**What is a Single Page Application?**

[MDN](https://developer.mozilla.org/en-US/docs/Glossary/SPA) provides a very good description:

*An SPA (Single-page application) is a web app implementation that loads only a single web document, and then updates the body content of that single document via JavaScript APIs such as [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest" \t "_blank) and*[*Fetch*](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)*when different content is to be shown.*

*This therefore allows users to use websites without loading whole new pages from the server, which can result in performance gains and a more dynamic experience, with some tradeoff disadvantages such as SEO, more effort required to maintain state, implement navigation, and do meaningful performance monitoring.*

Put simply, SPA allows developers to build websites that load data from multiple sources in the background to build dynamic layouts and allow for high amount of interactivity.

**Creating a new react app**

Make sure you have npx installed in your system. To verify type the following command in the terminal,

npx  
// or  
node -v  
npm -v

If the output is “command not found”, install npm and npx from — [Download NPM](https://docs.npmjs.com/downloading-and-installing-node-js-and-npm)

**NPM**

Node Package Manager or NPM is world’s largest software registry. Developers can share and borrow packages. It stores standalone tools for faster integration and updates.

Coming back to building our app, first we need to create a basic react app. To do so, use the commands:

npx create-react-app my-app  
cd my-app  
npm start

*Our web-app is up and running at —*[*http://localhost:3000/*](http://localhost:3000/)*.*

Graphical user interface

Description automatically generated

Create react app output

**Understanding the application**

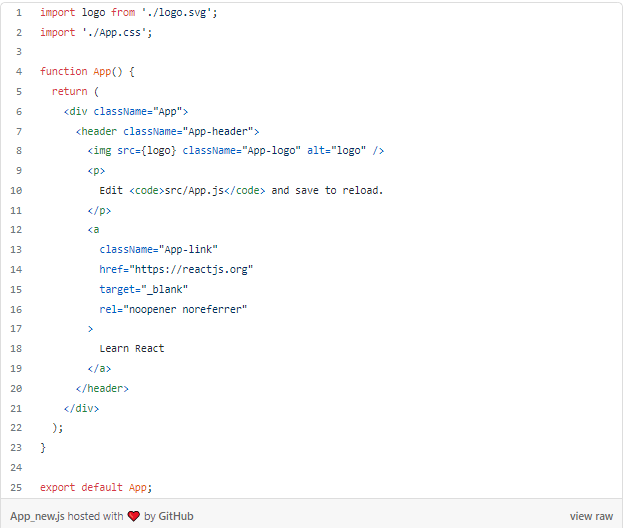
Before we start building the application, let’s understand how the code works.



Let’s start at index.js, as we can see in line 7, React uses [ReactDOM](https://reactjs.org/docs/react-dom.html" \t "_blank), which is used to render “data” to the browser. This “data” is nothing but JSX (Javascript + HTML) which is returned by the file, App.js (on which we will work on later). This is then, served in the div element with the id “**root**” in the HTML file located in the public folder as shown. This can be imagined with “root” element as an empty container which can be filled with anything (we will be filling it with our SPA)

// public/index.html  
<div id="root"></div>

The [React.StrictMode](https://reactjs.org/docs/strict-mode.html" \t "_blank) tag in line 8, is optional. It is used for highlighting potential problems in an application. It only runs in the development mode and has no impact in the production build.



This is a basic App.js file that is created by **create-react-app**which returns JSX from the function App.

Now that we understand the working of React, we can start building our very first single page application!

**Objective: Build a website for restaurant**

So from the objective, we know that a restaurant will have multiple dishes and people should be able to lookup these dishes, their prices and description. So, we need to display this data on our browser with the help of a layout, I chose the following grid layout:

Graphical user interface

Description automatically generated

Deployed Application

To display our data in grid, we need to install [Semantic UI React](https://react.semantic-ui.com/). Following command can be used to install it.

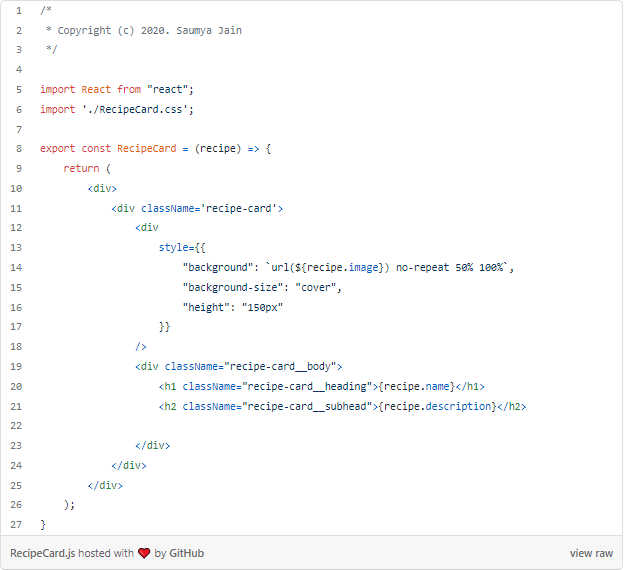
## yarn  
$ yarn add semantic-ui-react semantic-ui-css  
## Or NPM  
$ npm install semantic-ui-react semantic-ui-css

**Step 1: Create a Card**

Let’s create a new file named RecipeCard.js, this file will contain the JSX of recipe card that will be rendered and will take recipe object as a prop, which is of the type-

// This is one of the object given as a response of API call: <https://recipe-json.herokuapp.com/recipes>{  
 id: 123 //id of the recipe  
 name: pizza //name of the recipe  
 description: This consists of a base  
 and toppings included   
 onions, capsicum, etc  
 label: HOT //HOT, Trending, etc  
 category: italian //cuision type  
 price: $45 // price of recipe  
 image: [www.google.com](http://www.google.com/) // URL of recipe  
}

**NOTE:**This file will only contain the description of a single recipe.



The above HTML has 3 div elements each for — Main Card, Recipe Heading and Recipe body, creating a simple card with the following layout —

A pizza with cheese and toppings

Description automatically generated with low confidence

Recipe Card Output

Now that we have our HTML, we need CSS to make it look as shown in the image.

**NOTE:** I have already add the RecipeCard.css even though we haven’t created it yet but even if you run till here, no errors will be shown.

To add our CSS, create a file named RecipeCard.css and add the following code



The above CSS specifies the detailed information of font, card and spacing.

## ****Step 2:**** Getting data

Now that we have our card set up, we need to display some data on the website. For that any API that returns a list of following object will do. For this project, we are using — [**https://recipe-json.herokuapp.com/recipes**](https://recipe-json.herokuapp.com/recipes)

{ id, name, label, description, category, price, image }

*We have our API and know what kind of data would be returned but how to use it?*

To make use of the data returned by the API we need a function that calls the API and returns the response. To make API calls in React, we use **fetch**in the following way —



## ****Step 3:**** Generate the grid

We notice from the above function returns a list of recipes and hard-coding the cards for each recipe would be tedious and maybe not the right approach. We need a way to generate cards dynamically.

To do so, we first need to create a LandingPage. It will be the first page every viewer visits. It will have our grid of RecipeCards which will be generated dynamically.



The above code uses, Semantic UI React which was installed earlier, RecipeCard which stores details of a single recipe and recipes which are fetched by the API

Before we understand how the above code is working, we will take a small detour (I promise everything will make sense).

**Hooks** — They are lifecycle methods that can be used inside a function. They were introduced in React 16. Hooks allow you to reuse stateful logic without changing your component hierarchy.

*Confused? Okay, let’s start from the beginning…*

**Lifecycle Methods**

These methods, as the name suggests, maintains the lifecycle of a component. Lifecycle of a component means — its state.

Let us take an example of an input field, it is not static. A user can type some value that needs to be updated. From this we can see that lifecycle of input field will be: When it is shown on the screen, when the value updates, when it is not on the screen anymore. These states are named in react as: ComponentDidMount, ComponentDidUpdate, ComponentWillUnmount respectively.

The code for such example is as follows:

Text

Description automatically generated

Text

Description automatically generated

Hooks example

We can see from the left image (or experience :P) that classes uses a lot of boilerplate code and using licecycle method in class based components can introduce “wrapper hell”. All of these reasons (and more) led React developers introduce **HOOKS**(example in right image), they let us use lifeycyle methods inside a function based components. Let us discuss following types of HOOKS —

**useState: Used to update state**

const [name, setName] = React.useState('Saumya')

Here, name is a variable, setName is a function used to update name and ‘Saumya’ is the initial state of the variable, name

**useEffect: Allows you to perform side effects in your components**

Side effects are updating the DOM (Document Object Model), fetching data etc whenever a component is updated.

// useEffect(function, dependency)useEffect(() => {}, []) // equivalent to ComponentDidMount  
 // is called whenever the component comes in DOM  
useEffect(() => {}, JSX) // equvalent to ComponentDidUpdate  
 // is called when the JSX component is updated

Coming back to our code:



const [generatedRecipes, setGeneratedRecipes] = useState([])

generatedRecipes is a list of recipes that will contain all the recipes to be displayed. Initially it is an empty list.

useEffect(() => {  
 getRecipes().then((recipes) => {  
 setGeneratedRecipes(recipes)  
 })  
}, [])

When the component, LandingPage comes into the DOM, this Hook takes effect. It sets the generated recipes to the array of JSON objects returned by out API.

// iterates over recipes and returns a RecipeCard for each recipe  
recipes.map((recipe) => {  
 return <RecipeCard   
 id={recipe.id} // props for RecipeCard  
 {...recipe}  
 />  
});

In the div element with id = grid, we are using the CSS class called “ui grid”, it is defined in the Semantic UI React library. It creates a grid with responsive layout.

*We still see the boring React screen.. Where is our app?*

To see our app changes, we need to replace <App /> in index.js to <LandingPage.js /> because our “root” element is still getting its data returned by App.js(where our code changes are not present). So our index.js will look something like this —



By running the app, we will see something like this —

A picture containing text

Description automatically generated

Application output

**Step 4:** Finishing Touches..

We are alomst there… to complete the app, Header and search bar is missing.

Our Header consists of 2 segments — Restaurant Name and Search Bar

<Segment clearing> // Restaurant Name  
 <Header  
 as='h2'  
 // floated='center'  
 >  
 <Icon  
 name='utensil spoon'  
 />  
 <Header.Content>  
 Sam's Pizzeria  
 <Header.Subheader>  
 Select your space delicacy  
 </Header.Subheader>  
 </Header.Content>  
 </Header>  
</Segment>  
<Segment clearing> // Search Bar  
 <Input  
 size={'large'}  
 icon={  
 <Icon  
 name='search'  
 inverted circular link  
 />  
 }  
 placeholder='Filter...'  
 onChange={// function to be added}  
 />  
</Segment>

The above code gets us the required headers but we still need a function to search from the given recipes..

To do so, we will use **.filter**on the array of recipes and filter out the recipes searched with the help of following code —

({target}) => {  
 setGeneratedRecipes(  
 source.filter(({name}) => {  
 return name  
 .toLowerCase()  
 .includes(target  
 .value  
 .toLowerCase()  
 )  
 })  
 )  
}

In the above function we are filtering out the recipes that contain the input provided by the user and then finally saving it to generatedRecipes.

Lets add css for Headers and background in a file named LandingPage.css, which will look something like this-

So our Landing Page finally looks like this —

By running the app, we will see something like this —

A picture containing text

Description automatically generated

Application output

**Step 4:** Finishing Touches..

We are alomst there… to complete the app, Header and search bar is missing.

Our Header consists of 2 segments — Restaurant Name and Search Bar

<Segment clearing> // Restaurant Name  
 <Header  
 as='h2'  
 // floated='center'  
 >  
 <Icon  
 name='utensil spoon'  
 />  
 <Header.Content>  
 Sam's Pizzeria  
 <Header.Subheader>  
 Select your space delicacy  
 </Header.Subheader>  
 </Header.Content>  
 </Header>  
</Segment>  
<Segment clearing> // Search Bar  
 <Input  
 size={'large'}  
 icon={  
 <Icon  
 name='search'  
 inverted circular link  
 />  
 }  
 placeholder='Filter...'  
 onChange={// function to be added}  
 />  
</Segment>

The above code gets us the required headers but we still need a function to search from the given recipes..

To do so, we will use **.filter**on the array of recipes and filter out the recipes searched with the help of following code —

({target}) => {  
 setGeneratedRecipes(  
 source.filter(({name}) => {  
 return name  
 .toLowerCase()  
 .includes(target  
 .value  
 .toLowerCase()  
 )  
 })  
 )  
}

In the above function we are filtering out the recipes that contain the input provided by the user and then finally saving it to generatedRecipes.

Lets add css for Headers and background in a file named LandingPage.css, which will look something like this-

So our Landing Page finally looks like this —







*We have built our first Single Page Application!*

If we run the application, we would get something like this

Graphical user interface, website

Description automatically generated

Final Application

*Thank you for reading the article!*

Here’s the link to my GitHub repository

[saumyaaajain/HackedRecipe](https://github.com/saumyaaajain/HackedRecipe" \t "_blank)

### **[Contribute to saumyaaajain/HackedRecipe development by creating an account on GitHub.](https://github.com/saumyaaajain/HackedRecipe" \t "_blank)**

[github.com](https://github.com/saumyaaajain/HackedRecipe" \t "_blank)