React Tutorial #1, Boiler-plating to Customizing With Props & Hooks

Welcome to React, one of the most popular libraries supporting JavaScript to create lean, modularized and effective web pages!

The average time to learn React ranges from 1 month to 6 months, ***learn*** meaning become an experienced react developer. So, consider the weeks we will spend building React sample applications, an introduction to thinking in components, and familiarization with the concepts and techniques used in a component based architecture.

Start by installing the ***React developer tools*** on your browser, so you can demonstrate what components have been activated in your application.

Here is what you will be doing in this tutorial:

1) you will use the ***npx*** utility, with ***create-react-app***, to scaffold a boiler-plate React application
2) you will use ***JSX*** to represent a variable on your web page
3) you will create your own ***component*** and add this to the application
4) you will use the ***props*** object to pass data between react components
5) you will use the ***JavaScript Array map method*** to create a list of objects
6) you will use JSX to create a list of components, based on one component declaration
7) you will ***style*** a React component
8) you will use a ***hook*** in React to coordinate changing the state of a variable, with an action in your application

1 : Scaffolding a React Application

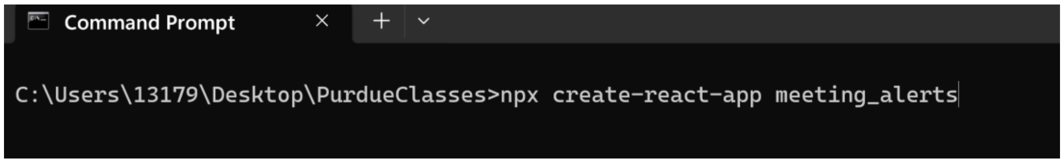
Use the ***npx tool***, to run ***create-react-app*** in the node library. This creates a shell application with React and ReactDom installed, and a package.json file with scripts and dependencies.

Why do we need both React and ReactDOM?

React *handles the logic and structure of the components, while* ***ReactDOM*** *takes care of rendering those components to the web page. This separation of concerns allows React to be used in different environments (like React Native for mobile apps) by swapping out ReactDOM for a different rendering engine.*

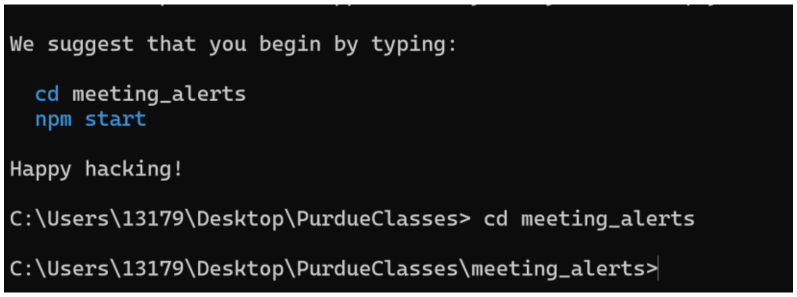
Run npx with create-react-app and the application name. We will build a sample meetings application, so let’s call this meeting_alerts. The folder/application name should use all lower case.

React, Introduction (Scaffold, Using JSX, Creating Components)



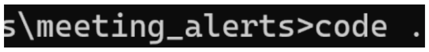
```
C:\Users\13179\Desktop\PurdueClasses>npx create-react-app meeting_alerts
```

When the scaffolding process is complete, navigate to the folder you have created, in this case meeting_alerts.



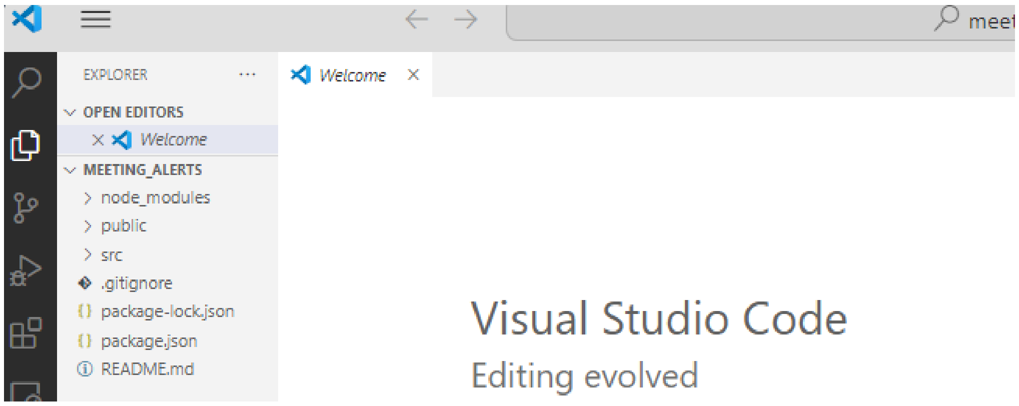
```
We suggest that you begin by typing:

  cd meeting_alerts
  npm start

Happy hacking!

C:\Users\13179\Desktop\PurdueClasses> cd meeting_alerts

C:\Users\13179\Desktop\PurdueClasses\meeting_alerts>
```

Now open the folder with Visual Studio Code.

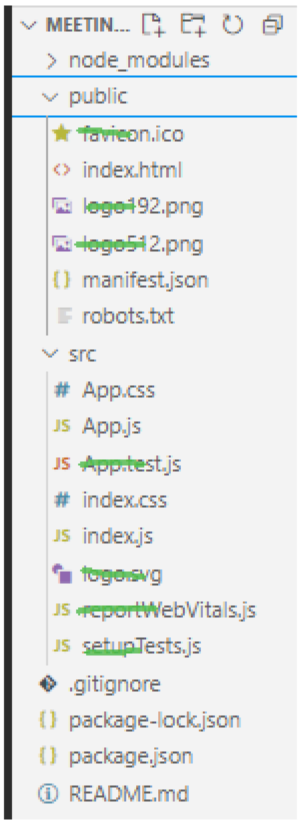


```
s\meeting_alerts>code .
```

This opens up in VSCode.



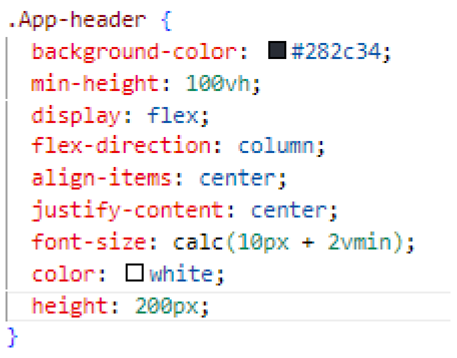
React, Introduction (Scaffold, Using JSX, Creating Components)



MEETIN...
- node_modules
- public
  - ~~favicon~~.ico
  - index.html
  - ~~logo1~~92.png
  - ~~logo51~~2.png
  - manifest.json
  - robots.txt
- src
  - App.css
  - App.js
  - ~~App.test~~.js
  - index.css
  - index.js
  - ~~logo.svg~~
  - ~~reportWebVitals~~.js
  - ~~setupTests~~.js
- .gitignore
- package-lock.json
- package.json
- README.md

... and now you should clear out all the content that you do not need, and resolve any errors in the remaining files after the this content has been removed.

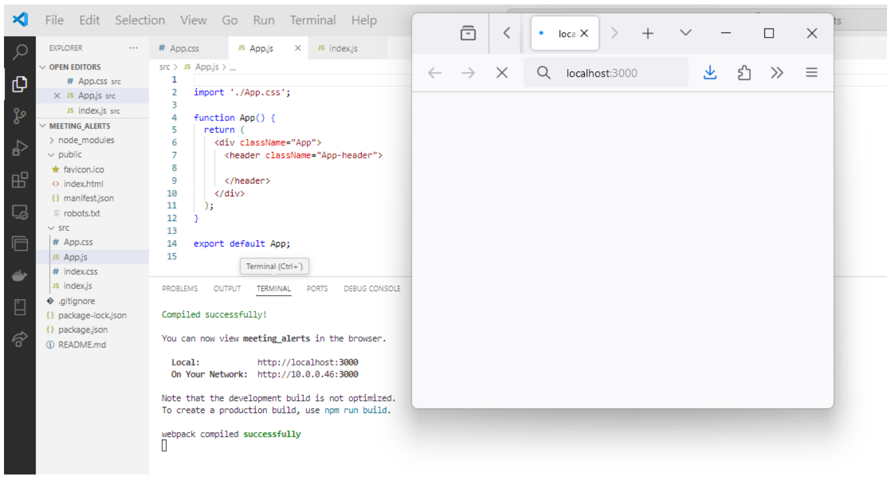
In App.css, give the header a new height, say 200px.



```css
.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
  height: 200px;
}
```

When all has been resolved, you should just see the following,

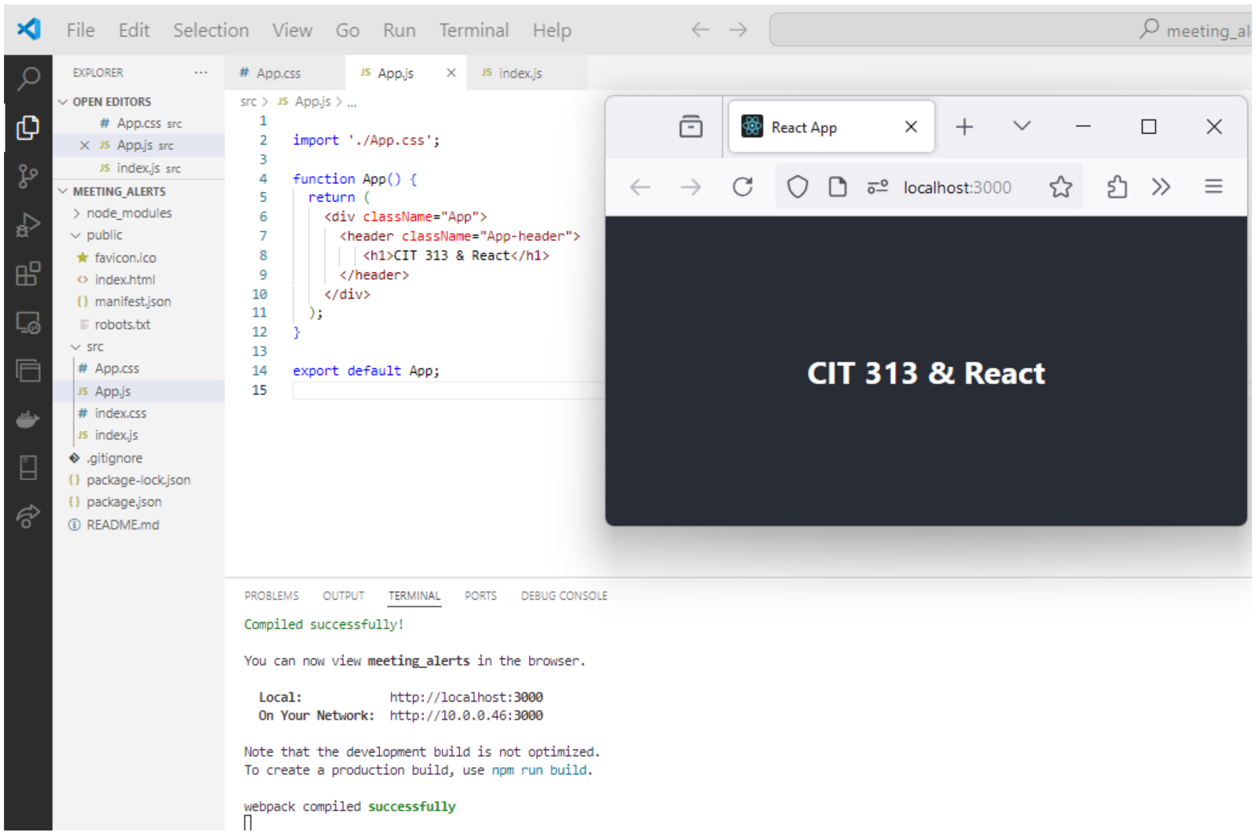
React, Introduction (Scaffold, Using JSX, Creating Components)



```
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">

      </header>
    </div>
  );
}

export default App;
```

```
Compiled successfully!

You can now view meeting_alerts in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://10.0.0.46:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Add a heading to the header,



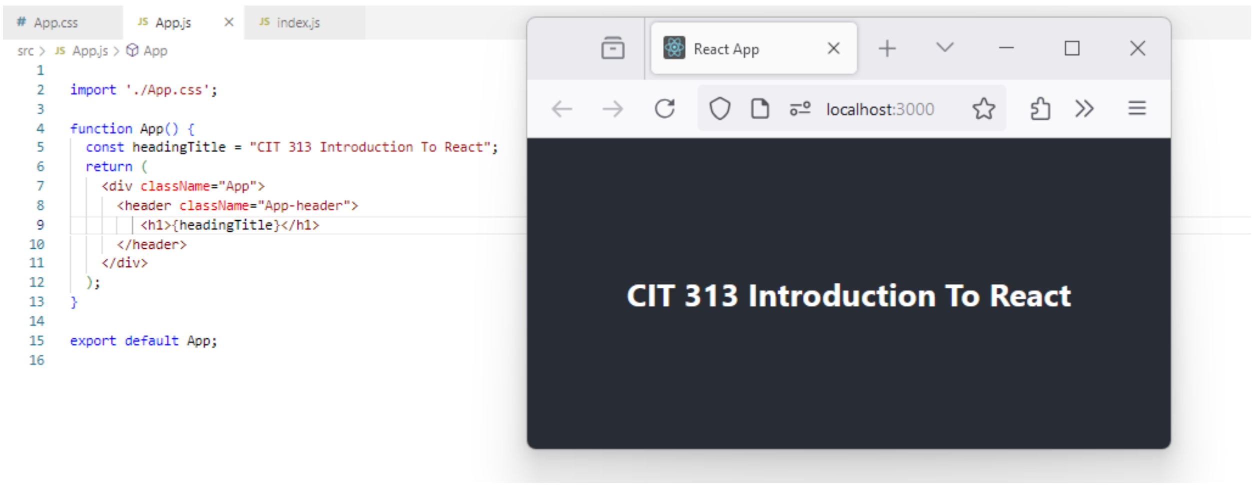
```
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <h1>CIT 313 & React</h1>
      </header>
    </div>
  );
}

export default App;
```

CIT 313 & React

```
Compiled successfully!

You can now view meeting_alerts in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://10.0.0.46:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

2: Using a variable with JSX, Representing JavaScript Content in JSX

Notice the markup being returned by function App? That is ***JSX***. Let's use a variable instead of the hard-coded heading text. The variable is included by using the ***{...} notation***, which brings standard JavaScript into JSX.



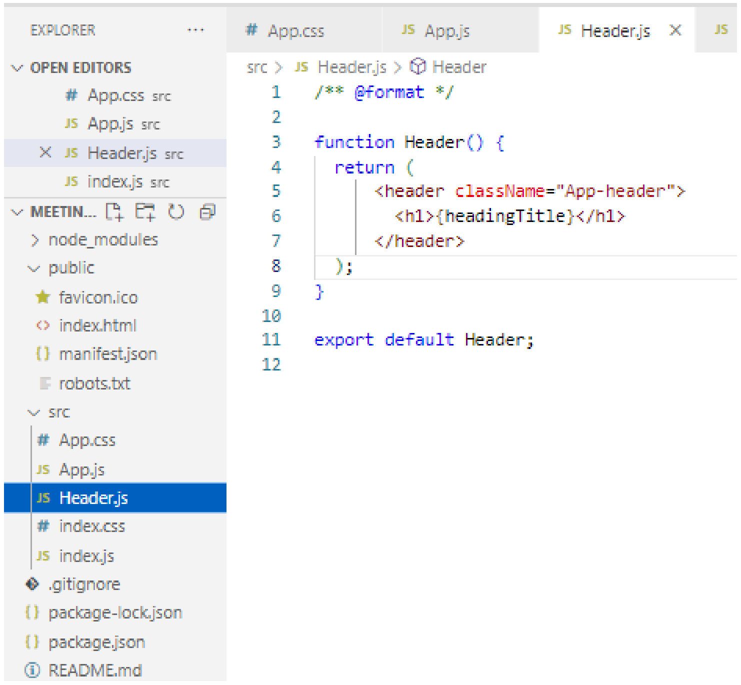
```

import './App.css';

function App() {
  const headingTitle = "CIT 313 Introduction To React";
  return (
    <div className="App">
      <header className="App-header">
        <h1>{headingTitle}</h1>
      </header>
    </div>
  );
}

export default App;

```

CIT 313 Introduction To React

3 : Custom Component

Let's create the header as a component and include it inside the App component.

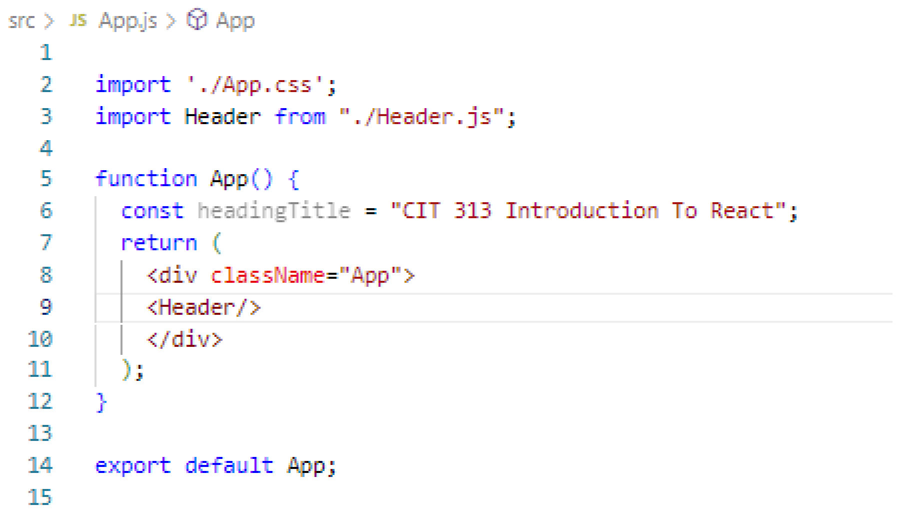
Create a new file, Header.js, and inside this add the content for the Header component, as follows.



```
/** @format */

function Header() {
  return (
    <header className="App-header">
      <h1>{headingTitle}</h1>
    </header>
  );
}

export default Header;

```

React, Introduction (Scaffold, Using JSX, Creating Components)

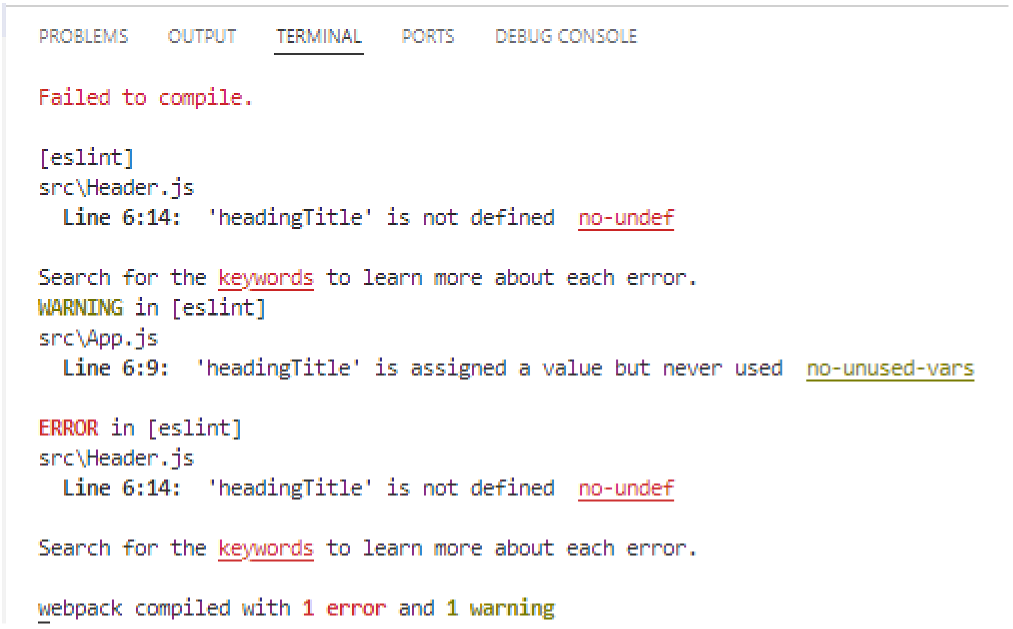
Change the App.js file and component, to include the new Header component.



src > JS App.js > App

```

import './App.css';
import Header from "./Header.js";

function App() {
  const headingTitle = "CIT 313 Introduction To React";
  return (
    <div className="App">
    <Header/>
    </div>
  );
}

export default App;

```

You will notice an error, because the title text is no longer local! We have created a new component, Header, and referenced a variable defined inside App.js.



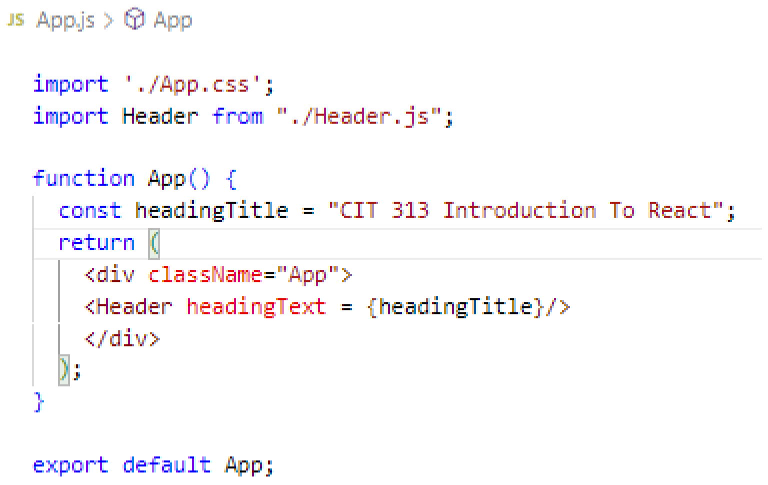
```
Failed to compile.

[eslint]
src\Header.js
  Line 6:14:  'headingTitle' is not defined  no-undef

Search for the keywords to learn more about each error.
WARNING in [eslint]
src\App.js
  Line 6:9:  'headingTitle' is assigned a value but never used  no-unused-vars

ERROR in [eslint]
src\Header.js
  Line 6:14:  'headingTitle' is not defined  no-undef

Search for the keywords to learn more about each error.

webpack compiled with 1 error and 1 warning
```

This will be fixed in the next step.

4: The Props Object - Passing Data between Components

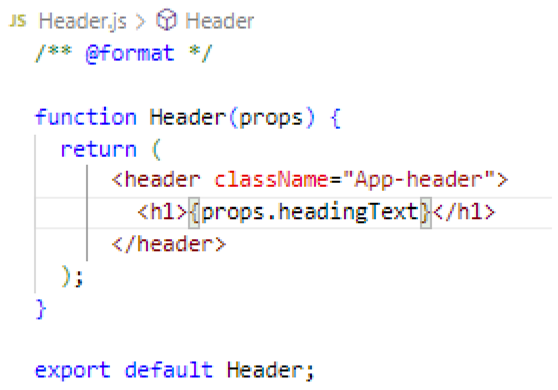
We have defined variable, headingTitle, with the title text for the heading, inside App.js. Yes, this could be moved to Header.js, but let's use the title text as an example of how to pass data between components, as this is definitely required.

Modify App.js to pass the title text to the Header component.



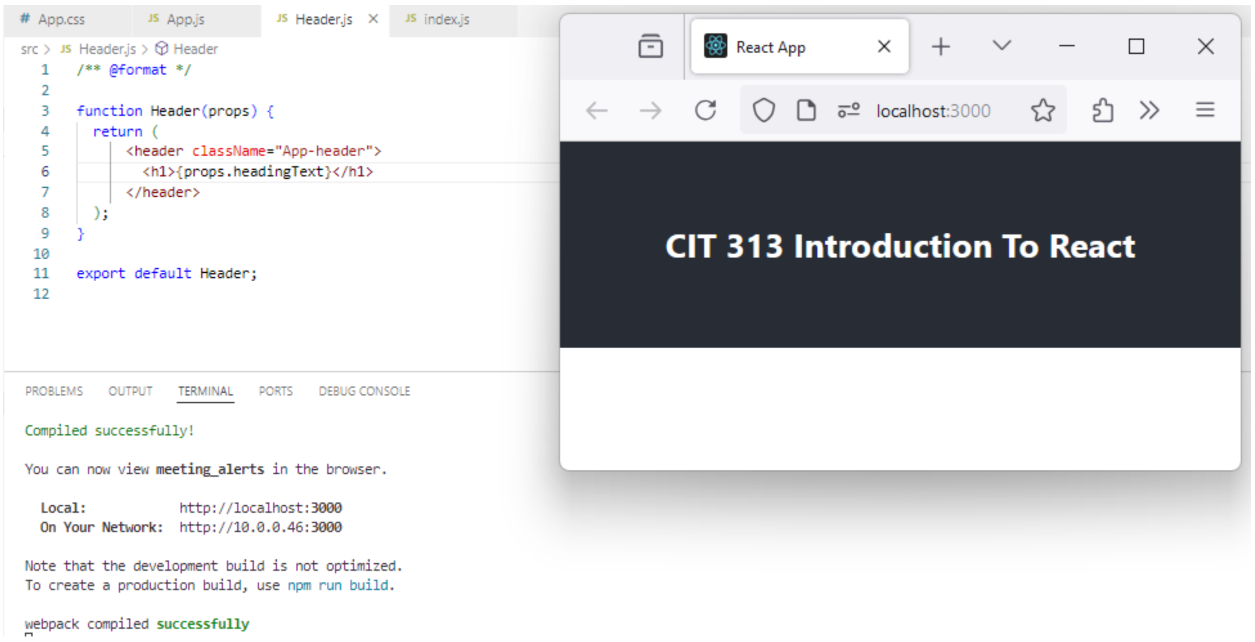
```
import './App.css';
import Header from "./Header.js";

function App() {
  const headingTitle = "CIT 313 Introduction To React";
  return (
    <div className="App">
    <Header headingText = {headingTitle}/>
    </div>
  );
}

export default App;
```

... and modify the Header component to receive this data, through what is termed the ***props object***.



```
/** @format */

function Header(props) {
  return (
      <header className="App-header">
        <h1>{props.headingText}</h1>
      </header>
  );
}

export default Header;
```

Now all the errors are resolved.



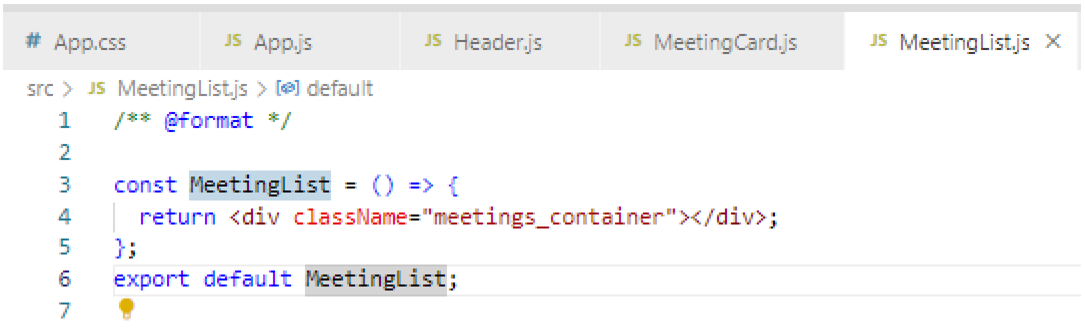
```
/** @format */

function Header(props) {
  return (
    <header className="App-header">
      <h1>{props.headingText}</h1>
    </header>
  );
}

export default Header;
```

```
Compiled successfully!

You can now view meeting_alerts in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://10.0.0.46:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

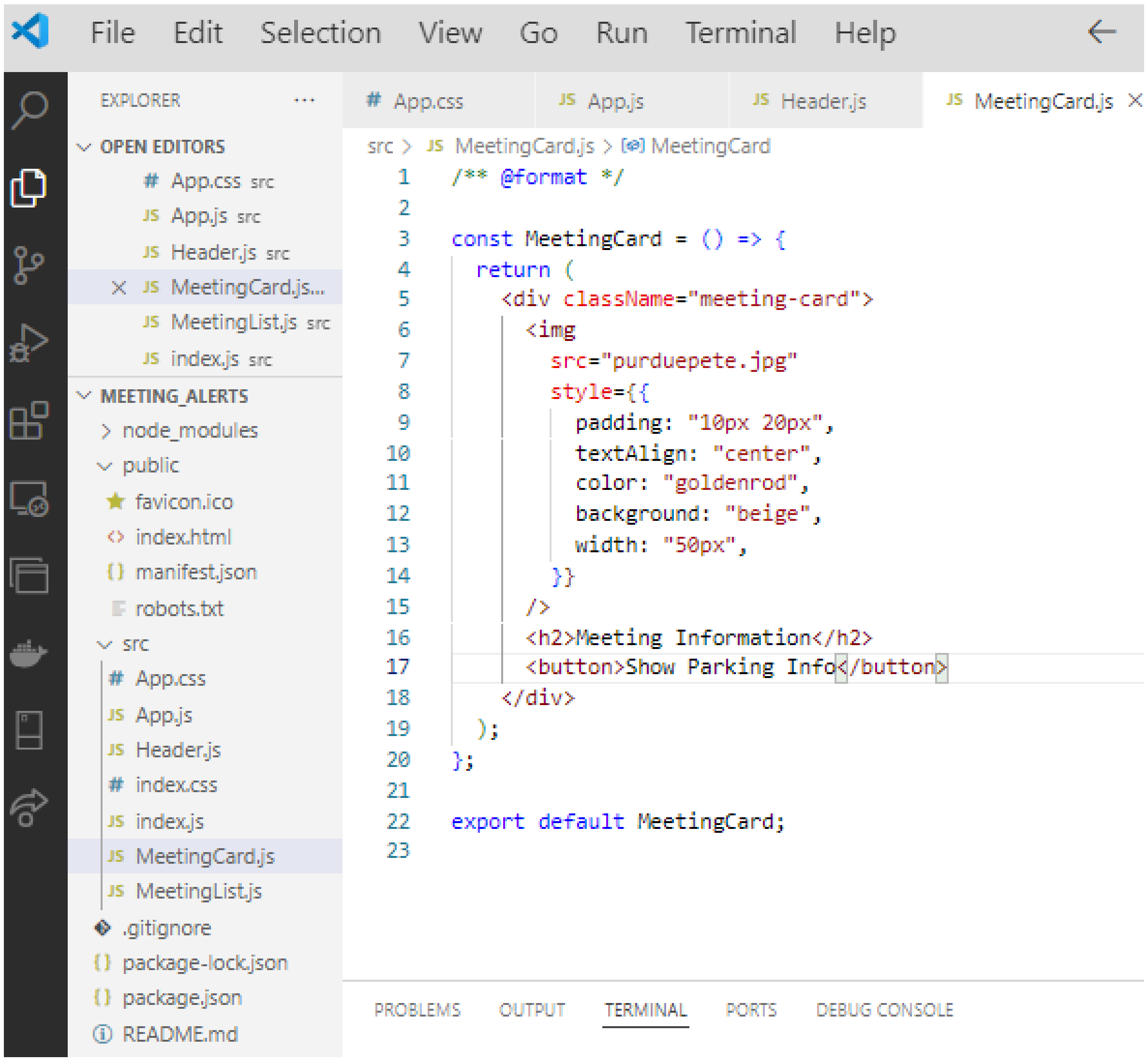
CIT 313 Introduction To React

5: Add Styling

Let’s create 2 new components, a Card , used to represent meeting information, and a container to store multiple such cards, starting with a single card.



```
/** @format */

const MeetingList = () => {
  return <div className="meetings_container"></div>;
};
export default MeetingList;
```

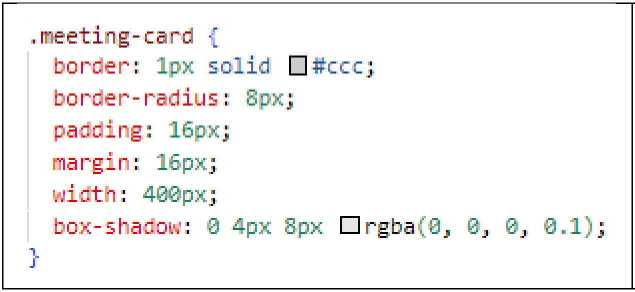


```
/** @format */

const MeetingCard = () => {
  return (
    <div className="meeting-card">
      <img
        src="purduepete.jpg"
        style={{
          padding: "10px 20px",
          textAlign: "center",
          color: "goldenrod",
          background: "beige",
          width: "50px",
        }}
      />
      <h2>Meeting Information</h2>
      <button>Show Parking Info</button>
    </div>
  );
};

export default MeetingCard;
```

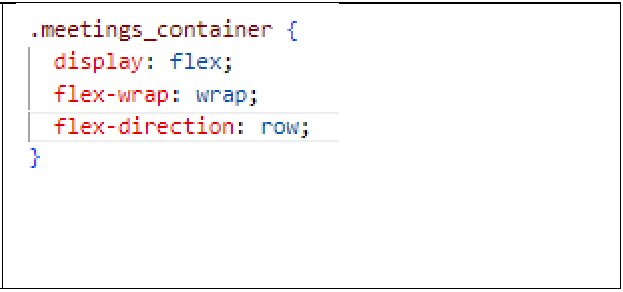
You can use this image, Purdue Pete, and add it to the public folder where it will be accessed by reference. We will deal with the button in a bit!

Note the ***inline styling*** for the MeetingCard component is specified using a style object, { ...}, surrounded by the JSX { ... }. Style rules can be integrated in several ways using react, including the use of a standard style file. Style objects can be created, assigned to variables and applied to elements.

Add the styles for the card and the card container, as ***external styling***, to App.css,

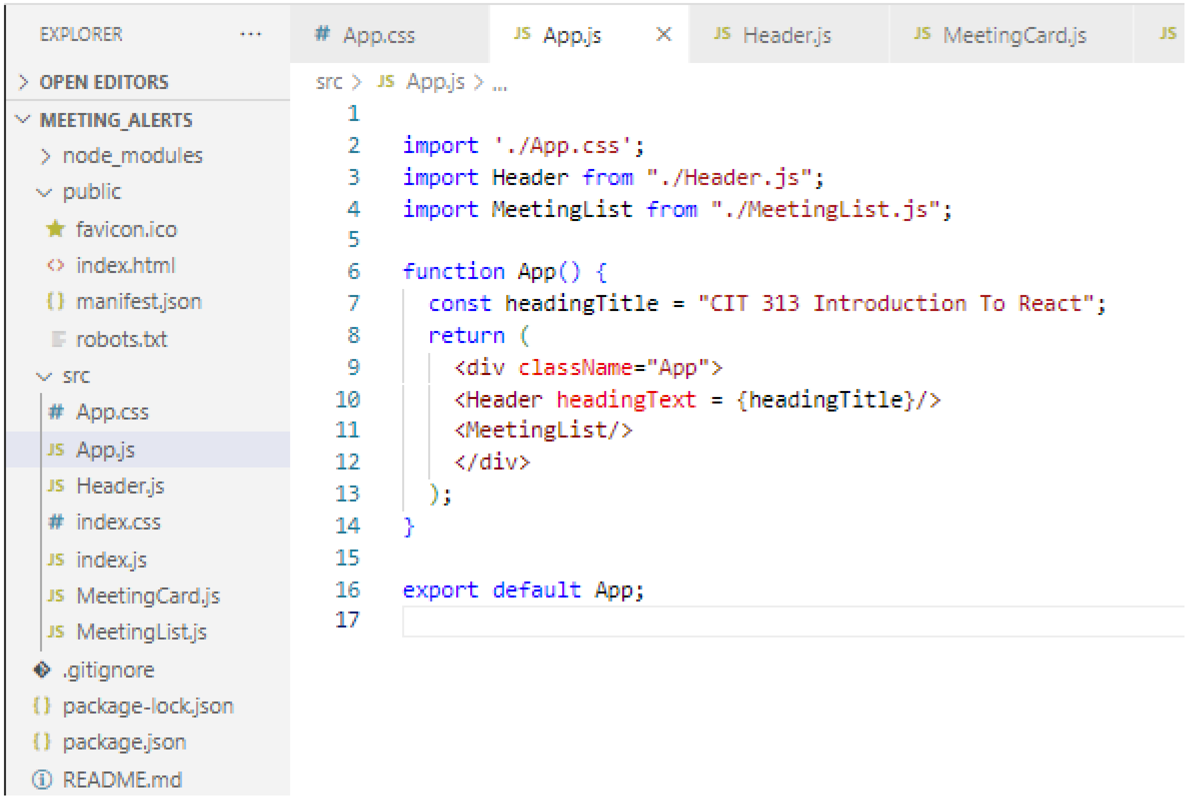


```css
.meeting-card {
  border: 1px solid #ccc;
  border-radius: 8px;
  padding: 16px;
  margin: 16px;
  width: 400px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
```



```css
.meetings_container {
  display: flex;
  flex-wrap: wrap;
  flex-direction: row;
}
```

First, add the element of class meetings_container (the flex container represented by the MeetingList component) to the App component.

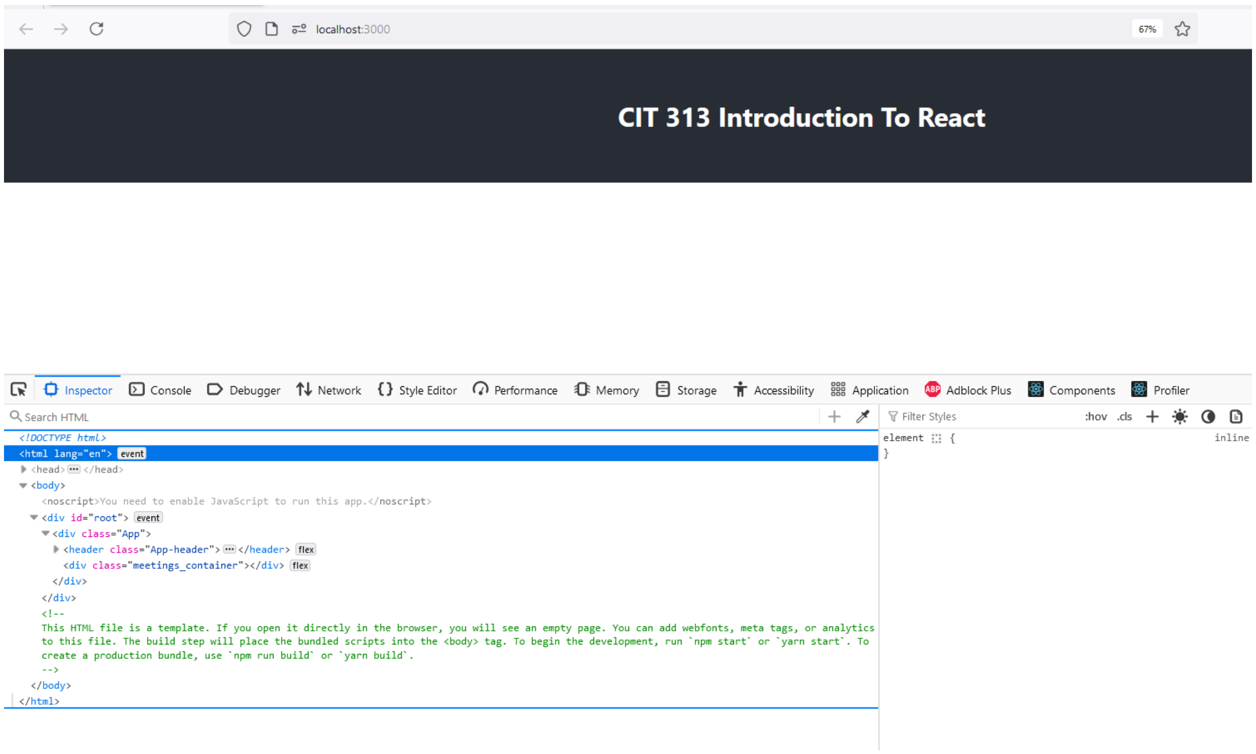


```javascript

import './App.css';
import Header from "./Header.js";
import MeetingList from "./MeetingList.js";

function App() {
  const headingTitle = "CIT 313 Introduction To React";
  return (
    <div className="App">
    <Header headingText = {headingTitle}/>
    <MeetingList/>
    </div>
  );
}

export default App;

```

We are using the React recommended way of adding components in separate and distinct files, so make sure to import the MeetingList component into App.js.

Use inspect to investigate the page structure, and you can see that the element of class meetings_container has been successfully integrated.

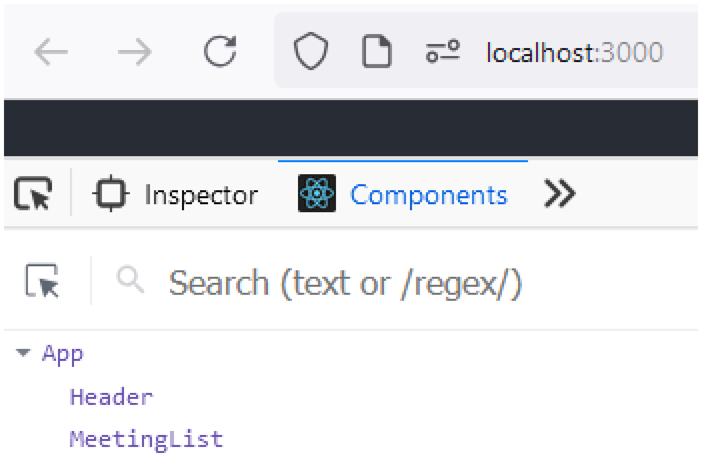
React, Introduction (Scaffold, Using JSX, Creating Components)



```
<!DOCTYPE html>
<html lang="en"> event
  <head> ••• </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"> event
      <div class="App">
        <header class="App-header"> ••• </header> flex
        <div class="meetings_container"></div> flex
      </div>
    </div>
    <!--
    This HTML file is a template. If you open it directly in the browser, you will see an empty page. You can add webfonts, meta tags, or analytics
    to this file. The build step will place the bundled scripts into the <body> tag. To begin the development, run `npm start` or `yarn start`. To
    create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

```
element {
}
```

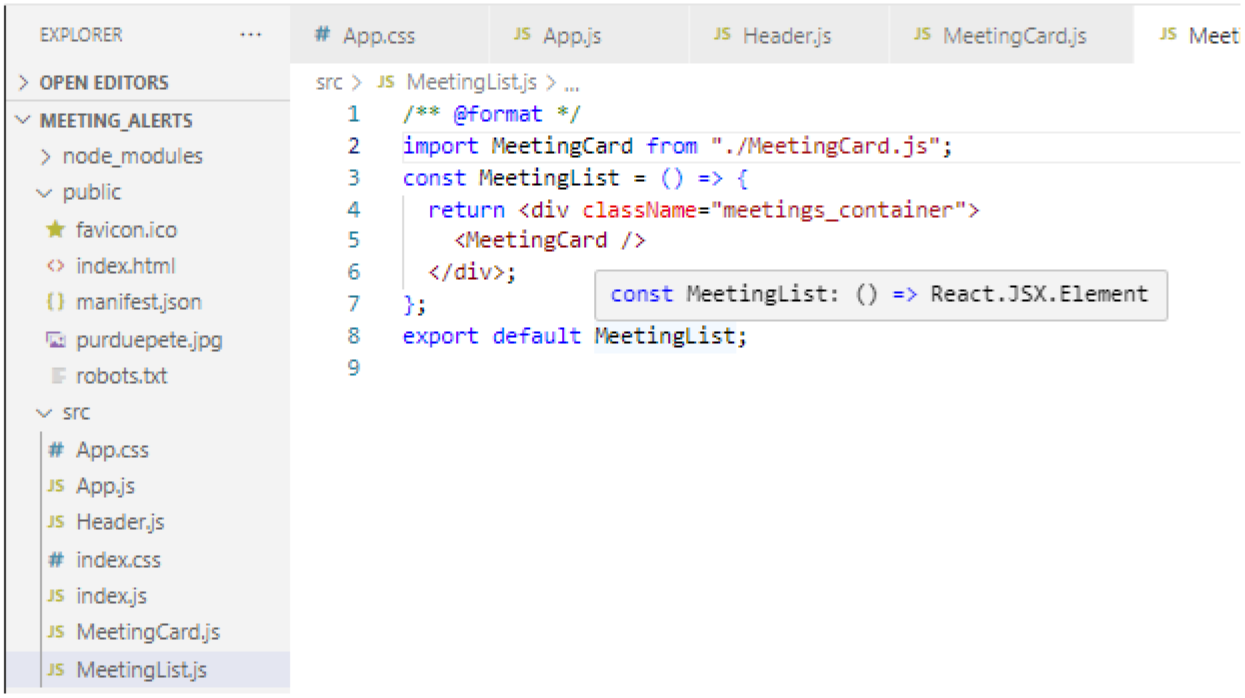
The Components tab shows the App component, the Header component and the MeetingList component.



```
App
  Header
  MeetingList
```

Now add a single Card component to the MeetingList component.

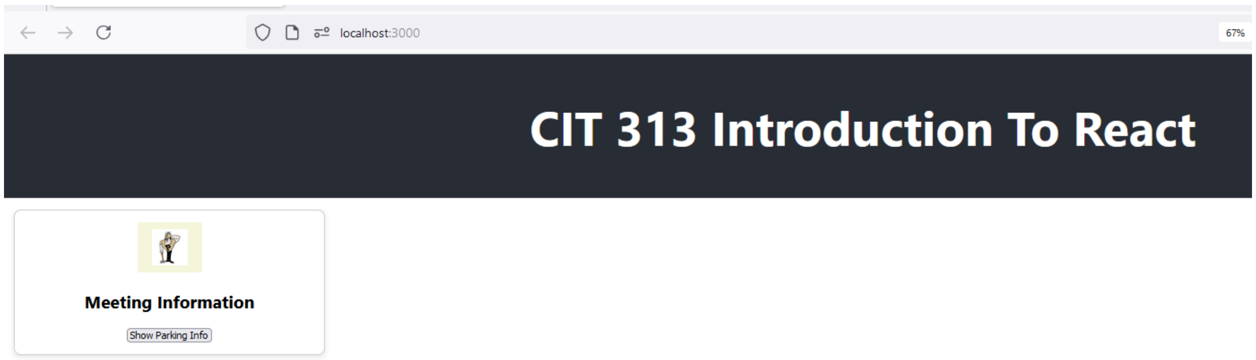
React, Introduction (Scaffold, Using JSX, Creating Components)



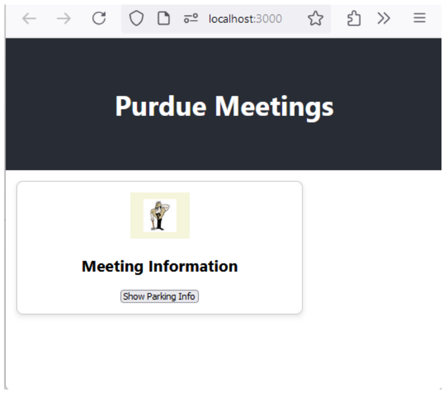
```
/** @format */
import MeetingCard from "./MeetingCard.js";
const MeetingList = () => {
  return <div className="meetings_container">
    <MeetingCard />
  </div>;
};
export default MeetingList;
```

```
const MeetingList: () => React.JSX.Element
```

... and you should see this,

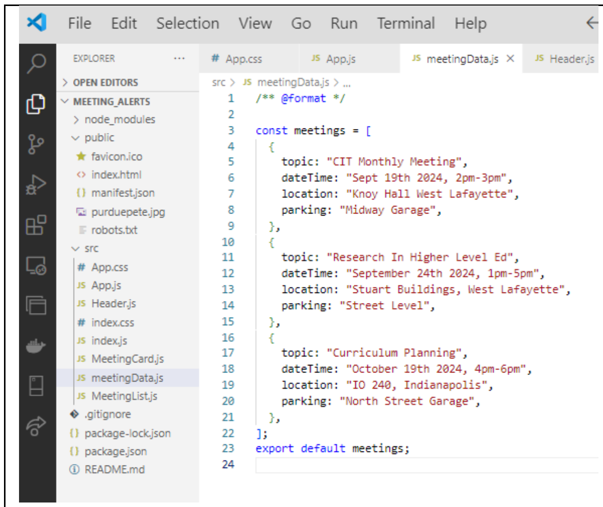


Now, let's change the title text before we move to the next step.
Do you know how?



6: Using Iteration in JSX

We have more than 1 meeting(!), so let's represent the meetings as an array of JavaScript objects, and we can add these to a new file in the src flolder, meetingData.js.

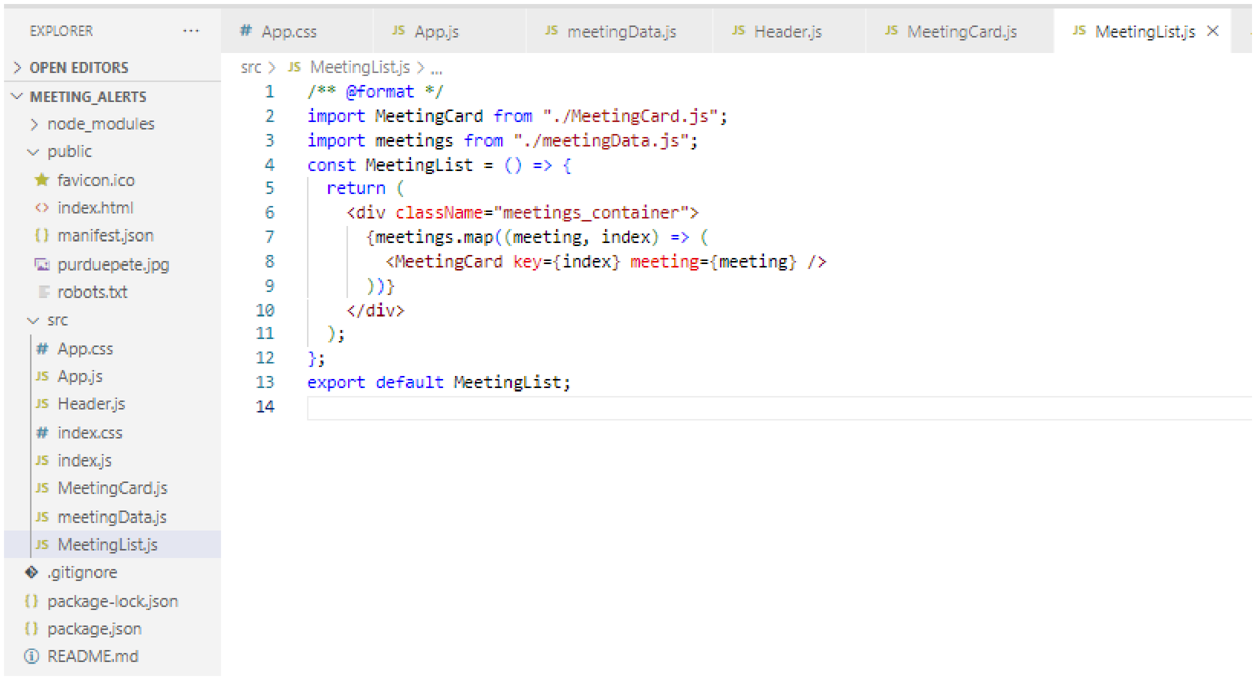


```
/** @format */

const meetings = [
  {
    topic: "CIT Monthly Meeting",
    dateTime: "Sept 19th 2024, 2pm-3pm",
    location: "Knoy Hall West Lafayette",
    parking: "Midway Garage",
  },
  {
    topic: "Research In Higher Level Ed",
    dateTime: "September 24th 2024, 1pm-5pm",
    location: "Stuart Buildings, West Lafayette",
    parking: "Street Level",
  },
  {
    topic: "Curriculum Planning",
    dateTime: "October 19th 2024, 4pm-6pm",
    location: "IO 240, Indianapolis",
    parking: "North Street Garage",
  },
];
export default meetings;
```

```
const meetings = [
 {
  topic: "CIT Monthly Meeting",
  dateTime: "Sept 19th 2024, 2pm-3pm",
  location: "Knoy Hall West Lafayette",
  parking: "Midway Garage",
 },
 {
  topic: "Research In Higher Level Ed",
  dateTime: "September 24th 2024, 1pm-5pm",
  location: "Stuart Buildings, West Lafayette",
  parking: "Street Level",
 },
 {
  topic: "Curriculum Planning",
  dateTime: "October 19th 2024, 4pm-6pm",
  location: "IO 240, Indianapolis",
  parking: "North Street Garage",
 },
];
export default meetings;
```

Now, use the JavaScript array ***map*** method, which returns an object as it iterates over all of the elements in the array. In this case, the object being returned is a markup entity – a React component to which is passed the meeting information in the props object.

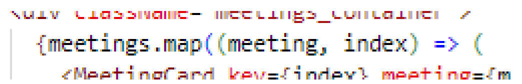
React, Introduction (Scaffold, Using JSX, Creating Components)



```
/** @format */
import MeetingCard from "./MeetingCard.js";
import meetings from "./meetingData.js";
const MeetingList = () => {
  return (
    <div className="meetings_container">
      {meetings.map((meeting, index) => (
        <MeetingCard key={index} meeting={meeting} />
      ))}
    </div>
  );
};
export default MeetingList;
```

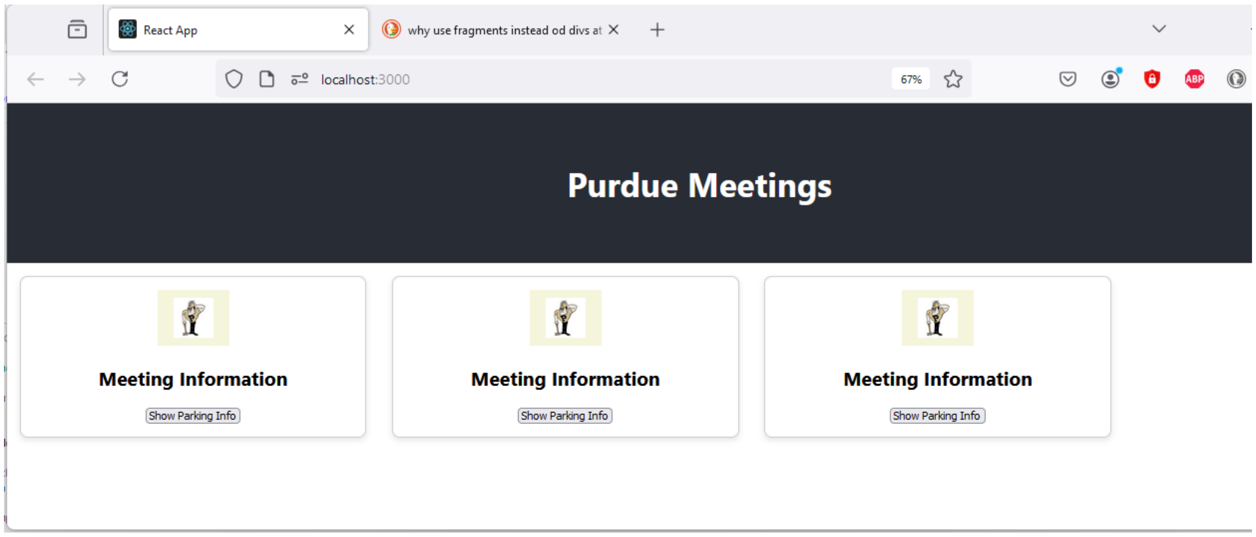
This operates on the content of the list and renders a Card for each item in the list.

Notice, in the code, we are using an index value in the map function

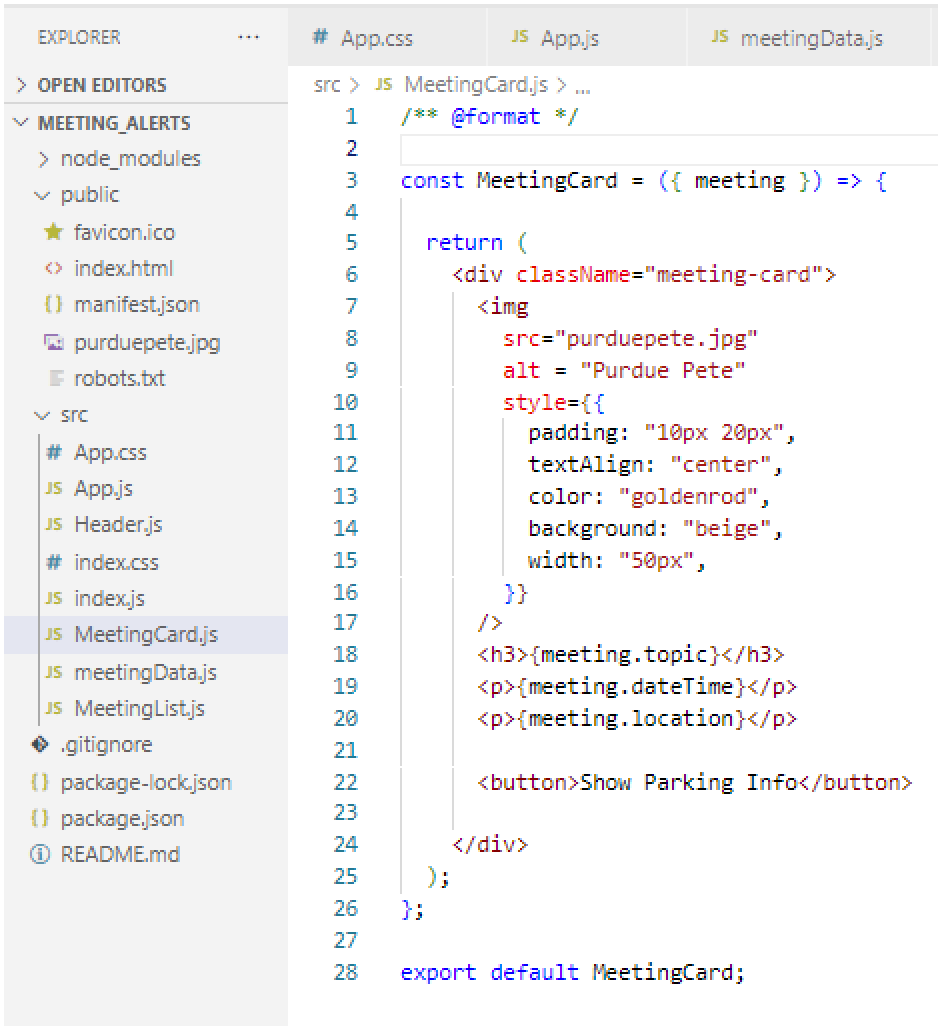


```
{meetings.map((meeting, index) => (
```

– this is a requirement for React, we have to use a unique value as we reference *list* content. Where did this come from? It is directly related to accessing each item in turn, selecting the correct item and making sure that the correct data is targeted.

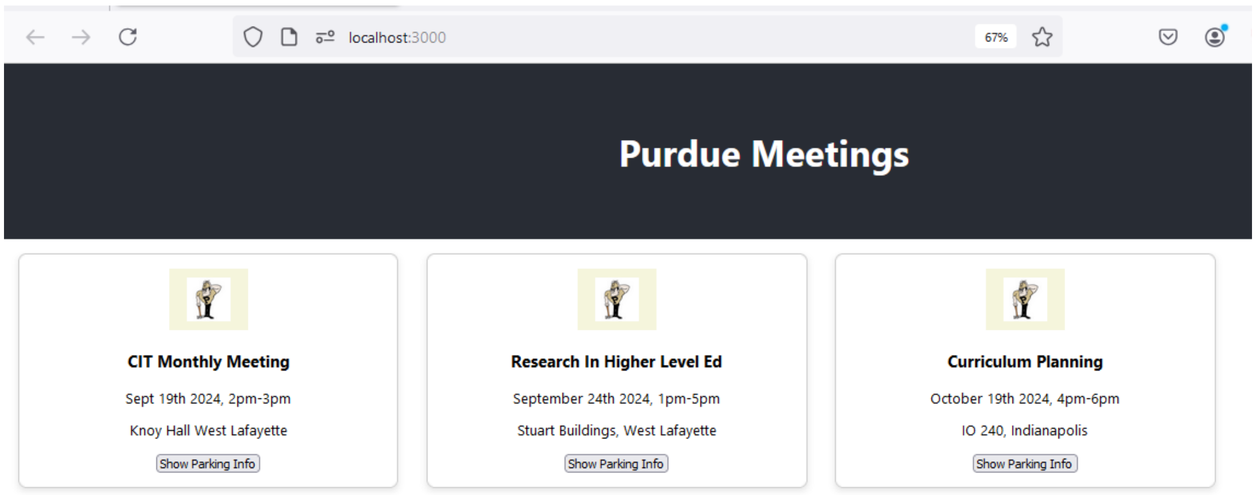


Now let's add some specific meeting information for each meeting object in the meeting list. We already have the meeting object, as it is passed in via props. So, we can use the meeting object and the dot access operator, to get the information from the object.



```
/** @format */

const MeetingCard = ({ meeting }) => {

  return (
    <div className="meeting-card">
      <img
        src="purduepete.jpg"
        alt = "Purdue Pete"
        style={{
          padding: "10px 20px",
          textAlign: "center",
          color: "goldenrod",
          background: "beige",
          width: "50px",
        }}
      />
      <h3>{meeting.topic}</h3>
      <p>{meeting.dateTime}</p>
      <p>{meeting.location}</p>

      <button>Show Parking Info</button>

    </div>
  );
};

export default MeetingCard;
```

This should result in



Any questions so far? Is it all wired together? Do you need any explanations?

The next step is the Parking modal.

The parking modal relates to the Meeting object and the Meeting Card, so it can be localized.

First, let's discuss ***hooks*** in React. ***Hooks*** were added to React in version 16.8.

Hooks allow components to be notified of changes, and to take action accordingly. There are multiple such hooks defined by React. Here, we will use the one that connects changing state, with a React function – ***useState***.

useState integrates a variable with a function used to change the value (state) of the variable. The variable is given an initial value, and a function to set a new value.

As before, we insert the markup for a modal, and hide/show the content depending on a button click, or a state variable.

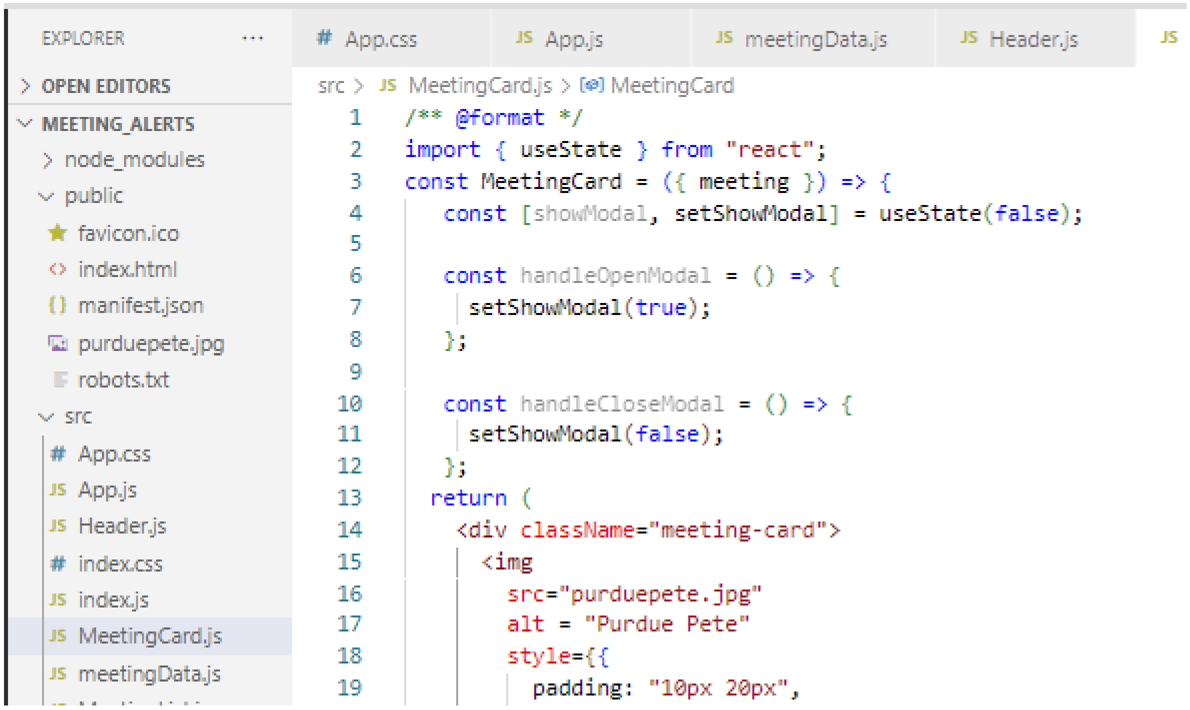
In order to use this hook, we need to include it. The parking modal button is in the MeetingCard component, so this is an appropriate place to add the registration for useState.

Below, the variable is *showModal*, and the function that sets it's value is *setShowModal*. The initial value for showModal is false.

const [showModal, setShowModal] = useState(false);

The ***useState*** function returns an array, and here we are using array destructuring to assign the values to showmodal and the setShowModal function, respectively.

We have 2 functions that coordinate with the state variable to display or hide the modal, ***handleOpenModal*** and ***handleCloseModal*** – these 2 functions set the value of the state variable to either true or false. You can give them any title so long as they are used correctly!



```
/** @format */
import { useState } from "react";
const MeetingCard = ({ meeting }) => {
  const [showModal, setShowModal] = useState(false);

  const handleOpenModal = () => {
    setShowModal(true);
  };

  const handleCloseModal = () => {
    setShowModal(false);
  };
 return (
   <div className="meeting-card">
     <img
       src="purduepete.jpg"
       alt = "Purdue Pete"
       style={{
         padding: "10px 20px",
```

When the parking button is clicked, the parking modal should open, so we should link the onClick event to the *handleOpenModal* function.

```
<button onClick={handleOpenModal}>Show Parking Info</button>
```

This function sets the value of the *showModal* state variable, to true.

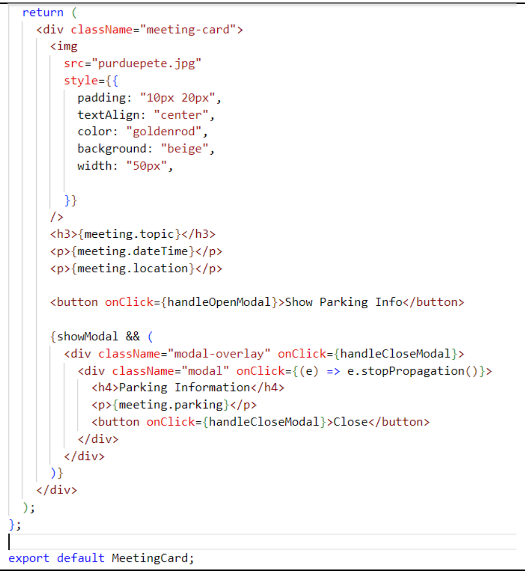
If the *showModal* variable is true, then the modal should be visible. We can use a shorthand notation to set the display property to true, for the modal content.

{***showModal &&*** (

<div .....

Let’s add this content first and then style the modal appropriately.

React, Introduction (Scaffold, Using JSX, Creating Components)



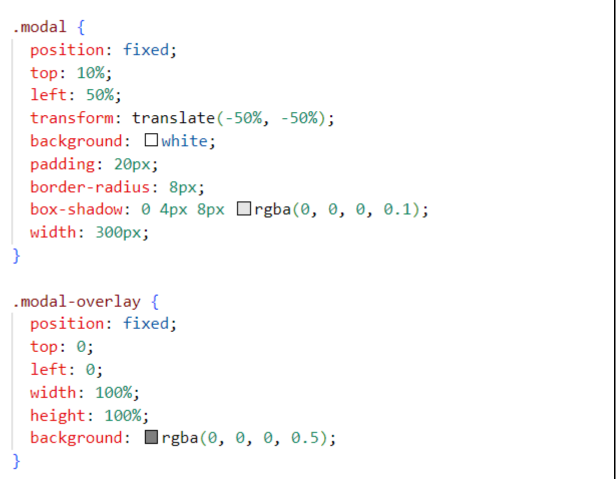
```
  return (
    <div className="meeting-card">
      <img
        src="purduepete.jpg"
        style={{
          padding: "10px 20px",
          textAlign: "center",
          color: "goldenrod",
          background: "beige",
          width: "50px",

        }}
      />
      <h3>{meeting.topic}</h3>
      <p>{meeting.dateTime}</p>
      <p>{meeting.location}</p>

      <button onClick={handleOpenModal}>Show Parking Info</button>

      {showModal && (
        <div className="modal-overlay" onClick={handleCloseModal}>
          <div className="modal" onClick={(e) => e.stopPropagation()}>
            <h4>Parking Information</h4>
            <p>{meeting.parking}</p>
            <button onClick={handleCloseModal}>Close</button>
          </div>
        </div>
      )}
    </div>
  );
};

export default MeetingCard;
```

```
{showModal && (
     <div className="modal-overlay" onClick={handleCloseModal}>
      <div className="modal" onClick={(e) => e.stopPropagation()}>
       <h4>Parking Information</h4>
       <p>{meeting.parking}</p>
       <button onClick={handleCloseModal}>Close</button>
      </div>
     </div>
   )}
```

Within the parking modal content, a close button calls the handleClose function.

```
<button onClick={handleCloseModal}>Close</button>
```

Add the specific styling to the App.css file, feel free to re-style the parking modal !



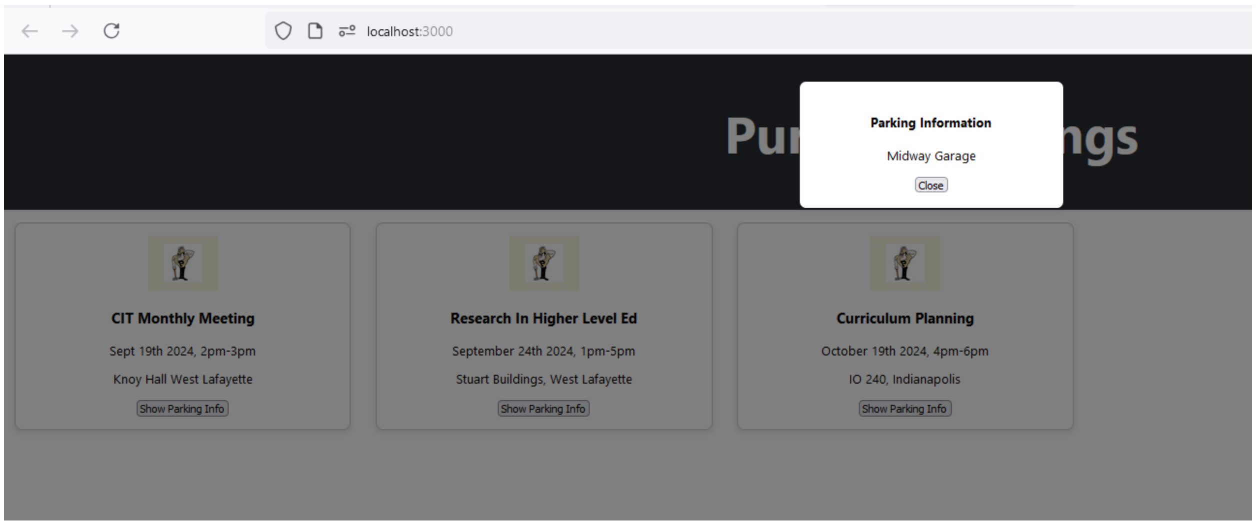
```
.modal {
  position: fixed;
  top: 10%;
  left: 50%;
  transform: translate(-50%, -50%);
  background: □white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 4px 8px □rgba(0, 0, 0, 0.1);
  width: 300px;
}

.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: ■rgba(0, 0, 0, 0.5);
}
```

```
.modal {
  position: fixed;
  top: 10%;
  left: 50%;
  transform: translate(-50%, -50%);
  background: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  width: 300px;
}

.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.5);
}
```

... and now you should see the parking modal become visible on clicking the

parking button - the modal content is displayed.

React, Introduction (Scaffold, Using JSX, Creating Components)



Parking Information

Midway Garage

CIT Monthly Meeting

Sept 19th 2024, 2pm-3pm

Knoy Hall West Lafayette

Research In Higher Level Ed

September 24th 2024, 1pm-5pm

Stuart Buildings, West Lafayette

Curriculum Planning

October 19th 2024, 4pm-6pm

IO 240, Indianapolis

App.css	
	```
	.App {
	  text-align: center;
	}

	.App-header {
	  background-color: #282c34;
	  display: flex;
	  flex-direction: column;
	  align-items: center;
	  justify-content: center;
	  font-size: calc(10px + 2vmin);
	  color: white;
	  height: 200px;
	}

	.App-link {
	  color: #61dafb;
	}

	.meeting-card {
	  border: 1px solid #ccc;
	  border-radius: 8px;
	  padding: 16px;
	  margin: 16px;
	```

```css
  width: 400px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.meetings_container {
  display: flex;
  flex-wrap: wrap;
  flex-direction: row;
}

.modal {
  position: fixed;
  top: 10%;
  left: 50%;
  transform: translate(-50%, -50%);
  background: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  width: 300px;
}

.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.5);
}
```

MeetingCard.js

```javascript
import React, { useState } from "react";
import "./App.css";
const MeetingCard = ({ meeting }) => {
  const [showModal, setShowModal] = useState(false);

  const handleOpenModal = () => {
    setShowModal(true);
  };

  const handleCloseModal = () => {
    setShowModal(false);
  };
  return (
    <div className="meeting-card">
      <img
        src="purduepete.jpg"
        alt="Purdue Pete"
        style={{
          padding: "10px 20px",
          textAlign: "center",
          color: "goldenrod",
          background: "beige",
          width: "50px",
        }}
      />
      <h3>{meeting.topic}</h3>
      <p>{meeting.dateTime}</p>
      <p>{meeting.location}</p>

      <button onClick={handleOpenModal}>Show Parking Info</button>
      {showModal && (
        <div className="modal-overlay" onClick={handleCloseModal}>
          <div className="modal" onClick={(e) => e.stopPropagation()}>
            <h4>Parking Information</h4>
            <p>{meeting.parking}</p>
            <button onClick={handleCloseModal}>Close</button>
          </div>
        </div>
      )}
    </div>
  );
};

export default MeetingCard;
```