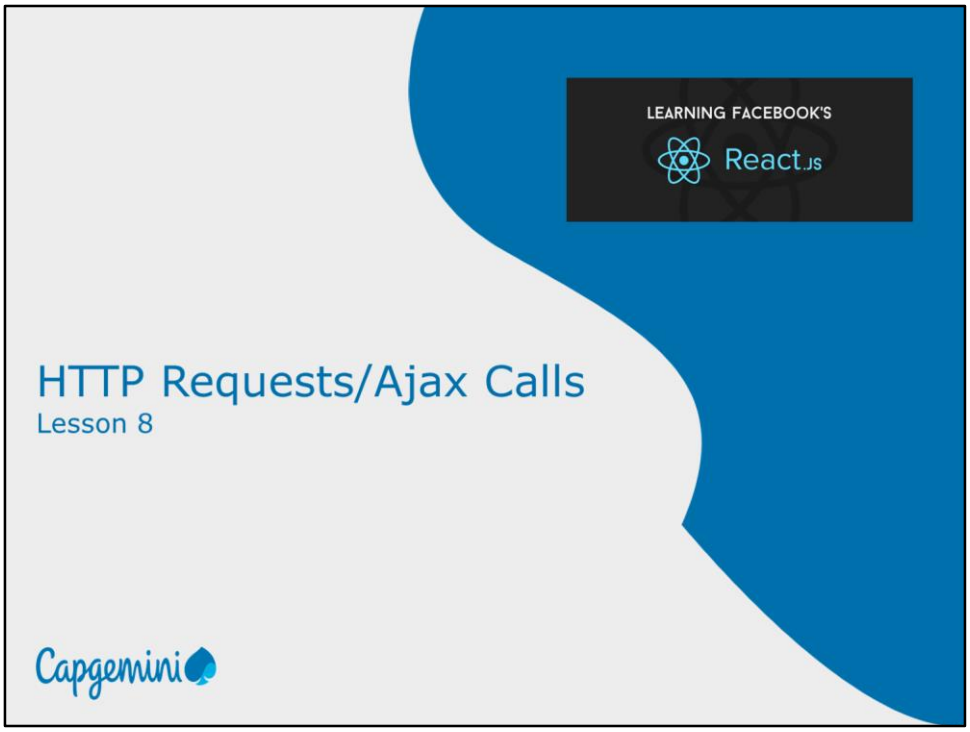
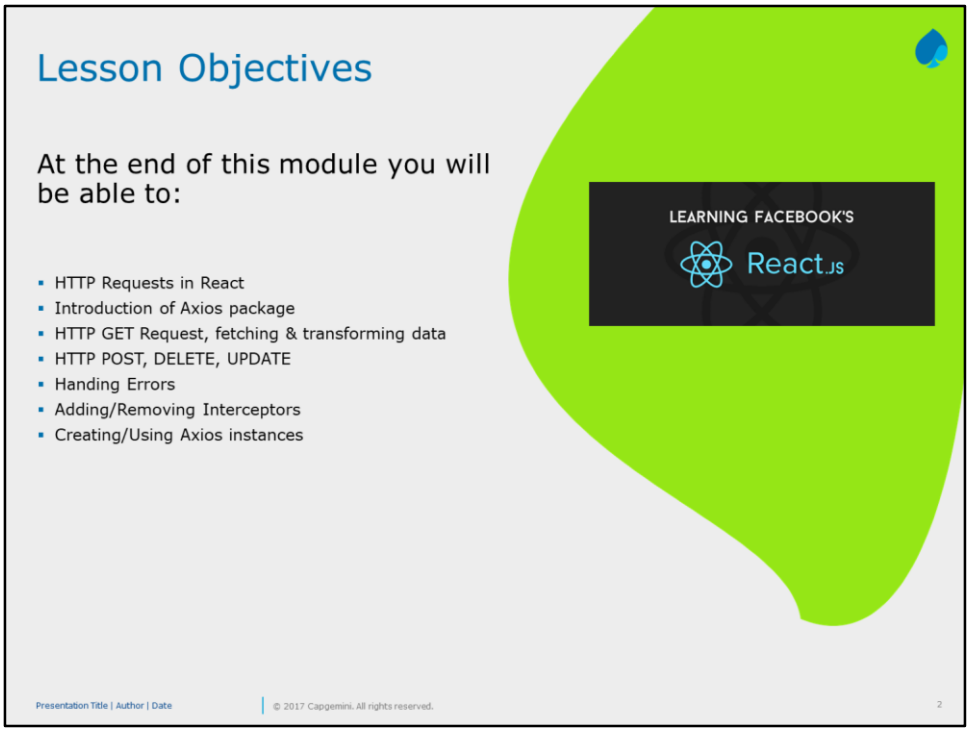
Instructor Notes:

Add instructor notes here.

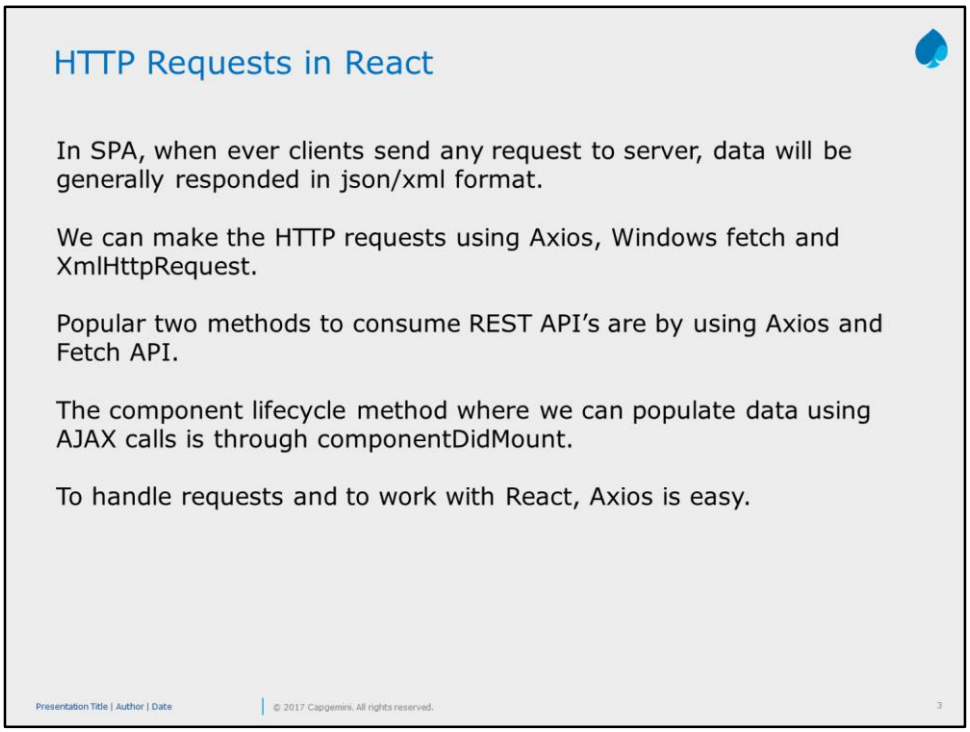


Instructor Notes:

Add instructor notes here.



Instructor Notes:



In SPA, when ever clients send any request to server, data will be generally responded in json/xml format.

We can make the HTTP requests using Axios, Windows fetch and XmlHttpRequest.

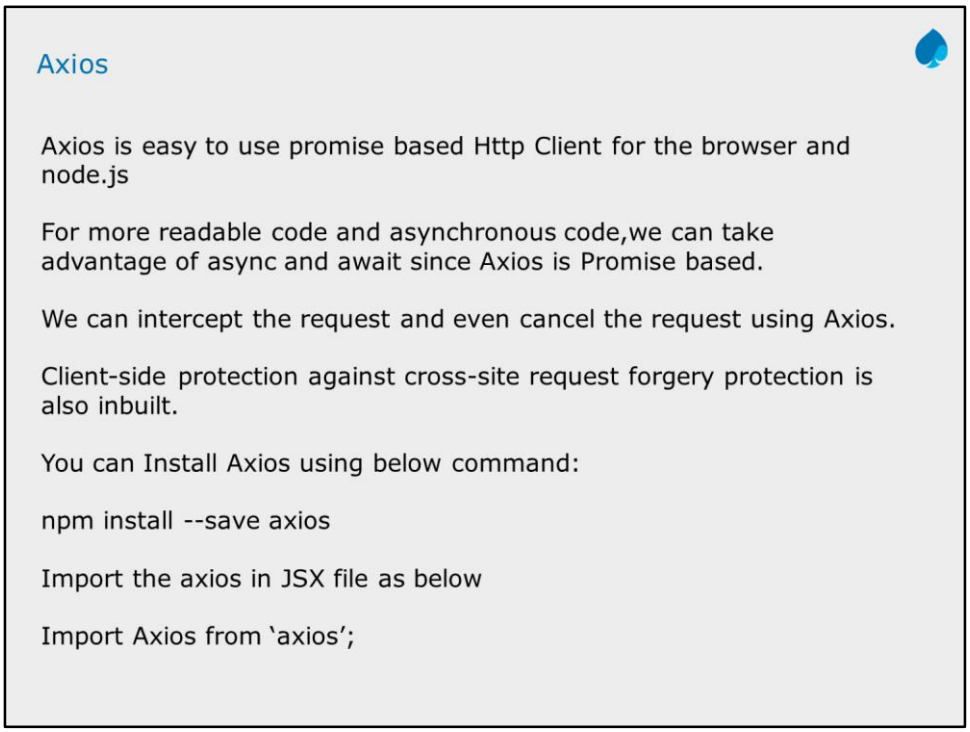
Popular two methods to consume REST API's are by using Axios and Fetch API.

The component lifecycle method where we can populate data using AJAX calls is through componentDidMount.

To handle requests and to work with React, Axios is easy.

Sending HTTP request from your react app is quite simple. In fact, you don't even need to use a library to do this.
All we need to do to send a simple GET request is to create a new XMLHttpRequest, add an event listener to it, open the URL and send the request.

Instructor Notes:



For small applications, using XMLHttpRequest works just fine.

As your apps get larger though, you might want to consider using an HTTP request library like axios.

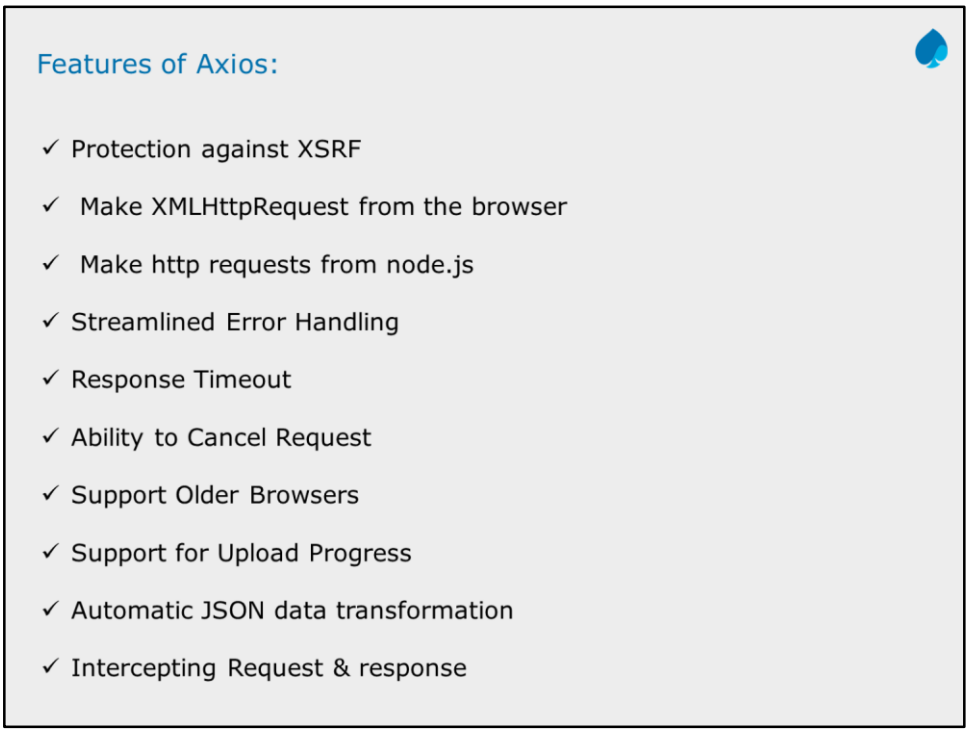
These libraries do the same things we did above but they provide a much simpler and cleaner API.

This is especially useful as your requests get more complex.

Also, the API uses promises instead of plain old callbacks which helps to keep things readable.

So let's take a look at how we can use axios to send HTTP requests.

Instructor Notes:



we need to install axios in our react project. We can do so using this command:
npm install axios
Now that we got that out of the way, let's send some request!

Instructor Notes:

It has been observed that componentDidMount is the best place to make side effects like making Http Requests.

Once Http Request is completed, state can be updated Asynchronously and then page will re-render with that updates.

Axios use the Promise to work it in Asynchronous way.

```
componentDidMount()
{
  Axios.get('url').then(response) => { console.log(response) };
}
```

In above code we have used Axios.get(url) from endpoint API to get Promise which returns response object.

Some more information also we can get such as status code under response.status object.

All we need to do is to call the axios.get method and provide the URL of the endpoint.

Instructor Notes:

Http Get method

```
import React, { Component } from 'react';
import axios from 'axios';

class CustomerList extends Component {
  state = {   customers: [ ]   }

  componentDidMount() {
    axios.get(`https://someurl.com/customers`)
      .then(response => {    const customers = response.data;
                this.setState({ customers });   })   }

  render( ) {
    return (
      <ul>
            { this.state.customers.map(customer =>
                <li>{customer.name}</li>)}
      </ul>
    )   }  }
```

First, you import React and Axios so that both can be used in the component. Then you hook into the componentDidMount lifecycle hook and perform a GET request.
You use axios.get(url) with a URL from an API endpoint to get a promise which returns a response object. Inside the response object, there is data that is then assigned the value of person.
You can also get other information about the request, such as the status code under res.status or more information inside of res.request.

Instructor Notes:

Making Http post method

In below code, you have created the class CustomerList.

It hold state such as name, email null values.

You have created handler named as handleChange where you update state using setState() method

```
import React from 'react';
import axios from 'axios';

export default class CustomerList extends React.Component {
  state = {
    name: ' ' ,   email : ' '
  }

  handleChange = event => {
    this.setState({ name: event.target.value, email:
                              event.target.value  });
    }
```

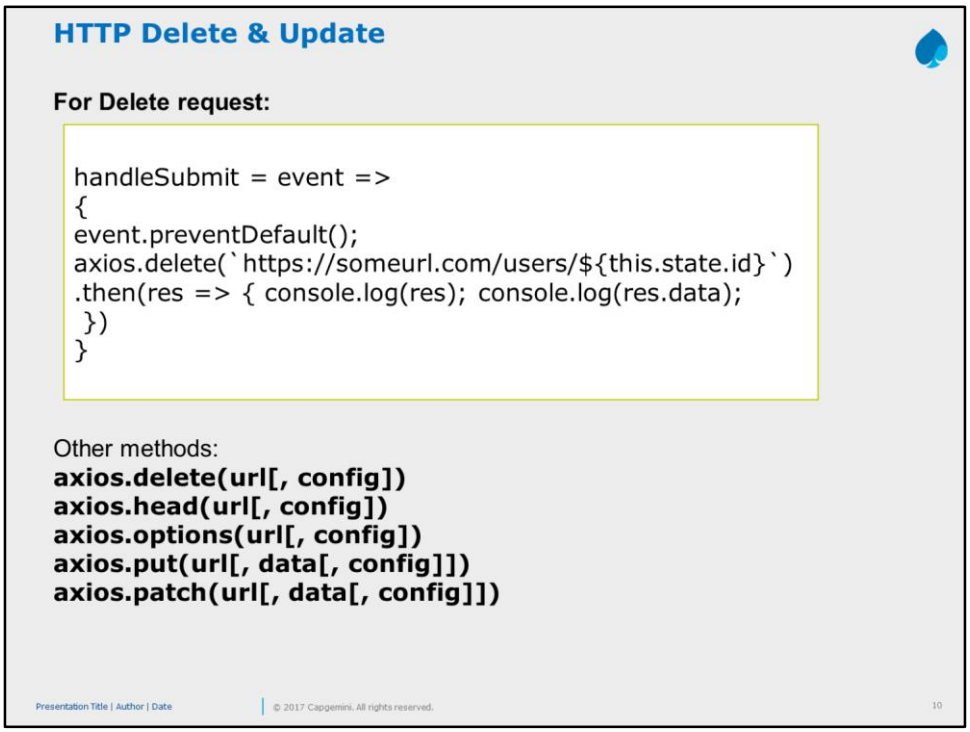
Instructor Notes:

```
handleSubmit = event => {
  event.preventDefault();

  const user = {
    name: this.state.name , email: this.state.email };

  axios.post(`https://someurl.com/users`, { user })
    .then(res => {
      console.log(res);
      console.log(res.data);
    })   }
```

```
render( ) {
  return (
    <div>
      <form onSubmit={this.handleSubmit}>
        Customer Name: <input type="text" name="name"
                              onChange={this.handleChange} />
        <button type="submit">Add Customer</button>
      </form>
    </div>
  )  } }
```

Inside the handleSubmit function, you prevent the default action of the form.
Then update the state to the user input.
Using POST gives you the same response object with information that you can use inside of a then call.
To complete the POST request, you first capture the user input.
Then you add the input along with the POST request, which will give you a response.
You can then console.log the response, which should show the user input in the form.

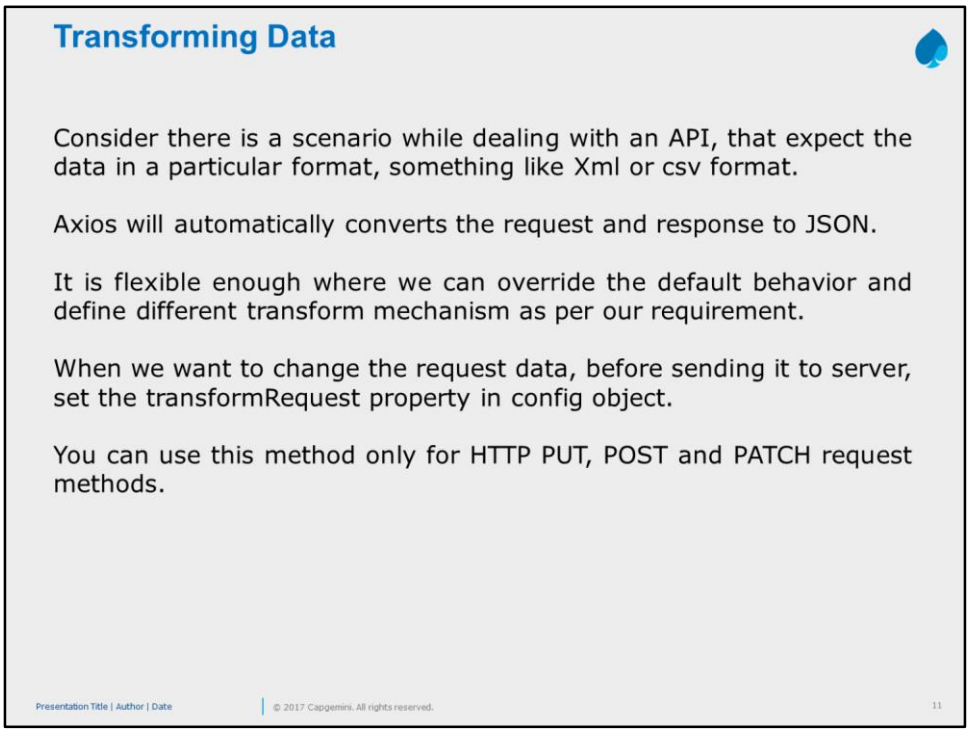
Instructor Notes:



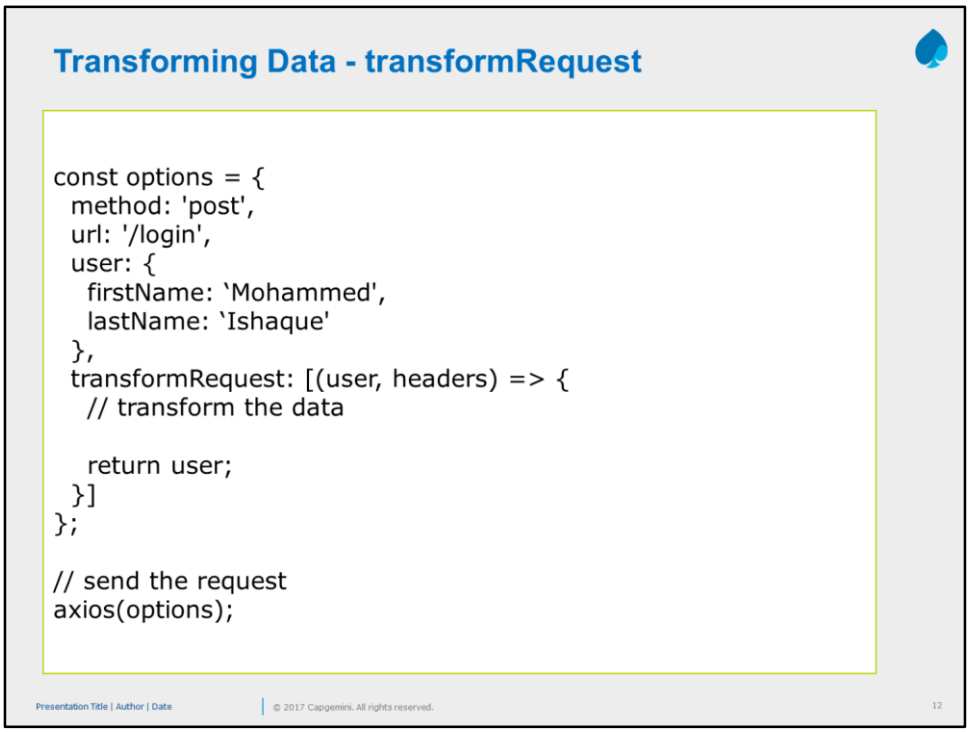
Add parameters to GET requests
A GET response can contain parameters in the URL, like this: https://site.com/?foo=bar.
With Axios you can perform this by using that URL:
axios.get('https://site.com/?foo=bar') or you can use a params property in the options:
axios.get('https://site.com/', { params: { foo: 'bar' } })
POST Requests
Performing a POST request is just like doing a GET request, but instead of axios.get, you use axios.post:
axios.post('https://site.com/') An object containing the POST parameters is the second argument:
axios.post('https://site.com/', { foo: 'bar' })

Axios offers methods for all the HTTP verbs, which are less popular but still used:
axios.put()
axios.delete()
axios.patch()
axios.options()
axios.head()

Instructor Notes:

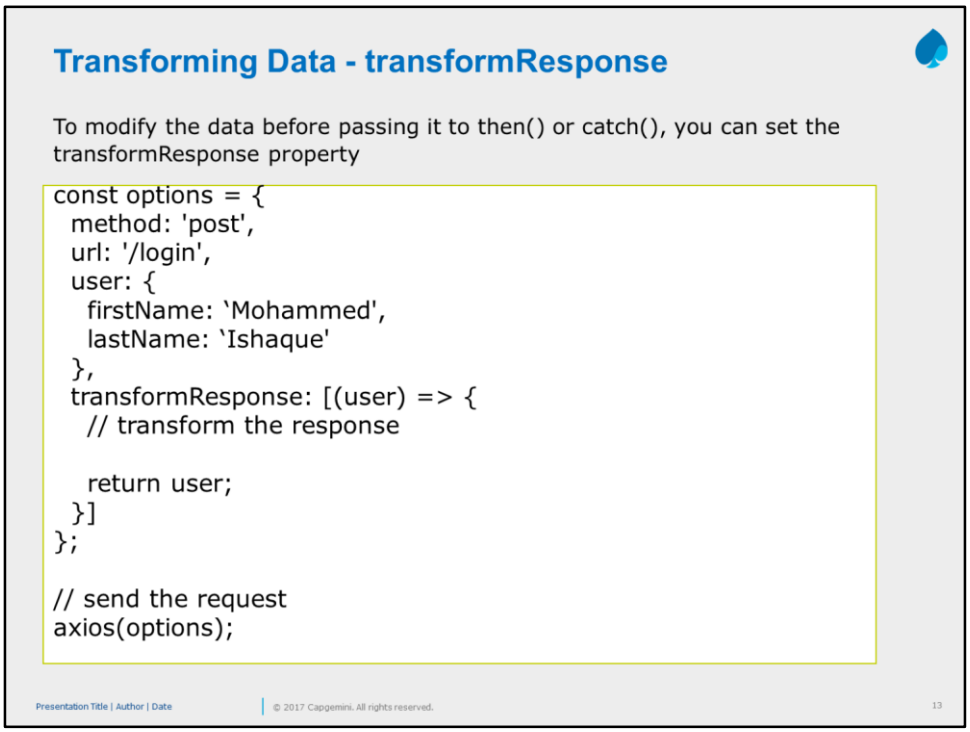


Instructor Notes:



To change the request data before sending it to the server, set the transformRequest property in the config object. Note that this method only works for PUT, POST, and PATCH request methods.

Instructor Notes:



To modify the data before passing it to then() or catch(), you can set the transformResponse property as given above.

Instructor Notes:

Add instructor notes here.

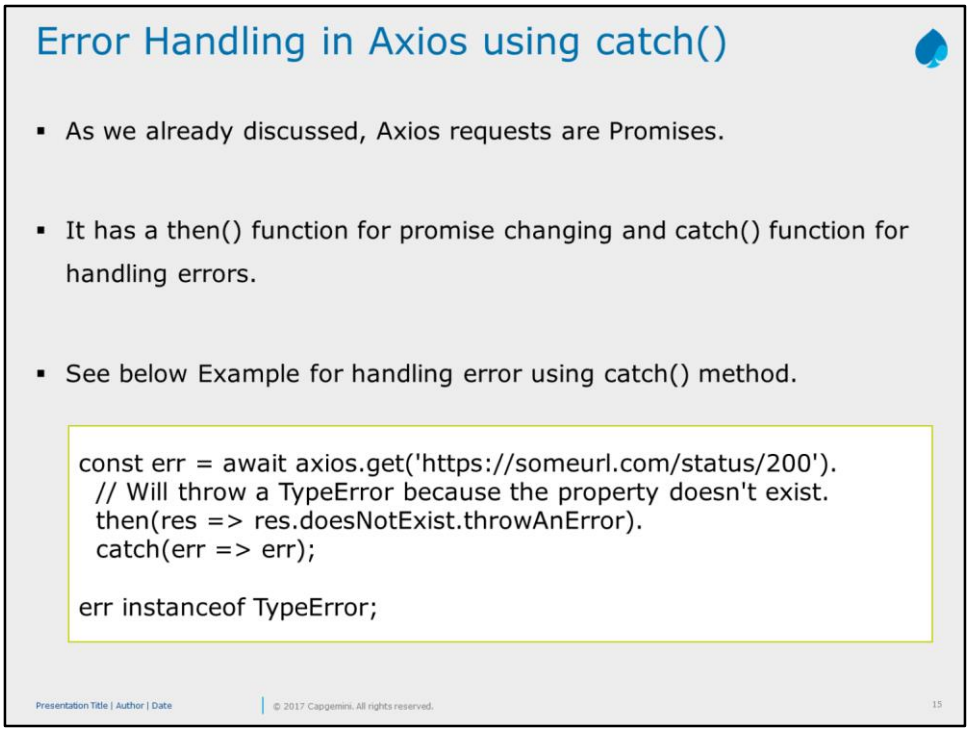
Demo

react-axios-get-demo
react-axios-demo-crud

Add the notes here.

Instructor Notes:

Slide explains regarding the SOAP format



Instructor Notes:

Slide explains regarding the SOAP format

Error Handling in Axios using catch()

- Axios' catch() behaves exactly the same as the promise catch() function. So you can use promise chaining and add a catch() at the end to handle any errors that occur in the promise chain.
- You can also use catch() to transform the error, just make sure you rethrow the error afterwards.

Instructor Notes:

Add instructor notes here.

Demo

React-debug-logical-errors

Add the notes here.

Instructor Notes:

Adding/Removing Interceptors

Axios has one powerful feature called Interceptors.

Interceptors are methods which are triggered before the main method.

It allow you to run your code or modify the request and/or response before the request and/or response is started.

There are two types of interceptors:

request interceptor: This is called before the actual call to the endpoint is made.

We will have two callbacks in request interceptor one with parameter config object and another one with the error object.

response interceptor: This is called before the promise is completed and the data is received by the then callback.

Instructor Notes:

Adding/Removing Interceptors

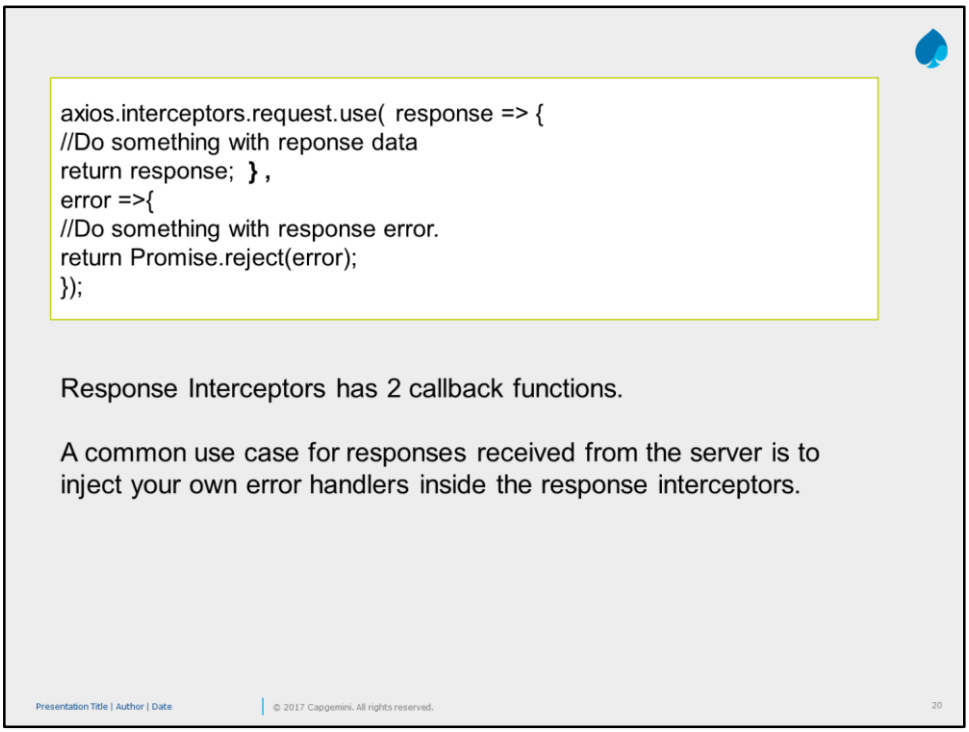
```
// Add a request interceptor
axios.interceptors.request.use((config) =>
 { // Do something before request is sent
return config; },
(error) => { // Do something with request error
return Promise.reject(error);
 }); // Add a response interceptor
```

Request Interceptors has 2 callback functions. First function is used to config your request before it is sent to the server.

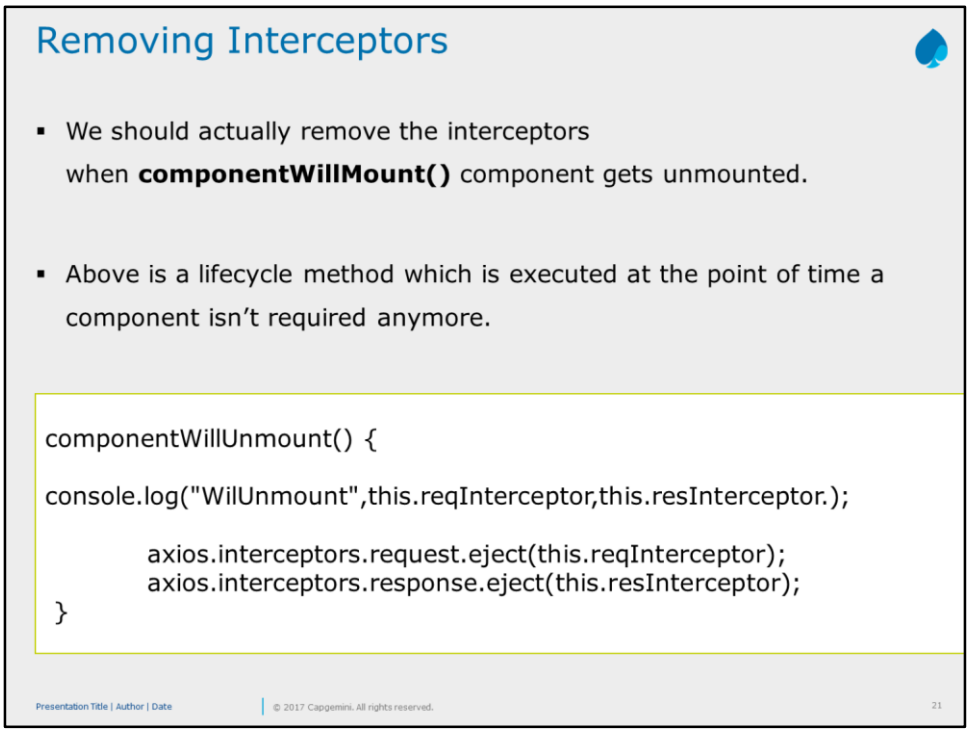
Config object includes option to modify your headers and authentication is a good use case for it.

You can get accessToken from your localStorage, then attach it to your headers with this line of code: *config.headers['Authorization'] = 'Bearer ' + token;*

Instructor Notes:

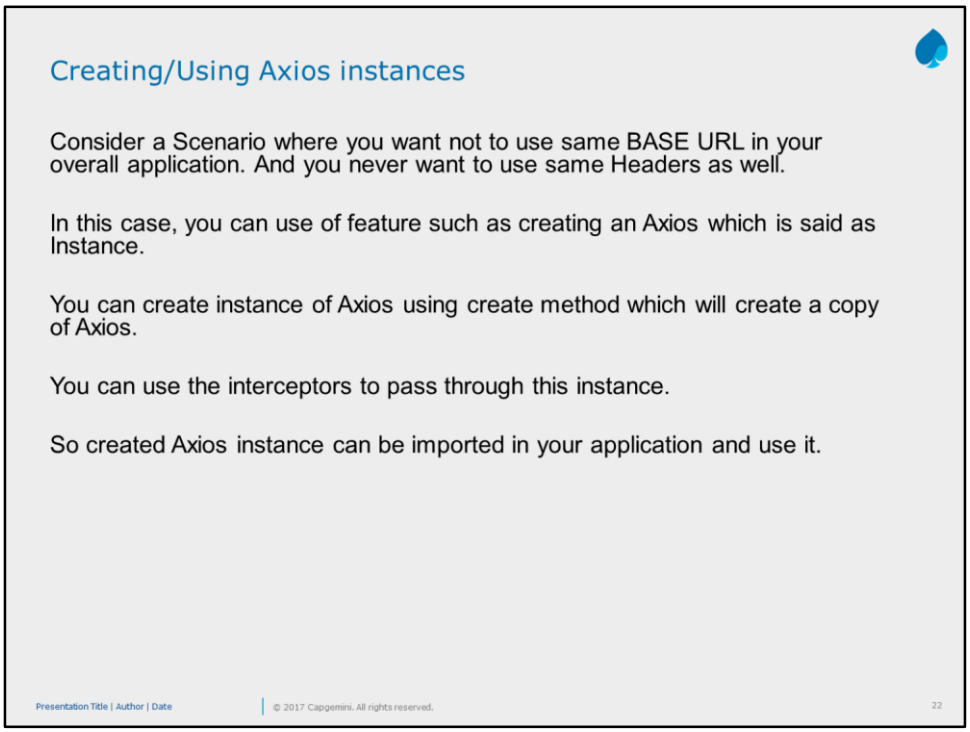


Instructor Notes:



If you may need to remove an interceptor later you can.
const myInterceptor = axios.interceptors.request.use(function () {/*...*/});
axios.interceptors.request.eject(myInterceptor);

Instructor Notes:



Consider a Scenario where you want not to use same BASE URL in your overall application. And you never want to use same Headers as well.

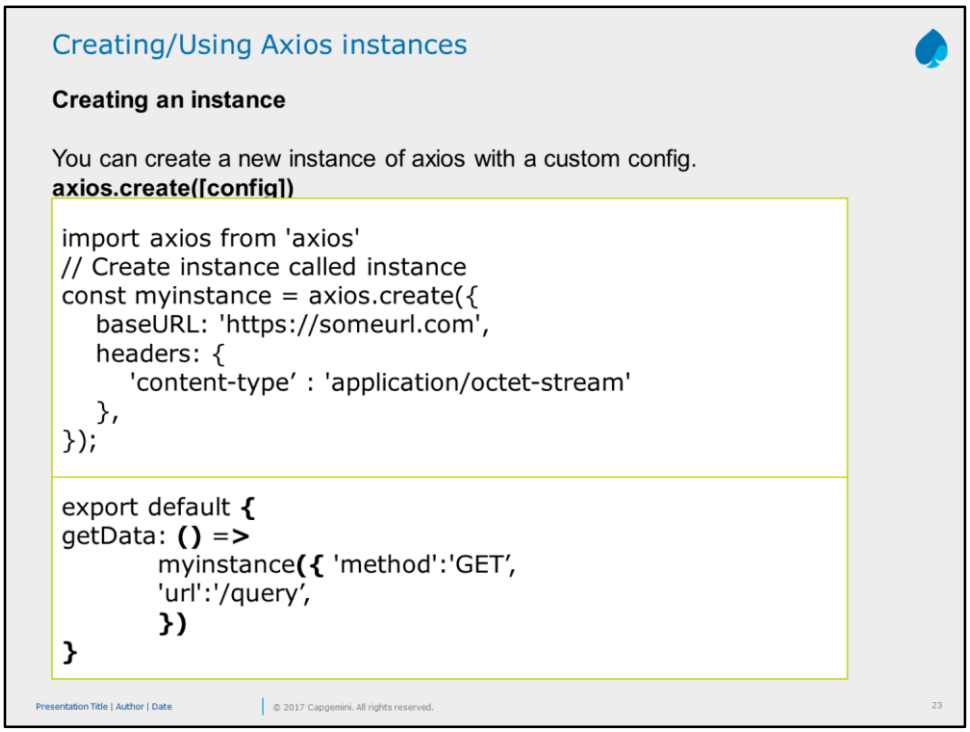
In this case, you can use of feature such as creating an Axios which is said as Instance.

You can create instance of Axios using create method which will create a copy of Axios.

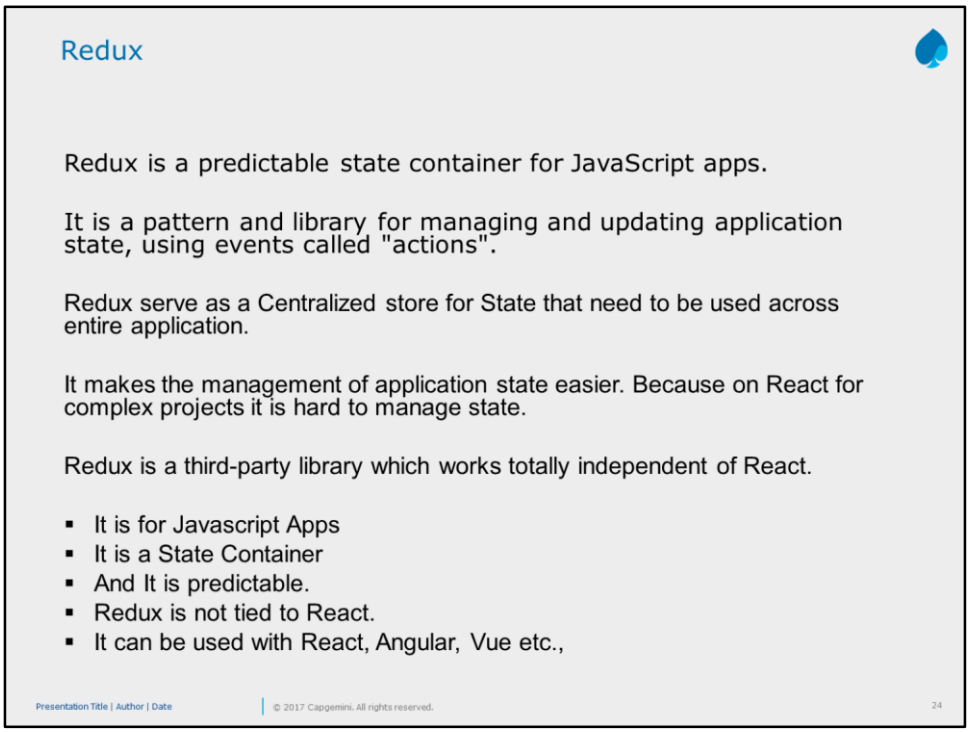
You can use the interceptors to pass through this instance.

So created Axios instance can be imported in your application and use it.

Instructor Notes:



Instructor Notes:



Redux is a predictable state container for JavaScript apps.

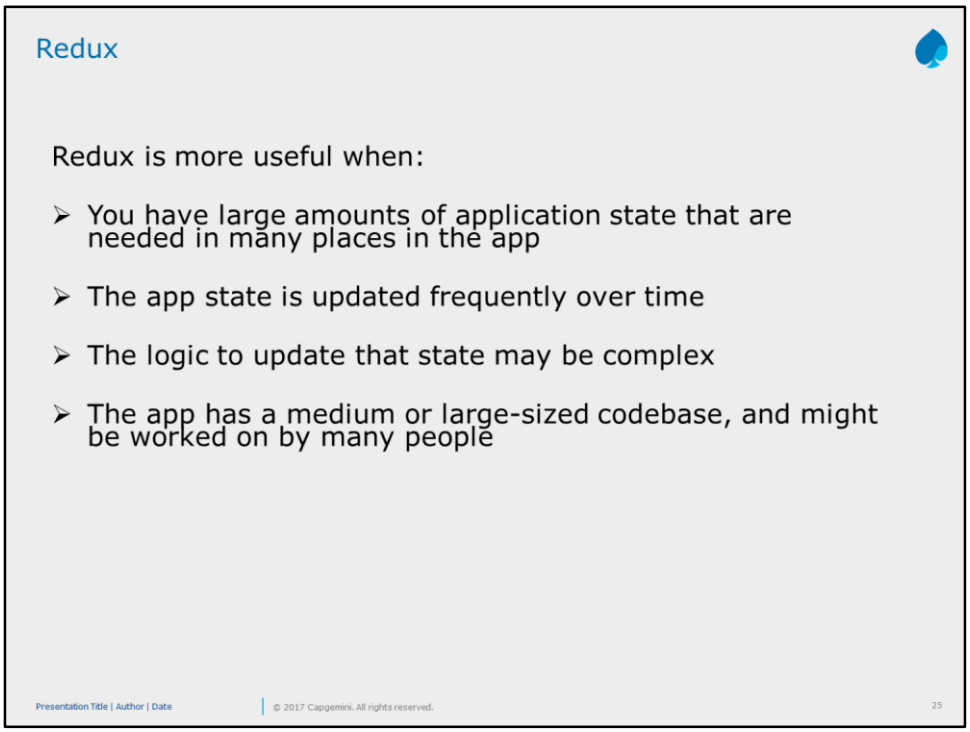
It is a pattern and library for managing and updating application state, using events called "actions".

Redux serve as a Centralized store for State that need to be used across entire application.

It makes the management of application state easier. Because on React for complex projects it is hard to manage state.

Redux is a third-party library which works totally independent of React.

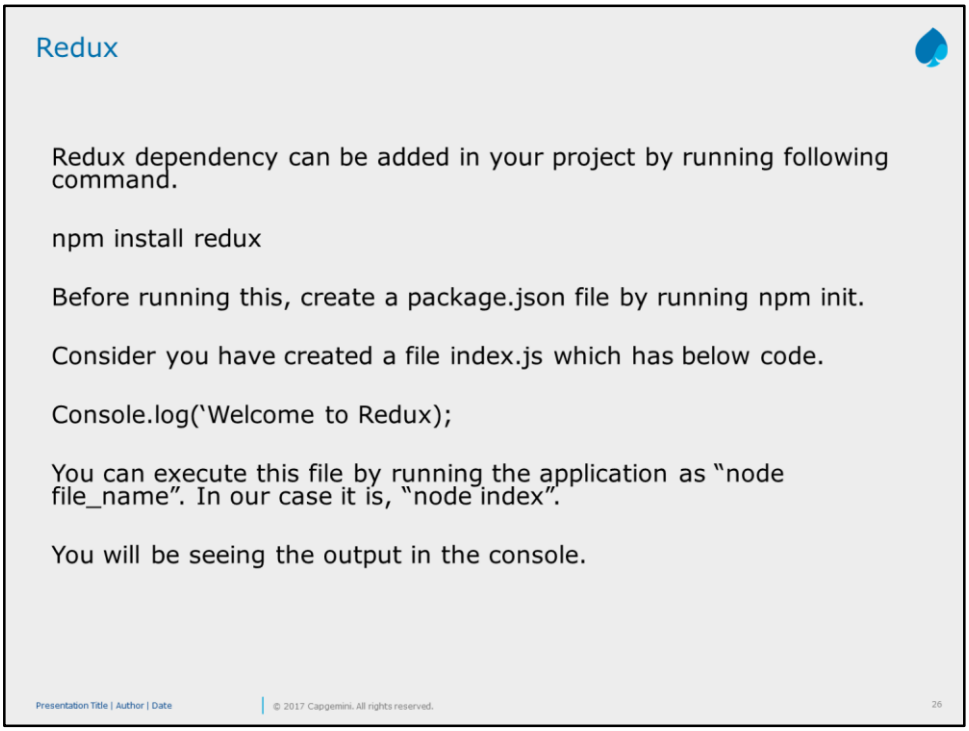
Instructor Notes:



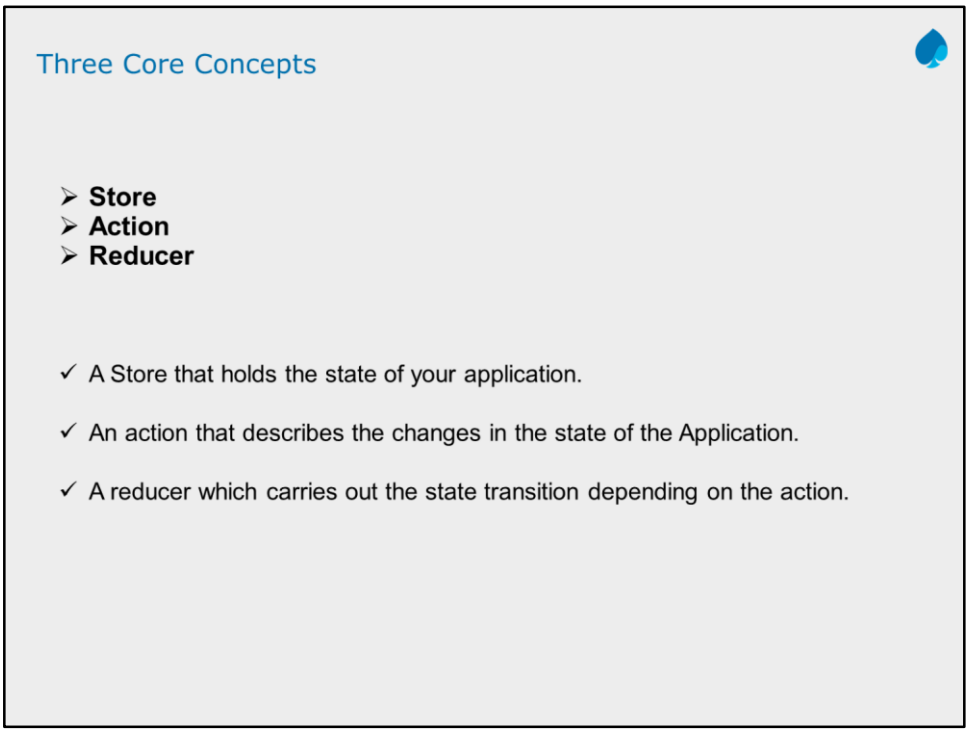
Redux is more useful when:

- You have large amounts of application state that are needed in many places in the app
- The app state is updated frequently over time
- The logic to update that state may be complex
- The app has a medium or large-sized codebase, and might be worked on by many people

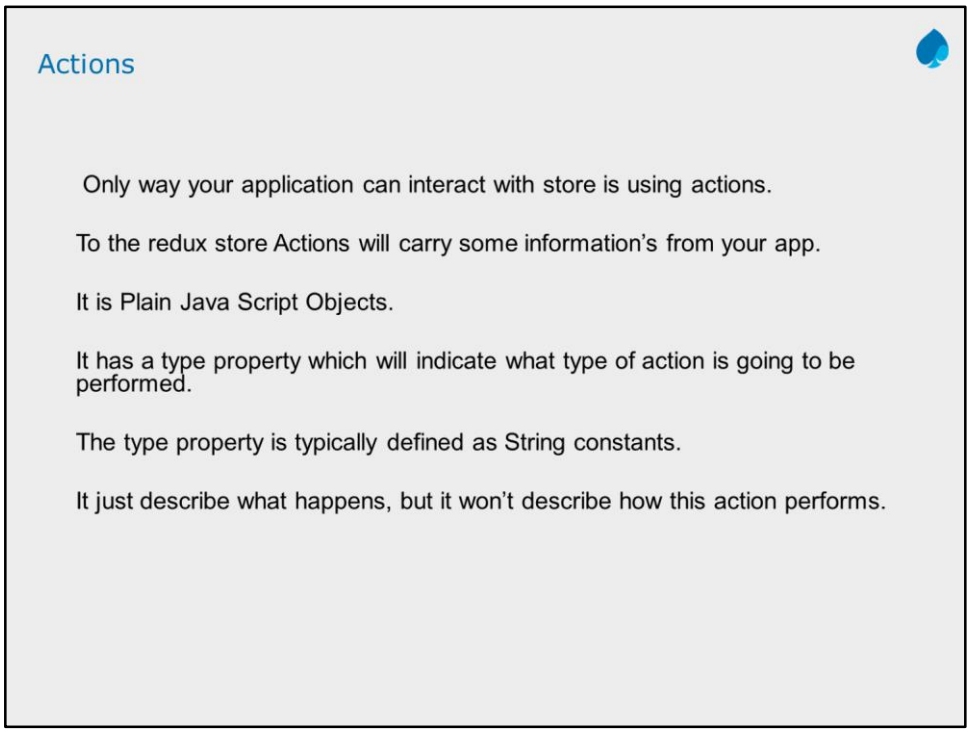
Instructor Notes:



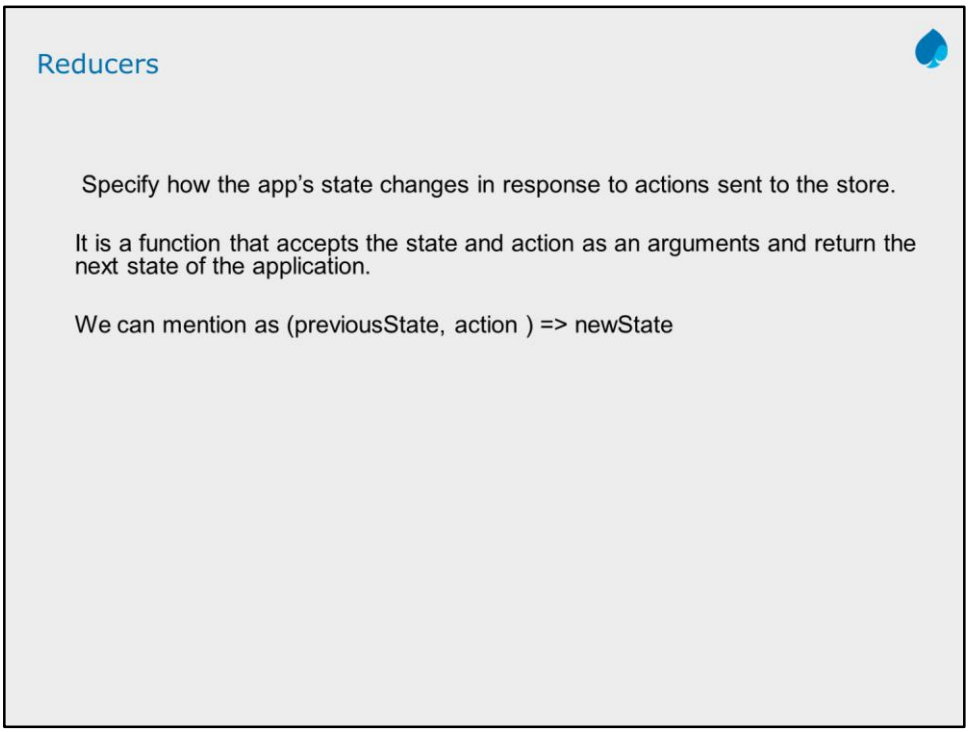
Instructor Notes:



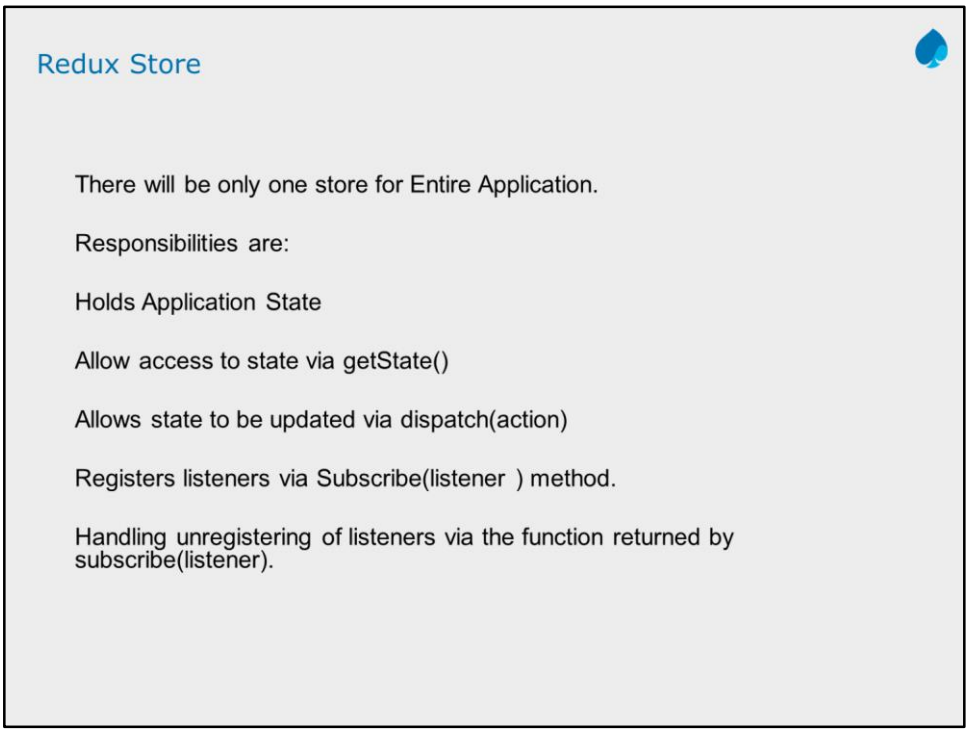
Instructor Notes:



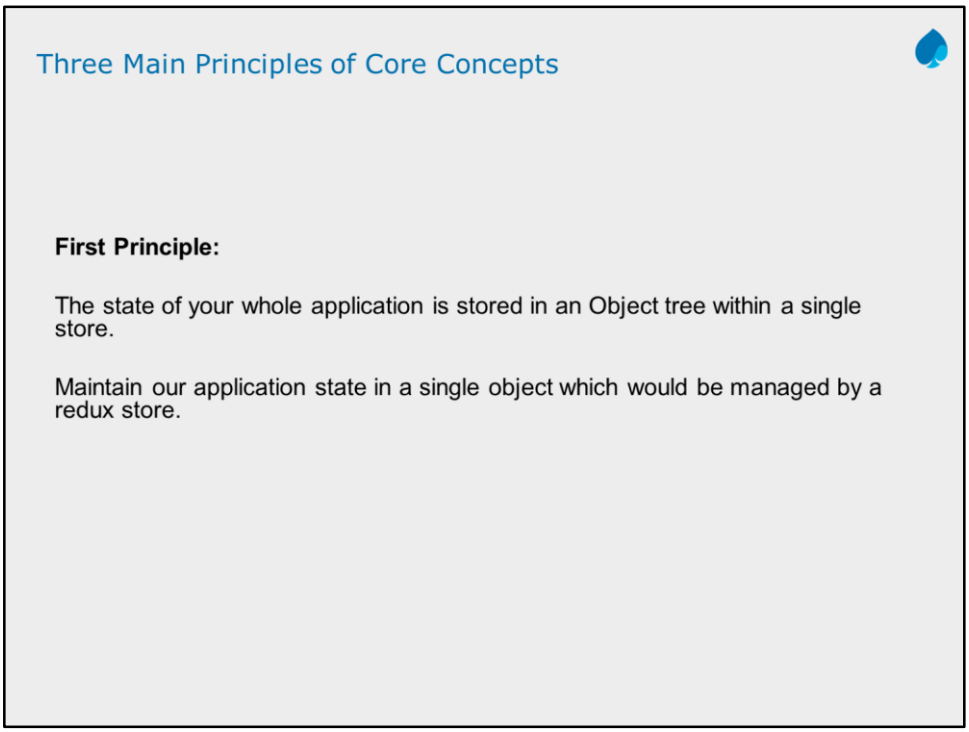
Instructor Notes:



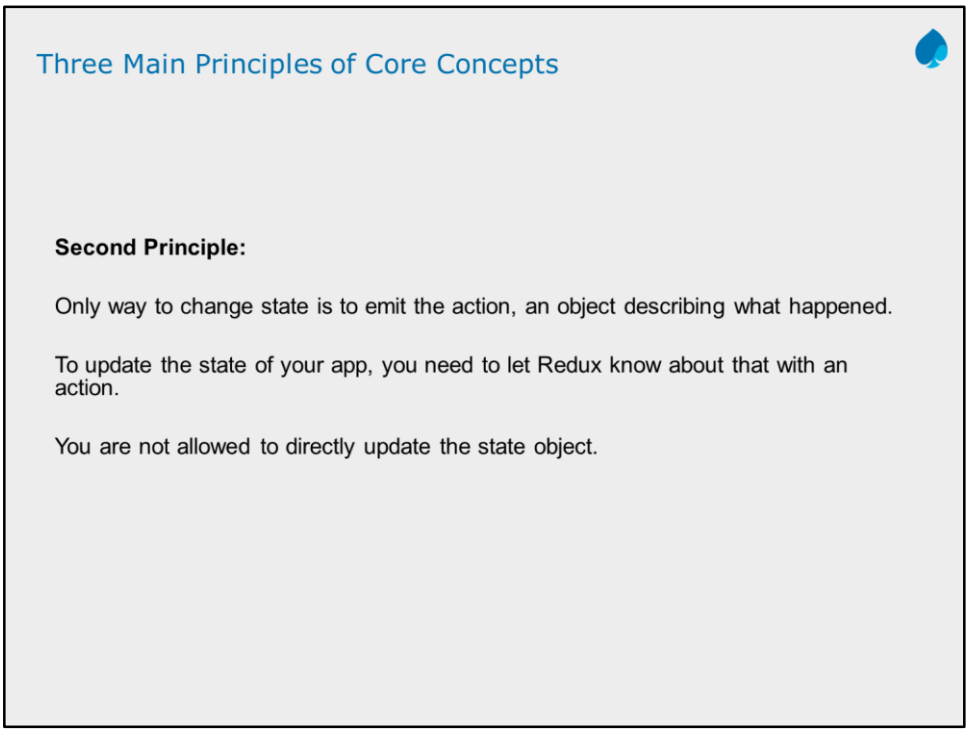
Instructor Notes:



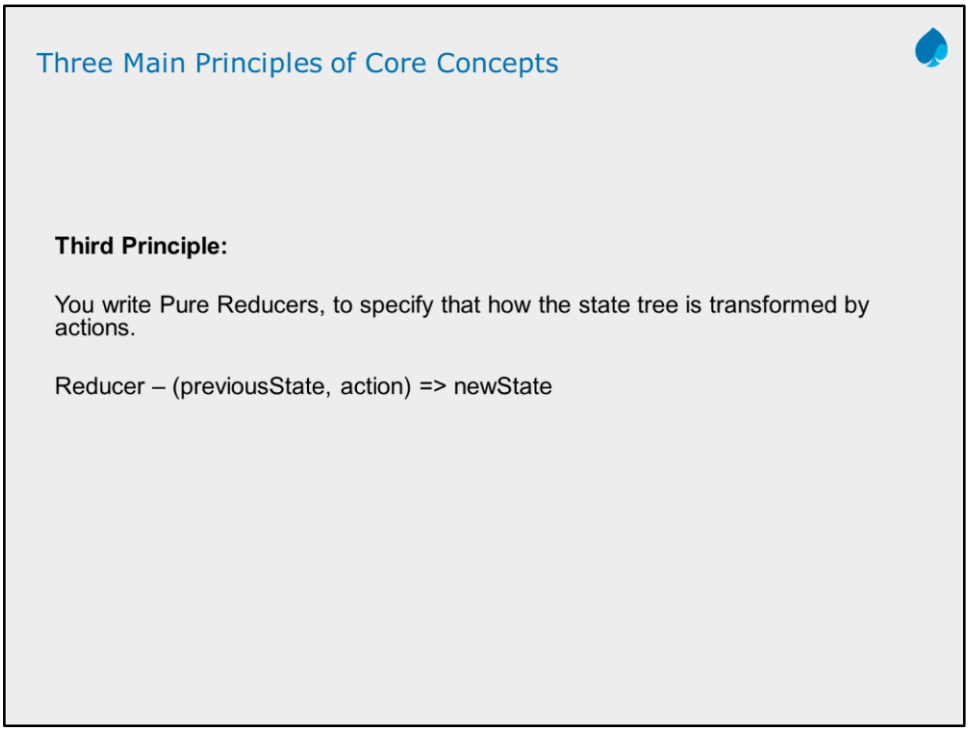
Instructor Notes:



Instructor Notes:



Instructor Notes:



Instructor Notes:

Add instructor notes here.

Demo

React-Redux-Demo

Add the notes here.

Instructor Notes:

Redux Thunk

A thunk is another word for a *function*. It's a special (and uncommon) name for a function that's returned by another. See Below:

```
function wrapper_function() {
// this one is a "thunk" because it defers work for later:
return function thunk() {
// it can be named, or anonymous
        console.log('do stuff now');
 }; }
```

With a plain basic Redux store, you can only do simple synchronous updates by dispatching an action.

Middleware extends the store's abilities, and lets you write async logic that interacts with the store.

Thunks are the recommended middleware for basic Redux side effects logic, including complex synchronous logic that needs access to the store, and simple async logic like AJAX requests.

Instructor Notes:

Redux Thunk

A thunk is another word for a *function*. It's a special (and uncommon) name for a function that's returned by another. See Below:

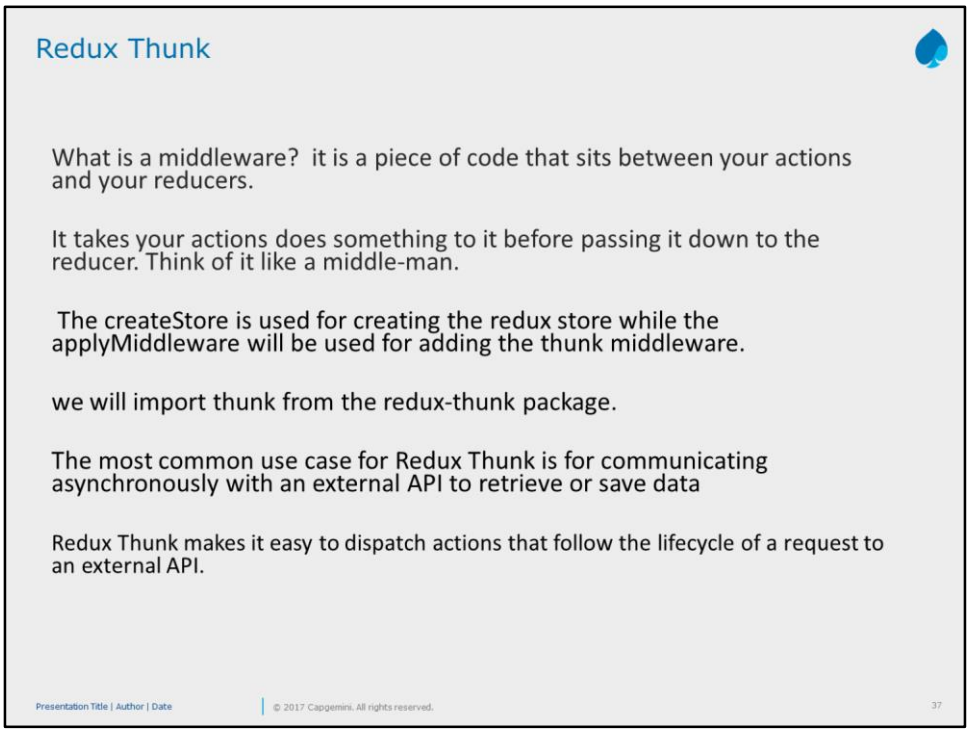
```
function wrapper_function() {
// this one is a "thunk" because it defers work for later:
return function thunk() {
// it can be named, or anonymous
        console.log('do stuff now');
 }; }
```

With a plain basic Redux store, you can only do simple synchronous updates by dispatching an action.

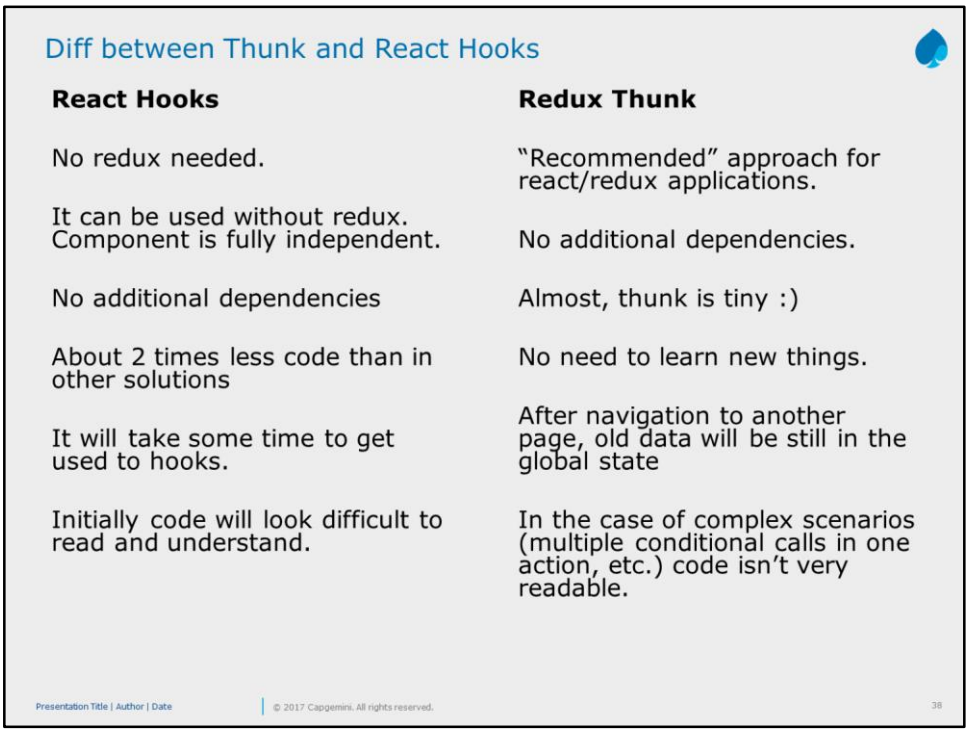
Middleware extends the store's abilities, and lets you write async logic that interacts with the store.

Thunks are the recommended middleware for basic Redux side effects logic, including complex synchronous logic that needs access to the store, and simple async logic like AJAX requests.

Instructor Notes:



Instructor Notes:



Instructor Notes:

Add instructor notes here.

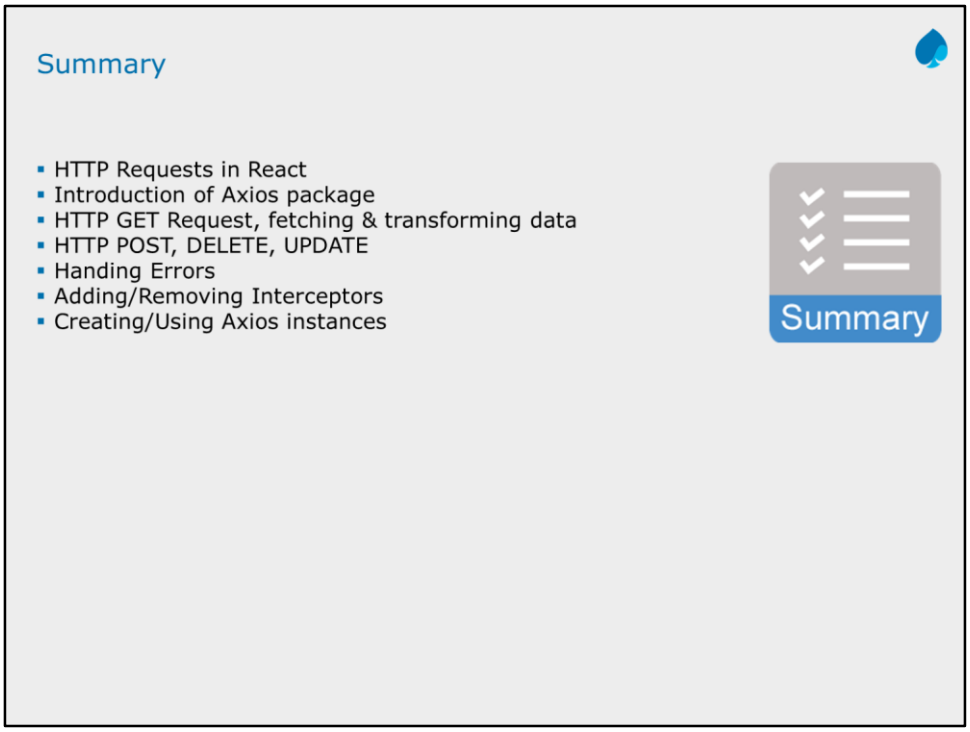
Demo

Redux-Thunk-Demo

Add the notes here.

Instructor Notes:

Add instructor notes here.



Add the notes here.