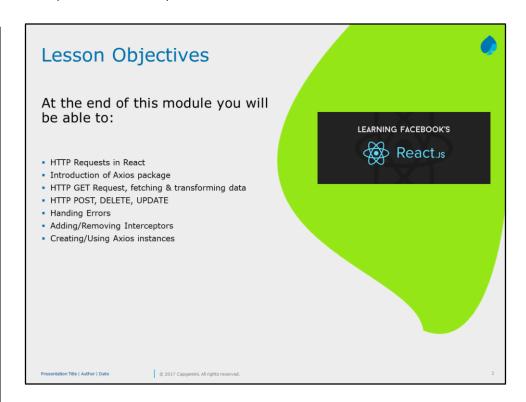
Add instructor notes here.



Add instructor notes here.



### HTTP Requests in React

0

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  $\mbox{API}'\mbox{s}$  are by using Axios and Fetch  $\mbox{API}.$ 

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.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

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.

### Axios



Axios is easy to use promise based Http Client for the browser and node.js

For more readable code and asynchronous code,we can take advantage of async and await since Axios is Promise based.

We can intercept the request and even cancel the request using Axios.

Client-side protection against cross-site request forgery protection is also inbuilt.

You can Install Axios using below command:

npm install --save axios

Import the axios in JSX file as below

Import Axios from 'axios';

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.

### Features of Axios:



- ✓ Protection against XSRF
- ✓ Make XMLHttpRequest from the browser
- ✓ Make http requests from node.js
- √ Streamlined Error Handling
- ✓ Response Timeout
- ✓ Ability to Cancel Request
- ✓ Support Older Browsers
- √ Support for Upload Progress
- ✓ Automatic JSON data transformation
- ✓ Intercepting Request & response

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!



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.

```
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 (
    { this.state.customers.map(customer =>
              {customer.name})}
    ) } }
```

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.

### 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

```
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.

```
### Proceedings of the International Configuration (International Configuration (Internation (Interna
```

### 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()
```

# **Transforming Data**



Consider there is a scenario while dealing with an API, that expect the data in a particular format, something like Xml or csv format.

Axios will automatically converts the request and response to JSON.

It is flexible enough where we can override the default behavior and define different transform mechanism as per our requirement.

When we want to change the request data, before sending it to server, set the transformRequest property in config object.

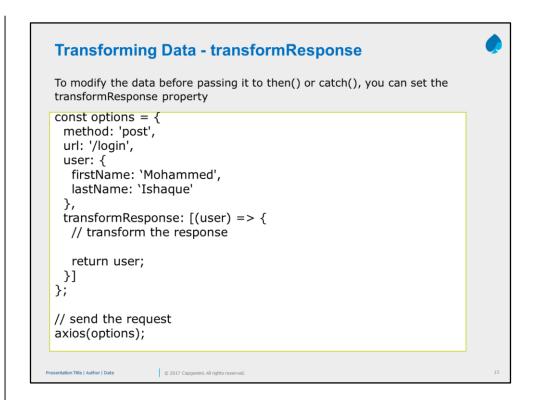
You can use this method only for HTTP PUT, POST and PATCH request methods.

Presentation Title | Author | Dat

© 2017 Capgemini. All rights reserved.

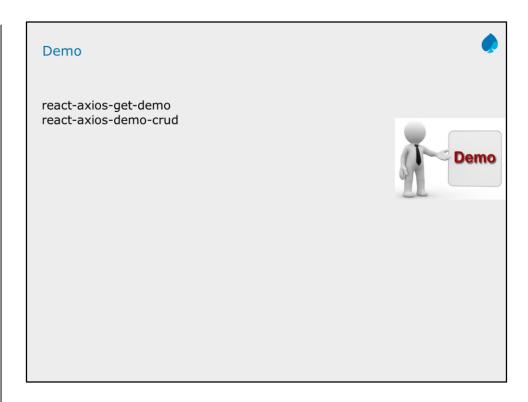
```
Transforming Data - transformRequest
 const options = {
  method: 'post',
  url: '/login',
  user: {
    firstName: 'Mohammed',
    lastName: 'Ishaque'
  transformRequest: [(user, headers) => {
    // transform the data
    return user;
  }]
 };
 // send the request
 axios(options);
Presentation Title | Author | Date
                   © 2017 Capgemini. All rights reserved.
```

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.



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

Add instructor notes here.



Add the notes here.

Slide explains regarding the SOAP format

# Error Handling in Axios using catch()



- As we already discussed, Axios requests are Promises.
- It has a then() function for promise changing and catch() function for handling errors.
- See below Example for handling error using catch() method.

```
const err = await axios.get('https://someurl.com/status/200').
// Will throw a TypeError because the property doesn't exist.
then(res => res.doesNotExist.throwAnError).
catch(err => err);
```

err instanceof TypeError;

Presentation Title | Author | Date

2017 Capgemini, All rights reserve

Page 01-15

Slide explains regarding the SOAP format

# Error Handling in Axios using catch()

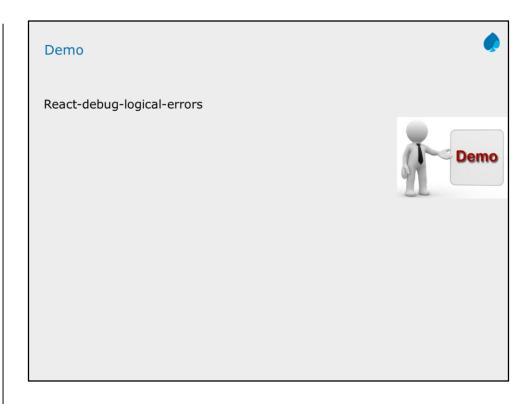


- Axios' catch() behaves exactly the same as the promise catch() function.
   So you can use promise chaining and add a catch() at the end to handle any errors that occur in the promise chain.
- You can also use catch() to transform the error, just make sure you rethrow the error afterwards.

Presentation Title | Author | Da

© 2017 Capgemini. All rights reserv

Add instructor notes here.



Add the notes here.

# 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.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

. . . .

# 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.

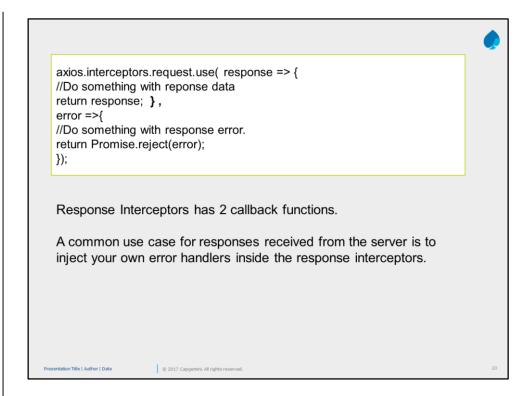
Config object includes option to modify your headers and authentication is a good use case for it.

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

Presentation Title | Author | Date

© 2017 Capgemini. All rights reser

Page 01-19



# Removing Interceptors



- We should actually remove the interceptors when componentWillMount() component gets unmounted.
- Above is a lifecycle method which is executed at the point of time a component isn't required anymore.

```
componentWillUnmount() {
  console.log("WillUnmount",this.reqInterceptor,this.resInterceptor.);
        axios.interceptors.request.eject(this.reqInterceptor);
        axios.interceptors.response.eject(this.resInterceptor);
}
```

Presentation Title | Author | Dat

© 2017 Capgemini. All rights reserved.

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

### Creating/Using Axios instances

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

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.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

2.

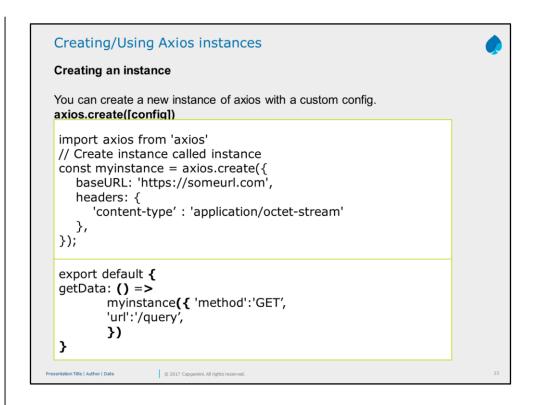
Consider a Scenario where you want not to use same BASE URL in your overall application. And you never want to use same Headers as well.

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.



### Redux

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.

- It is for Javascript Apps
- It is a State Container
- And It is predictable.
- Redux is not tied to React.
- It can be used with React, Angular, Vue etc.,

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

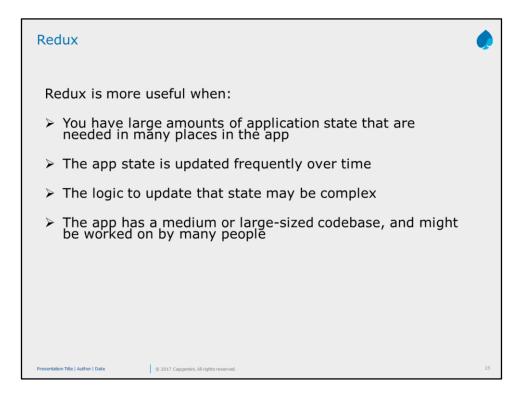
Redux is a predictable state container for JavaScript apps.

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.

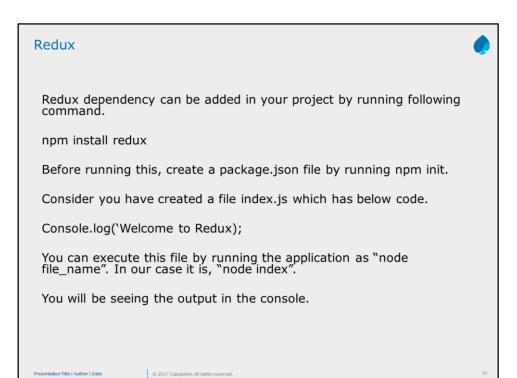
Redux is a third-party library which works totally independent of React.



### 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

Page 01-25



# Three Core Concepts

- > Store
- > Action
- > Reducer
- ✓ A Store that holds the state of your application.
- $\checkmark\,$  An action that describes the changes in the state of the Application.
- $\checkmark$  A reducer which carries out the state transition depending on the action.

### Actions



Only way your application can interact with store is using actions.

To the redux store Actions will carry some information's from your app.

It is Plain Java Script Objects.

It has a type property which will indicate what type of action is going to be performed.

The type property is typically defined as String constants.

It just describe what happens, but it won't describe how this action performs.

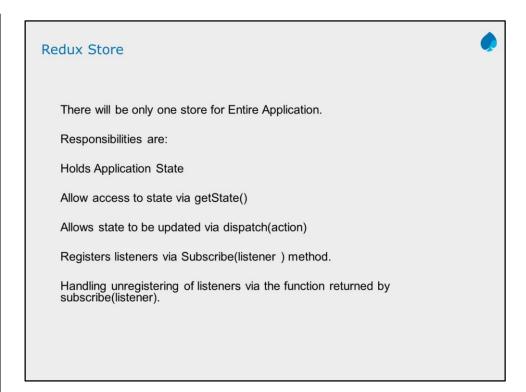
### Reducers



Specify how the app's state changes in response to actions sent to the store.

It is a function that accepts the state and action as an arguments and return the next state of the application.

We can mention as (previousState, action ) => newState



# Three Main Principles of Core Concepts

### First Principle:

The state of your whole application is stored in an Object tree within a single store.

Maintain our application state in a single object which would be managed by a redux store.

# Three Main Principles of Core Concepts



### Second Principle:

Only way to change state is to emit the action, an object describing what happened.

To update the state of your app, you need to let Redux know about that with an action.

You are not allowed to directly update the state object.

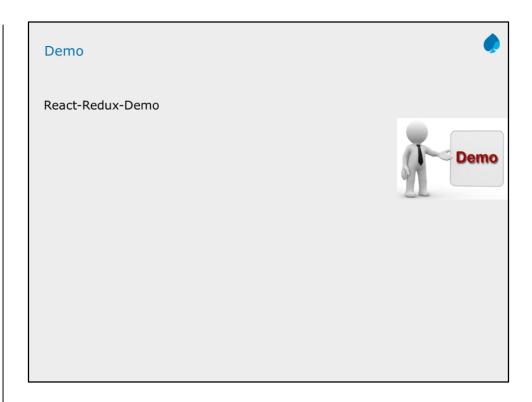
Three Main Principles of Core Concepts

### Third Principle:

You write Pure Reducers, to specify that how the state tree is transformed by actions.

Reducer – (previousState, action) => newState

Add instructor notes here.



Add the notes here.

### 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:

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.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

### 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:

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.

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

### Redux Thunk



What is a middleware? it is a piece of code that sits between your actions and your reducers.

It takes your actions does something to it before passing it down to the reducer. Think of it like a middle-man.

The createStore is used for creating the redux store while the applyMiddleware will be used for adding the thunk middleware.

we will import thunk from the redux-thunk package.

The most common use case for Redux Thunk is for communicating asynchronously with an external API to retrieve or save data

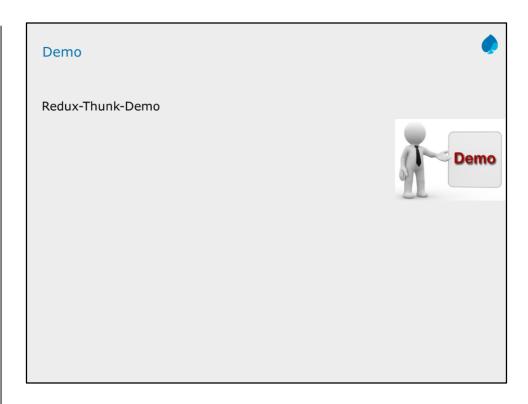
Redux Thunk makes it easy to dispatch actions that follow the lifecycle of a request to an external API.

Presentation Title | Author | Date

© 2017 Capgemini, All rights reserved.

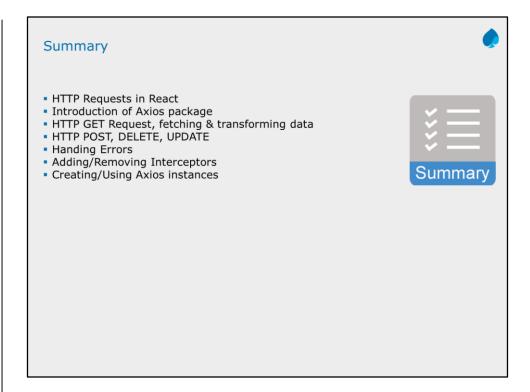
Diff between Thunk and React Hooks	
React Hooks	Redux Thunk
No redux needed.	"Recommended" approach for react/redux applications.
It can be used without redux. Component is fully independent.	No additional dependencies.
No additional dependencies	Almost, thunk is tiny :)
About 2 times less code than in other solutions	No need to learn new things.
It will take some time to get used to hooks.	After navigation to another page, old data will be still in the global state
Initially code will look difficult to read and understand.	In the case of complex scenarios (multiple conditional calls in one action, etc.) code isn't very readable.
Presentation Title   Author   Date © 2017 Capgemini. All rights reserved.	38

Add instructor notes here.



Add the notes here.

Add instructor notes here.



Add the notes here.