Axios
React Context

Agenda

Introduction to Axios

API Methods

Error Handling

Interceptors

Setting up Default Config

React Context API

HOCs

Axios is a promise-based HTTP client that works both in the browser and in a node.js environment

Installation:
Npm install axios –save
Bower install axios

Running Axios

Node

```
var axios = require('axios')
axios.get('https://api.github.com/users/codeheaven-io')
```

Browser

```
<script src="./bower_components/axios/dist/axios.js"></script>
<script>
axios.get('https://api.github.com/users/codeheaven-io');
</script>
```

Using Axios

```
import <any alias name> from 'axios';
 <alias name>.get(`http://www.reddit.com/r/${this.props.subreddit}.json`)
      .then(res => {
        const posts = res.data.data.children.map(obj => obj.data);
        this.setState({ posts });
      });
```

`http://www.reddit.com/r/${this.props.subreddit}.json` : template string part of ES6 syntax.

Axios API

```
// Performing a GET request
axios.get('https://api.github.com/users/' + username)
  .then(function(response){
    console.log(response.data); // ex.: { user: 'Your User'}
    console.log(response.status); // ex.: 200
  });
```

```
// Performing a POST request
axios.post('/save', { firstName: 'Marlon', lastName: 'Bernardes' })
  .then(function(response){
    console.log('saved successfully')
  });
```

To execute multiple requests in parallel, simply provide an array argument to **axios.all**. When all requests are complete, you'll receive an array containing the response objects in the same order they were sent. Alternatively you can use

Axios All

```
axios.all([
    axios.get(<URL>);
    axios.get(<URL>)
  ])
  .then(axios.spread(function (res1, res2) {
    console.log('User', res1.data, res2.data);
   }));
```

Custom Headers

```
var config = {
  headers: {'X-My-Custom-Header': 'Header-Value'}
};
axios.get(<URL>, config);
axios.post(<URL>, <DATA>, config);
```

Error Handling

```
axios.get(<URL>)
.catch(function(error){
if(error.response) {
Console.log(error.response)
}
Else{
console.log(error.message)
}
})
```

Interceptors

You can intercept **requests** or **responses** before they are handled by then or catch.

```
Axios.interceptors.request.use(function(config){
//TODO: before request is sent
return config
})
Axios.interceptors.response.use(function(response){
//TODO: modify response Data
return response;
})
```

Add or Remove Interceptors

```
Var interceptor = axios.interceptors.request.use(function(){...})
axios.interceptos.request.eject(interceptor)
```

Setting Up Default Config

- axios.defaults.baseURL = 'https://api.example.com';
- axios.defaults.headers.common['Authorization'] = AUTH_TOKEN;
- axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';

React Context API

Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language

Example (Upto react 15)

In Parent Component

```
getChildContext() {
  return {
       isLoggedIn: AuthService.isLoggedIn,
       user: this.props.user
  }
}
```

In Child

```
static contextTypes = {
  isLoggedIn: PropTypes.bool,
  user: PropTypes.object
}
```

And we can use the values in child component by using this.context

Example (React 16+)

```
const ThemeContext = React.createContext('light');

class App extends React.Component {
  render() {
    return (
      <ThemeContext.Provider value="dark">
        <Toolbar />
      </ThemeContext.Provider>
    );
  }
}

class ThemedButton extends React.Component {
  static contextType = ThemeContext;
  render() {
    return <Button theme={this.context} />;
  }
```

Example (React 16+)

```
<ThemeContext.Consumer>
  {value => /* render something based on the context value */}
</ThemeContext.Consumer>
```

HOC

A higher-order component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API, per se. They are a pattern that emerges from React’s compositional nature.

HOC Example

```
export const someHoc = (WrappedComponent) => {
  class SomeHOC extends Component {
   render() {
     return (
       <WrappedComponent {...this.props} userData={AuthService.userData} />
     );
   }
}
```

Exercise

create an HOC in React to get and set the login related info of current user in component props. The data can be logginStatus, userInfo etc.

use the context api of react to set the data and top and use the HOC to get the data in the child components.