UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT
CHAPTER 8 : CLIENT-SIDE SCRIPTING WITH JAVASCRIPT LIBRARY

LOO YIM LING
yllloo@utar.edu.my

Client-Side Scripting with Javascript Library

1) Front-end and Back-end Development
2) Client-Side Scripting Javascript Libraries
3) React JS for Client-Side Scripting
4) Communication between Client-Side and Server-Side

Client Side and Server Side Technologies

Information available on https://www.ironhack.com/en/web-development/front-end-vs-back-end-what-s-the-difference

Front-end and Back-end Development

1) Front-end development; practice of converting data and functionalities into a graphical interface.
   - HTML, CSS, JavaScript
2) Back-end development; practice of accessing, manipulating, delivering data and functionalities to a potential user.
   - PHP, Python, Databases
3) Full-stack development; practice of both front-end and back-end.

Client-Side Scripting Javascript Libraries

1) Bootstrap, Vue.js, Inertia.js, React.js, Angular.js...
2) One Global Objective
   - To construct the visuals and design of the app or website in question, in order to therefore generate a certain feeling among its users, so that they want to come back. Certainly not an easy feat.

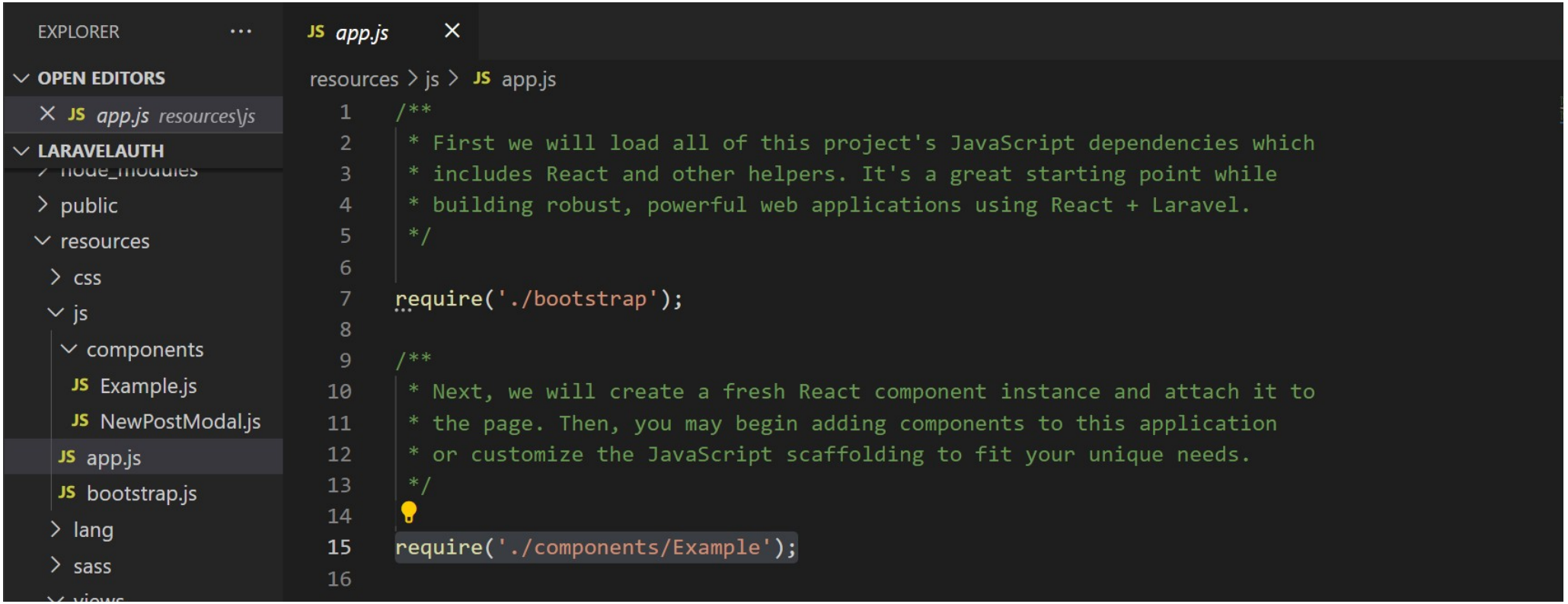
React JS for Client-Side Scripting

1) React is a JavaScript library for building user interfaces
2) As a pre-requisite to configure integration of React.js in a Laravel web application, dependency of laravel/ui need to be required through composer CLI
3) As covered in the previous topic, you can replace the Vue.js scaffolding or even create a new React.js Laravel project with artisan CLI.
4) Then NPM need to be installed to manage Javascript packages.

```
composer require laravel/ui
php artisan ui react
npm install && npm run dev
```

React JS Compiled Assets

1) In the file `resources/assets/js/app.js`, the Example component is included for any Laravel file that call/implement the component.



React JS Component

1) Notice that the `./bootstrap` required, bootstraps React and other required helper, then loads a React component named Example.
2) In React, build React components.
3) Components that are to load automatically when a web page is loaded should be added to the `resources/assets/js/app.js` file.

React JS Component

1) React Components are found in resources/js/components folder and identified by documentID.
2) Naming convention of a React component; Sentence case.

resources > js > components > JS Example.js > Example

```
1   import React, { Component } from 'react';
2   import ReactDOM from 'react-dom';
3   import { Table, Button, Modal, ModalHeader, ModalBody, ModalFooter, Input, FormGroup, Label
4   import axios from 'axios';
5   import NewPost from './NewPostModal';
6
7   export default class Example extends Component {
```

```
139   if (document.getElementById('example')) {
140       ReactDOM.render(<Example />, document.getElementById('example'));
141   }
```

Integrating React JS Component in Views

1) React Components are identified by documentID, thus integrating a React component in Laravel view file can be done by binding of documentID to tag.
2) Import the `app.css` and `app.js` into view file in order to integrate the css and js scaffold within the view file.

resources > views > welcome.blade.php

```
11        <link href="/css/app.css" rel="stylesheet">
12    </head>
13    <body class="antialiased">
14        <div id="example">></div>
15    </body>
16    <script src="/js/app.js"></script>
17 </html>
```

React JS Concepts: Components

1) Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
2) Conceptually, components are like JavaScript functions. They return React elements describing what should appear on the screen.
3) To define a Component:

```
function Example()
{
  return <h1>Hello</h1>;
}
export default Example;
```

React JS Concepts: Components

1) However, it is more common to use ES6 class to define a component. (NOTE: If you are doing React Native for Android/iOS development, you MUST use ES6 class):

```
class Example extends Component
{
  render(){
    return (<h1>Hello</h1>);
  }
}
export default Example;
```

React JS Concepts: Components

1) Components can refer to other components in their output. This lets allows usage of the same component abstraction for any level of detail.
   - A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.
   - For example, we can create an App component that renders Example components many times

React JS Concepts: Components

```
class Example extends Component
{
   render(){
      return (<h1>Hello</h1>);
   }
}
export default Example;

import Example from ‘./Example’
class App extends Component
{
   render(){
      return (
         <Example />
         <Example />
   );}
}
export default App;
```

React JS Concepts: States

1) State allows React components to change their output over time in response to user actions, network responses, and anything else.

```
7   export default class Example extends Component {
8       constructor(){
9           super()
10          this.state = {
11              posts: [], //response of API into post state
12              newPostModal: false,
13              newPostData: {title:"", content:"", user_id:""},
14              updatePostModal: false,
15              updatePostData: {id:"", title:"", content:"", user_id:"" }
16          }
17      }
18      loadPost(){
19          axios.get('http://127.0.0.1:8000/api/posts').then((response) => {
20              this.setState({
21                  posts:response.data
22              })
23          })
24      }
```

React JS Concepts: Props

1) Props are used to pass attributes to a component.
2) Props are readonly and the value must never be modified during the lifecycle of a component.

React JS Concepts: Props

```
class Example extends Component
{
   render(){
      return (<h1>Hello, {this.props.name}</h1>);
   }
}
export default Example;

import Example from `./Example'
class App extends Component
{
   render(){
      return (
         <Example name="Mary Jones"/>
         <Example name="Treble John"/>
   );}
}
export default App;
```

React JS Commonly-used Lifecycle Methods: render()

1) The only <u>required</u> method in a class component.
2) When called, it should examine this.state and this.props and return either React elements/components, arrays or fragments, portals, strings & numbers, Boolean or null.

```
render()
```

React JS Commonly-used Lifecycle Methods: constructor()

1) Called before the component is mounted.
   - When implementing the constructor for a React.Component subclass, call `super()` before any other statement.
2) Typically, constructors are only used for two purposes:
   - Initializing local state by assigning an object to `this.state`
   - Binding event handler methods to an instance

```
constructor(){
    super()
    this.state = { posts: [] }
}
```

React JS Commonly-used Lifecycle Methods: componentDidMount()

1) Invoked immediately after a component is mounted (inserted into the tree).
   - Initialization that requires DOM nodes should go here.
   - If you need to load data from a remote endpoint, this is a good place to instantiate the network request.
2) This method is a good place to set up any subscriptions.

```
componentDidMount()
```

React JS Commonly-used Lifecycle Methods: componentDidUpdate()

1) Invoked immediately after updating occurs. This method is NOT called for the initial render.
2) Use this as an opportunity to operate on the DOM when the component has been updated.
3) This is also a good place to do network requests as long as you compare the current props to previous props (e.g. a network request may not be necessary if the props have not changed)

```
componentDidUpdate(prevProps, prevState, snapshot)
```

React JS Commonly-used Lifecycle Methods: componentWillUnmount()

1) Invoked immediately before a component is unmounted and destroyed.
2) Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount() .
3) Do not call setState() because the component will never be re-rendered. Once a component instance is unmounted, it will never be mounted again.

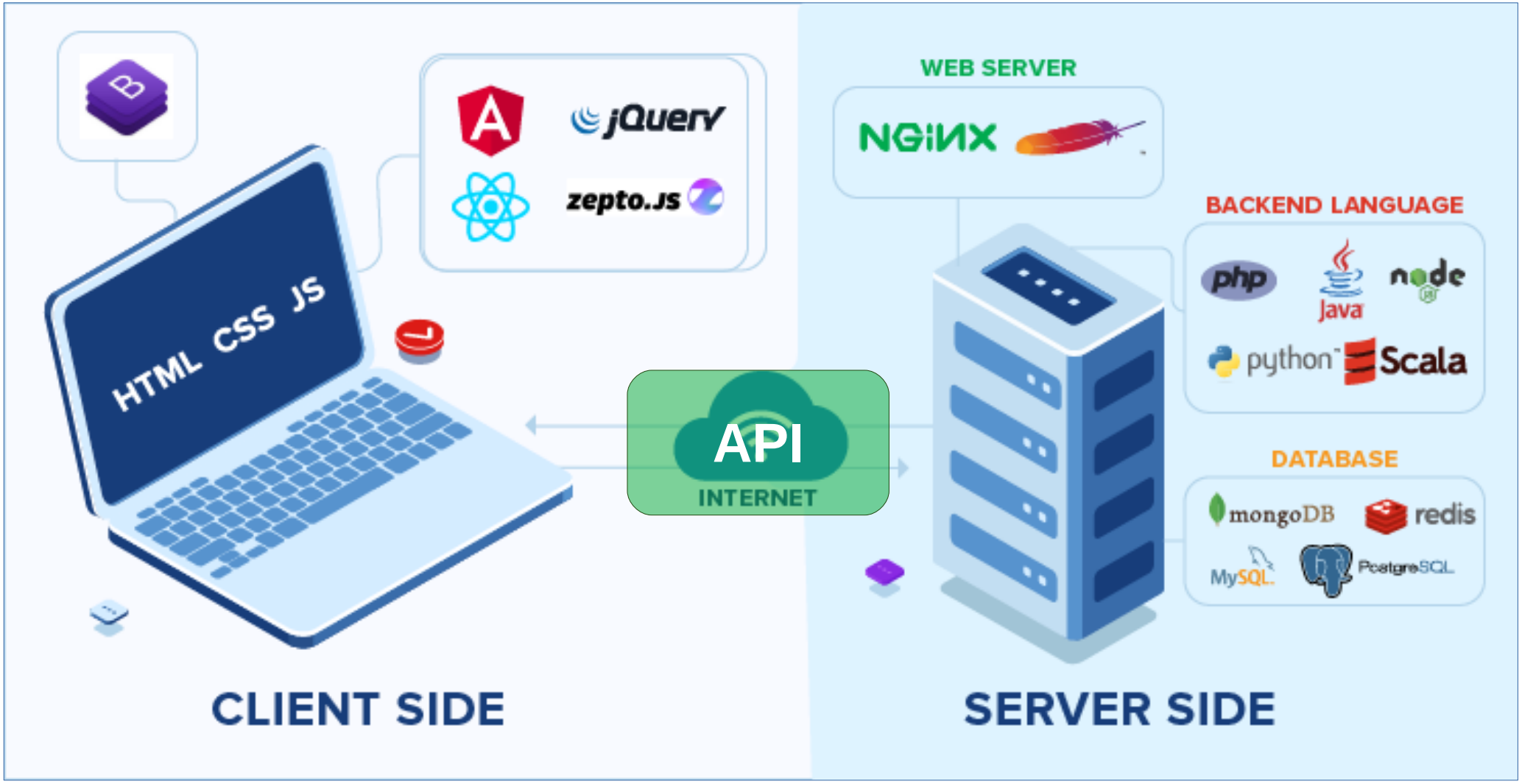
```
componentWillUnmount()
```

React JS Event Handling

1) A common pattern in ES6 is for an event handler to be a method on the class:

```
<Button color="success" size="sm" className="mr-2"
onClick={this.callUpdatePost.bind(this, post.id,
post.title, post.content, post.user_id)}> Edit
</Button>
...
onChange={(e) =>{
   let { updatePostData } = this.state
   updatePostData.content = e.target.value
   this.setState({updatePostData})
}}></Input>
```

Communication between Client-Side and Server-Side



Information available on https://www.ironhack.com/en/web-development/front-end-vs-back-end-what-s-the-difference

API: Application Programming Interface

1) API is software intermediary that allows two applications to talk to each other.
2) It is a set of rules that allow programs to talk to each other. The developer creates the API on the server and allows the client to talk to it.
3) In order to enable React JS client-side to communicate with Laravel framework server-side database, API need to be created.

RESTful API

1) REST determines how the API looks like. It stands for "Representational State Transfer".
2) It is a set of rules that developers follow when they create their API. One of these rules states that one should be able to get a piece of data (called a resource) when one link to a specific URL.
3) Each URL is called a request while the data sent back is called a response.

RESTful API: Methods

1) The method is the type of request you send to the server. You can choose from these five types below:
   - GET
   - POST
   - PUT
   - PATCH
   - DELETE
2) They are used to perform four possible actions: Create, Read, Update and Delete (CRUD).

RESTful API: Endpoints

1) RESTful API endpoints consist of the method, API name and the database access application logic as follows:

routes > api.php

```
Route::get('posts', [PostController::class, 'index']);
Route::post('post', [PostController::class, 'store']);
Route::put('post/{id}', [PostController::class, 'update']);
Route::delete('post/{id}', [PostController::class, 'destroy']);
```

RESTful API: JSON

1) JSON (JavaScript Object Notation) a common format for sending and requesting data through a REST API.
2) A JSON object looks like a JavaScript Object. In JSON, each property and value must be wrapped with double quotation marks.

```
{
   "name": "Doctor Strange",
   "email": "doctor@gmail.com",
   "password": "doctor1234"
}
```

RESTful API: HTTP Status Codes And Error Messages

1) HTTP status codes tells the status of the response quickly. The range from 100+ to 500+:
   - 200+ means the request has succeeded.
   - 300+ means the request is redirected to another URL
   - 400+ means an error that originates from the client has occurred
   - 500+ means an error that originates from the server has occurred.

```
return response()->json([
   'message' => 'User successfully registered',
   'user' => $user
   ], 201);
```

END OF LECTURE 09