Full Stack Development - I

COMP 3123

What is React Route ?

- A tool that allows you to handle routes in a web app, using ***dynamic routing***.
- Dynamic routing takes place as the app is rendering on your machine, unlike the old routing architecture where the routing is handled in a configuration outside of a running app.
- React router implements a component-based approach to routing.
- It provides different routing components according to the needs of the application and platform.
- React Router can be installed using the **npm cli** utility:

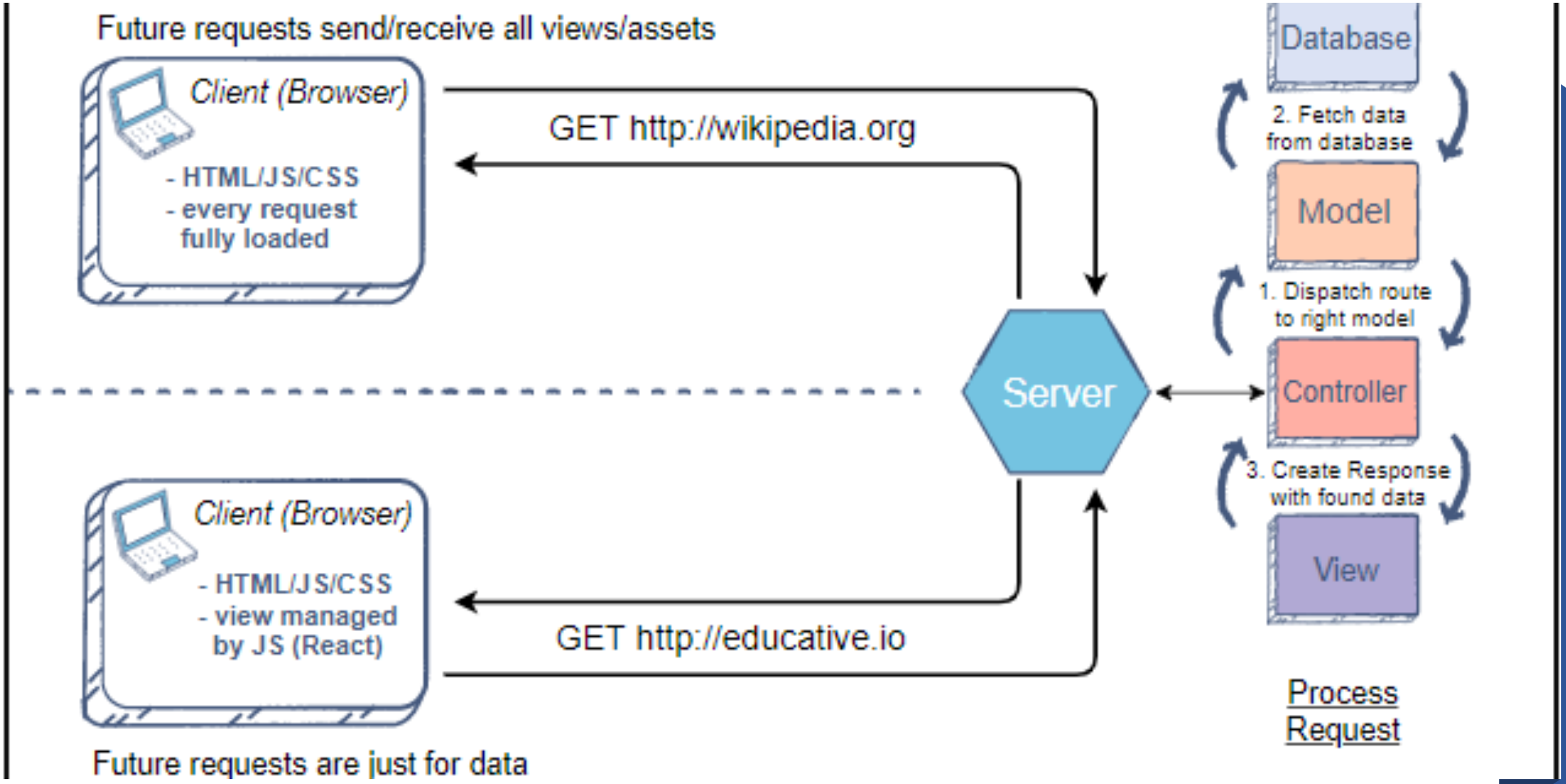
  npm install –save react-router

  npm install –save react-router-dom

React Route cont…

- In React, ***routers*** help create and **navigate** between the different URLs that make up your web application.
- They allow your user to **move between the components** of your app while preserving user state, and can provide unique URLs for these components to make them more shareable.
- With routers, you can improve your app's user experience by **simplifying** site.

Web reference Link


Future requests send/receive all views/assets
Client (Browser)
- HTML/JS/CSS
- every request fully loaded
GET http://wikipedia.org
Server
Database
2. Fetch data from database
Model
1. Dispatch route to right model
Controller
3. Create Response with found data
View
Process Request
Client (Browser)
- HTML/JS/CSS
- view managed by JS (React)
GET http://educative.io
Future requests are just for data

The Router

- When starting a new project, we need to determine which type of router to use.
- For browser based projects, there are **<BrowserRouter>** and **<HashRouter>** components.
  - <BrowserRouter> should be used when you have a server that will handle dynamic requests (knows how to respond to any possible URI).
  - <HashRouter> should be used for static websites (where the server can only respond to requests for files that it knows about).
- Usually it is preferable to use a <BrowserRouter>, but if your website will be hosted on a server that only serves static files, then the <HashRouter> is a good solution.

Web reference Link

Syntax- React Route

```
<Route exact path="/specifiedPath" component={componentName} />
```

Create Components

In this step, we will create four components. The **App** component will be used as a tab menu. The other three components **(Home), (About)** and **(Contact)** are rendered once the route has changed.

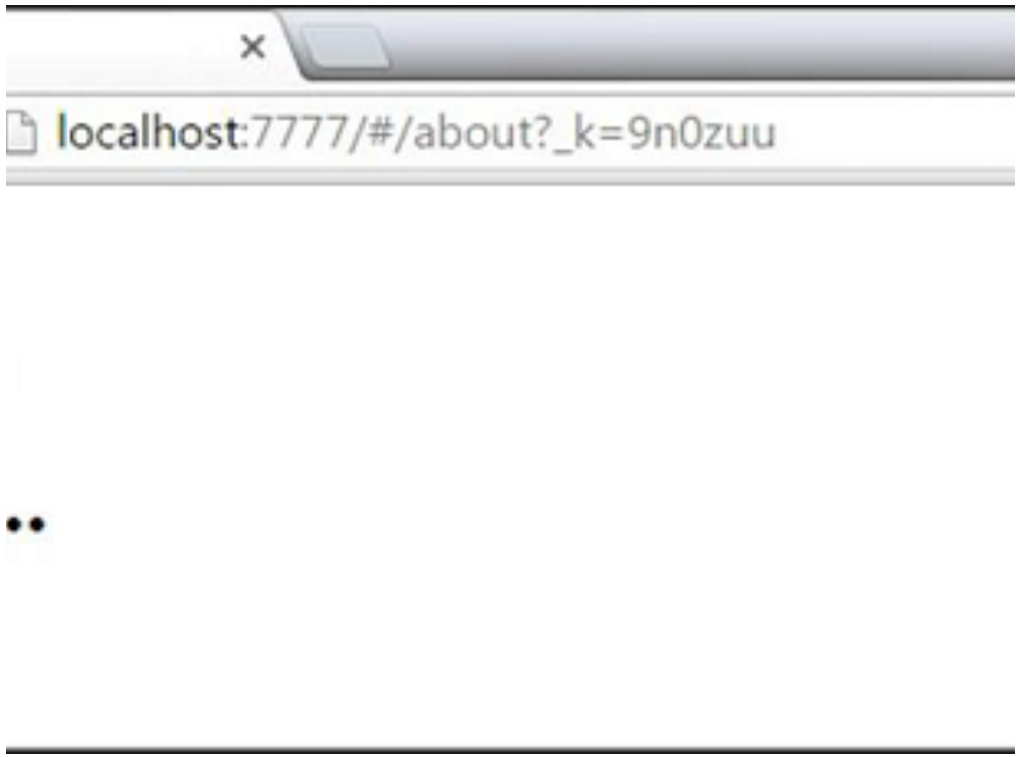
Web reference Link

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Router, Route, Link, BrowserHistory, IndexRoute } from 'react-router'

class App extends React.Component {
  render() {
    return (
      <div>
        <ul>
        <li>Home</li>
        <li>About</li>
        <li>Contact</li>
        </ul>
        {this.props.children}
      </div>
    )
  }
}
export default App;
```

Add a Router

```
ReactDOM.render((
  <Router history = {browserHistory}>
    <Route path = "/" component = {App}>
      <IndexRoute component = {Home} />
      <Route path = "home" component = {Home} />
      <Route path = "about" component = {About} />
      <Route path = "contact" component = {Contact} />
    </Route>
  </Router>
), document.getElementById('app'))
```



React Forms

- HTML form elements work a little bit differently from other DOM elements in React, because form elements naturally keep some internal state.
- For example, this form in plain HTML accepts a single name:

```
<form>
  <label>
    Name:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Submit" />
</form>
```

- This form has the default HTML form behavior of browsing to a new page when the user submits the form.
- If you want this behavior in React, it just works. But in most cases, it's convenient to have a JavaScript function that handles the submission of the form and has access to the data that the user entered into the form.
- The standard way to achieve this is with a technique called "**controlled components**".

Web reference Link

Controlled Components

- In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input.
- In React, **mutable state** is typically kept in the state property of components, and only updated with setState().
- We can combine the two by making the React state be the “single source of truth”.
- Then the React component that renders a form also controls what happens in that form on subsequent user input.
- An input form element whose value is controlled by React in this way is called a “controlled component”.
- For example, if we want to make the previous example log the name when it is submitted, we can write the form as a controlled component:

View Demo

```
class NameForm extends React.Component {
 constructor(props) {
  super(props);
  this.state = {value: ''};

  this.handleChange = this.handleChange.bind(this);
  this.handleSubmit = this.handleSubmit.bind(this);
 }

 handleChange(event) {
  this.setState({value: event.target.value});
 }

 handleSubmit(event) {
  alert('A name was submitted: ' + this.state.value);
  event.preventDefault();
 }
```

```
 render() {
  return (
   <form onSubmit={this.handleSubmit}>
    <label>
     Name:
     <input type="text" value={this.state.value} onChange={this.handleChange} />
    </label>
    <input type="submit" value="Submit" />
   </form>
  );
 }
}
```

Since the value attribute is set on our form element, the displayed value will always be this.state.value, making the React state the source of truth. Since handleChange runs on every keystroke to update the React state, the displayed value will update as the user types.
With a controlled component, the input's value is always driven by the React state. While this means you have to type a bit more code, you can now pass the value to other UI elements too, or reset it from other event handlers.

React Form input validation

The Form component takes in the following props:

- **errors:** a key value pair object where the key is the field, and the value is the error detected for that field. If there is no error on the field, the key will not be in the object.
- **touched:** a key value pair object where the key is the field, and the value is true if the field has been modified. The key is present only if the user has touched the field (modified its value or triggered an onBlur event). This allows us to know whether or not to display an error.
- **values:** a key value pair where the key is the name of the field, and the value is the value of the field (either initial value, or what the user entered)
- **handleSubmit:** a function passed to the handleSubmit prop of the form tag. This is the logic that will be executed when the user clicks on the Submit button.
- **handleChange:** a function passed to the handleChange prop of the input tags. This is the logic that will save what the user enters in the field.
- **handleBlur:** a function passed to the handleBlur prop of the input. This is the logic that will validate the field value.

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
class MyForm extends React.Component {
 constructor(props) {
  super(props);
  this.state = {
   username: '',
   age: null,
  };
 }
 mySubmitHandler = (event) => {
  event.preventDefault();
  let age = this.state.age;
  if (!Number(age)) {
   alert("Your age must be a number");
  }
 }
 myChangeHandler = (event) => {
  let nam = event.target.name;
  let val = event.target.value;
  this.setState({[nam]: val});
 }
 render() {
  return (
   <form onSubmit={this.mySubmitHandler}>
   <h1>Hello {this.state.username} {this.state.age}</h1>
   <p>Enter your name:</p>
   <input
    type='text'
    name='username'
    onChange={this.myChangeHandler}
   />
   <p>Enter your age:</p>
   <input
    type='text'
    name='age'
    onChange={this.myChangeHandler}
   />
   <br/>
   <br/>
   <input type='submit' />
   </form>
  );
 }
}

ReactDOM.render(<MyForm />, document.getElementById('root'));
```

Using Axios

- Many projects on the web need to interface with a REST API at some stage in their development.
- Axios is a lightweight **HTTP client** based on the **$http service** within Angular.js v1.x and is similar to the **native JavaScript Fetch API**.
- Axios is **promise-based**, which gives you the ability to take advantage of JavaScript's async and await for more readable asynchronous code.
- You can also **intercept and cancel requests**, and there's built-in client-side protection against cross-site request forgery.

Web Reference Link

Adding Axios to the Project

- To add Axios to the project, open your terminal and change directories into your project: run this command to install Axios:

  npm install –save axios

- 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 a GET Request

```
import React from 'react';
import axios from 'axios';
export default class PersonList extends React.Component {
  state = {
    persons: []
  }
  componentDidMount() {
    axios.get(`https://jsonplaceholder.typicode.com/users`)
      .then(res => {
        const persons = res.data;
        this.setState({ persons });
      })
  }
  render() {
    return (
      <ul>
        { this.state.persons.map(person => <li>{person.name}</li>)}
      </ul>
    )
  }
}
```

Making a POST Request

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

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

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

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

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

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

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

Thank You