Welcome to the CoGrammar Tutorial: Hooks and Routing

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: **Feedback on Lectures**

Skills Bootcamp
8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

CoGrammar

State Hook

Hook used for state management, allowing components to store and retrieve information.

- The **useState** hook declares a **state variable**, which is **preserved between function calls** and whose **change triggers a rerender**.
- The function **accepts** the **initial state** of the variable as input.
- The function **returns** a pair of values: the **state variable** and the **function that updates it**.

```
const [number, setNumber] = useState(10);
const [string, setString] = useState("");
const [object, setObject] = useState({
    attribute1: "Name",
    attribute2: 23,
    attribute3: false });
```

CoGrammar

Function Components Recap: JavaScript functions which accept a single prop object as input and use hooks to create reusable pieces of UI by returning React elements.

```
import React, { useState } from 'react';

function Counter () {
    let [count, setCount] = useState(0);

    function inc () {
        setCount(count + 1);
    }

    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={inc} >Increment</button>
        </div>
    )
}

export default Counter;
```

CoGrammar

Effect Hook

Hook used for connecting to and synchronizing external systems after your components are rendered, known as performing side effects.

- The **useEffect** hook is used for tasks like **fetching data**, directly **updating the DOM** and setting up **event listeners**.
- The function takes in two arguments: a **block of code** which will be executed when the component is loaded, and a **dependencies list**, which is a list of variables whose change will trigger the first argument to be rerun.
- If **no dependency argument** is passed, the first argument will run on **every render**.
- If an **empty dependency argument** is passed, the first argument will on be run on the first render of the component.

CoGrammar

Fetch Data from API

```
import React, { useState, useEffect } from 'react';

function API() {
  let [funFact, setFunFact] = useState(null);

  useEffect(() => {
    async function fetchData() {
      let response = await fetch("https://catfact.ninja/fact/");
      let data = await response.json();
      console.log(data.fact)
      setFunFact(data.fact);
    }
    fetchData();
  },[])

  return (
    <h1>{funFact}</h1>
  )
}
export default API;
```

CoGrammar

Cleanup Function

Function returned by the useEffect hook which gets executed before every rerun of the component and after the component is removed.

- Tasks that can be performed in the useEffect hook, may need to be **aborted or stopped** when the **component is removed** or when **state changes**.
- For example, API calls may need to be aborted, timers stopped and connections removed.
- If this is not handled properly, your code may attempt to update a state variable which no longer exists, resulting in a **memory leak**.
- This is done with a **cleanup function**, which is **returned by the useEffect hook**. This function will run when the component is **removed** or **rerendered**.

CoGrammar

Cleanup Function

```
import { useEffect } from 'react';

function SweepAway () {
  useEffect(() => {
    const clicked = () => console.log('window clicked')
    window.addEventListener('click', clicked)

    // return a clean-up function
    return () => {
      window.removeEventListener('click', clicked)
    }
  }, [])

  return (
    <div>When you click the window you'll find a
          message logged to the console</div>
  )
}
```

CoGrammar

Ref Hook

Hook used to store mutable values which do not trigger re-renders and update DOM elements directly.

- The **useRef** hook is store values which **persist between re-renders**, but **do not cause the component to re-render** when changed.
- We can also access DOM elements using useRef by passing the returned object to elements in the **ref** attribute.
- The function accepts an **initial value** as an **input**.
- The function returns an **object** with the property **current** initialised to the value passed as input to the function.

CoGrammar

Ref Hook

```
import { useRef } from 'react';

function PetCat () {

    let pet = useRef(0);

    function handleClick() {
        pet.current = pet.current + 1;
        alert('You clicked ' + pet.current + ' times!');
      }

  return (
    <div>
        <button onClick={handleClick}> Pet the virtual cat! </button>
    </div>
  )
}

export default PetCat;
```

CoGrammar

Routing in React

- In the context of React, **client side routing** is executed.
- This allows your app to **update the URL from a link click** without making another request for another document from the server, making your application **render immediately**.
- In simple terms, routing in React involves **dynamically updating the content** of the website without reloading the entire page.
- Routing in React is mostly implemented using **routing libraries** or frameworks. Two common libraries in use for a seamless routing experience are **React Router DOM** and **Reach Router**.

CoGrammar

React Router DOM

Achieves client side routing in your React application by using its inbuilt routing APIs.

- To use React Router in your application, you need to install it first using npm or yarn

Terminall.sh

```
$ npm install react-router-dom
```

CoGrammar

Configuration

- After installing React Router, you need to configure your app to use it. This will be done in the root of you Javascript file (index.js).

index.js

```
//other React imports
import { createBrowserRouter, RouterProvider } from 'react-router-dom';

const paths = createBrowserRouter([
  {
    path: '/',
    element: <h1>Hello World</h1>
  }
])


const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <RouterProvider router={paths} /> {/** replaced <App/> */}
  </React.StrictMode>
);
```

CoGrammar

React Router APIs

- From the configuration example shown, we made two important imports:
  1. **createBrowserRouter**: this configures Browser Router which enables client side Routing in our React application.
     - It is a function that takes in a list of available paths in our application, the paths will be defined by objects.
     - Currently, we've only created one path which is the home path using a '/' and it renders a <h1> text saying Hello World.

CoGrammar

React Router APIs

2. **RouterProvider**: All path objects created by the createBrowserRouter API are passed to the provider component as a value of the router prop to render your app and enable routing.

- After this configuration, upon running your React server, you will have a text displaying Hello World on the home page.

CoGrammar

Multiple Pages

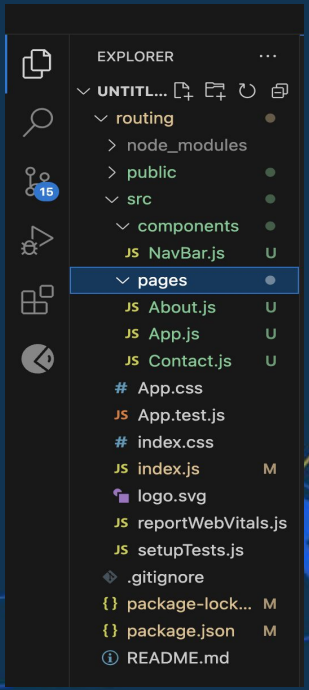
index.js

```
//other React imports
import App from './pages/App';
import About from './pages/About'
import Contact from './pages/Contact'
import { createBrowserRouter, RouterProvider } from 'react-router-dom';

const paths = createBrowserRouter([
  {
    path: '/',
    element: <App/>
  },
  {
    path: '/about',
    element: <About/>
  },
  {
    path: '/contact',
    element: <Contact/>
  }
])


const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <RouterProvider router={paths} />
  </React.StrictMode>
);
```

Folder structure:



CoGrammar

Navigating through React Router pages

- For hyperlinks, we are used to utilizing the <a> tag in HTML. Using <a href=''> causes a page refresh which can lead to losing an application's state.
- To achieve complete client side routing with React Router, we use its <Link> element to navigate from page to page. Instead of the {href='/path'} attribute in <a> tags, the link element provides a { to='/path'} property to direct the link to the desired URL path.
- The <Link> element does not cause a page refresh hence the application's state cannot be lost.

CoGrammar

Example

Note that the structure of the App component is also implemented on the About and Contact component

- The { Link } element is imported from ‘react-router-dom’
- You can also user { NavLink } to know whether a page is active or not.

NavBar.js

```
import { Link } from "react-router-dom"

const NavBar = () =>{
    return (
        <nav>
            <Link to="/">Home</Link>
            <Link to="/about">About</Link>
            <Link to="/contact">Contact</Link>
        </nav>
    )
}

export default NavBar
```

App.js

```
import NavBar from "../components/NavBar"

function App () {
    return (
        <section>
            <NavBar/>
            <h1>Home</h1>
        </section>
    )
}

export default App
```

CoGrammar

Dynamic Routing

- Dynamic routing is a way of rendering a new component by updating a particular segment in the URL called params.
- We achieve this by adding { :id } to the path, the colon section of the path will represent the dynamic segment. The suffix of the path will be replaced by respective path id or name.
- Note that you can name the id to anything as long as it rhymes with the intention. i.e { :itemId }, { :userId }.

Example

index.js

```javascript
//other React imports
import App from './pages/App';
import About from './pages/About'
import Contact from './pages/Contact'
import User from './pages/User';
import { createBrowserRouter, RouterProvider } from 'react-router-dom';


const paths = createBrowserRouter([
  {
    path: '/',
    element: <App/>
  },
  {
    path: '/about',
    element: <About/>
  },
  {
    path: '/contact',
    element: <Contact/>
  },
  {
    path: '/user/:userId', //dynamic path, has the /:userId suffix
    element: <User/>
  }
])
//other configurations
```

NavBar.js

```javascript
import { Link } from "react-router-dom"

const NavBar = () =>{
    return (
        <nav>
            <Link to="/">Home</Link>
            <Link to="/about">About</Link>
            <Link to="/contact">Contact</Link>
            <Link to="/user/1">User 1</Link>
            <Link to="/user/2">User 2</Link>
            <Link to="/user/3">User 3</Link>
        </nav>
    )
}

export default NavBar
```

CoGrammar

useParams() Hook

- The useParams hook returns an object of key/value pairs of the dynamic params from the current URL matched by the dynamic path.
- We created a User.js component that utilized the useParams to access the params of the { /user/:userId } path.

User.js

```
import { useParams } from "react-router-dom";

const User = ()=>{
    const { userId } = useParams()
    return (
        <section>
            User: { userId }
        </section>
    )
}

export default User;
```

CoGrammar

Passing State Variables

- State can be passed via the <Link> element in the same way we pass props to components. We use an extra prop called { state }.
- State can also be passed via a useNavigate hook provided by React Router which returns a function that lets you navigate programmatically.
- To access the state, we use a useLocation hook which returns a location object with the state property containing the state passed from the component.

Passing State

Using <Link state={data}>

NavBar.js

```
import { Link } from "react-router-dom"

const NavBar = () =>{

    const user1 = {
        id: 1,
        name: 'user1',
        role: 'Frontend Developer'
    }
    const user2 = {
        id: 2,
        name: 'user2',
        role: 'Backend Developer'
    }

    return (
        <nav>
            <Link to="/">Home</Link>
            <Link to="/about">About</Link>
            <Link to="/contact">Contact</Link>
            <Link to="/user/1" state={user1}>User 1</Link>
            <Link to="/user/2" state={user2}>User 2</Link>
            {/**other Link tags */}
```

Using useNavigate hook

NavBar.js

```
import { Link, useNavigate } from "react-router-dom"

const NavBar = () =>{
    const navigate = useNavigate()

    const user1 = {
        id: 1,
        name: 'Dan',
        role: 'Frontend Developer'
    }
    const user2 = {
        id: 2,
        name: 'Walobwa',
        role: 'Backend Developer'
    }

    const handleNavigatestate = (id, userData)=>{
        navigate(`/user/${id}`, { state: userData})

    }

    return (
        <nav>
            <Link to="/">Home</Link>
            <Link to="/about">About</Link>
            <Link to="/contact">Contact</Link>
            <button onClick={()=>handleNavigatestate(user1.id, user1)}>User 1</button>
            <button onClick={()=>handleNavigatestate(user2.id, user2)}>User 2</button>
            {/**other Link tags */}
```

CoGrammar

useLocation hook

- The useLocation hook is used to access the state passed from its respective dynamic path. We access state from the location object returned by the useLocation hook.
- You need to import useLocation from React Router in order to use it. This gives access to data passed from both the <Link> element and the useNavigate hook.

```
import { useParams, useLocation } from "react-router-dom";

const User = ()=>{
    const { userId } = useParams()

    //accessing state using use location
    const location = useLocation()
    const userData = location.state
    return (
        <section>
            <p>User: { userId }</p>
            <p>Name: { userData.name}</p>
            <p>Role: { userData.role}</p>
        </section>
    )
}

export default User;
```

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar