Welcome to the CoGrammar

Tutorial: Introduction to React Routing and Version Control with Git and GitHub

The session will start shortly...

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

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

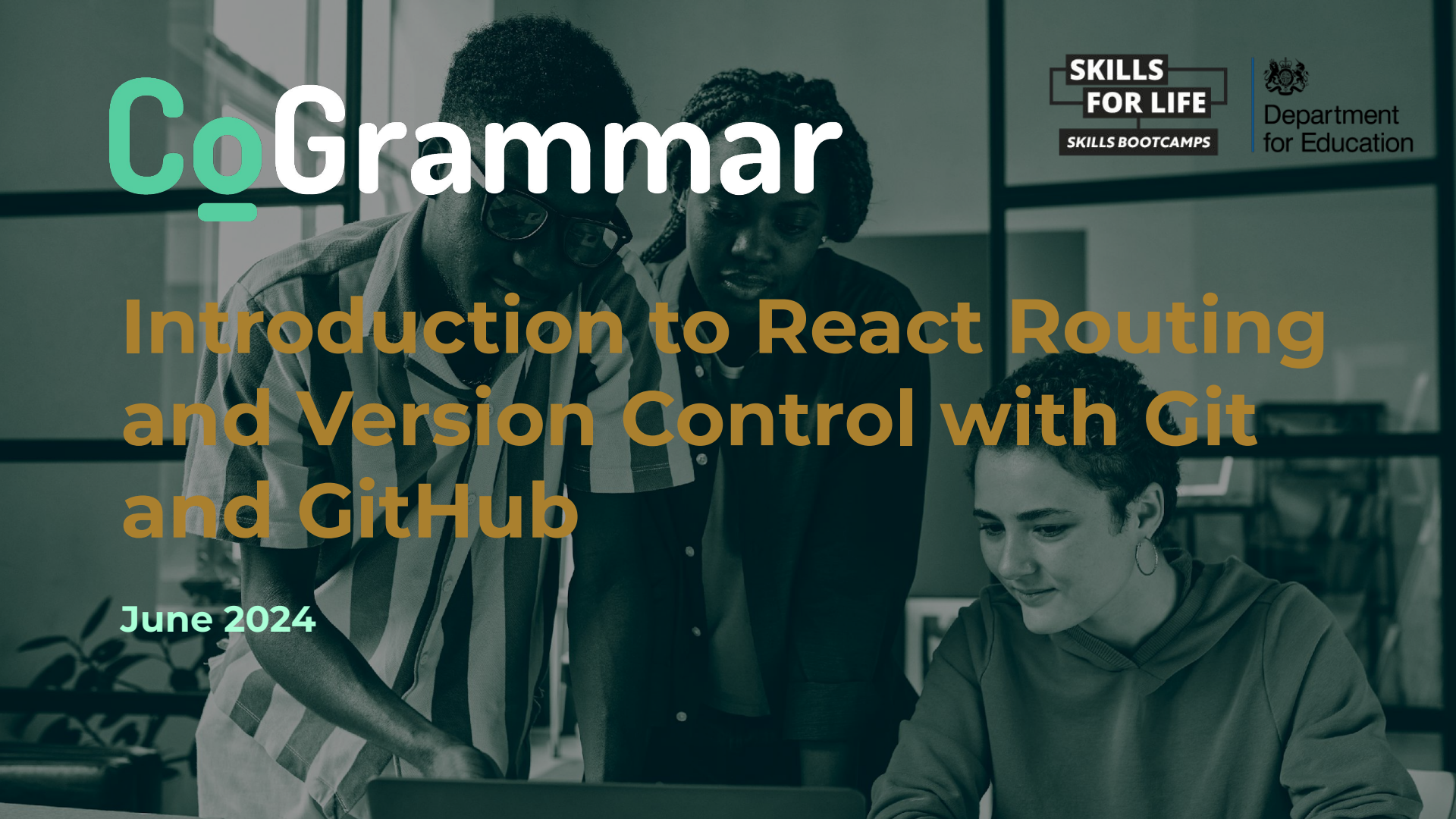
Skills Bootcamp Progression Overview

✅ 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

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Introduction to React Routing and Version Control with Git and GitHub

June 2024

Lesson Objectives

- Understanding the concept of Routing and its purpose in web applications.
- Configure Routing using **react-router-dom** and understand it’s core components.
- Understanding Version Control with Git and GitHub

Routing

Definition and Use Cases

- Routing can be termed as the **conditional rendering** of components based on the **URL** in the browser.
- Routing allows users to **navigate between different pages or views** within a web application.
- Routing with plain HTML/CSS used to be **file based**, the anchor (<a>) were used to create hyperlinks that link to different web pages which were the different (.html) files in your project.

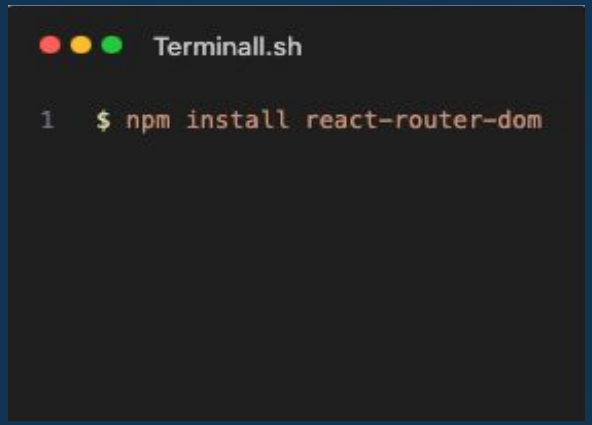
Routing in React

- 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**.

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



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

```
7   //other React imports
8   import { createBrowserRouter, RouterProvider } from 'react-router-dom';
9
10  const paths = createBrowserRouter([
11    {
12      path: '/',
13      element: <h1>Hello World</h1>
14    }
15  ])
16
17
18  const root = ReactDOM.createRoot(document.getElementById('root'));
19  root.render(
20    <React.StrictMode>
21      <RouterProvider router={paths} /> {/** replaced <App/> */}
22    </React.StrictMode>
23  );
```

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.

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.

Multiple Pages

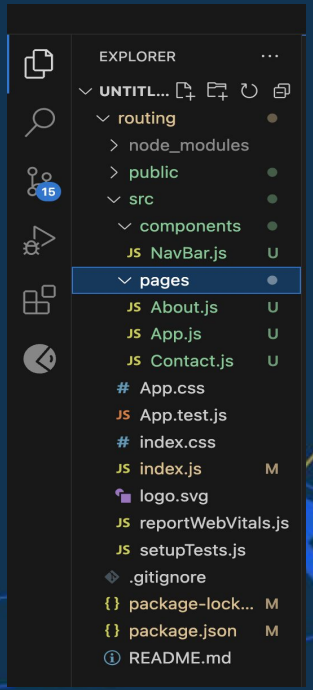
- Having multiple pages in our React app is one of the main achievements of routing.
- We do this by creating **other path objects** and **pointing** the path elements to their specific components.
- The **element property** of the path object will be replaced by a **React component from your project**.
- In this case, we have three components representing three pages and all are stored in a folder called pages for best practice purpose.

Multiple Pages

index.js

```
6   //other React imports
7   import App from './pages/App';
8   import About from './pages/About'
9   import Contact from './pages/Contact'
10  import { createBrowserRouter, RouterProvider } from 'react-router-dom';
11
12  const paths = createBrowserRouter([
13    {
14      path: '/',
15      element: <App/>
16    },
17    {
18      path: '/about',
19      element: <About/>
20    },
21    {
22      path: '/contact',
23      element: <Contact/>
24    }
25  ])
26
27
28  const root = ReactDOM.createRoot(document.getElementById('root'));
29  root.render(
30    <React.StrictMode>
31      <RouterProvider router={paths} />
32    </React.StrictMode>
33  );
```

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.

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

```
1   import { Link } from "react-router-dom"
2
3   const NavBar = () =>{
4       return (
5           <nav>
6               <Link to="/">Home</Link>
7               <Link to="/about">About</Link>
8               <Link to="/contact">Contact</Link>
9           </nav>
10      )
11  }
12
13  export default NavBar
```

App.js

```
1   import NavBar from "../components/NavBar"
2
3   function App () {
4       return (
5           <section>
6               <NavBar/>
7               <h1>Home</h1>
8           </section>
9       )
10  }
11
12  export default App
```

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

```
6   //other React imports
7   import App from './pages/App';
8   import About from './pages/About'
9   import Contact from './pages/Contact'
10  import User from './pages/User';
11  import { createBrowserRouter, RouterProvider } from 'react-router-dom';
12
13
14  const paths = createBrowserRouter([
15    {
16      path: '/',
17      element: <App/>
18    },
19    {
20      path: '/about',
21      element: <About/>
22    },
23    {
24      path: '/contact',
25      element: <Contact/>
26    },
27    {
28      path: '/user/:userId', //dynamic path, has the /:userId suffix
29      element: <User/>
30    }
31  ])
32  //other configurations
```

NavBar.js

```
1   import { Link } from "react-router-dom"
2
3   const NavBar = () =>{
4       return (
5           <nav>
6               <Link to="/">Home</Link>
7               <Link to="/about">About</Link>
8               <Link to="/contact">Contact</Link>
9               <Link to="/user/1">User 1</Link>
10              <Link to="/user/2">User 2</Link>
11              <Link to="/user/3">User 3</Link>
12          </nav>
13      )
14  }
15
16  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

```
1   import { useParams } from "react-router-dom";
2
3   const User = ()=>{
4       const { userId } = useParams()
5       return (
6           <section>
7               User: { userId }
8           </section>
9       )
10  }
11
12  export default User;
```

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

```
1   import { Link } from "react-router-dom"
2
3   const NavBar = () =>{
4
5       const user1 = {
6           id: 1,
7           name: 'user1',
8           role: 'Frontend Developer'
9       }
10      const user2 = {
11          id: 2,
12          name: 'user2',
13          role: 'Backend Developer'
14      }
15
16      return (
17          <nav>
18              <Link to="/">Home</Link>
19              <Link to="/about">About</Link>
20              <Link to="/contact">Contact</Link>
21              <Link to="/user/1" state={user1}>User 1</Link>
22              <Link to="/user/2" state={user2}>User 2</Link>
23              {/**other Link tags */}
```

Using useNavigate hook

NavBar.js

```
1   import { Link, useNavigate } from "react-router-dom"
2
3   const NavBar = () =>{
4       const navigate = useNavigate()
5
6       const user1 = {
7           id: 1,
8           name: 'Dan',
9           role: 'Frontend Developer'
10      }
11      const user2 = {
12          id: 2,
13          name: 'Walobwa',
14          role: 'Backend Developer'
15      }
16
17      const handleNavigatestate = (id, userData)=>{
18          navigate(`/user/${id}`, { state: userData})
19
20      }
21
22      return (
23          <nav>
24              <Link to="/">Home</Link>
25              <Link to="/about">About</Link>
26              <Link to="/contact">Contact</Link>
27              <button onClick={()=>handleNavigatestate(user1.id, user1)}>User 1</button>
28              <button onClick={()=>handleNavigatestate(user2.id, user2)}>User 2</button>
29              {/**other Link tags */}
```

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.

```jsx
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;
```

Let’s Breathe!

Let’s take a small break before moving on to the next topic.

CoGrammar

The Concept of Version Control

- ★ Version control is a system that records changes to a file or set of files over time so that specific versions can be recalled later.

- ★ Version control allows multiple people to work on a project simultaneously, helps track and revert changes if needed, and aids in project management by keeping a history of all changes.

Overview of Git

- ★ Git was created by Linus Torvalds in 2005 for development of the Linux kernel with efficiency and speed in mind.

- ★ Key Features:
  - Distributed Version Control → Each developer's working copy of the code is also a repository that can contain the full history of all changes.
  - Speed → Designed to handle large projects like the Linux kernel efficiently.
  - Data Integrity → Ensures cryptographic integrity of the project, securing it against both corruption and deliberate change.
  - Branching → Allows multiple versions of a repository to be out at once which can be merged back into the main source.

The Concept of Version Control

- ★ Version control is a system that records changes to a file or set of files over time so that specific versions can be recalled later.

- ★ Version control allows multiple people to work on a project simultaneously, helps track and revert changes if needed, and aids in project management by keeping a history of all changes.

Overview of Git

- ★ Git was created by Linus Torvalds in 2005 for development of the Linux kernel with efficiency and speed in mind.

- ★ Key Features:
  - Distributed Version Control → Each developer's working copy of the code is also a repository that can contain the full history of all changes.
  - Speed → Designed to handle large projects like the Linux kernel efficiently.
  - Data Integrity → Ensures cryptographic integrity of the project, securing it against both corruption and deliberate change.
  - Branching → Allows multiple versions of a repository to be out at once which can be merged back into the main source.

Setting up Git

- ★ Download and install Git from the official website. Ensure it's configured with the correct user name and email to attribute commits properly.

- ★ Configuration:
  - Use the following commands to set up identity:-
    - `git config --global user.name "Your Name"`
    - `git config --global user.email "your_email@example.com"`.

Git Commands

- ★ Initialization: `git init` transforms a directory into a Git repository, starting the tracking process of changes.
- ★ Staging and Committing: `git add <filename>` stages files.
- ★ `git commit -m "commit message"` records file snapshots permanently in version history.
- ★ Viewing Changes: `git status` shows the status of changes in the working directory and staging area. git log displays committed snapshots.

Introduction to GitHub

Connecting Git with GitHub

- GitHub provides a web-based graphical interface and access control for managing project repositories.
- The command `git remote add origin <repository URL>` links your local repository to a remote one, allowing for pushing and pulling changes.
- `git push origin main` sends changes to the main branch of the remote repository.
- `git pull` updates the local line of development with updates from its remote counterpart.

Collaboration Using GitHub

- Branching:
  - Use `git branch <branchname>` to create a new branch
  - `git checkout <branchname>` to switch branches.

- Merging:
  - `git merge <branch>` combines the history of the specified branch into the current branch, which is useful when completing features.

Pull Requests and Code Review

- After pushing a branch to GitHub, you can issue a pull request via GitHub's website to propose your changes.

- Team members review, discuss, and eventually merge the pull request into the main branch.

Git Cheat Sheet

- https://education.github.com/git-cheat-sheet-education.pdf

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar