

Task 2: Advanced React Rendering Techniques for a Profile Website

Objective:

Explore component lifecycle methods, render lists, and implement advanced rendering techniques using React, with a focus on a profile website. This task aims to enhance your understanding of React's lifecycle, rendering lists, and advanced concepts like render props, refs, event handling, and higher-order components.

Pre-requisites:

- Basic understanding of HTML, CSS, and JavaScript
- Familiarity with a code editor like Visual Studio Code
- Basic knowledge of React

Concepts Covered:

- Component Lifecycle
- Rendering Lists and Keys
- Render Props and Refs
- Event Handling and Higher Order Components

Concepts:

1. Component Lifecycle:

Implement `useEffect` to log messages during different lifecycle stages (mount, update, unmount).

```
// src/components/ProfileList.js
import React, { useEffect } from 'react';

const ProfileList = () => {
  useEffect(() => {
    console.log('Component Mounted');
    return () => {
      console.log('Component Unmounted');
    };
  }, []);

  useEffect(() => {
    console.log('Component Updated');
  });

  return (
    <div>
      <h2>Profile List</h2>
    </div>
  );
};
```

```
export default ProfileList;
```

2. Rendering Lists and Keys:

Create a `ProfileList` component that renders a list of profiles using `.map()`, each with a unique key.

```
// src/components/ProfileList.js
import React from 'react';

const profiles = [
  { id: 1, name: 'John Doe', occupation: 'Software Engineer' },
  { id: 2, name: 'Jane Smith', occupation: 'Graphic Designer' },
  { id: 3, name: 'Sam Johnson', occupation: 'Product Manager' }
];

const ProfileList = () => {
  return (
    <div>
      <h2>Profile List</h2>
      <ul>
        {profiles.map(profile => (
          <li key={profile.id}>
            {profile.name} - {profile.occupation}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default ProfileList;
```

3. Render Props and Refs:

Use render props to create a flexible `HoverInfo` component and implement a ref in a `SearchBar` component to focus on the input field on component mount.

```
// src/components/HoverInfo.js
import React from 'react';

const HoverInfo = ({ render }) => {
  return (
    <div onMouseOver={() => render(true)} onMouseOut={() => render(false)}>
      {render(false)}
    </div>
  );
};

export default HoverInfo;
```

```
// src/components/SearchBar.js
import React, { useRef, useEffect } from 'react';

const SearchBar = () => {
  const inputRef = useRef(null);

  useEffect(() => {
    inputRef.current.focus();
  }, []);

  return (
    <input type="text" placeholder="Search profiles..." ref={inputRef} />
  );
};

export default SearchBar;
```

4. Handling Events and Higher Order Components:

Create a `withLogger` higher-order component (HOC) that logs every click event on wrapped components like `ProfileItem`.

```
// src/components/withLogger.js
import React from 'react';

const withLogger = WrappedComponent => {
  return props => {
    const handleClick = () => {
      console.log('Component clicked');
    };

    return <div onClick={handleClick}><WrappedComponent {...props} /></div>;
  };
};

export default withLogger;

// src/components/ProfileItem.js
import React from 'react';

const ProfileItem = ({ profile }) => {
  return (
    <div>
      <h3>{profile.name}</h3>
      <p>{profile.occupation}</p>
    </div>
  );
};
```

```

export default ProfileItem;

// src/components/ProfileList.js
import React from 'react';
import withLogger from './withLogger';
import ProfileItem from './ProfileItem';

const ProfileItemWithLogger = withLogger(ProfileItem);

const profiles = [
  { id: 1, name: 'John Doe', occupation: 'Software Engineer' },
  { id: 2, name: 'Jane Smith', occupation: 'Graphic Designer' },
  { id: 3, name: 'Sam Johnson', occupation: 'Product Manager' }
];

const ProfileList = () => {
  return (
    <div>
      <h2>Profile List</h2>
      <ul>
        {profiles.map(profile => (
          <ProfileItemWithLogger key={profile.id} profile={profile} />
        ))}
      </ul>
    </div>
  );
};

export default ProfileList;

```

Setup:

1. Install Visual Studio Code (VS Code):

Download and install VS Code from [Visual Studio Code](#).

2. React Developer Tools:

Install the React Developer Tools browser extension for [Chrome](#) or [Firefox](#).

3. Git:

Install Git for version control. You can download it from [Git](#).

4. Node.js and npm:

Ensure Node.js and npm are installed on your machine. You can download and install them from [Node.js](#).

Install Dependencies:

- Navigate to the project directory and run:

```
bash
```

```
Copy code
```

```
npm install
```

5. Setting Up the React Project:

- Open your terminal.
- Navigate to the directory where you want to create your project.
- Run the following command to create a new React project:

```
npx create-react-app profile-website
```

6. Installing Additional Dependencies:

- Navigate to the project directory:

```
cd profile-website
```

- Install any additional dependencies if required. For example:

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

Tasks:

Part 1: Understanding Component Lifecycle and Lists (30 minutes)

1. Explore Component Lifecycle (15 minutes):

- Implement `useEffect` in a component like `ProfileList` to log messages during different lifecycle stages (mount, update, unmount).
- Example:

```
// src/components/ProfileList.js
import React, { useEffect } from 'react';

const ProfileList = () => {
  useEffect(() => {
    console.log('Component Mounted');
    return () => {
      console.log('Component Unmounted');
    };
  }, []);

  useEffect(() => {
    console.log('Component Updated');
  });

  return (
    <div>
      <h2>Profile List</h2>
    </div>
  );
};
```

```
export default ProfileList;
```

- Goal: Understand the component lifecycle in React, specifically how components are created, updated, and destroyed.

2. Rendering Lists and Keys (15 minutes):

- Create a `ProfileList` component that renders a list of profiles using `.map()`, each with a unique key.
- Example:

```
// src/components/ProfileList.js
import React from 'react';

const profiles = [
  { id: 1, name: 'John Doe', occupation: 'Software Engineer' },
  { id: 2, name: 'Jane Smith', occupation: 'Graphic Designer' },
  { id: 3, name: 'Sam Johnson', occupation: 'Product Manager' }
];

const ProfileList = () => {
  return (
    <div>
      <h2>Profile List</h2>
      <ul>
        {profiles.map(profile => (
          <li key={profile.id}>
            {profile.name} - {profile.occupation}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default ProfileList;
```

- Goal: Learn how to render lists in React and the importance of keys for list items, particularly in the context of a profile website.

Part 2: Advanced Rendering Techniques (30 minutes)

1. Implementing Render Props and Refs (15 minutes):

- Use render props to create a flexible `HoverInfo` component that shows additional information when hovering over a product. Implement a ref in a `SearchBar` component to focus on the input field on component mount.
- Example:

```

// src/components/HoverInfo.js
import React from 'react';

const HoverInfo = ({ render }) => {
  return (
    <div onMouseOver={() => render(true)} onMouseOut={() =>
render(false)}>
      {render(false)}
    </div>
  );
};

export default HoverInfo;

// src/components/SearchBar.js
import React, { useRef, useEffect } from 'react';

const SearchBar = () => {
  const inputRef = useRef(null);

  useEffect(() => {
    inputRef.current.focus();
  }, []);

  return (
    <input type="text" placeholder="Search profiles..." ref={inputRef} />
  );
};

export default SearchBar;

```

- Goal

: Master the concept of render props for component flexibility and learn to use refs for direct DOM access.

2. Handling Events and Higher Order Components (15 minutes):

- Create a `withLogger` higher-order component (HOC) that logs every click event on wrapped components like `ProfileItem`.
- Example:

```

// src/components/withLogger.js
import React from 'react';

const withLogger = WrappedComponent => {
  return props => {
    const handleClick = () => {
      console.log('Component clicked');
    };

```

```

        return <div onClick={handleClick}><WrappedComponent {...props} /
    ></div>;
    };
};

export default withLogger;

// src/components/ProfileItem.js
import React from 'react';

const ProfileItem = ({ profile }) => {
    return (
        <div>
            <h3>{profile.name}</h3>
            <p>{profile.occupation}</p>
        </div>
    );
};

export default ProfileItem;

// src/components/ProfileList.js
import React from 'react';
import withLogger from '../withLogger';
import ProfileItem from '../ProfileItem';

const ProfileItemWithLogger = withLogger(ProfileItem);

const profiles = [
    { id: 1, name: 'John Doe', occupation: 'Software Engineer' },
    { id: 2, name: 'Jane Smith', occupation: 'Graphic Designer' },
    { id: 3, name: 'Sam Johnson', occupation: 'Product Manager' }
];

const ProfileList = () => {
    return (
        <div>
            <h2>Profile List</h2>
            <ul>
                {profiles.map(profile => (
                    <ProfileItemWithLogger key={profile.id}
profile={profile} />
                ))}
            </ul>
        </div>
    );
};

export default ProfileList;

```


- Goal: Understand event handling in React and the concept of higher-order components for enhancing component functionality.







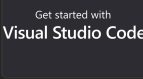

Instructions:

1. Write the required code in the appropriate files in the `src/components` directory.
2. Open the terminal and navigate to the project directory.
3. Run the React project using:

```
npm start
```

4. Open the project in a web browser to ensure the code displays correctly.
5. Use the browser's developer tools and React Developer Tools to debug and inspect the elements.

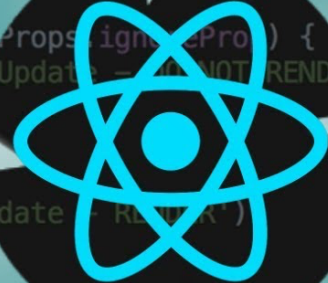
Resources:

-  **Getting Started – React**
A JavaScript library for building user interfaces
 <https://reactjs.org/docs/getting-started.html>
-  **Create React App**
Set up a modern web app by running one command.
 <https://create-react-app.dev/>
-  **New React Developer Tools – React Blog**
This blog site has been archived. Go to react.dev/blog to see the recent posts. A month a...
 <https://reactjs.org/blog/2015/09/02/new-react-developer-tools.html>
-  **Documentation for Visual Studio Code**
Find out how to set-up and get the most from Visual Studio Code. Optimized for building and d...
 <https://code.visualstudio.com/docs>

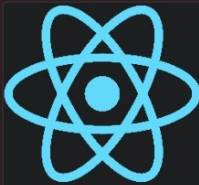
Videos:

TUTORIAL

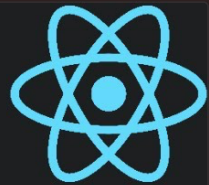
React Component Lifecycle



freeCodeCamp.org 25:39 min 158,966 Views 3,161 Likes



RENDER LISTS



Fruits

apple: 95
orange: 45
banana: 105
coconut: 159
pineapple: 37

Vegetables

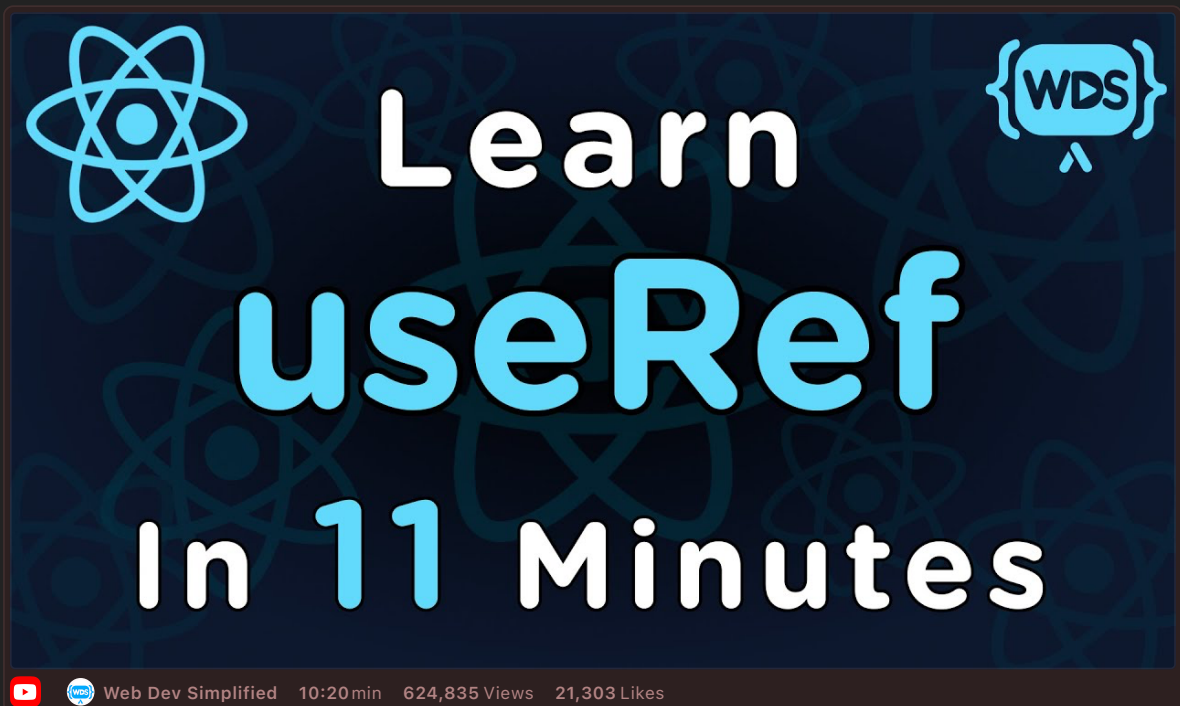
potatoes: 110
celery: 15
carrots: 25
corn: 63
broccoli: 50

Bro Code 26:40 min 25,319 Views 776 Likes

•



•



GitHub Instructions:

1. **Open in Visual Studio Code:**

After clicking on the "Open in Visual Studio Code" button from the GitHub Classroom confirmation page, VSCode will open the repository directly. If prompted, select "Open" or "Allow" to open the repository in VSCode.

2. **Open the Terminal in VSCode:**

In VSCode, open a terminal by selecting Terminal > New Terminal from the top menu.

3. Complete the Task:

In VSCode, write your solution in the appropriate files in the `src/components` directory.

4. Run and Test Your Code:

Open your terminal, navigate to your project directory, and run:

```
npm start
```

5. Commit Your Changes:

In the VSCode terminal, add your changes to git:

```
git add src/components/*
```

Commit your changes with a meaningful message:

```
git commit -m "Completed task 22"
```

6. Push Your Changes to Your Repository:

Push your changes to your forked repository:

```
git push origin main
```

7. Create a Pull Request:

Go to your repository on GitHub.

Click on the "Pull Requests" tab.

Click the "New Pull Request" button.

Ensure the base repository is the original template repository and the base branch is `main`.

Ensure the head repository is your forked repository and the compare branch is `main`.

Click "Create Pull Request".

Add a title and description for your pull request and submit it.

Summary of Commands:

```
# Open in Visual Studio Code

# Open terminal in VSCode

# Complete the task by editing src/components/*

# Navigate to the project directory
cd profile-website

# Run your code
npm start

# Add, commit, and push your changes
git add src/components/*
git commit -m "Completed task 2"
git push origin main

# Create a pull request on GitHub
```