Assignment 7 - Finding The Path

1. **What are various ways to add images into our App? Explain with code examples**

Directly in HTML:

We can use the <img> element directly in your HTML to include an image.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Image Example</title>
</head>
<body>
  <img src="path/to/your/image.jpg" alt="Description of the image">
</body>
</html>
```

Using CSS Background:

We can set the background of an element using CSS.

```
.my-container {
  background-image: url("images/background.jpg");
  background-size: cover;
}
```

Using JavaScript:

We can dynamically create and append an <img> element using JavaScript.

```
<script>
  const imageContainer = document.getElementById('image-container');
  const imageElement = document.createElement('img');

  imageElement.src = 'path/to/your/image.jpg';
  imageElement.alt = 'Description of the image';

  imageContainer.appendChild(imageElement);
</script>
```

Using React (JSX):
In a React application, we can include images using JSX.

```jsx
import React from 'react';

const App = () => {
  return (
    <div>
      <img src="path/to/your/image.jpg" alt="Description of the image" />
    </div>
  );
};

export default App;
```

2. **What would happen if we do console.log(useState())?**

Calling console.log(useState()) in React would result in logging an array with two elements:

The current state value: This is the value that the component's state currently holds. It depends on how you initialized the state using useState and any subsequent updates you may have made.
A setter function: This function allows you to update the state value. Calling the setter function with a new value will trigger a re-render of the component.

```jsx
const [count, setCount] = useState(0);

console.log(useState()); // Logs: [0, f] where f is the setter function
```

3. **How will useEffect behave if we don't add a dependency array ?**

Runs after every render: The useEffect callback executes after every single render of the component, including the initial render. This can lead to unexpected behavior and performance issues if the effect is expensive or causes side effects.
Mimics componentDidMount and componentDidUpdate: This behavior is similar to a combination of React's class component lifecycle methods componentDidMount (runs only once after initial render) and componentDidUpdate (runs after updates).

Potential for infinite loops: If the effect updates the state or props that trigger a re-render, it could create an infinite loop as the effect triggers another render, and so on.

4. **What is SPA?**

SPA, which stands for Single-Page Application, is a web application design and development approach where the entire website or app is loaded in a single HTML page. Instead of reloading the entire page for each navigation click like traditional websites, SPAs dynamically update content within the same page using JavaScript and frameworks like React, Angular, or Vue.js.

5. **What is difference between Client Side Routing and Server Side Routing?**

Client-Side Routing:

- Navigation on the Client: In client-side routing, the navigation and rendering of pages are handled on the client side (in the user's browser) without making additional requests to the server for each page.

- JavaScript Frameworks: Client-side routing is often associated with single-page applications (SPAs) that use JavaScript frameworks or libraries (e.g., React, Angular, Vue.js). These frameworks manage the application state and handle the dynamic updating of content.

- Smooth Transitions: Since pages are not fully reloaded from the server, transitions between pages are generally smoother and faster. Only the content that needs to change is updated dynamically.

- Reduced Server Load: The server is responsible for initially serving the application and providing data through APIs. However, once the application is loaded, subsequent page changes do not require additional server requests unless new data is needed.

Server-Side Routing:

- Navigation on the Server: In server-side routing, navigation triggers requests to the server, and the server responds by rendering and sending back the appropriate HTML page.

- Traditional Approach: In traditional multi-page applications (MPAs), server-side routing is the default behavior. Each link or navigation action typically results in a request to the server, and the server sends back a new HTML page.

- Full Page Reloads: When navigating between pages, the entire HTML page is often reloaded from the server. This can result in slower transitions and a less responsive feel compared to SPAs.

- More Server Interaction: Server-side routing may involve more frequent interactions with the server, leading to increased server load, especially in applications with complex navigation.