React – Routing

Single Page Applications, Blueprint for SPA

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#react

Table of Contents

Routing Overview

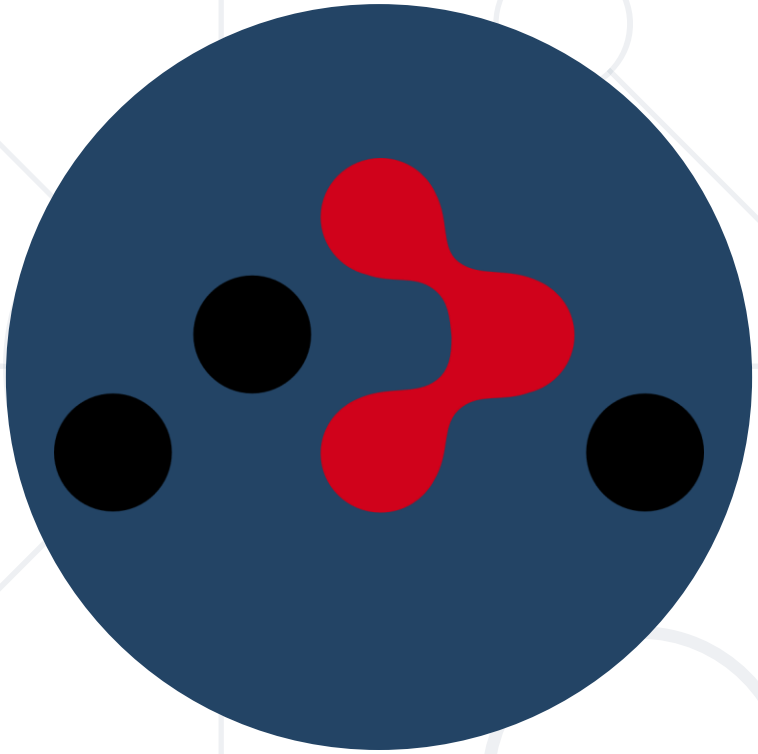
Navigation for Single Page Apps

What is Client-side Routing?

- **Client-side routing** is internal handling of a route
- It's a pivotal element of writing **SPA's**
- Allows navigation, **without a full reloading of** the page
- Loading only the initial **HTML**, **CSS** and **JS**
- Gives better UX

Single Page Applications

- A **Router** loads the correct content when the **location changes**
- Change in content is reflected in the address bar
- Benefits
  - Load all scripts only once
  - Maintain state across multiple pages
  - Browser history can be used
  - Build User Interfaces that react quickly



React-Router

Routing Library Tailored for React

React Router

- **React Router** is an **API** for React applications
- Uses component structure

```
const App =(
  <Routes>
    <Route path="/catalog" element={<Catalog />} />
    <Route path="/about" element={<About />} />
  </Routes>);

ReactDOM.render((
  <BrowserRouter>
    <App />
  </BrowserRouter>
), document.getElementById('root'))
```

Installation and Setup

- Install using npm from the terminal

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

- **Route**, **Link**, **BrowserRouter**, **Routes** components helps to implement the routing

```
import {
      BrowserRouter as Router,
      Routes,
      Route,
      Link,
} from 'react-router-dom';
```

Adding More Scenes

- React components can be wrapped in a **Route** and bound to a specific path

```
<BrowserRouter>
   <Routes>
      <Route path="/" element={<App />} />
      <Route path="/catalog" element={<Catalog />} />
      <Route path="/about" element={<About />} />
   </Routes>
</BrowserRouter>
```

Route Matching

- Route path is exact by default
- Use * as a wildcard

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/about" element={<About />} />
  <Route path="/user/:id" element={<User />} />
  <Route path="*" element={<NotFound />} />
</Routes >
```

Default route

Navigating with Link

- **Link** replaces **<a>** and automatically prevents **page reload**

```
class App extends Component {
  render() {
    return (
      <div>
        <h1>React Router Tutorial</h1>
        <Link to="/catalog">Catalog</Link>
        <Link to="/about">About</Link>
      </div>
    );
  }
}
```

useParams

- Parameters are dynamic parts of the URL

```
/catalog/elecronics/XYZ5538
```

- Configure the Route to work with params

```
<Route path="/catalog/:category/:userId"
component={Catalog}/>
```

- Access from the component

```
const {category, userId} = useParams();
```

useLocation

- The **location** object represents
  - Where the app is now
  - Where you want it to go
  - Where it was
- A **location** object is never mutated

Redirects

- You can redirect the user by rendering a **Navigate** component
  - you can replace route state instead of push with replace attribute
- You can redirect with useNavigate hook

```
...
if (condition) {
      return <Navigate to="/home" />
}
```

Determine the component to render at run-time

Active Links

- **NavLink** knows when it's currently active
  - We can style them with **style**, **className** or **children**

```
<NavLink to="/catalog"
      style={({isActive}) =>
            isActive ? { color: 'red’ } : {}}>
  Catalog
</NavLink >
```

```
<NavLink to="/catalog"
      className={({isActive}) =>
            isActive ? activeStyle : undefined}>
  Catalog
</NavLink >
```

Nested Routes

- You can dynamically nest routes

```
const About = () => (
  <div>
    <h1>About Page</h1>
    <Route
      path="contact"
      element={<Contact />}
    />
  </div>
)
```

AbortController

Purpose, Usage, Error Handling

AbortController in React

- **AbortController** is used to cancel ongoing fetch requests, preventing potential memory leaks and unnecessary network usage, especially in situations where a component unmounts before the request completes.
  - Create an instance of **AbortController**, pass its signal property to the fetch request, and call **abort()** on the controller to cancel the request.
  - When a fetch request is **aborted**, a specific **AbortError** is thrown, which should be caught and handled separately from other errors.

AbortController Example

```
function MyComponent() {
  const [data, setData] = useState(null);
  useEffect(() => {
   const controller = new AbortController();
   const signal = controller.signal;
    fetch('https://my-api.com/endpoint', { signal })
     .then(response => response.json())
     .then(json => setData(json))
     .catch(error => {
  if (error.name === 'AbortError') { console.log('Fetching data was aborted');
       } else {
      console.error(error);
       }
      });
   return () => {
      controller.abort();
    };
  }, []);
```

Lazy Loading

Code-Splitting, Bundling, React.lazy

Code-Splitting – Bundling

- Most React apps will have their files "**bundled**" using tools like **Webpack** or **Browserify**
- Bundling is the process of
  - Following **imported** files
  - Merging them into a **single file** (bundle)

Code-Splitting Bundling

- The bundle can be included on a webpage to **load** an entire app **at once**

```
export function add(a, b) {
  return a + b;
}
```

```
import { add } from './math.js';
console.log(add(16, 26));
```

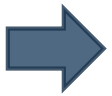
```
function add(a, b) {
  return a + b;
}
console.log(add(16, 26));
```

Dynamic Import

- The best way to introduce **code-splitting** into your app is through the dynamic **import()** syntax

```
import { add } from './math';

console.log(add(16, 26));
```



```
import("./math").then(math => {
  console.log(math.add(16, 26));
});
```

Using React.lazy

- The **React.lazy** function lets you render a dynamic import as a **regular component**

```
const OtherComponent = React.lazy(() =>
import('./OtherComponent'));

function MyComponent() {
  return (
    <div>
      <OtherComponent />
    </div>
  );
}
```

Suspense – Showing Indicators

- The **Suspense** component shows **fallback content** while we're waiting for another component to load

```
function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <OtherComponent />
      </Suspense>
    </div>
  );
}
```

Accepts any React element

Route-based Code Splitting

- An example of how to setup route-based code splitting

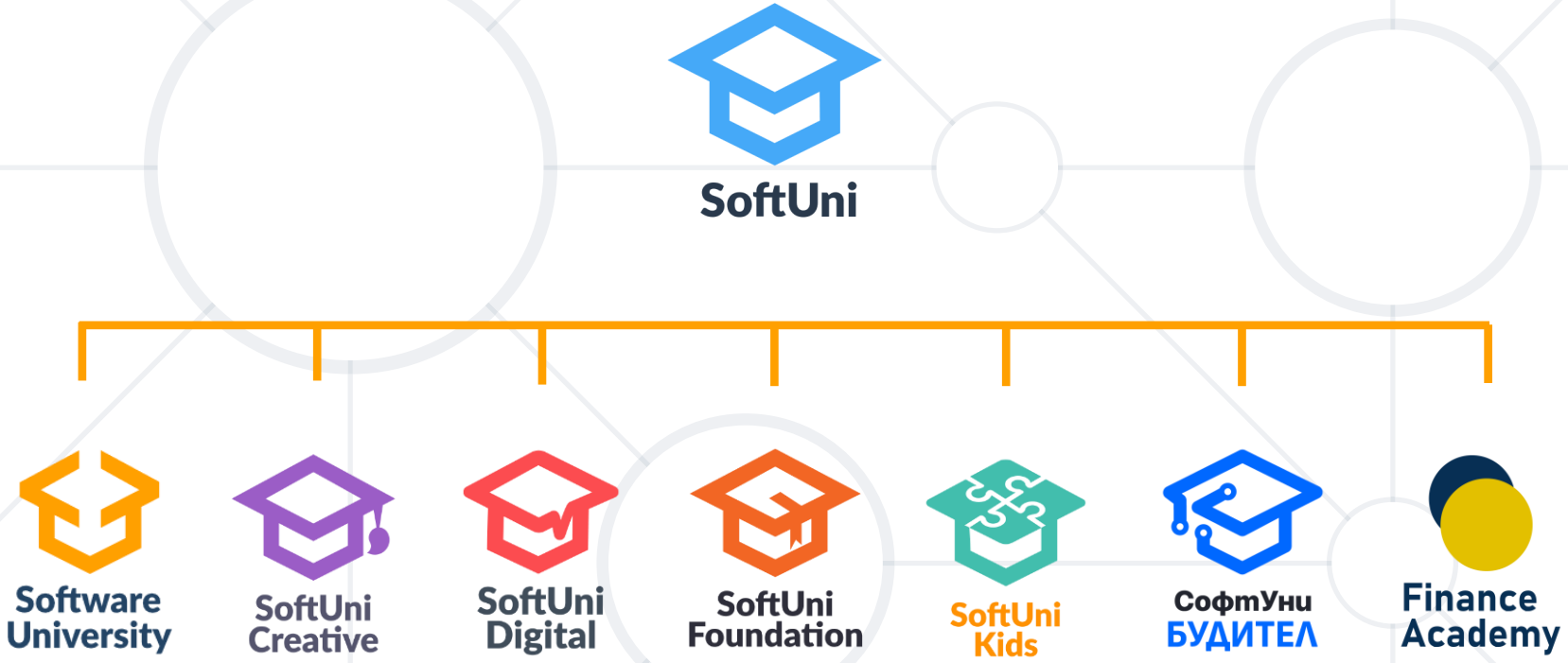
```
const Home = lazy(() => import('./routes/Home'));
const About = lazy(() => import('./routes/About’));

const App = () => (
  <Router>
    <Suspense fallback={<div>Loading...</div>}>
      <Routes>
        <Route path="/" component={<Home />} />
        <Route path="/about" component={<About />} />
      </Routes>
    </Suspense>
  </Router>
);
```

Summary

- Virtual Dom
  - The virtual DOM - VDOM
- Routing Overview
  - Internal handling of a route - Client-side routing
  - Single Page Applications - Router
- React Router
- AbortController
- React Lazy & Suspense

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank
Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR
Serving the high achievers

SOFTWARE GROUP

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg