Next Gen Engineering
School of Tech - INDIA

React.js 4

Routing

Overview

- Using React Router
- Building a React app for deployment

Objectives

- Have a working knowledge of `react-router-dom`
- Know why and how to use `vite`
- Know how to build a React app for deployment
- Understand what environment variables are and how to use them

The React sessions

We have 4 React sessions:

- Components: Introduction to React, Components & JSX/TSX, Testing Components ✅
- State: Virtual DOM, Data Flow, State Management with the `useState` React Hook ✅
- Side Effects: Component Lifecycle, Fetching Data with the `useEffect` React Hook, Testing User Events ✅
- Routing: React Router, Building & Deployment (this one)

Routing and Navigation

We have been using React to create single-page applications (SPA) which means dynamically rewriting the current web page with new data instead of loading entire new pages.
But we may want our users to be able to navigate directly to specific pages and share links with others.

Routing and Navigation in React

Routing and navigation functionality is not built-in to React by default. We can use a library called React Router to create specific URLs for our components. In other words, we can use React Router to create "pages" for our application.

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

React Router Components

React Router provides us with useful components, e.g.:

- `<BrowserRouter>` A parent router built for the browser
- `<Routes>` Used like a switch to render the correct route
- `<Route>` Defines a new route using the `path` prop
- `<Link>` Renders an `<a>` element to a `route` using the `to` prop
- `<NavLink>` A special kind of `<Link>` that knows if it is active

> Let's have a look at `./examples/react-router` together to see this in action

Dynamic Routes

We can create dynamic routes to match `urls` that begin with `/blog` (`/blog/1`, `/blog/2` etc) by defining a route parameter:

```
<Route path="/blog/:id" />
```

We can then grab the value of this param in our components using the `useParams` hook:

```
import { useParams } from "react-router-dom"
const { id } = useParams()
```

> Let's have a look at `./examples/react-router` together to see this in action

Programmatic Navigation

Sometimes we need to navigate or change the `url` programmatically. We can use the `useNavigate` hook for this:

```
import { useNavigate } from "react-router-dom"
const navigate = useNavigate()
navigate("/some-path")
```

> Let's have a look at `./examples/react-router` together to see this in action

Emoji Check:

How do you feel about using React Router to implement routing in a React application?

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Exercise - Country App Routing - 30mins

Instructor to distribute exercise:

> See `./exercises/react-countries-routing/README.md`

Emoji Check:

How do you feel about implementing routing in a React application?

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

React in the Real World

In order to create fully functional React Apps running in the browser, we need to do a few things:

- We need to transpile TSX into JS capable of producing HTML (Babel)
- We should bundle our TS and CSS files together into consolidated `.ts` and `.css` files (WebPack)
- For dev, we should automate this process when our code changes (hot-reload)
- For prod, we should minify these files to make them smaller (WebPack)

Bootstrap a React App

We used `vite` in React 1:

```
cd <projects-dir>
npm create vite@latest
type in your react app name
<select React>
<select Typescript + SWC>
cd <app-name>
npm install
npm run dev
```

All of the examples and exercises were set up using `vite`.

Project Structure

`vite` generates a basic structure.
As our app grows and becomes more complex, it could take on a structure like the following:

```
├──public
│   └── favicon.ico
├──src
│   ├── /api          Pure functions that make API calls
│   ├── /components   Atomic components with minimal logic
│   ├── /hooks        Custom hooks
│   ├── /views        Pages that make up the app
│   ├── /utils        Helper functions
│   ├── App.tsx        Top level app component
│   └── index.tsx      React entry point
└── index.html   Browser entry point
```

Deployment

We don't run a React App in production like we do a Node Express App. Instead we run a build process to generate static files which we then serve like any other website.
If we used `vite` then this build process already exists:

```
npm run build
```

Environment Variables

There will be times when you need your app to behave differently depending on which environment it is running in.
For example:

- to connect to a different API url depending on whether the app is running in production (e.g. `api.example.com`) or in a test environment (e.g. `api.test.example.com`)
- to switch off certain functionality (e.g. analytics) if the app is not running in production

We can use environment variables (usually abbreviated to env vars) for this, these are variables that are specific to the environment the app is running in and have been declared in that environment.

Environment Variables in React

`node.js` provides us access to have and use environment variables via an object `process.env`.
With some settings changes we can allow `vite` to use this same object.
On this `process.env` object there is a built-in environment variable called `NODE_ENV`, which is set to 'development' when running `npm run dev` and is set to 'production' when you run `npm run build`.
You can set any other custom environment variables.
In the app the environment variables can be read from `process.env`, so `process.env.NODE_ENV` and `process.env.<YOUR_VARIABLE>`

Environment Variables in React

First we need to make a few changes to our `vite.config.ts` file to tell vite we want to mimic the 'process.env' object.

```
import { defineConfig, loadEnv } from 'vite'
import react from '@vitejs/plugin-react-swc'

export default defineConfig(({mode}) => {
  process.env = loadEnv(mode, process.cwd(), '')

  return {
    plugins: [react()],
    server: {
      open: true,
      port: 3000
    },
    define: {
      'process.env.MY_NAME': JSON.stringify(process.env.MY_NAM
      'process.env.MY_BOOLEAN': process.env.MY_BOOLEAN,
    }
  }
})
```

Environment variables

`process` is a feature made available to use by `node.js`. So Typescript doesn't know what this is. We can use an npm package to let typescript know what 'process' is for us.
Make sure you in the root folder then run the below command.

```
npm i --save-dev @types/node
```

Environment variables

Let's try it out!

- Add a `.env` file to the root of your project
- Add an env var into the `.env` file like `MY_NAME=<yourname>`
- Use the env var in your app with `process.env.MY_NAME`

Now let us override this env var during the build process:

- Run `MY_NAME=JOEBLOGGS npm run build`
- Serve the application with serve

Emoji Check:

How do you feel about using `env vars` in a React application?

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

What's next?

React is a very popular tool and the likelihood you will be using it in your professional life is very high. These modules have aimed to teach the core principles that underpin React, however we only have time to cover the basics! If you work with React on a real-world project you will definitely come across many of the following concepts or tools:

- Higher-Order Components
- Component libraries (e.g. Material-UI)
- Advanced styling (e.g. styled-components / Emotion)
- Storybook (build your own component library)
- State Management (e.g. Context / Redux / MobX)
- Server Side Rendering (e.g. Next.js)

You can create cross-platform mobile apps using React Native

Overview - recap

- Using React Router
- Building a React app for deployment

Objectives - recap

- Have a working knowledge of `react-router-dom`
- Know why and how to use `vite`
- Know how to build a React app for deployment
- Understand what environment variables are and how to use them

Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively