Single Page Applications with React

What are we going to cover

Single Page Applications

- Pros and cons

React-Router-Dom library

Single Page Applications

Single Page Applications are almost completely loaded at startup

- All HTML, CSS and JavaScript resources are loaded at startup

Once loaded the application only communicates with the server to load or update data

- Normally using AJAX JSON requests

Normally the user can still use the browsers back and forward buttons

- Or bookmark individual pages

Even though the URL changes there are no full page reloads

- State can be maintained in JavaScript objects

Navigation is done using the browsers History API

- Or using hash navigation

Pros and Cons of an SPA

Advantages

- Fast navigation between different views
- Browser based client can become stateful where needed

Disadvantages

- Tighter coupling between different parts
- Possible problems with memory leaks as pages are not reloaded

Best of both worlds

Create Single Page Modules
- AKA Mini SPA’s
- A bounded context is often a good module boundary

Minimize the number of full page reloads
- Only when navigating from one distinct module to another

All interaction inside a module uses the SPA paradigm
- Fast response

Less coupling between modules
- Easier to version modules independently

React Router Dom

Not part of React itself

- The most popular routing library for React
- Originally inspired by Ember's router

Render a Router with routes

- Component per path
- Optional redirecting or catch all routes

Routes can be nested

- Using either absolute or relative URL’s

The react-router-dom components are real React components

- Everything you know from React is the same

Choosing a Router

There are several Router implementations to chose from

- BrowserRouter
  - Use the **HTML 5 History API**
  - The HTML 5 History API isn’t available in Internet Explorer 9
  - This is the most commonly used router
- HashRouter
  - Uses the **hash** part of the URL
  - Works in every browser but produces “ugly” url’s
- StaticRouter
  - Useful for server side rendering
- MemoryRouter
  - Useful for unit tests where are router is required

Defining Routes

```
<BrowserRouter>
  <div>
    <h1>Animals</h1>
    <Route path="/cats" exact component={Cats} />
    <Route path="/cats/:name" exact
      component={Cats} />
    <Route path="/dogs" exact
      render={() => <Dogs dogs={dogs} />} />
  <div>
</BrowserRouter>
```

The Switch component

Normally React-Router will render all components where the route **path** matches

- If exact is specified the component will only be renderded if there is an exact match
- Otherwise it will also be rendered with a partial match

By wrapping the **Route** components in a **Switch** component only the first match is rendered

Easy way to handle unknown routes

- Either render a 404 style not found component or redirect to a know route at the end of the known routes

Render a single component

```
 <BrowserRouter>
  <h1>Animals</h1>
  <Switch>
    <Route path="/cats" exact component={Cats} />
    <Route path="/cats/:name" exact
      component={Cats} />
    <Route component={NotFound}/>
  </Switch>
</BrowserRouter>
```

Using route parameters

Route **parameters** can be added to the path using a colon

- Use multiple parameters if needed

Child components of a <Route /> receive a **match** prop that contains the **params** object

- When using a render prop you need to be explicit about passing props on

Other components can use the **withRouter()** higher order functions to get the same **match** prop

- As long as they are children of the router

Defining route parameters

```
 <BrowserRouter>
  <h1>Animals</h1>
  <Switch>
    <Route path="/cats" exact component={Cats} />
    <Route path="/cats/:name" exact
      component={Cats} />
    <Route component={NotFound}/>
  </Switch>
</BrowserRouter>
```

Extracting route parameters

```
import React, { Component } from 'react';
import { withRouter } from 'react-router-dom';

const DisplayRouteParams =
  ({ match: { params } }) => (
  <div>Route params: {JSON.stringify(params)}</div>
);

export default withRouter(DisplayRouteParams);
```

The Redirect component

A **Redirect** component can be used to redirect to another route

- Use the **to** property to specify the new route

By default this will replace the current route in the history table

- Use the **push** property to add it instead

Redirecting to a known page

```
 <BrowserRouter>
  <h1>Animals</h1>
  <Switch>
    <Route path="/cats" exact component={Cats} />
    <Route path="/cats/:name" exact
      component={Cats} />
    <Redirect to="/cats" />
  </Switch>
</BrowserRouter>
```

Navigating with the React-Router-Dom

Navigating can be done with the **Link** component

- Generates an anchor tag in the markup
- Can be styled as a button with Bootstrap if needed

There is also a **history** prop passed to routed components

- Use this.props.history.push('/my-new-location')

Navigating with the React-Router-Dom

The **Link** component is used to render an HTML anchor tag and navigate.

- The child can be a node or a function. The function is passed if the link is active amongst other parameters

The **NavLink** is similar with additional styling capabilities.

- Used when it matches the active route

Beware: Do not use an HTML anchor tag as this will do a full page load

Navigating

```
<nav>
  <Link to="/cats">Cats</Link>
  <Link to="/cats/zorro">Zorro</Link>
  <Link to="/dogs">Dogs</Link>
</nav>
```

The Prompt component

Render the **Prompt** component to let the user confirm navigating away from a route.

- Useful with dirty data entry forms

The **message** property can also be a function

- The new **location** is passed as a parameter
- Return a string to prompt the user or true to continue

Using a Prompt

```
<Prompt when={isDirty}
  message="There are unsaved changes."/>
```

Conclusion

Single Page Applications are popular these days

- Usually provide a much better user experience
- But at the price of more coupling

React-Router is a great way to create React SPAs

- Takes care of all routing needs
- Provides a simple yet powerful API