Lab 3

The third lab is about routing and form validation, *objectives*:

1. Understanding how a React application is loaded by the browser.
2. Understanding how a web application can be split into different pages using the React router.
3. Get experience with passing props between components and combining them with parameters from the url.
4. Get some experience with form validation and the html 5 form validation api.

Background

The assignments here assumes you have a working solution for lab 2, i.e. a working react app with three components: `App`, `ComposeSalad`, and `ViewOrder`.

Assignments

1. We are going to move the `ComposeSalad` and `ViewOrder` to separate pages in the application. First, make sure you know what a router is and the basics of the react router, for example by reading the official tutorial:
   `https://reactrouter.com/docs/en/v6/getting-started/tutorial`

2. We will use the react router. Add it to your project using npm and start the development web server, in the terminal (press `ctrl-c` first, if you are running the development web server from lab 2):

   ```
   > npm install react-router-dom
   > npm start
   ```

3. All React router components, for example `<Link> and <Route>`, must be a child of a router, we will use `BrowserRouter` in this lab. To ensur that nothing ends up outside the router by mistake we can add the `BrowserRouter` to the top of the React component tree. But where is that? How is `<App>` instantiated? To answer these questions lets see what happens when the browser loads your app. It will fetch default file from the servers root, which by default on most servers is a file called `index.html`. There is a `public/index.html` file in your project. When you open it, there is no mention of `<App>`, just a `<div id="root">`. Change the the page title (`<title>React App</title>`), or add some html to the page, save it. Look in the browser and you see that the title in the tab changes so this is the file viewed. If you open the development tools and view the source file it looks similar, but with one important addition: `<script defer src="/static/js/bundle.js"></script></head>`. This is the works of the React tool chain and `bundle.js` contains all the from your project. Specifically, it will include `src/index.js`. Open it and you will find code that adds `App` to the DOM. Add a `<BrowserRouter>` here:

```
import { BrowserRouter } from "react-router-dom";
ReactDOM.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
  document.getElementById('root')
);
```

Reflection question 1: How do you replace the application icon, `favicon.ico`?

4. Next up is a navigation bar. Let's place it in `App`, between the header and the salad bar components. It is a good idea to clean up the render function before it gets too complex. The html/JSX code quickly becomes hard to read when it grows, especially when you add wrapping `<div>` elements for styling. It is a good idea to separate the different parts of the page to separate React function components. If there are many dependencies to the component state you can move the code to functions inside the class. When adding a new portion to a page, it is good practice to place that part in a separate render function. This is what i have after adding an empty placeholder for the navigation bar:

```
class App {
  render() {
    return (
        <div className="container py-4">
          <Header />
          <Navbar />
          {this.renderPageContent()}
          <Footer />
        </div>
  );}

  renderPageContent(){
     <ViewOrder order={this.state.order} />
     <ComposeSalad inventory={inventory} orderSalad={this.orderSalad} />
    return (
    );
  }
}

function Header() {
  return (
    <header className="pb-3 mb-4 border-bottom">
      <span className="fs-4">Min egen salladsbar</span>
    </header>
  );
}
```

When using the React router you use the `<Link to="my-path">` elements for the user to click on to navigate in your app, similar to native `<a href="my-path">` html elements in static web pages. What about universal design? Do not worry, all the right `aria-*` attributes et.c. are set so screen readers know these are navigation elements. Use bootstrap classes to style the links, see `https://getbootstrap.com/docs/5.1/components/navs-tabs/`. Bellow is the example code adapted for the react router:
Option: If you want a responsive design where the menu collapse to an icon on small screens, use navbar `https://getbootstrap.com/docs/5.1/components/navbar/`.

```
function Navbar() {
  return (
  <ul className="nav nav-tabs">
    <li className="nav-item">
      <Link className="nav-link" to="/compose-salad">
        Komponera en sallad
      </Link>
    </li>
    {/* more links */}
  </ul>);
}
```

Dod you notice that the paths are hyphen separated word (”compose-salad”). This is by tradition for urls, which we will discuss later in the course. Add a second link for the `ViewOrder` page. Go to your browser and klick on the links. The path in your browser changes (url), the selected page is highlighted in the menu, and you use the browser back icon to go to the previous page. However, all pages look the same.
Reflection Question 2: If you use `nav-pills` instead of `nav-tabs` the selected page is no longer highlighted in the menu, why? *Hint:* `<NavLink>` and the `active` css class.

5. Let's render the page content based on the navigation bar selection, either the `ComposeSalad`, or the `ViewOrder` component. For this you use `<Routes>` and `<Route>`. If you followed my code clean up advice above you should replace `this.renderPageContent()` with `this.renderRouter()` and place your router code in this function. Your solution should handle four cases:
`http://localhost:3000/compose-salad` → render the `ComposeSalad` component
`http://localhost:3000/view-order` → render the `ViewOrder` component
`http://localhost:3000/` → This is the first page of your app, render a welcome message. (You can also add a home link in the menu) *hint:* `index` attribute
`http://localhost:3000/other-stuff` → all other urls, render a “page not found” message. *hint:* `path="*"`

6. Now you should have a working app with two components. Try to order a salad and then switch to the view order page. This user experience is not so good. When the customer order a salad they only see an empty form, no confirmation. Fix this by jumping to the view order page on a successful form submit. Read about navigating programmatically at `https://reactrouter.com/docs/en/v6/getting-started/tutorial#navigating-programmatically`. React router hooks can only be used in functional components, so you can not use `useNavigate()` in your `ComposeSalad` class. This can be solved using higher order components, see `https://reactjs.org/docs/higher-order-components.html#convention-pass-unrelated-props-through-to-the-wrapped-component`. Here is a wrapper:

```
import { useNavigate } from "react-router-dom";
import ComposeSalad from './ComposeSalad';

function ComposeSaladWrapper(props){
  const navigate = useNavigate();
  return (
    <ComposeSalad navigate={navigate} {...props}/>
  );
}
export default ComposeSaladWrapper;
```

Update `ComposeSalad` so it jumps to the view order page when a salad is ordered.

7. Create a component, `ViewIngredient`, that shows the information from the `inventory` object about an ingredient, i.e vegan, lactose et.c. You should be able to navigate to the `ViewIngredient` component by clicking on the extras in the `ComposeSalad` component. To solv this, you should:

   - Add a `<Link ...>` around each ingredient name in `ComposeSalad`.
   - Create a functional component: `ViewIngredient`. The ingredient name can be found in the params. Pass `inventory` to it as a prop.
   - Add `<Route path='/view-ingredient/:name' ...>` to your router.

   Reflection Question 3: What is the difference between `<Link to="/view-ingredient/:name"` and `<Link to="view-ingredient/:name"`. Try it, look in the browser url field.

   Oprional assignemnt: Add a "view info" button next to the select dropdowns. When the user clicks on it, jump to the `view-ingredient` page for the selected ingredient.

   Note: If you return to the `compose-salad` page after viewing the info about an ingredient the form will be empty. This is because React will create a new `ComposeSalad` object when you navigate to the page (even if you use the browser history). Consequently any earlier state is lost. Normally you would use pop-ups to show the ingredient info, but here I want you to get experience using router params.

8. Now your app is split to different pages, where each page have a clear functionality. This is good, do not confuse the user by putting too many unrelated things on one page. Let's move on to another important part of the user experience, form validation and feedback. When a user orders a salad we want to make sure that:

   - one foundation is selected
   - one protein is selected
   - one dressing is selected

   If these conditions are not met, an error message should be displayed and the form submission should stop. We will use html 5 form validation, which have a set of predefined constraints. One of them is `required`, which ensures that a value is provided for the form field. Html is text, and the default action is to send a http request, which is also text based, so in this context "a value" means anything but the empty string. Remember, the empty sting is falsy in JavaScript. First, add `required` to the `<select>` fields. To test error handling with select fields, make sure that there is an invalid selection (`<option value="">`) and that your component form state is set to an empty sting in the constructor and after form submission.

   ```
   <select required ...>
     <option value=''>make a choice...</option>
     ... more options
   </select>
   ```

   Now press the submit button. You should get an error message from your browser. Let's replace this your own error message and style it with bootstrap. There are two css classes in bootstrap for this: `valid-feedback`, and `invalid-feedback`. They can be used on a text element, for example `<div>` to show messages. The `<div>` should be sibling to the form field (`<select>`). The bootstrap css class will hide the element until any of the pseudo classes `:valid` or `:invalid` is set for the form input field and the css class `was-validated` is set on any parent element. We do not want to show error messages for fields the user have not interacted with, so set the `was-validated` in your `handleChange` or `handleSubmit` functions:

```
handleChange(event) {
  event.target.parentElement.classList.add("was-validated");
  // ...
}
handleSubmit(event){
  event.preventDefault();
  event.target.classList.add("was-validated");
  // ...
}
```

If you only want to show error messages when the form is submitted, you skip the part in `handleChange`. Note, in `handleChange`, `event` points to the `<select>` element, but we want to update the style for parent `<div>` element. Hence the `parentElement`. Next, add an error message to your `<select>` fields. This adds your custom error messages. There is one more thing you need to do, remove the browser error messages:

```
<form onSubmit={this.handleSubmit} noValidate>
```

The attribute noValidate tells the browser not to show its own error message. The browser still does html 5 form validation and updates the pseudo classes `:valid/:invalid`. You can check if a form is valid by calling `formElement.checkValidity()` on the form element, in `handleSubmit`:

```
if(event.target.checkValidity() === false){ ... }
```

9. *Optional assignment 3*: Add validation of the following constraint:

   - at least three, but not more than nine extras are selected

   This error is not related to a single input, but rather a group of checkboxes. It is not a good idea to write an error message on each checkbox, rater add an alert box below the group headline, see `https://getbootstrap.com/docs/5.1/components/alerts/`. You can check the constraint in your `handleChange` and `handleSubmit` functions and store the result in the state:

   ```
   constructor(props) {
     super(props);
     this.state = {
       formErrors: {
         ignoreInvalid: false,
         extras: false,
       }
     };
   }
   ```

   Use this part of the component state to show and hide your alert boxes. Do not bother the user with an error when the first extra is selected. Wait until the form is submitted. After a failed submission, you want to clear the error as soon as the problem is fixed. The attribute `ignoreInvalid` can be used for this, or you might prefer to call it `submissionFailed`.

Editor: Per Andersson

Contributors in alphabetical order:
Alfred Åkesson
Oscar Ammkjaer
Per Andersson

Home: `https://cs.lth.se/edaf90`

Repo: `https://github.com/lunduniversity/webprog`

This compendium is on-going work.
Contributions are welcome!
Contact: `per.andersson@cs.lth.se`