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 some experience with url parameters.
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`. The basic functionality is there, but the user experience is not so good. we will adress this by adding form validation and navigation in this lab.

Bootstrap

You included bootstrap css in lab 2. This add style to your app. Some functionality of bootstrap requires JavaScript code, for example closing an alert box. Include it in `main.jsx`:

```
import '../node_modules/bootstrap/dist/js/bootstrap.esm.js'
```

Assignments

1. Let's start with form validation and feedback. When a user adds a salad to the shopping basket 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, so in this context "a value" means anything but the empty string. This is inline with JavaScript, the empty sting is falsy. First, add `required` to the `<select>` fields. We also need an incorrect option for the select fields. Make sure the invalid option is pre-selected:

   ```
   const [foundation, setFoundation] = useState("");

   <select required ...>
     <option value="">Gör ditt val</option>
     ... more options
   </select>
   ```

Now press the submit button. You should get an error message from your browser. When the form validation fails, a `submit` event vill not be generated and `handleSubmit` will not be called. Let's replace the error message with your own and style it with bootstrap. There are two css classes in bootstrap for this: `valid-feedback`, and `invalid-feedback`. They will show or hide the element based on the `:valid`/`:invalid` pseudo classes. Use them to show feedback messages. The `<div className="invalid-feedback">Message</div>` should be sibling to the form field (`<select>`). The bootstrap css class will hide the element until the pseudo classes `: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 has not interacted with, so add the `was-validated` class on form submission. As an alternative, you can add it on the blur event for the fields to show the feedback on a per field base. Add the information of user interaction to the component state. Either a singel value for the entire field as shown bellow, or one flag for each field:

```
const [touched, setTouched] = useState(false);

handlesubmit(event) {
  setTouched(true);
  // ...
}

<form className={touched ? "was-validated" : ""} ...>
```

There is one more thing you need to do:

```
<form 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` and will generate a `submit` event even if the validation failds. You can check if a form is valid by calling `formElement.checkValidity()` on the form element, in `handleSubmit`:

```
if(!event.target.checkValidity()){ /* run when invalid */ }
```

Your `handleSubmit` should either:

- Create a salad and send it to `App` (`props.addSalad(newSalad)`), clear the form and hide any error messages: `setTouched(false)`
- Display the error message if the form validation fails: `setTouched(true)`

2. *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/5.3/components/alerts/`. Also, there are no html5 constraints that you can use to let the browser do the validation for you. Instead you need to check the form validity in your event handlers and manage the state manually (`onChange` in the check boxes).

   ```
   const [showExtrasError, setExtrasError] = useState(false);
   ```

   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.

3. Navigation is next. 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 beginning of the tutorial on react routers home page (until "Loading Data"):
`https://reactrouter.com/en/main/start/tutorial`

4. We will use the react router. Add it to your project using npm and start the development web server, in the terminal (type `q` and press enter, if you are running the development web server):

```
> npm install react-router-dom
> npm run dev
```

5. To use the features of a router all components needs to be a child of a `<RouterProvider>`. To ensur that nothing ends up outside the router by mistake we can add the `<RouterProvider>` to the top of the React component tree, above `<App>`. But where is that? How is `<App>` instantiated? To answer these questions let's see what happens when the browser loads your app. When you enter `http://localhost:5173/` in the url field of the browser, it will ask a web server at the local machine for the content to show. The server will answer with a default content, normally called `index.html`. You could ask for this file directly: `http://localhost:5173/index.html`. There is a `index.html` file in the root of your project. When you open it, there is no mention of `<App>`, just a `<div id="root">`. Change the page title (`<title>My Salad Bar</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 we know that this is the file viewed. There is one more thing in `index.html`: `<script type="module" src="/src/main.jsx"></script>`. This script bootstraps react and you can find the file in your project. Open it and you will find code that adds `App` to the DOM. Replace `App` with a `<RouterProvider>` here:

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import 'bootstrap/dist/css/bootstrap.css'
import { RouterProvider } from 'react-router-dom'
import router from './router.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>,
)
```

You will also need a router configuration. Place it in a new file, `router.jsx`

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

const router = createBrowserRouter([
  {
    Component: App,
    children: [
      {
```

```
        path: "compose-salad",
        Component: ComposeSalad,
      }, {
        index: true,
        element: <p>Welcome to my own salad bar</p>
      }]
  },
]);
export default router;
```

`App` do not have a path. This is a layout route. It will always be rendered. Place content that should be rendered on all pages here, for example headers and menus.

You also have an index page in the list of children. The index page will be rendered if the path ends after matching the parent (equivalent to `path=""`), making sure there is a match for `http://localhost:5173/` among the children. Note, it will not match `http://localhost:5173/index.html`.

Did you notice that the paths are hyphen separated word (”compose-salad”), commonly referred to as kebab case. This avoids percentage encoding of the urls, making it much nicer for humans to read.

There are two properties used in the configuration to describe what to render on a match, `Component` and `element`. `Component` takes a react component name (a reference to the function-object) while `element` takes a JSX expression as value. Use `element` if you do not have a component for this part of the UI, but do not put too much GUI details in the router configuration.

When you look in the browser, it still shows the `ComposeSalad` and `ViewOrder` components and not the welcome message. This is because `<App>` is rendered and you have not changed it. Open it and remove `ComposeSalad` and `ViewOrder`. Still no welcome message. To fix this you need to tell where the child content should render. This is done using the `<Outlet>` component. It is a good idea to clean up the render function before it gets too complex, so I moved the header and footer to separate components:

```
import { Outlet } from 'react-router-dom';
//...

function App() {
  // ...
  return (
    <div className="container py-4">
      <Header />
      <Outlet context={{ inventory, addSalad, shoppingBasket }} />
      <Footer />
    </div>
  );
}
```

This should show the welcome message, but you will get an error if you go to `http://localhost:5173/compose-salad`. This is because `props` becomes `undefined` in `ComposeSalad`. `props` set in `<Outlet>` will not be forwarded to the children. Instead we must use `context` to pass data to the rendered child routes. All children will see the same data, so `context` must be the union of the data any child needs. The data can be read in the child using the `useOutletContext` hook:

```
import { useOutletContext } from 'react-router-dom';

function ComposeSalad() {
```

```
  const { inventory, addSalad } = useOutletContext();
  //...
  return //...
}
```

Now `ComposeSalad` should work. Add a route for the view order and update the component to get the shopping basket from `useOutletContext()`. Your app should support the following paths:
`http://localhost:5173/compose-salad` → render the `ComposeSalad` component
`http://localhost:5173/view-order` → render the `ViewOrder` component
`http://localhost:5173/` → render the welcome message.

Assignment: If the user enters an invalid url the app will blow up with an error. This is not nice. Handle this by showing a "page not found" page. The header, navbar and footer should still be shown. *Hint* `https://reactrouter.com/en/main/route/route#splats`

6. We need a navigation bar so the user can jump between pages. Let's place it in `App`, between the header and the `Outlet` components:

```
function App (props) {
  // code to deal with state
  return (
    <div className="container py-4">
        <Header />
        <Navbar />
        <Outlet context={{ inventory, addSalad, shoppingBasket }} />
        <Footer />
    </div>
  );
}
```

When using the React router you use the `<Link to="my-path">` or `<NavLink to="my-path">` elements for the user to click on to navigate in your app, instead of native `<a href="my-path">` html elements. What about accessibility? Do not worry, `<Link>` use proper html tags and sets `aria-*` attributes as needed so screen readers know these are navigation elements. Use bootstrap classes to style the links, see `https://getbootstrap.com/docs/5.3/components/navs-tabs/#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/5.3/components/navbar/`.

```
import { Link } from "react-router-dom";
export default 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>);
}
```

Reflection Question: What is the difference between using `<Link>` and `<NavLink>` in your navigation bar? Try it: click on a tab and then outside it to move the focus away from the tab. You can also reload the page.

Try this: Start to compose a salad by selecting a few ingredients, switch to the view

order page, and the back to the compose salad page. What do you see? An empty form. The compose salad component it is deleted when you navigate away from and a new component is created when you return to the page. Consequently any earlier state is lost. It is tricky to preserve the form content. Options are not to use a router for this part of the app, or store the state outside the component. Both options bring a new set of challenges, so do not spend time to fix the lost form content annoyance.

7. Now you should have a working app with three pages. What happens when you order a salad? The only thing you see is that the form is cleared. You need to switch to the view order page to see it in the shopping basket. This user experience is not so good. Fix this by jumping to the view order page on a successful form submit. Read about navigating programmatically at `https://reactrouter.com/en/main/hooks/use-navigate`. Update `ComposeSalad` so it jumps to the view order page and show a confirmation message when a salad is ordered.

   Hint: Add a child route to `view-order`, example url:
`/view-order/confirm/123e4567-e89b-12d3-a456-426614174000`
This should show the shopping basket and a confirmation for salad 123e4567-e89b-12d3-a456-426614174000.

   Note: `useOutletContext()` will return the context from the nearest `<Outlet>`. Remember to forward the shopping basket in `ViewOrder`.

   Reflection question: What happens if the user writes an invalid uuid in the url?

   Reflection Question: What is the meaning of a leading `/` in a path, the difference between `navigate("view-order/confirm/123e4567-e89b-12d3-a456-426614174000")` and `navigate("/view-order/confirm/123e4567-e89b-12d3-a456-426614174000")`. Try it, look in the browser url field.

8. *Optional assignment:* Create a component, `ViewIngredient`, that shows the information from `inventory` 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 name of the extras, and a ”view info” button next to the select dropdown in `ComposeSalad`.
   - Add one route that matches `path='/compose-salad/view-ingredient/:name'` to your router configuration. This should be a child of the `compose-salad` path. A match should render a `ViewIngredient` component.
   - create a `ViewIngredient` component. It should view information about the `:name` ingredient, see `https://reactrouter.com/en/main/hooks/use-params`
   - Add a `<Outlet />` to `ComposeSalad`. This should render a `ViewIngredient` component.

9. This is all for lab 3. 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. The user gets relevant feedback when the form is not correctly filled. This covers the basic features and user experience in the app. In lab 4 we will add persistent storage to preserve state when the app is reloaded, and add communication to a server for fetching ingredient data and placing orders.

Editor: Per Andersson

Contributors in alphabetical order:
Ahmad Ghaemi
Alfred Åkesson
Anton Risberg Alaküla
Mattias Nordal
Oskar Damkjaer
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`