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.

Assignments

1. Let's start with 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 option and that the component state is is initialised to an empty sting, `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. Let's replace this with 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` and/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, do not set the class 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()){ ... }
```

On a successful form submission, clear the form and remove all `was-vaildated` classes. You do not want to show error messages befor the user have interacted with the form.

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.

   ```
   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 (press `ctrl-c` first, if you are running the development web server):

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

5. We need a navigation bar so the user can jump between pages. Let's place it in `App`, between the header and the `ComposeSalad/ViewOrder` 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 components. This is what i have after adding an empty placeholder for the navigation bar:

```
function App (props) {
  // code to deal with state
  return (
    <div className="container py-4">
        <Header />
        <Navbar />
        <ViewOrder />
        <ComposeSalad />
        <Footer />
    </div>
  );
}
function Header() {  return (
    <header className="pb-3 mb-4 border-bottom">
      <span className="fs-4">Min egen salladsbar</span>
    </header>
)}
function NavBar(props) { return <></> }
function Footer(props) { return ... }
```

When using the React router you use the `<Link 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 universal design? Do not worry, all the relevant `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/5.3/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/`.

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

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. Add a second link for the `ViewOrder` page. Go to your browser and klick on the links. The path in your browser changes (the url) and you see the navigations in the browser history. However, all pages look the same.

Reflection Question 2: What is the difference between using `<Link>` and `<NavLink>` in your navigation bar?

6. To view different content based on the url we need a router. 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. But where is that? How is `<App>` instantiated? To answer these questions lets see what happens when the browser loads your app. When you enter `http://localhost:3000/` in the url field, 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:3000/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 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 we know that this is the file viewed. If you open "View Source", under view/development in the browser, it looks similar but with one important addition: `<script defer src="/static/js/bundle.js"></script>`. This is the works of the React tool chain and `bundle.js` contains all the JavaScript code from your project. Specifically, it will include `src/index.js`. Open it and you will find code that adds `App` to the DOM. Replace `App` with a `<RouterProvider>` here:

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

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

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

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

const router = createBrowserRouter([
  {
    element: <App />,
    children: [
      {
        path: "compose-salad",
        element: <p>replace with compose salad component</p>
      }]
  },
]);
export default router;
```

Note, `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 her, for example headers and menus.
Reflection question 1: How do you replace the application icon, `favicon.ico`?

7. Let's render the page content based on the url. In the router configuration we already have a layout rout rendering `App` and a child for compose salad. The path `compose-salad` renders the text "replace with compose salad component", replace it with `<ComposeSalad`

/>. What about the props? *Either follow this instructions here, or use a context and reducer to pass data to children as described in the optional assignment in 8 bellow.* We do not have access to the state of App in the router configuration, so we can not pass inventory and the shopping cart to ComposeSalad here. Instead, pass the data as props to <Outlet > and use useOutletContext in the component where it is needed. <Outlet /> declares where the child rout element is rendered in its parent, which is App in our program. The same props are passed to all children, so pass all data any of the children needs in one object. Replace <ViewOrder/><ComposeSalad/> with <Outlet context={{ shoppingBasket, handleAddSalad, inventory }} /> in the JSX returned by App(). Now <ComposeSalad /> should render if you have the right url in the browser. Add more routs:
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. *hint:* index attribute, https://reactrouter.com/en/main/start/tutorial#index-routes
You can also add a *home* button in the menu for the welcome page.

Note: 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. When you navigate away from the compose salad component it is deleted. A new component is created when you return to the page. Consequently any earlier state is lost. To avoid this, you can lift the form state up in the component tree (bad design) or store the component state in a persistent storage, for example localstorage. You will get hands on experience with localstorage in lab 4. There are other tricks that can be used as well, for example keep ComposeSalad in the react component tree (move it from the rout configuration to App) and hide it using CSS (display:none) when the compose salad tab is not selected.

8. *Optional assignment*. Create a reducer for the shopping basket and pass it to the component tree using context as described in *Scaling Up with Reducer and Context* https://react.dev/learn/scaling-up-with-reducer-and-context. The inventory object never changes, so you do not need to pass it using props. Instead import it directly in any file where it is used. We will adress this in lab 4.

9. 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 heder, navbar and footer should still be shown. *Hint* https://reactrouter.com/en/main/route/route#splats

10. 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 rout to view-order, example url: "/view-order/confirm/123e4567-e89b-12d3-a456-426614174000". This should show a confirmation for salad 123e4567-e89b-12d3-a456-426614174000 and the shopping basket. What happens if the user writes an invalid uuid in the url?

    Note, you need to import "bootstrap/dist/js/bootstrap.bundle.min" for the dismiss button of bootstraps alerts to work. Place the import in App.

    Reflection Question 3: What is the meaning of a leading / in a path, the difference between <Link to="view-order/confirm/123e4567-e89b-12d3-a456-426614174000" /> and

`<Link to="/view-order/confirm/123e4567-e89b-12d3-a456-426614174000" />`. Try it, look in the browser url field.

11. *Optional assignment:* 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 `<NavLink ...>` around each ingredient name in `ComposeSalad`.
    - Add one route with `path='/compose/salad/view-ingredient/:name' ...>` to your router configuration.
    - Add a `<Outlet />` to `ComposeSalad`

    Add a ”view info” button next to the select dropdown. When the user clicks on it, show the information for the selected ingredient.

12. 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`

You can use this work if you respect this *LICENCE*: CC BY-SA 4.0
`http://creativecommons.org/licenses/by-sa/4.0/`
Please do *not* distribute your solutions to lab assignments and projects.

Copyright © 2015-2024.
Dept. of Computer Science, LTH, Lund University. Lund. Sweden.