Lab 2

This lab is about the react and bootstrap, *objectives*:

1. Understanding how a web page can be styled using css classes.
2. Get experience with basic react usage: components and props.
3. Get some experience using html forms.

This lab will take significantly more time to finish compared to lab 1.

Bootstrap

Open the bootstrap documentation to get an overview of the different bootstrap components to choose from. The pages contains examples, so it is easy to reuse the basic building blocks by copying the template code. Note, the examples uses HTML attribute names, but in you must use the equivalent React names. Replace `class` and `for` in the examples with `className` and `htmlFor`, for example replace `class="btn btn-primary"` with `className="btn btn-primary"`
`https://getbootstrap.com/docs/5.3/components/buttons/`

Set up

In the first lab you created JavaScript code to manage custom made salads. In this lab you will create a web page where a user can compose and order salads. On the course canvas page you find the instructions for creating a new react project. Follow this and replace `App.js` with the one in canvas. You should now have an app with a headline, a container box listing the extras, and a footer.

Assignments

1. Study the relevant material for lecture 3 and 4.

2. You will create React components during the lab. You may use function components with the `useState` hook, or class components.

3. Create a component for composing salads (download the template file from Canvas). All source files are stored in `./src` directory of your project:

```
import { useState } from 'react';

function ComposeSalad(props) {
  let foundations = Object.keys(props.inventory).filter(name => props.inventory[name].fou
  const [foundation, setFoundation] = useState('Pasta');
  const [extra, setExtra] = useState({Bacon: true, Fetaost: true});

  return (
    <div className="container col-12">
      <div className="row h-200 p-5 bg-light border rounded-3">
        <h2>Välj bas</h2>
        {foundations.map(name => <div key={name} className="col-4">{name}</div>)}
      </div>
    </div>
```

```
  );
}
export default ComposeSalad;
```

A few observations:

- Remember to export the component name, otherwise you can't instantiate it outside the file.
- Note the JSX code in the function, it looks like HTML with embedded JavaScript.
- `key={name}` helps react track witch part of the DOM to render when data changes, read about it in the react documentation.
- `className="container"` is a bootstrap class that adds some styling to the page so it looks nicer. Style other html elements using bootstrap css classes.
- JSX does not have comments, but you can use embedded JavaScript for that:

```
<span>  {/* this part won't do anything right now */}  </span>
```

Next we will instantiate ComposeSalad in App. Make sure `src/App.js` imports everything you will use. These are my imports:

```
import 'bootstrap/dist/css/bootstrap.css';
import { useState } from 'react';
import inventory from './inventory.ES6';
import ComposeSalad from './ComposeSalad'
```

Add a `ComposeSalad` component in App, in the returned JSX expression add:

```
<ComposeSalad inventory={inventory} />
```

Open the java script console in the web browser and read the output. jslint gives some warnings. Always have the java script console open to see messages from jslint and react. There should not be any warnings or errors when you have finished the lab.

Reflection question 1: As an alternative to the function component above you can use a class component: `class ComposeSalad extends react.Component {}`. Is there a difference between class components and function components concerning features (use cases where only one of them can be used)?

Reflection question 2: The render function must be a pure function of `props` and the component state, the values returned by `useState()`. What happens if the output of the render function is depending on other data that changes over time?

Reflection question 3: In the code above, the `foundations` array is computed every time the component is rendered. The inventory changes very infrequent so this is inefficient. Can you cache `foundations` so it is only computed when `props.inventory` changes?

4. In your `ComposeSalad` react component, add a HTML form for composing a salad. This requires a fair amount of code and knowledge. Divide the work to small incremental steps, for example:

   1. Read and understand how forms should be implemented in react, see `https://reactjs.org/docs/forms.html`.
   2. Read all requirements and hints bellow.
   3. `ComposeSalad` must be a controlled component, the React state is the "single source of truth"

4. Start with something small, perhaps a `<select>` for the foundation.
   - First write the code to render the `<select>` and `<option>` elements for the foundation. You wrote the code to render the `<option>` list in assignment 1 in lab 1.
   - Make sure the component state controls which option is selected ("Pasta" is the initial state in the code above)
   - Add an event handler to update the component state when the user selects a different foundation. Inspect the component state using the React Developer Tools plugin for Chrome or write the state to `console.log` to ensure things work correct. Remember, component state must be an immutable data structure. Do not modify the `extras` object, instead make a copy and use it as the next state. Also, to update the state you need to pass a function to `setExtras`: `setExtras( oldState => copyAndUpdate(oldStat) )`.
5. The `Salad` class is not suitable for holding the form state. Use strings for foundation, protein and dressing and an object containing the selected extras. Create a `Salad` object when the form is submitted.
6. When this works, add fields for protein, extras and dressing to the form. Structure your code. Use JavaScript functions, or components for the different parts of the form. For example create an option component that can be reused for choosing foundation, protein and dressing.

   ```
   <OptionsComponent
       options={foundations}
       value={foundation}
       onChange={onFoundationChange}
   />
   ```

7. *optional assignment:* add a "Caesar Salad" quick compose button. When the user clicks the button, the form is pre-filled with the selections for a Caesar sallad.

Reflection question 4: What triggers react to call the render function and update the DOM?
Reflection question 5: When the user change the html form state (DOM), does this change the state of your component?
Reflection question 6: For a class based component, what is the value of `this` in the event handling call-back functions?
Requirements:

- The form should limit the selection to one foundation, one protein, any number of extras, and one dressing when selecting html form elements and internal state.
- You do not need to support portion size (gourmet salad).
- One learning outcome of this lab is for you to get familiar with html and css. Therefore you must use native html tags, e.g. `<input> and <select>`, and style them using `className`. Most real world applications use frameworks, such as ReactBootstrap, which encapsulate the html tags and styling in react components. You should use this approach in the project bur not in the labs.
- You must use controlled components to handle form state. In the project you can use any from handling frameworks you desire.
- Your code must be flexible. If the content of `inventory` changes, your form should reflect these changes. Use iterations in JavaScript (`Array.map` is recommended), avoid hard coding each ingredient (you may not assume which ingredients are present in inventory)

Some hints:

- It is a good idea to create additional react components, or functions to render the elements that are repeated, for example SaladCheckbox, and/or SaladSelect (three instances: proteins, extras, and dressing).
- React is based on the *model-view* design pattern. `ComposeSalad` is the view and component state and `this.props` is the model. `ComposeSalad` contains all functionality for viewing the model. `Salad` is not aware of how it is visualised. Do not put any view details, such as html/react elements, in this class. This makes your data structures portable. You can reuse the `Salad` class in an Angular or Vue.js application.
- If you use class components, remember to bind your callback functions, or use arrow functions: `this.handleChange = this.handleChange.bind(this);` Read why you sometimes need to bind your callbacks here:
  `https://reactjs.org/docs/handling-events.html`.
- Use checkboxes when more than one item can be selected, see the bootstrap documentation on how to style them. Use the html elements `<input type='checkbox'>` and `<label>`.
- For checkboxes, the state of the DOM-element is stored in the property named `checked` (for other `<input>` types, the DOM state is stored in the property `value`). Do not assign `undefined` to it. To avoid this, you can use the JavaScript short cut behaviour of `||`
  `<input checked=(extras['Tomat'] || false)>`.
- `<imput>` elements have a `name` attribute. For checkboxes you can use it to store which ingredient it represents. In your callback function it is available in `event.target.name`.
- You may assume correct input for now, we will add form validation in the next lab.

5. Handle form submission. The salad in the form should be added to a shopping cart when the user submits the form. The shopping cart should be stores in the `App` component.
   - When the form is submitted, you must create a `Salad` object from assignment 1 to store it. To import the file, you need to change it to a ES6 module, see the `inventory.ES6.js` as an example.
   - The shopping cart is just a list of salad objects, use an array if you did not do the optional task in lab 1.
   - When the form is submitted, pass a Salad object to the parent, i.e. `App`. `App` should only handle `Salad` objects and not bother about the internals of the `ComposeSalad`, i.e. creating the object from the the html form state. Remember, the user might want to compose several salads, so make sure to copy/create objects when needed.
   - `onSubmit` is the correct event for catching form submission. Avoid `onClick` on the submit button, it will miss submissions done by pressing the enter key in the html form.
   - Clear the form after a salad is ordered, so the customer can start composing a new salad from scratch.
   - The default behaviour of form submission is to send a http GET request to the server. We do not want this since we handle the action internally in the app. Stop the default action by calling `event.preventDefault()`. If you forget this then the app will be reloaded and JavaScript/component state will be lost (submit will result in an empty shopping cart).

Reflection question 7: How is the prototype chain affected when copying an object with `copy = {...sourceObject}`?

6. Create a react component, `ViewOrder`, to view the shopping cart and add it in `App`. The shopping cart should be an input to the component, as `inventory` is in `ComposeSalad`.

7. Add the `ViewOrder` component to `App`, i.e. `<ViewOrder shoppingCart={shoppingCart}>`. This demonstrates the declarative power of react. When the state changes all affected subcomponents will automatically be re-renderd.

   An order can contain several salads. Remember to set the `key` attribute in the repeated html/JSX element. Avoid using array index as key. This can break your application when a salad is removed from the list. This is explained in many blog posts, for example `https://medium.com/@robinpokorny/index-as-a-key-is-an-anti-pattern-e0349aece318`.
   Hint 1: use the `uuid` property in the `Salad` objects as key.

8. *Optional assignment 1:* Add a remove button to the list of salads in the `ViewOrder` component. Remember, `props` are read only. The original list is in the `App` component.

9. *Optional assignment 2:* Add functionality so the user can edit a previously created salad. Add an edit button to each row in the list of salads in the `ViewOrder` component. For conditional rendering of a component you can use any JavaScript condition, `if...then...else` or `{editMode && <ComposeSalad />}` You also need modify the `ComposeSalad` component so it can be used for editing. Use `props` to pass the salad to be edited. `App` will not initialise this prop, so it will be `undefined`. Use this to determine if the `ComposeSalad` component is in create or edit mode when needed, for example the the text for the submit button (create/update). Note: do not update the salad object in the order until the update button is pressed.

   The edit scenario is a good use case for a modal wrapper around the `ComposeSalad` component. For edit, a pop-up window will appear, and when done the user is back in the list of the salads.
   Hint: Do this assignment in two steps, first add the functionality to view the salad, then continue with changes needed to save the updated salad.

10. This is all for now. In the next lab we will introduce a router and move the `ComposeSalad` and `ViewOrder` to separate pages.

Editor: Per Andersson

Contributors in alphabetical order:
Alfred Åkesson
Anton Risberg Alaküla
Mattias Nordal
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`