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 copy the template code. Note, the examples ar HTML and uses `class="btn btn-primary"`. In a react component you must use the react template syntax:`className="btn btn-primary"`
`https://getbootstrap.com/docs/5.1/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, see `https://canvas.education.lu.se/courses/17001/pages/react-skapa-ett-projekt`.

In this lab we use ECMAscript modules, so download the ES6 variant of the inventory file from the canvas page (labb 2 assignment).

Assignments

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

2. If you are using the linux system at LTH, remember to run `initcs` to add `node` to the path.

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

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

class ComposeSalad extends Component {
  constructor(props) {
    super(props);
    this.state = {};
  }

  render() {
    let extras = Object.keys(this.props.inventory)
                 .filter(name => this.props.inventory[name].extra);
    return (
      <div classNameer="continer col-12">
      <div className="row h-200 p-5 bg-light border rounded-3">
        <h2>Välj innehållet i din sallad</h2>
```

```
            {extras.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 `render()` function, it is a mix of HTML and 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 you add with 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 { Component } from 'react';
import inventory from './inventory.ES6';
import ComposeSalad from './ComposeSalad';
```

Add `ComposeSalad` in the render function:

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

Reflection question 1: `ComposeSalad` is a class extending `react.Component`. Is there a difference between class based components and function components, for example `function App(){...}` that the react template builder created for you?
Reflection question 2: The render function must be a proper function of `this.props` and `this.state`. What happens if the output of the render function is depending on other data?
Reflection question 3: In the code above, `extras` are computed every time a 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 understanding. Divid it to smaller steps:
   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. start with something small, perhaps a `<select>` for the foundation.
   4. first write the code to render the `<select>` and `<option>` elements. When this looks good, add the calback functions and update the component state.

5. make sure the component state is updated when you interact with the form (view the state with React Developer Tools plugin for Chrome or `console.log`) *Remember,* you must use `this.setState({stateProp: newValue})` to update the state. You may not modify the objects referred to by the state. Therefore, do not use `Salad` objects to represent the form state. Create the `Salad` object when the form is submitted.
6. when this works, continue with the the proteins, extras and dressing.
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: What is the value of `this` in the event handling call-back functions?
Requirements:

- You should assume one foundation, one protein, any number of extras, and one dressing when selecting html form elements ind 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.
- 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). You can pass bound functions to subcomponents to keep the state and callback functions in `ComposeSalad`.
- React is based on the *model-view* design pattern. `ComposeSalad` is the view and {`this.state, 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.
- Use the html `name` attribute to pass additional data to your callback functions. When you have a `<select name="foundation" value=this.state.["foundation"]>` element, you can use this generic event handle function:

```
handleSwitchChange(event) {
   this.setState({[event.target.name]: event.target.value});
 }
```

  You must understand this code if you use it. For check boxes, replace `value` with `checked`

- 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. The html elements to use are `<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=(this.state['Tomat'] || false)>`.
- `<select>` and `<option>` might be good alternatives for selecting the foundation and dressing.
- `<imput>` elements have a `name` attribute. 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.
   - 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()`.

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

6. Create a react component to view the shopping cart. 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 order='this.state.order'>`. This demonstrates the declarative power of react. When the state changes all affected subcomponents will automatically be re-renderd. Remember to use `this.setState(newValues)` to update the state.

   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 `uid` 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. 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 'codeundefined. 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
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`