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.

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.
`https://getbootstrap.com/docs/4.4/components/jumbotron/`

Background

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 home page you find the instructions for creating a new react project, see `http://cs.lth.se/edaf90/labs/`.

If you do not intend to use bootstrap modals, skip the `npm install jquery popper.js` step. If you change your mind, you can always add them later.

In this lab we use ECMAscript modules, so download the ES6 variant of the inventory file:

```
> cd lab2
> curl -o src/inventory.ES6.js
         http://fileadmin.cs.lth.se/cs/Education/EDAF90/labs/lab2/inventory.ES6.js
```

Assignments

1. Study the relevant material for lecture 3, see `http://cs.lth.se/edaf90/reding-instructions/lecture-3/`.

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

3. To compose a salad we will need to know what it can contain. In `src/App.js` add:

   ```
   import inventory from './inventory.ES6';
   ```

4. Create a component for composing salads. Pass `inventory` to it using `props`. I suggest you name it ComposeSalad:

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

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

```
  render() {
    const inventory = this.props.inventory;
    let foundations = Object.keys(inventory).filter(name => inventory[name].foundation);
    return (
      <div className="container">
        <ul>
          {foundations.map(name => <li key={name}>{name}</li>)}
        </ul>
      </div>
    );
  }
}

export default ComposeSalad;
```

A few observations: 1, remember to export the component name, otherwise you can't instantiate it outside the file. 2, note how you can mix JavaScript and JSX in the `render()` function. 3, `key={name}` helps react track witch part of the DOM to render when data changes, read about it in the react documentation. 4, `className="container"` is a bootstrap class that adds some styling the page looks better. Style other html elements you add with bootstrap css classes. 4, JSX does not have comments, but you can use embedded JavaScript for that:

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

5. Lets use the component, instantiate it in `App.js`:

```
import ComposeSalad from './ComposeSalad';

// add this line to the existing JSX in your render() function:
<ComposeSalad inventory={inventory} />
```

6. In your `ComposeSalad` react component, add a html form for composing a salad, see `https://reactjs.org/docs/forms.html`. Some hints:

   - The `ComposeSalad` should only render the html form. If you want to use modals, place that code in a separate component, `ComposeSaladModal`, or in `App`. `ComposeSaladModal` is recommended since it makes your code more reusable. We will use a router later and then there is no use for a modal.
   - React is based on the *model-view* design pattern. `ComposeSalad` is the view and `Salad` is the model. `ComposeSalad` contains all functionality for viewing the model. `Salad` are not aware of how it is visualised. Do not put any view details, such as html/react elements, in this class. This makes your code portable. You can reuse the `Salad` class in an Angular or JavaFX application without modification.
   - Remember to bind your callback 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, 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 `value` is used). 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.
- Use iterations in JavaScript (`Array.map` is recommended), avoid hard coding each ingredient (your may not assume which ingredients are present in inventory, so the 'Tomato' part of the example above is not ok)
- It is a good idea to create additional react components, for example SaladCheckbox, and/or SaladSection (two instances, one for extras and one for proteins). You can pass bound functions to subcomponents if you prefer to keep the callback functions in `ComposeSalad`.
- `<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 part two of this lab.
- When the form is submitted, create a new Salad object and pass it to the parent, i.e. `App`. The `App` should only handle `Salad` objects and not bother about the internals of the `ComposeSalad`, i.e. the form state. The `onSubmit` is the right trigger. Avoid `onClick` on the submit button, it will miss submissions done by pressing the enter key in the form.
- Clear the form after a salad is ordered, so the customer can start on a new salad from scratch. Note, you can not remove properties from the state object, just change their value. There are two alternatives: set values to `false` to indicate that this option is not selected, or store the form state in an object stored in a state property, i.e. `this.setState({formValues: {}})};`.
- 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()`.

7. Store the orders salads in the state of `App`.

8. Create a react component to view the salad order. The order should be an input to the component, as `inventory` is in `ComposeSalad`.

9. 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 kay. This can break your application when a salad is removed from the list. This is explained in many blog posts, for example see `https://medium.com/@robinpokorny/index-as-a-key-is-an-anti-pattern-e0349aece318`.

10. This ends part one of the lab. In the next part 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`