Introduction to React

Bjarte Wang-Kileng

HVL

September 15, 2025

Western Norway
University of
Applied Sciences

About React

- React is also named ReactJS
- The V of MV* (*Model-View-something*).
- Maintained by Facebook and a community of individual developers and companies.
- React is usually used togheter with CSS Modules and JSX.
  - JSX was influenced by XHP, an HTML component library for PHP.
- CSS Modules and JSX must be translated to CSS and JavaScript.
  - CSS Modules and JSX are not understood by browsers.
- For development, React is possible through JavaScript libraries only.
  - CSS modules and more do require the build approach though.

Create React application using the Vite framework

- Approach depend on choice of build tool and development server.
  - I will use Vite, a common choice for React development.
- Create React application *dat152-project* from React template:

```
npm create vite@latest dat152-project -- --template react
```

- Install all project dependencies:

```
cd dat152-project
npm install
```

- Start the development web server:

```
cd dat152-project
npm install
```

- Build CSS, JavaScript and HTML for production deployment:

```
npm run build
```

React application file structure

- File **index.html** – Starting point for the application.
- File **vite.config.js** – Main configuration file.
  - Add property *base* unless hosted at web server root.

```
// Set base path relative to the current HTML file
export default defineConfig({
    plugins: [react()],
    base: './',
})
```

- Folder **public folder** – Can be used by **index.html**.
- Folder **src** – Application source code.
- Folder **dist** – Result of application build.
- Folder **node_modules** – Used by **npm** to create the web application.

The index file

- The file **index.html**:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" ... />
    <title>Welcome to DAT152</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

- Tag *DIV* with *id* **root** will be the container of the React application.
- The build process will prepend the correct path to the URLs.
  - Property *base* set in **vite.config.js**.
- Build process will translate content of **src** to JavaScript and CSS.

The file **main.jsx**

- The file **main.jsx**:

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

- Build process will replace tags with function calls.
  - Older versions of React was class based.
- *StrictMode* is a development tool that highlights potential issues with the application.
  - Has no impact on a production build.

The file **App.jsx**

- The file **App.jsx**:

```
import './index.css'

function Welcome({courseName}) {
    return (
        <h1 className='courseintro'>Welcome to {courseName}</h1>
    );
}

function WelcomeDAT152() {
    return (
        <Welcome courseName='DAT152' />
    );
}

export default WelcomeDAT152;
```

- *Welcome* is called with attributes of tag enveloped in an object.
- With React, JavaScript can import CSS files.
- Observe that HTML attribute *class* is replaced with JS *className*.

React with Eclipse

- The commercial plugin *React::CodeMix* is no longer maintained.
- The *Wild Web Devloper* (WWD) plugin has support for React.

React development with Eclipse

- Build first a React template project from command line.
- Use e.g. *Eclipse IDE for Enterprise Java and Web Developers*.
- Create a new project, e.g. a *Static Web Project*.
- Import the React template project into the new Eclipse project.
- Start of development server and project build from command line.

CSS modules

- Class names and animation names are scoped locally by default.

```
import studentStyle from './student.module.css';

function Person() {
    return (
        <p className={studentStyle.person}>Ole Olsen</p>
    );
}
```

- Class composition lets CSS classes inherit from other classes.

```
.person {
    background-color: red;
}

.student {
    composes: person;
    color: blue;
}
```

- Class can be scoped globally using the keyword *:global*.

JSX

- JSX creates React elements.

```
element = <p>Pi is approximately 3,14.</p>;
```

- Observe that JSX must be a single element.

```
/* The code below is illegal */
element = <em>One</em> <em>two</em>;
```

```
/* The code below is OK */
element = <span><em>One</em> <em>two</em></span>;
```

- React has a container element that can gather multiple elements.

```
element = <React.Fragment><em>One</em> <em>two</em></React.Fragment>;
```

- The container element has an alternativ short syntax:

```
element = <><em>One</em> <em>two</em></>;
```

JSX expressions

- JSX can embed expressions.

```
const number = 3.14;
const element = <p>Pi is approximately {number}.</p>;
```

- JavaScript expressions and functions can be used in JSX expressions.

```
const number = Math.PI;
const element = <p>Double Pi is approximately {2*number}.</p>;
```

```
const noFmt = new Intl.NumberFormat("nb-NO");
const element = <p>Pi is approx {noFmt.format(Math.PI)}.</p>;
```

- Arrays are expanded.

```
const taskRows = [
        <tr key={1}><td>Wash windows</td><td>ACTIVE</td></tr>,
        <tr key={2}><td>Clean floor</td><td>DONE</td></tr>,
    ];

const taskTable = <table>{taskRows}</table>;
```

JavaScript logical expressions for JSX expressions

- Using logical && in JSX expression:

```
<p>Found {count} task{count!==1 && 's'}.</p>
```

- Using inline if-else in JSX expression:

```
<p>Found {count===1 ? 'one task' : `${count} tasks`}.</p>
```

- Observe that the expression must return a single element.

```
{/* Below code is illegal since last part are two elements */}
<p>Found {count===1 ? 'one task' : {count} tasks}.</p>
```

- We can use the container element if several elements.

```
<p>Found {count===1 ? 'one task' : <>{count} tasks</>}.</p>
```

JSX and JavaScript

- JSX becomes JavaScript when translated.
  - JSX:

```
const element = <a href="https://eple.hvl.no">eple</a>;
```

  - Translated JavaScript:

```
// Simplified
const element = React.createElement(
    'a',
    {href: 'https://eple.hvl.no'},
    'eple'
);
```

- JSX is protected from XSS attacks from text strings.

React components

- React elements are immutable.
  - Element can be replaced by a new element, but
  - element can not be modified.
- Functional component can return a React component.
- Name of function of component must start with a capital letter:
  - Correct:

```
function Person() {
    ....
}
```

  - Wrong:

```
function person() {
    ....
}
```

React component state

- React components have an immutable state.
  - State properties can not be modified, but can be replaced.
- View is updated if React component changes state.

```
const [count,setCount] = React.useState(initvalue);
```

  - Will initialize state if not set (*initvalue*).
  - Will return the current value (*count*), or initial value on initialization.
  - Returns method to update state to a different value (*setCount*).
- Update state to a different value:

```
setCount(newcount);
```

React hook *useEffect*

- Hooks give functional component access to React features.
  - We have already met the hook *React.useState*.

- Example on the use of the *useEffect* hook:

```
function componentDidMount() { ... }
function componentWillUnmount() { ... }

React.useEffect(() => {
    componentDidMount(); // Running method

    return componentWillUnmount; // Function reference
});
```

  - The function argument is run when component is inserted into view.
  - The return value function is run before component is removed.

- For more hooks, see the React documentaton.

Parameters to React components

- Parameters can be specified when using React components:

```
<NumTasks count="1" />
```

- Functional component is given an object as argument:

```
function NumTasks(props) {
    if (props.count === 1) {
        return (<OneTask />);
    } else {
        return (<MultipleTasks count={props.count} />);
    }
}
```

  - Object **props** have properties corresonding to the tag attributes.

- Observe that the property object is read-only.

Modifying the View

- In JavaScript DOM we modify the View, e.g. add element.
  - E.g. DOM methods *appendChild*, *removeChild* and *innerHTML*.
- Using Angular we modify JavaScript objects.
  - Angular will update the View accordingly.
- Using React, we always produce the full View.
  - No remove or add methods on the View.
  - Views are not updated, but replaced with a new full View.
  - Similar to a movie, each frame is a complete new image, not the difference from the previous frame.
  - We remove a row from an HTML table by creating a new HTML table with the row gone.
- Observe, we create the full new view in React, but internally React only propagates the necessary changes to the real DOM.

Virtual DOM

- Concept used by e.g. React and Vue.
- Real DOM is the DOM that is viewed in the browser.
- Virtual DOM is a representation in memory of real DOM.
- Any state change will produce a new virtual DOM tree.
- React maintains both the updated and old virtual DOM tree.
- Uses the difference beween the virtual DOMs to modify real DOM.
  - Only the difference is propagated to real DOM.

Events in React

- Mixture of W3C DOM and HTML on-tag approach.

```
function handleEvent(event) {
    window.alert(`Event ${event.type} on ${event.currentTarget.tagName}`);
}

function EventIntro() {
    return (
        <button onClick={handleEvent}>Click me</button>
    );
}
```

- Event names are camelCase.
  - *onClick*, not *onclick*.
- Argument must be a function reference, not a function call.
  - Correct: `onClick={function}`
  - Wrong: `onClick={function()}`

More on Events

- The React SyntheticEvent encapsulates the underlying DOM Event.
  - Not all DOM Events exists in React as a SyntheticEvent.
- Not possible with multiple event handlers for same event on target.
- Events handlers by default trigger in bubbling phase.
- Add **Capture** to event name for capturing phase.

```
<button onClickCapture={handleEvent}>Click me - Capturing phase</button>
```

HTML rows or lists from JavaScript Arrays

- The Array method *map* can create a new Array from an Array.
  - A supplied function is applied on every array element.

- JSX with *map* is handy when making HTML row and list elements.

```
function TaskList() {
    const tasks = [
        {"id": 1, "title":"Paint roof","status":"WAITING"},
        {"id": 3, "title":"Wash windows","status":"ACTIVE"}
    ];

    const taskHTMLRows = tasks.map(
        task => <tr key={task.id}><td>{task.title}</td><td>{task.status}</td></tr>
    );

    return (
        <table>
            {taskHTMLRows}
        </table>
    );
}
```

- React requires that each Array element must produce a unique attribute *key* in the React element.
  - The *key* attribute only needs to be unique within the same array (ref).

Read-only and modifiable form control elements

- Some form control elements are read-only.
  - Value can be read, but not modified with JavaScript.

```
<input type="file" />
```

- Most form control elements are modifiable.
  - Value can both be read and modified with JavaScript.

```
<input type="text" />
```

- Observe that React requires an end tag for the input element.

Modifiable form elements in React

- Form element state should be controlled by React.

```
function MemberInput() {
    const [value, setValue] = React.useState('');

    const valueChange = (event) => {
        setValue(event.target.value);
    };

    return (
        <form>
            <input type="text" value={value} onChange={valueChange}/>
        </form>
    );
}
```

- To-way binding between state and view:
  - React component state should reflect HTML control element value.
  - HTML control element value should reflect React component state.

Form control elements in JSX and HTML

- Elements *textarea* and *select* have different syntax in JSX and HTML.

Form control element *textarea*

HTML: Content is given by its text child.

```
<textarea>Value of element</textarea>
```

JSX: Content is given by an attribute *value*.

```
<textarea value="Value of element" />
```

Form control element *select*

HTML: Default is specified by an attribute *selected* on an *option*.

```
<select>
    <option value="apple">Apple</option>
    <option value="pear" selected>Pear</option>
    <option value="banana">Banana</option>
</select>
```

JSX: Default value is specified by an attribute *value* on *select*.

```
const fruit =
    <select value="pear">
        <option value="apple">Apple</option>
        <option value="pear">Pear</option>
        <option value="banana">Banana</option>
    </select>
```

Communication between components

- A React component example:

```
function CustomerView() {
    return (
        <>
            <CustomerForm/>
            <CustomerList/>
        </>
    );
}
```

  - **CustomerForm** adds a customer, to be viewed in **CustomerList**.
  - Customer id from **CustomerForm** must not exist in **CustomerList**.

- Ids must propagate from **CustomerList** to **CustomerForm**.
- Customer must propagate from **CustomerForm** to **CustomerList**.
- Callbacks must be used to propagate the changes.

Lifting the state

- When components must be consistent on data, React recommends to propagate to the state of the closest common ancestor.
- **CustomerView** is the closest common ancestor in the example.
  - Form control elements of **CustomerForm** should be controlled by **CustomerView**.
  - Customer list of **CustomerList** should be set as a state property of **CustomerView**.
- Changes in **CustomerForm** or **CustomerList** must trigger a *setState* in **CustomerView**.
  - No use of *setState* in **CustomerForm** or **CustomerList**.

What Next?

- Next.js is a React framework for SPA.
- Server side solution for React.
- Routes pages from server to a React SPA.