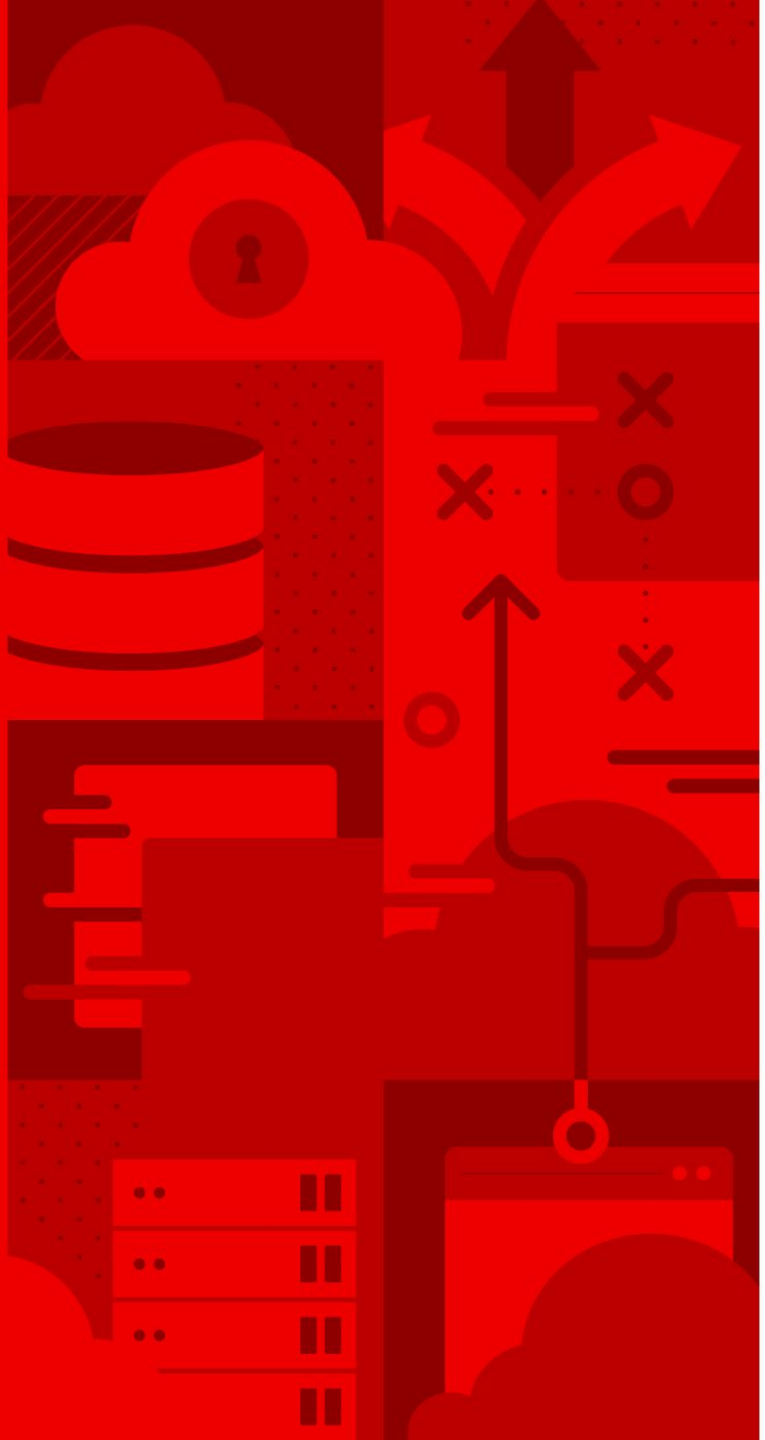


JavaScript & React basics

Let's mix some HTML and JS together

Red Hat

Agenda

History of Javascript - the old and the new

What do we need to create a web UI?

Tools and technologies

New goodies in JavaScript

Web UI key principles

React basics - props, state, component lifecycle

Project setup

JavaScript Dark age ended like 10 years ago...

1995
Brendan Eich

React
A JavaScript library for building user interfaces
Uses Shadow DOM

ES6
Big improvement to JS
Prepared for many years
Introduces a lot of new features

2006-03-22

2013-05-29

2015-01-12

2015-06-17

2016+

jQuery
The most ❤️ and 💩 DOM manipulation library

Babel
Previously known as 6t5 babel
Transpiles new JS to old one

ES7 end ES next
From now on ES committee creates every year new standard

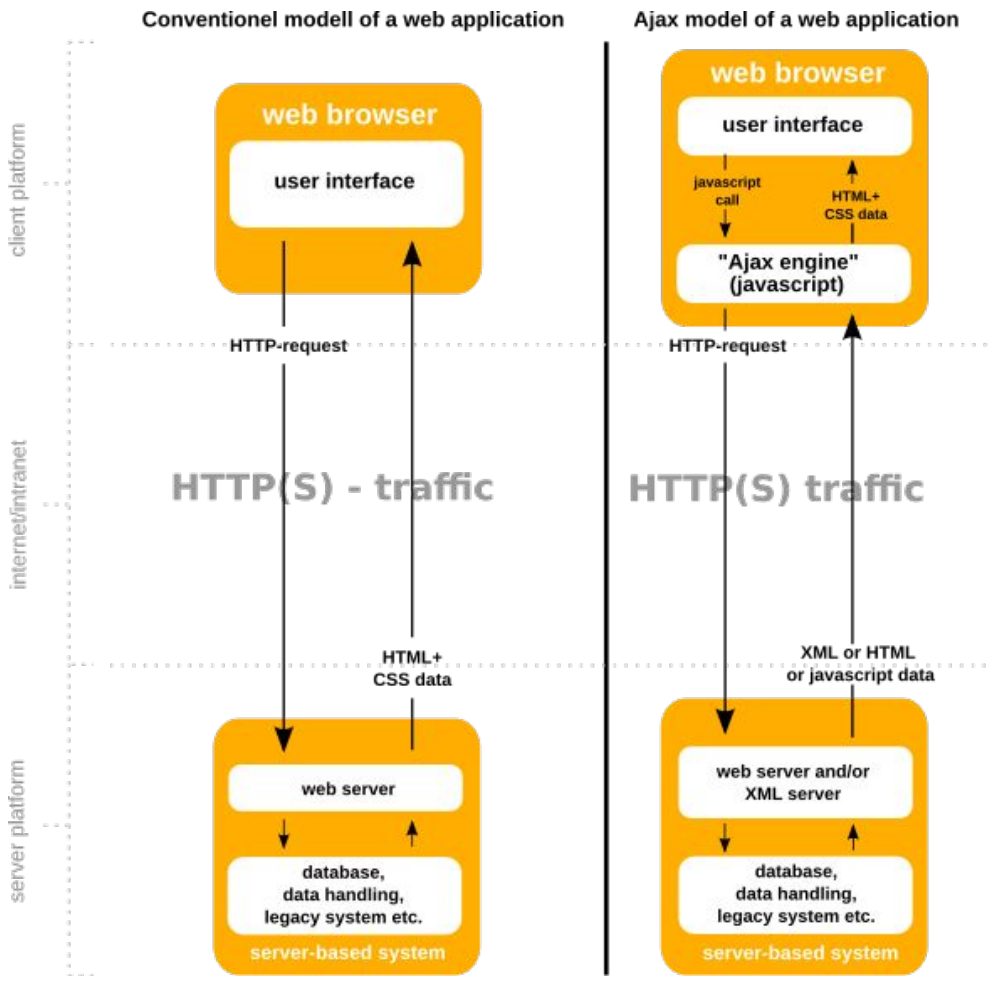
Red Hat

What do we need to create a web UI?

- web browser (see the page & debug)
- code editor (VS code)
- programming language (JavaScript)
- library to help us create the app (React)
- package manager (NPM)
- transpile new code features for older web browsers (Babel)
- bundle JS in a single file (Webpack)
- deploy the app (OpenShift, Surge)

Ajax (Asynchronous JavaScript and XML)

- Also known as XHR (XMLHttpRequest)
- At first used to exchange XML documents (Extensible Markup Language)
- Asynchronously exchange data with server
- No longer needed to reload entire page just to get the data
- Need for clever way how to identify user
- Polling was used to update data on the fly



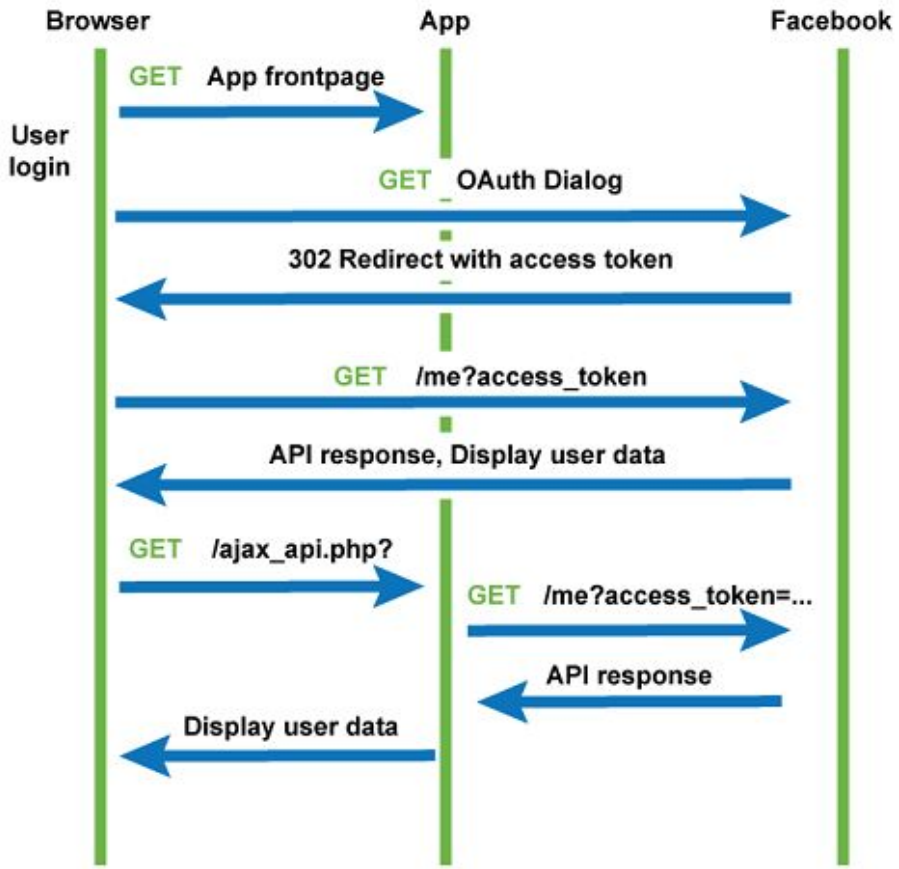
https://en.wikipedia.org/wiki/Ajax_(programming)

Single session

- User logs in and is given a specific ID on server called session ID
- User performs asynchronous operations
- Once user is done, browser is closed and server resources are cleaned
- Resources consuming
- User has to log in every time new browser is opened (if application allows that)
- Session hijacking (no need to know the password, just listen on specific events and get the session ID)

Single sign on/out

- Transfers the need of tracking the log-in state from server to browser
- Better user experience - no need to log in for each application
- Allows you to login with social media and other providers
- Allows user to automatically log out of all devices if there are some suspicions
- Uses JWT and Oauth 2
- JWT - header, pyload and signature
- Oauth 2 - protocol for authorization
- Multiple services to take care of for us
  - Keycloak, Auth0



Red Hat

Internet explorer was the master

- First there was Netscape
- Internet explorer followed and gained a lot of traction because they were improving browser API and were the leading giants
- IE used to had some quirks in order to support their paid customers and these changes were not adopted by other browsers because they were not standardized
- Firefox was at that time heavily improving standards and adopting them
- Opera and Chrome followed
- Now Chrome has the leading (63.56%) followed by Safari (19.85%) with IE and Edge on third place (5.43%)

https://www.w3counter.com/globalstats.php

jQuery

- Easy to understand wrapper around JavaScript
- In its prime, it was the most used JavaScript library
- People could program in jQuery, not JavaScript
- To this date, for many people it is still a go to library when they need some quick prototype
- With the evolution of the language “$” is becoming obsolete
- And here is why:

```
$(el).fadeIn();
```

→

```
el.classList.add('show');
el.classList.remove('hide');


.show {
  transition: opacity 400ms;
}
.hide {
  opacity: 0;

}
```

New vs old

```javascript
class Something {

  constructor(logMeIn) {
    this.logMe = logMeIn;
  }
  method = () => {
    console.log(this?.logMe || 'I am method!')
  }
}

const firstStuff = new Something();
const secondStuff = new Something('I like trains');

firstStuff.method();
secondStuff.method();
```

```javascript
"use strict";

function _instanceof(left, right) { if (right != null && typeof Symbol !== "undefined" && right[Symbol.hasInstance]) { return !!right[Symbol.hasInstance](left); } else { return left instanceof right; } }

function _classCallCheck(instance, Constructor) { if (!_instanceof(instance, Constructor)) { throw new TypeError("Cannot call a class as a function"); } }

function _defineProperty(obj, key, value) { if (key in obj) { Object.defineProperty(obj, key, { value: value, enumerable: true, configurable: true, writable: true }); } else { obj[key] = value; } return obj; }

var Something = function Something(logMeIn) {
  var _this = this;

  _classCallCheck(this, Something);

  _defineProperty(this, "method", function () {
    console.log((_this === null || _this === void 0 ? void 0 : _this.logMe) || 'I am method!');
  });

  this.logMe = logMeIn;
};

var firstStuff = new Something();
var secondStuff = new Something('I like trains');
firstStuff.method();
secondStuff.method();
```


Red Hat

New is always better!

```javascript
class Something {

  constructor(logMeIn) {
    this.logMe = logMeIn;
  }
  method = () => {
    console.log(this?.logMe || 'I am method!')
  }
}

const firstStuff = new Something();
const secondStuff = new Something('I like trains');

firstStuff.method();
secondStuff.method();
```

```javascript
"use strict";

function _instanceof(left, right) { if (right != null && typeof
```


```javascript
_this.logMe) || 'I am method!');
  });

  this.logMe = logMeIn;
};

var firstStuff = new Something();
var secondStuff = new Something('I like trains');
firstStuff.method();
secondStuff.method();
```


Red Hat

Variables and their scoping - old and new

- Global - **var**
  - Variable leaking
  - Variable overriding

- Local scope - let and const (new way)
  - **let** - can be changes, meaning its data type and value can be changed
  - **const** - can't be changed, meaning its data type can't be changed but inner value can

```
const myVariable = 'hello';
// error!
myVariable = 'something';

const otherVar = { foo: 'bar' };
// correct!
otherVar.foo = 'baz';
```

Callback hell

- A way how to execute some code after asynchronous action happened
- Chain functions together
- Choose function based on some conditions

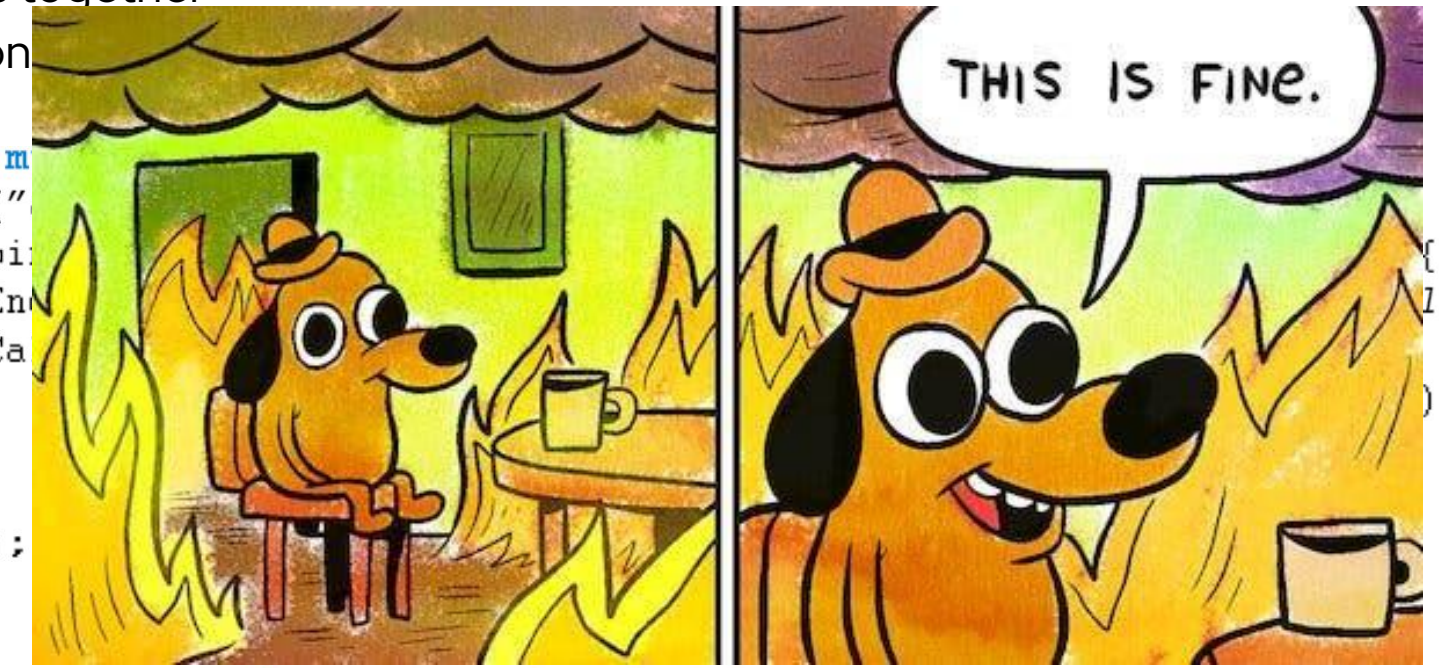
```
// First, we must get user by id
CallEndpoint("api/getidbyusername/hotcakes", function(result) {
    CallEndpoint("api/getfollowersbyid/" + result.userID, function(result) {
        CallEndpoint("api/someothercall/" + result.followers, function(result) {
            CallEndpoint("api/someothercall/" + result, function(result) {
                CallEndpoint("api/someothercall/" + result, function(result) {
                    // Ahhhhhhh... but you didn't believe you'd end up here
                });
            });
        });
    });
}); // Always use semicolons; at work and at home.
```

Callback hell

- A way how to execute some code after asynchronous action happened
- Chain functions together
- Choose function

```
// First, we m
CallEndpoint("
    CallEndpoi                                                    {
        CallEn                                                    lt) {
            Ca
                                                                  ) {


            });
        });
    });
}); // Always use semicolons; at work and at home.
```



Callback hell

- A way how to execute some code after asynchronous action happened
- Chain functions together
- Choose function based on some conditions

```
// first we must get user by id
const hotcakes = await
CallEndpoint('/api/getidbyusername/hotcakes');
const followers = await CallEndpoint('/api/getfollowersbyid');
const other = await CallEndpoint('/api/someothercall');
const another = await CallEndpoint('/api/someothercall');
const andOther = await CallEndpoint('/api/someothercall');
```

Transpilers

- Transpiler is a tool, that takes a code written in one language and transforms it into a different language (or different version in some cases)
- Due to browser backwards compatibility we can't just use the latest features of JS. They would not work.
- Code has to be transformed into something browsers understand
- There are also many variations of the language that will never be supported in browsers. Yet we want to use them.
  - JSX
  - Typescript
- JS is becoming more and more compiled language
- Many speak of an "Age of a Transpilers"

Transpilers

- Transpi[illegible]akes a code w[illegible]ten in [illegible]ge and [illegible] it int[illegible] diff[illegible]guage (or [illegible]erent vers[illegible]n s[illegible]ses)
- [illegible]ue to [illegible]ser back[illegible]s compa[illegible] we c[illegible] just [illegible]e [illegible]st features of [illegible] They would [illegible] work.
- Code [illegible]formed [illegible] thin[illegible]wsers [illegible]le [illegible]nd
- The [illegible] also ma[illegible] of t[illegible]lanc[illegible]ge that [illegible]e [illegible] be supported [illegible]rowsers. Ye[illegible]e want to u[illegible]em
  - [illegible]USX
  - Ty[illegible]t
- [illegible]coming more and more [illegible]mpiled language
- Many speak of an “Age of a Tr[illegible]spilers”

Red Hat

Type checkers

- Typechecker is a tool to help developers keep code stable by introducing types
- Since many languages (JavaScript included) have no types it can be hard to do complex features

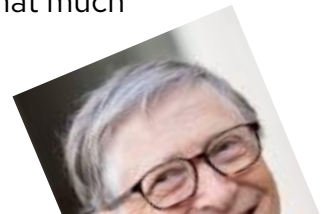
Type checkers - Typescript

- > TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.
- Created by Microsoft, very similar to c#
- First introduction to classes - JavaScript has no understanding of classes, even new classes are just syntax sugar
  - Adds option to private and public properties
- Generic types
  - Really powerful, but hard to understand
- Type checking when building your application

JS is slowly moving towards functional programming

- With a release of ES6 (ES2015) JS specification the language drastically shifted towards functional programming
- Introduction of lambda functions, many many new prototype functions
  - Prototypes (Array, Object, Map, Set, ...) received a huge upgrade when it comes to natively supported functions which were previously only possible thanks to certain libraries. (Map, Reduce, Entries, ...)
- There are whole frameworks based on FP and shifting away from OOP
- There are also many JS extensions which are based entirely upon FP
  - RxJS, CircleJS, LodashFP, Ramda
- Can fully replace Objects with closures and composition

Although there are some who don't like this that much



Improved Browser API

- Browsers (not IE) have transformed significantly in a past few years
- Most of the are actually following and keeping up with ECMA (Javascript) specifications
- Many JS utility libraries are becoming obsolete because browsers support is getting better and better
- Browsers support I18N (Internationalization and localization) natively
- And many more
  - Browser plugins
  - Security
  - Performance
  - Debugging tools

https://www.ecma-international.org/

Build tools - module based codebase

- Script tags are the history, now you need only one and the rest will be loaded on demand
- Move from CommonJS to MJS
  - You can now actually reference a different module from JS file in a browser?!. (import/export)
- We no longer need global variables everywhere
- JS is sadly becoming compiled language (Thanks IE)
- Thanks to modules, code splitting is becoming really simple and easy
  - Scripts are no longer loaded all at once
  - They can be loaded on demand
  - Huge performance and also UX improvements

rollup.js

webpack

PARCEL
Blazing fast, zero configuration web application bundler

Gulp

Red Hat

Dependency headache

- Thanks to the explosion of JS in recent years, your bundles will have thousands of external dependencies
- Libraries depend on libraries which depend on libraries of libraries
- After installing React, ReactDOM and Patternfly you will have over 100 external dependencies
- At larger scale, any breaking change to a single dependency may cause the whole project collapse and requires extreme caution
- Any upgrade must be considered and planned and will likely require tons of refactoring
- **YOU DON’T NEED LIBRARY FOR EVERYTHING**
- Some libraries has become so popular that everybody uses them and you can end up with multiple copies of the same code but in different versions

NPM vs Yarn

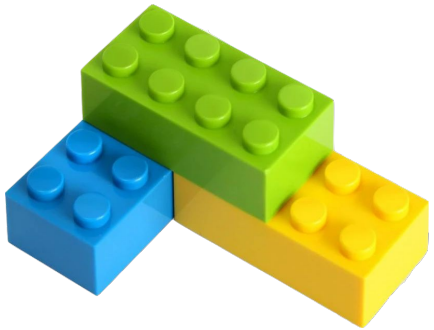
NPM

- Created by Google
- Included by default with node
- Finally supports monorepo (from version 7)
- Hosts over 1.3 mil of packages
- Option to audit your packages

Yarn

- Created by Facebook
- First introduction of lock file
- First introduction of monorepo
- Mirror of all npm packages
- Yarn 2.0 adds plg'n'play - option to minimize download of packages on install

Web UI key principles (React based)



1. Component-Based Approach

- UI is divided into reusable pieces of code called components (like a LEGO)
- Components can have a hierarchy, allowing the creation of complex user interfaces from simple parts.

2. JSX (JavaScript XML)

- Syntax similar to HTML, used in React to describe what the UI should look like.
- During compilation, JSX is transformed into JavaScript function calls.

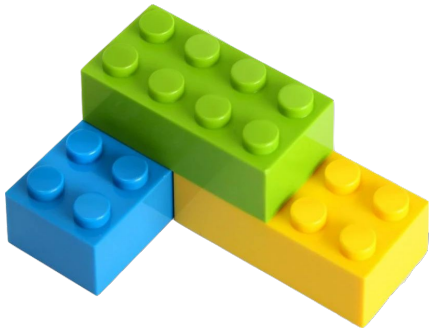
MyComponent.jsx U ×

packages > module > src > components > MyComponent.jsx > ...

```
1   import React from 'react';
2
3   const MyComponent = (props) => {
4     const text = props.text ?? 'My First Component';
5     return (
6       <h1>{text}</h1> // JSX
7     );
8   }
9
10  export default MyComponent;
```


Red Hat

Web UI key principles (React based)



3. One-Way Data Binding

- Data can be passed to components via read-only props.
- Components can have their own internal state, which can change.
- When the state changes, the component re-renders to reflect it.

4. Virtual DOM

- It is a lightweight representation of the actual DOM.
- Minimizes the number of updates made to the actual DOM by batching them together. It only updates parts of the DOM that have changed, rather than re-rendering the entire page.

5. Declarative Approach

- Instead of telling the system step-by-step how to change the UI, React describes what the final state should look like.

6. State Management

- In smaller applications, state management can be handled directly within components, while in larger applications, libraries like Redux or Context API are often used to help manage global state.

What is React? Definition

Declarative, component based JavaScript library for building UI

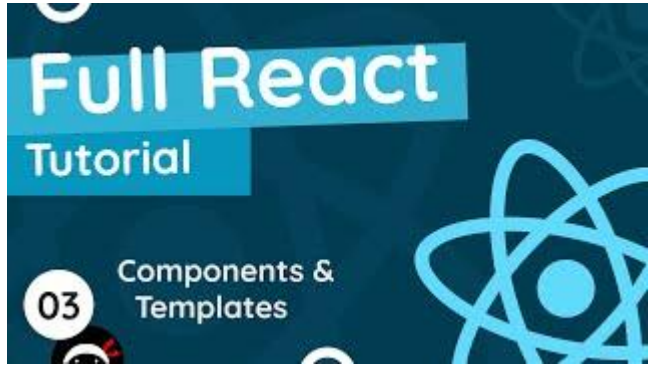
- Components are (sometimes) stand alone units with well defined API.
- You don't need to know how the component works under the hood to be able to use it.
- React is build on rather simple principles which allow developers build complex solutions to satisfy their customers needs.

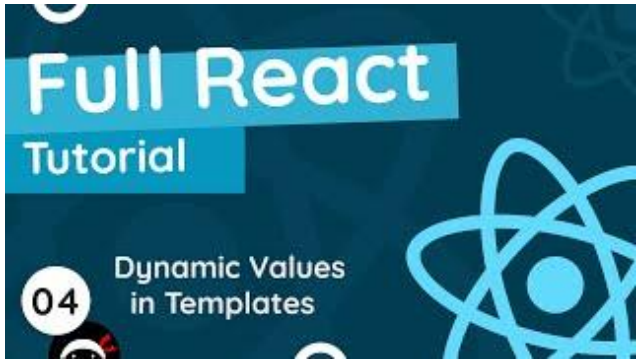
React IS NOT:

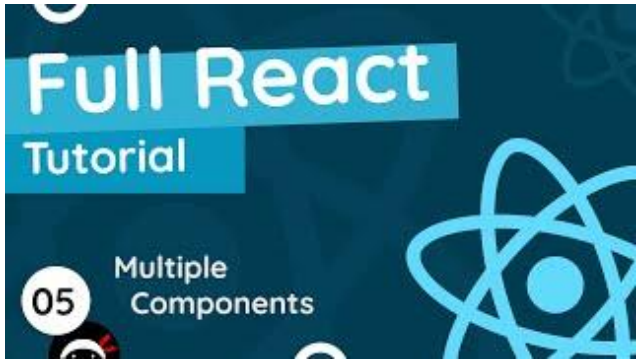
- state management library
- data fetching library
- data visualization library
- framework

https://reactjs.org/

React tutorials







https://www.youtube.com/playlist?list=PL4cUxeGkcC9gZD-Tvwfod2gaISzfRiP9d

Component and props

- **Component** is a stand alone UI piece with well defined API
- API can provide a set of props (you can send multiple props to single component)

- **Props** is an object
- Props are given to component from its parent
- Prop can be required or optional
- Key is a prop name, and value is its value
- Prop is a variable which is consumed and used by the component
- Component will most likely throw an error if its missing a required props
- Prop requirement and props can be defined via propTypes
- PropTypes is a static attribute on a component type
- Each prop type definition is a function that checks the prop content

Props manipulating

- There are two types of props
  - **Values** - passing data down
  - **Callbacks** - user (or async) interactions
- Use spread to control multiple props
- Combine them together
- Spread them to pass them down

```
const InputComponent = ({ onChange, ...props }) => {
 return (
   <input
    { ...props }
     onChange={(event) =>
onChange(event.target.value)}
   />
 )
}
```

Component props examples

```
const TitleComponent = props => {
 const {text, size} = props;  // ES6 destructuring
 return (
   <h1 style={{ fontSize: size }}>{text}</h1>
 )
};

const Parent = () => {
 return (
   <TitleComponent text="Hello" size={48} />
 )
}
```

Component prop types examples

```
import PropTypes from 'prop-types';
const TitleComponent = props => {
 const {text, size} = props;
 return (
   <h1 style={{ fontSize: size }}>{text}</h1>
 )
};

TitleComponent.propTypes = { // prop types static attribute definition
 text: PropTypes.string.isRequired, // will throw an error if the prop missing
 size: PropTypes.number // will be set to 24 if the prop is missing
};

TitleComponent.defaultProps = { // default values for the props
 size: 24
};
```

Class vs function component - Function

- Components are state and props in a nutshell
- React is moving towards function component
  - Hooks to control lifecycle
  - Hooks to connect to redux and router
  - Hooks to control everything
- Prefer functions for calculation to be moved out of function component as they are re-initialized on every render

```
const InputComponent = () => {
 return (
   <input />
 )
}
```

Class vs function component - Class

- As some hooks are still missing so classes will still be around, but rare
- Binding of this in classes is necessary
- To ease binding @babel/plugin-proposal-class-properties should be used
- Do not call setState from render method as you end up with infinite loop

```
class InputComponent extends Component {
 render() {
  return (
   <input />
  )
 }
}
```

Component and state

- Component can have its internal state
  - **It should not hold application state (data)**
    - You have state management libraries for that
  - State should only hold component specific data

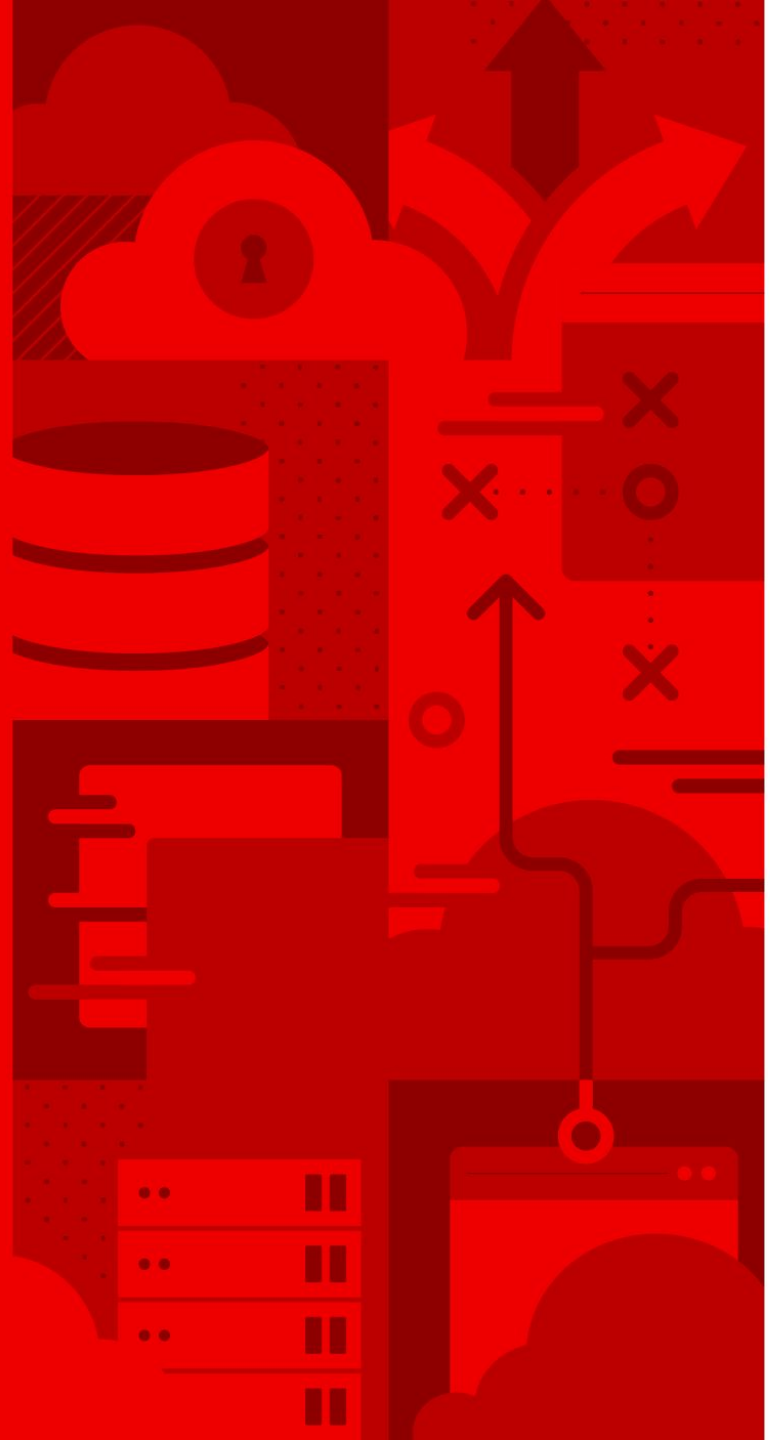
    What is the value of input? Is the dropdown expanded?

    ~~What are the attributes of value logged in user?~~

- State is directly accessible only within the component
- If you want to expose it to children, it must be sent to them as a prop

Component state example

```
const InputComponent = () => {
 const [value, setValue] = useState(''); // storing the value here
 const handleInputChange = (({ target: { value } }) => setValue(value));
 return (
   <input
       name="controlled-input"
       value={value}
       placeholder="No value"
       onChange={handleInputChange}
    />
 )
};
```



Project start!

Red Hat

Clone the repository!

- https://gitlab.fi.muni.cz/qhala/dashboard-PV278

Install, start and build

- Run `npm install`
- Run `npm start`
- Optionally run `npm run build`
- Optionally run `npm run deploy`

APIs

- Price per country
- Sunshine
- Air
- Temperture

Homework 2

- Deploy application - https://gitlab.fi.muni.cz/qhala/dashboard-PV278
- Record a short video and submit it to the IS homework vault
- **Deadline 3.11.2024**