React crash course

Or, why do we need yet another framework for frontend development?

Oslo, 10.04.2019
Kodekata med PenDevCrew - S2, rom 4060

Overview

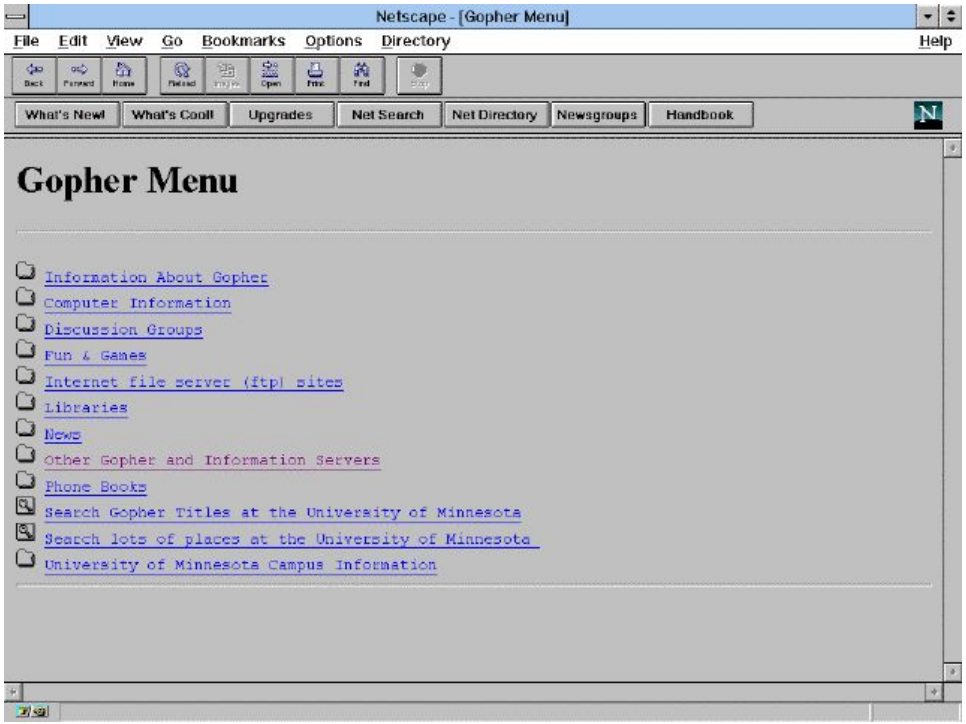
- Web throughout the decades
  - From the 80s until today
- Single Page Applications (SPA)
  - Javascript frameworks for SPAs
- React
  - What/Why/How to React?
  - Demo first hands
  - React advanced: inheritance/composition, components, lifecycles, Redux
  - React testing
- Sample Apps
  - ToDo app with / without Redux
  - ToDo app with Hooks/Context API

Internet in the 80s

- **Mail** - exchange messages
- **FTP** - exchange files
- **Gopher** - get documents

Gopher features:

- Basic folders
- Basic links
- Static document render



Internet in the 90s

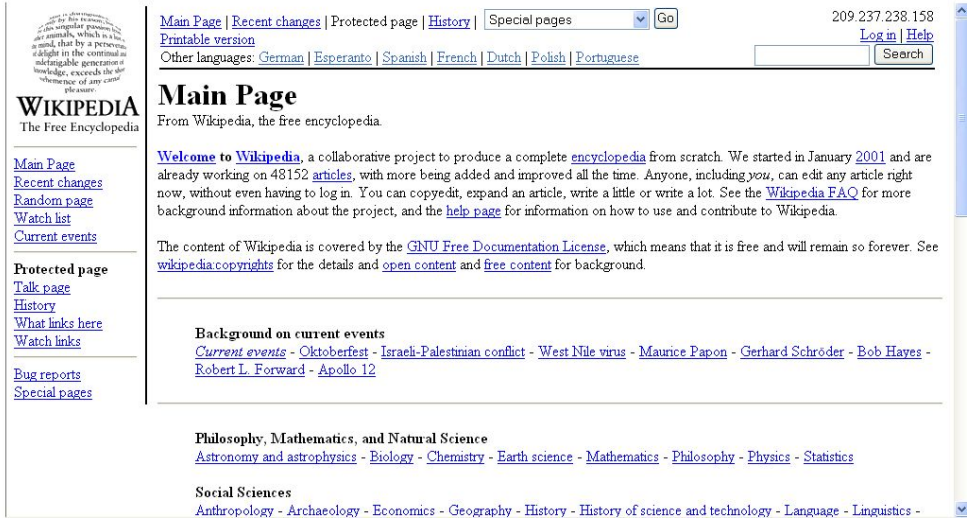
- **HTTP / www**
- **Web servers** → Dynamic pages
  - Perl CGIs, PHP
  - DBs
- **JavaScript**
  - Now web pages can also be dynamic in the browser

Still, too static and confusing…

- Link click → Page load
- Menus, headers, all in tables
- Browser wars (Netscape/IE)

Internet in the 2000s

- **Web 2.0**
  - user adds content
- More **apps**, less pages
- **Ajax** calls
  - do HTTP calls “in background”
  - don’t need to reload pages
  - → REST APIs
- **Javascript** improvements
  - more useful (JQuery, …)
  - takes over HTML rendering from server
- Better **standards**
  - Firefox, Safari, Opera, Chrome



Internet in the 2010s

- Memes

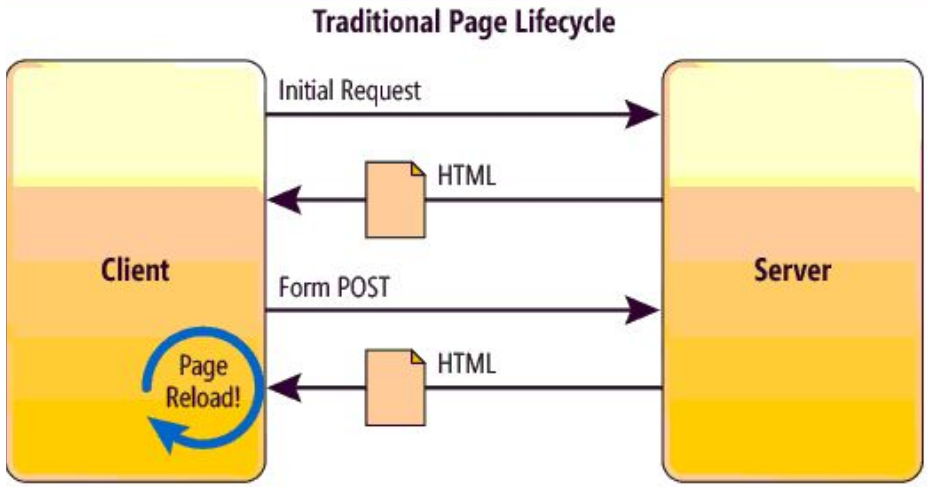
Also:

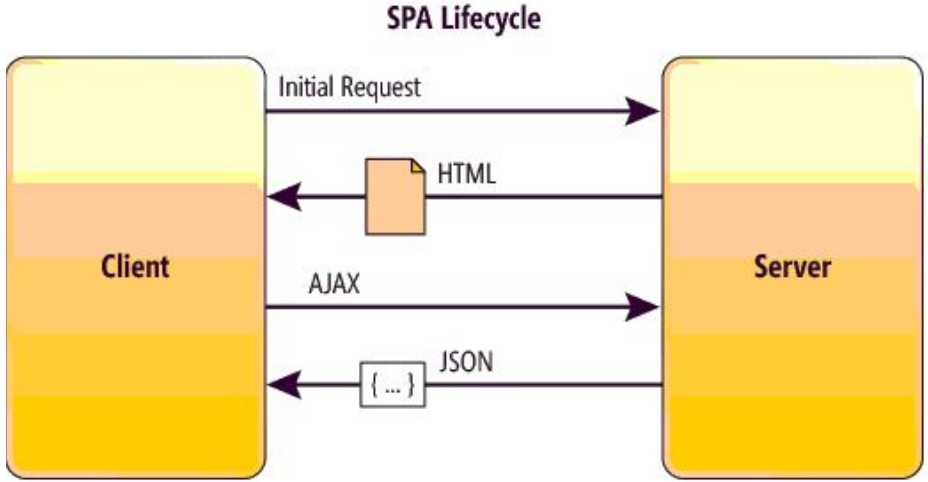
- HTML 5
- Mobile / Responsive
- Better browsers
  - Local/Session Storage
  - Better JavaScript engines
- Full-stack Javascript
  - NodeJS
  - NPM
  - Javascript frameworks

Frontend challenges today

- Think more as **web applications**, less as web sites
- Load all scripts **once**, exchange only data **afterwards**
- UI **challenges**:
  - Work on every computer, mobile phone, tablet, TV, car, fridge,
  - Accessible to and for everyone
  - Intuitive, simple, no learning curve

- **Single Page Applications (SPA)**
  - DOM changes, no HTML page loads
  - Fetch data from REST APIs
  - Use local / session storage in browser


Traditional Page Lifecycle
Initial Request
HTML
Client
Server
Form POST
Page Reload!
HTML


SPA Lifecycle
Initial Request
HTML
Client
Server
AJAX
JSON
{ ... }

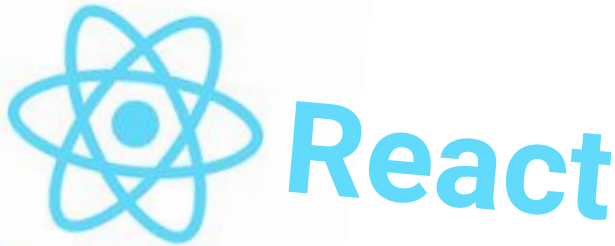
Javascript frameworks for SPAs

2 big ones:

- **Angular JS** (angularjs.org - Google)
- **React JS** (reactjs.org - Facebook)

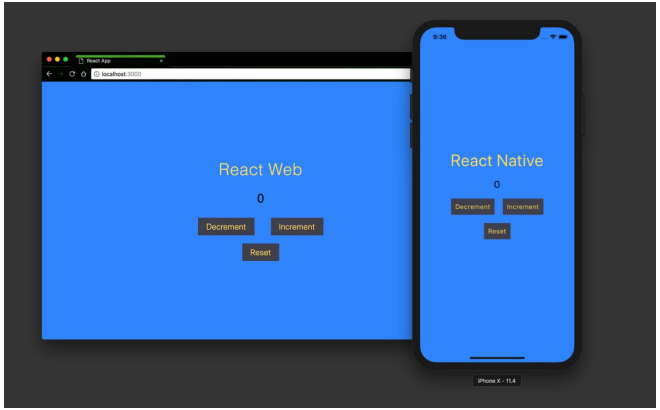
Others:

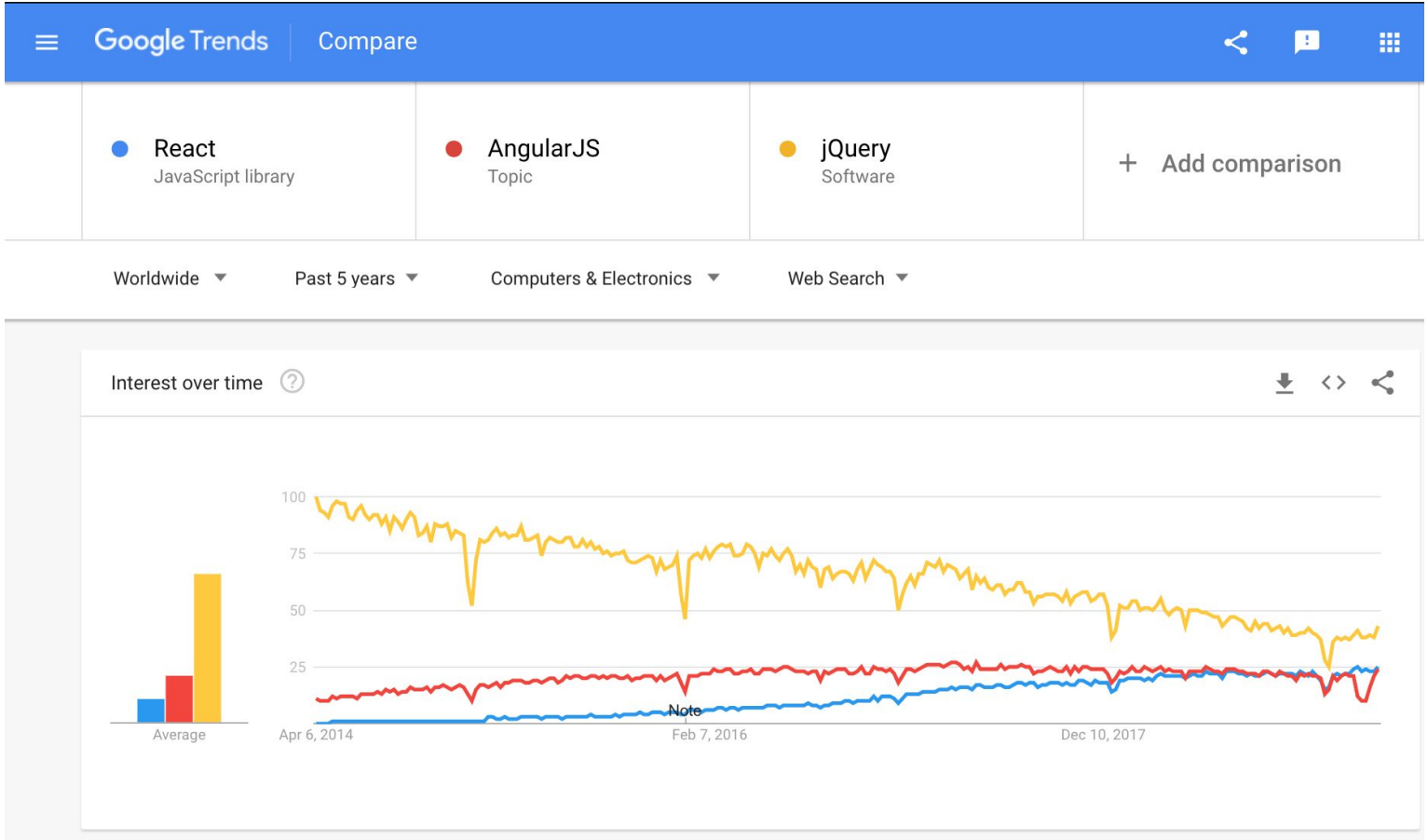
- Ember (emberjs.com)
- Backbone (backbonejs.org)
- ExtJS (sencha.com/products/extjs/)



What is React?

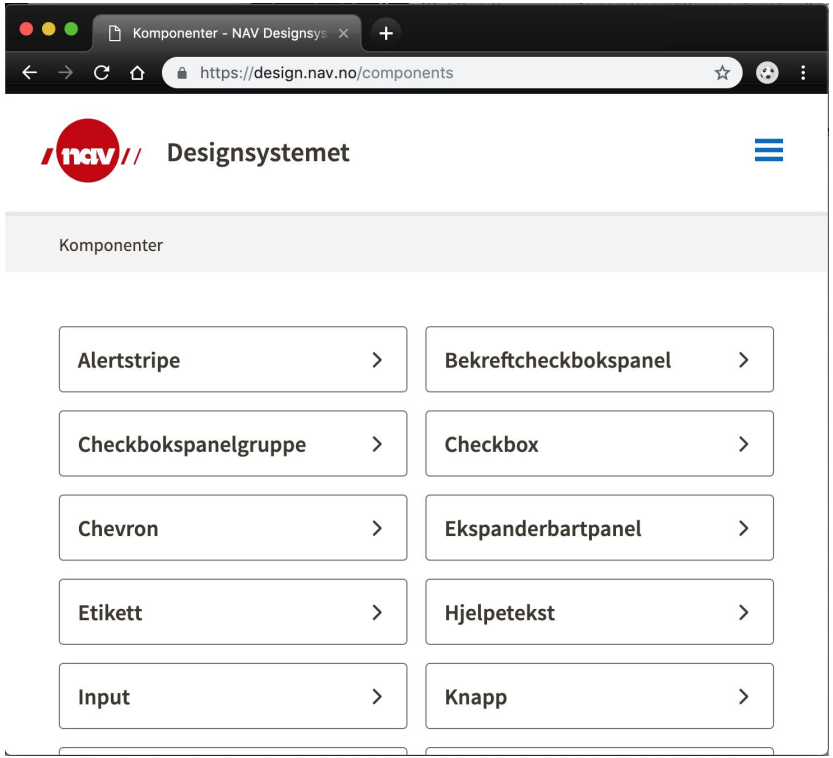
- JavaScript library for Single Page Application UIs
- Made by Facebook
- https://github.com/facebook/react/
- V1.0 released April 2017
- Current v16.8
- 2 “flavors”:
  - **ReactJS** for web apps
  - **ReactNative** for mobile native apps




Google Trends
Compare
React
JavaScript library
AngularJS
Topic
jQuery
Software
Add comparison
Worldwide
Past 5 years
Computers & Electronics
Web Search
Interest over time
100
75
50
25
Note
Average
Apr 6, 2014
Feb 7, 2016
Dec 10, 2017

Why React?

- Create/reuse React **components**
  - Less reinventing, more reusing
- Great for **design systems**
  - Consistent UI through all webapps
- Same components for **web/mobile**
  - “Learn once, code anywhere”
- Proper project **structure**
  - Classes, components, utils, actions, etc
- Single data flow, **reactive** updates
  - You worry about data flows, React worries about update/render



How to React?

- First, **NodeJS** (nodejs.org)

  Javascript runtime, running on desktop

- Second, **NPM** (mpnjs.org)

  Node Package Manager
  Think of RPM/DEB, but for Javascript

- Third, **Create react app** - facebook.github.io/create-react-app/


STOP
DEMO TIME
memegenerator.net

Demo: npm useful commands

- `npm install -g create-react-app`
- `create-react-app <your-app-name>`

`package.json` - Javascript's `pom.xml` or `build.gradle` file
`node_modules` - All your webapp dependencies go here

- `npm start` - Development server at `localhost:3000` with hot reload
- `npm test` - Run tests
- `npm run build` - Uglify into JS / CSS chunks ready for production
- `serve -s build` - See your production build at `localhost:5000`

Demo: first hands down

Tasks:

- Change text in App.js
- Create a new Component, MyParagraph.js
- Move content into <MyParagraph> component
- Export default class MyParagraph
- Import MyParagraph.js in App.js

- Add custom text as props to the MyParagraph component
- Import/reuse external package, such as NAV AlertStripe (see design.nav.no)
  - NOTE: You have to process index.less into index.css, for proper rendering

Demo: handling events

- **Input**
  - import { Input } from 'nav-frontend-skjema'
  - Add to App.js, add onChange function
  - Add state, call setState() with new values

- **Button**
  - Import KnappBase from 'nav-frontend-knapper'
  - Add to App.js
  - <KnappBase>Normal</KnappBase>
  - Add onClick function

Now, for more advanced stuff

No inheritance in React

```
class Animal extends React.Component {
  render () {
    return <div>I am a {this.props.type || 'animal'} </div>
  }
}
```

How to make a `class Dog`?

`class Dog extends Animal {...}` makes sense in OOP… but not in React

React favors **composition** instead of **inheritance**

Composition in React

```
class Animal extends React.Component {
  render () {
    return <div>I am a {this.props.type || 'animal'} </div>
  }
}
```

```
class Dog extends React.Component {
  render () {
    return <Animal type='dog'>
  }
}
```

React.Component functions and lifecycle

https://reactjs.org/docs/react-component.html

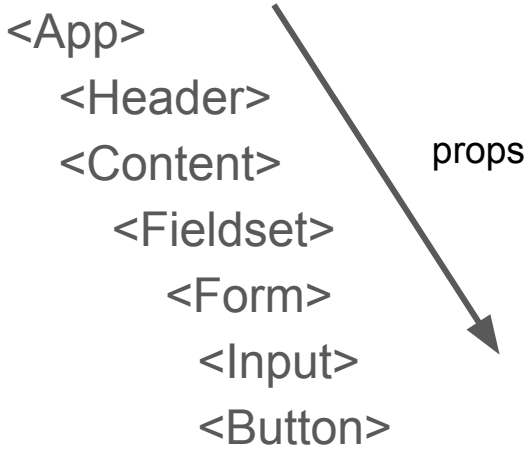
- **componentDidMount** ← Called once, useful for fetch initial data
- **componentDidUpdate** ← Called everytime props change, useful for reactive logic
- componentWillUnmount
- shouldComponentUpdate
- getSnapshotBeforeUpdate
- **static getDerivedStateFromProps** ← Useful to sync component state from existing/new props
- static getDerivedStateFromError
- componentDidCatch
- **render** ← That is your view

React.Component lifecycle example

```
class Example extends React.Component {
  state = {
   posts: []
  }
  componentDidMount() {
    fetchPosts().then(response => {
      this.setState({
        posts: response.posts
      });
    });
  }
  render() {
    return <div>
      <ul>{this.state.posts.map(post => {
         return <li>{post}</li>
      })}</ul>
    </div>
  }
}
```

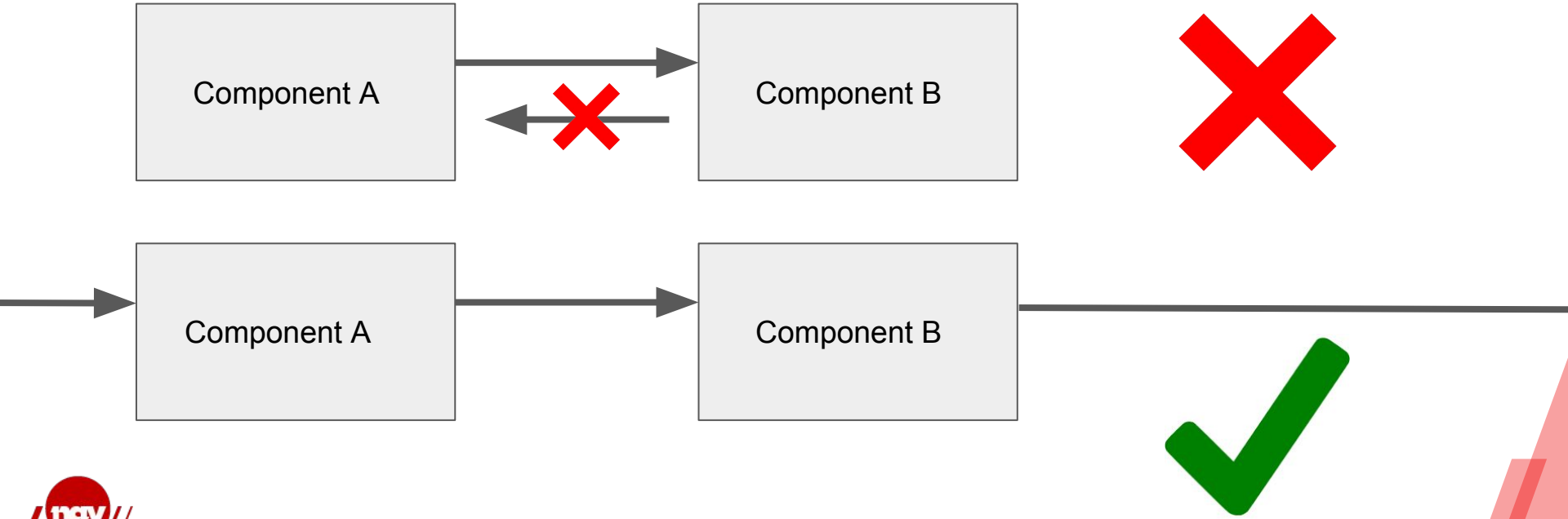
Data flow in React

Props flow from top components to children components



But how to update parent props from children components?

Answer: you go Magellan style



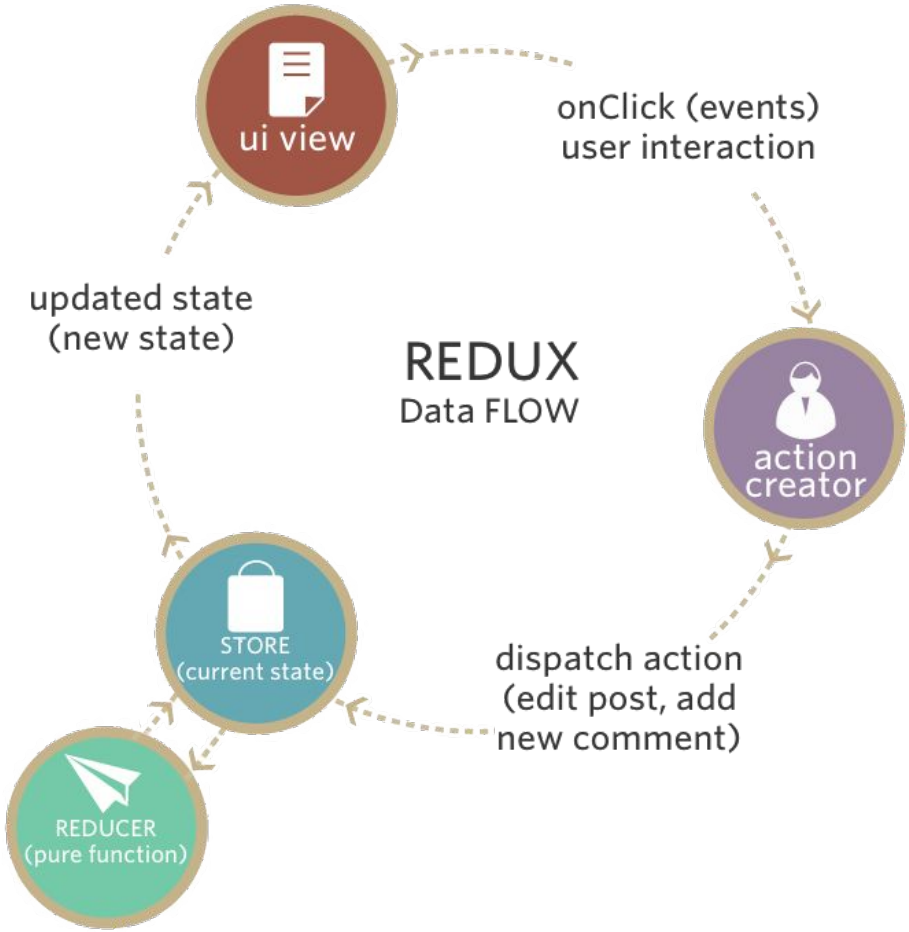
Redux

Redux (redux.js.org) - A predictable **state container** for JavaScript apps

React bindings for Redux: react-redux.js.org
(`npm install --save redux react-redux`)

Think of Redux as a app-wide **global state** that all React components can see, read and change. By doing that, components who subscribe to part of that state will update accordingly.

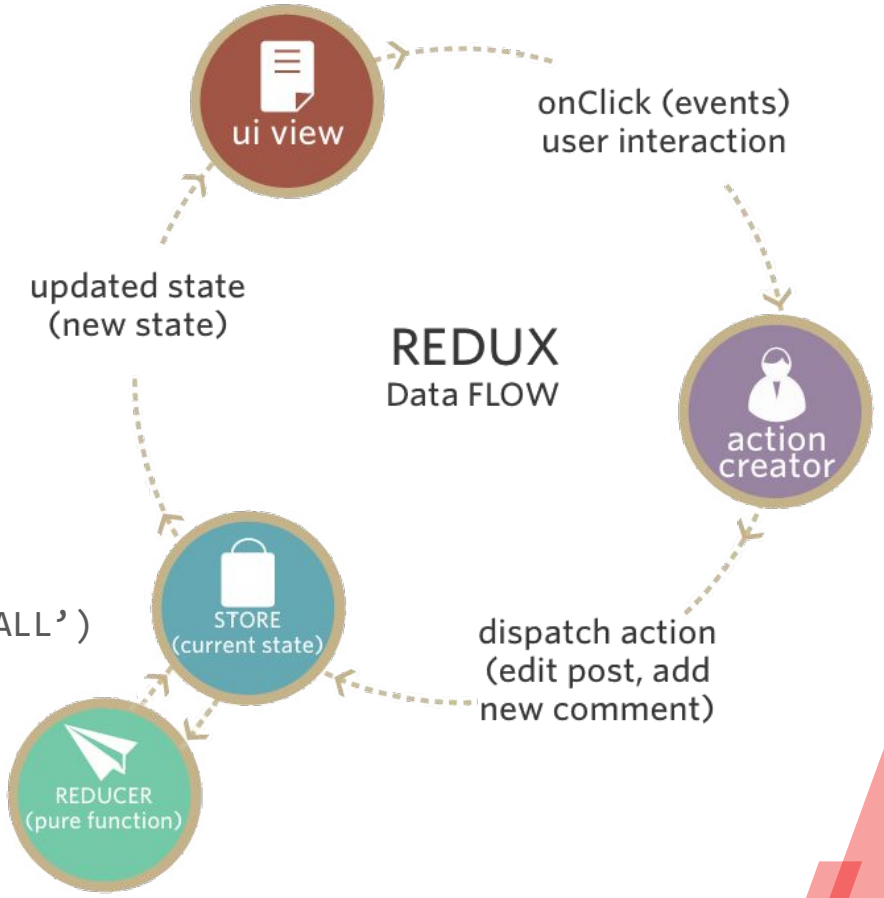
Redux


ui view
onClick (events)
user interaction
action creator
dispatch action
(edit post, add
new comment)
STORE
(current state)
REDUCER
(pure function)
updated state
(new state)
REDUX
Data FLOW

React + Redux data flow

- **View:** `mapStatetoProps = (state) => {`
  `return { items: state.items } }`
- **View:** button “Remove all items” clicked
  `this.props.actions.removeAllItems()`
- **Action creator:** `function removeAllItems() {`
  `return { type: ‘ITEMS/REMOVE_ALL’} }`
- **Reducer:** `if (action.type === ‘ITEMS/REMOVE_ALL’)`
  `return {items: []}`
- **View** render():

```
this.props.items.map(item => {
   return <div>{item}</div>
})
```



Do you really need Redux? No.

Yet, to let your app scale and grow graciously, you want to delegate responsibilities, separate concerns, namespace different pages/tools

- **Components** should worry about subscribing to store's state, map it to props, render, and dispatch actions

- **Action creators** should worry about describing what needs to change (fetch more data, dispatch one or more actions, ...)

- **Reducers** should worry about describing what/how part of the store will be affected with each dispatched action

How to actions / reducers

The basics:

1. Combine all **reducers**
2. Create a **store** with reducers
3. Define a `<Provider>` that wraps the whole app with the **store**

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './App'

import { createStore, combineReducers } from 'redux'
import { Provider } from 'react-redux'

import * as reducers from './reducers'

const initialState = {}

const reducer = combineReducers({
  ...reducers
})

const store = createStore(reducer, initialState)

ReactDOM.render(<Provider store={store}>
                  <App />
                </Provider>,
document.getElementById('root'));
```

How to actions / reducers (cont)

```
1  import React, { Component } from 'react';
2  import { connect } from 'react-redux'
3  import { bindActionCreators } from 'redux'
```

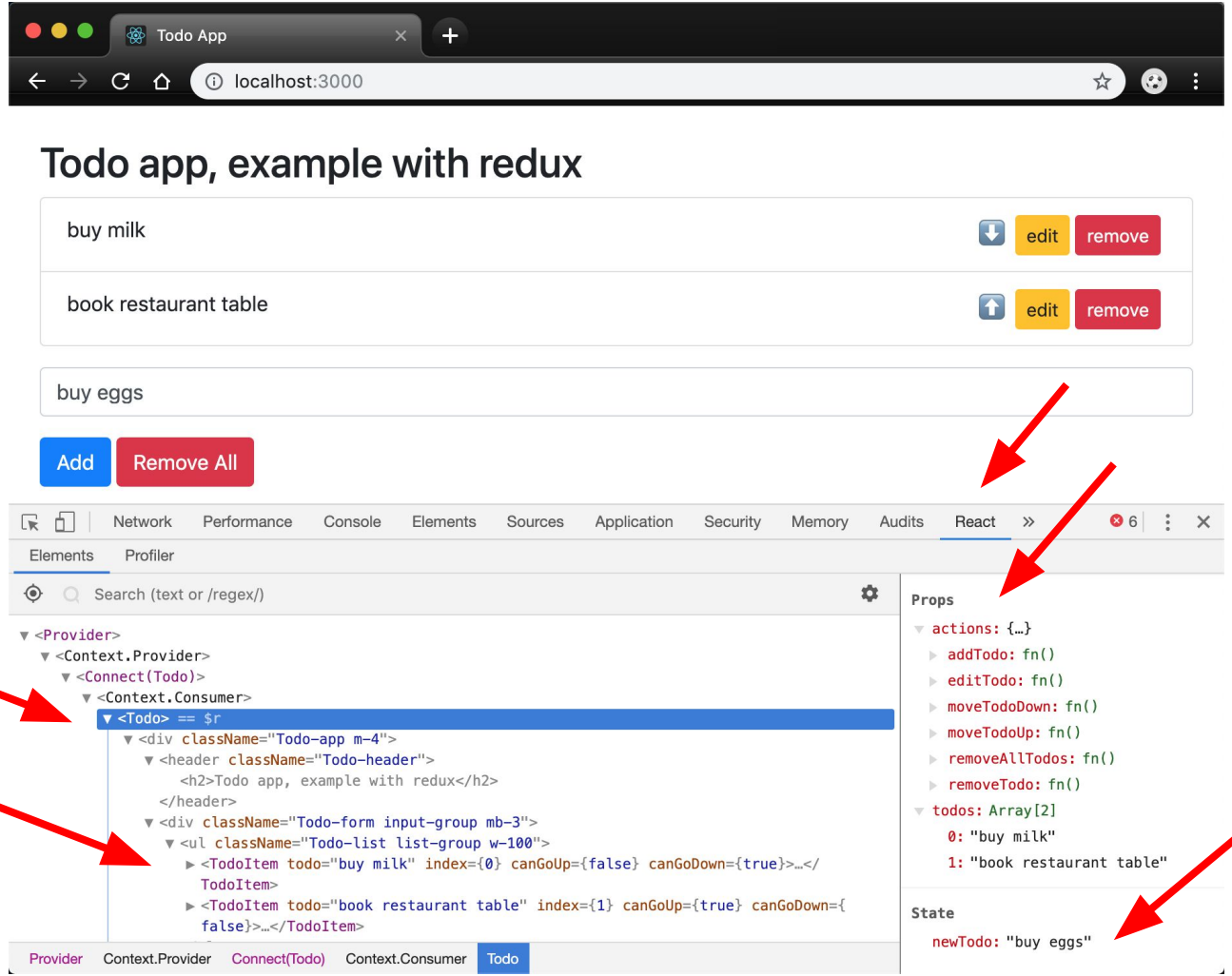
```
6  import * as todoActions from './actions/todo'
7
8  const mapStateToProps = (state) => {
9    return {
10     todos: state.todo.todos
11   }
12 }
13
14 const mapDispatchToProps = (dispatch) => {
15   return { actions: bindActionCreators(Object.assign({}, todoActions), dispatch) }
16 }
17
18 class Todo extends Component {
```

```
66 }
67
68 export default connect(
69   mapStateToProps,
70   mapDispatchToProps
71 )(Todo)
72
```


STOP
DEMO TIME
memegenerator.net

React testing

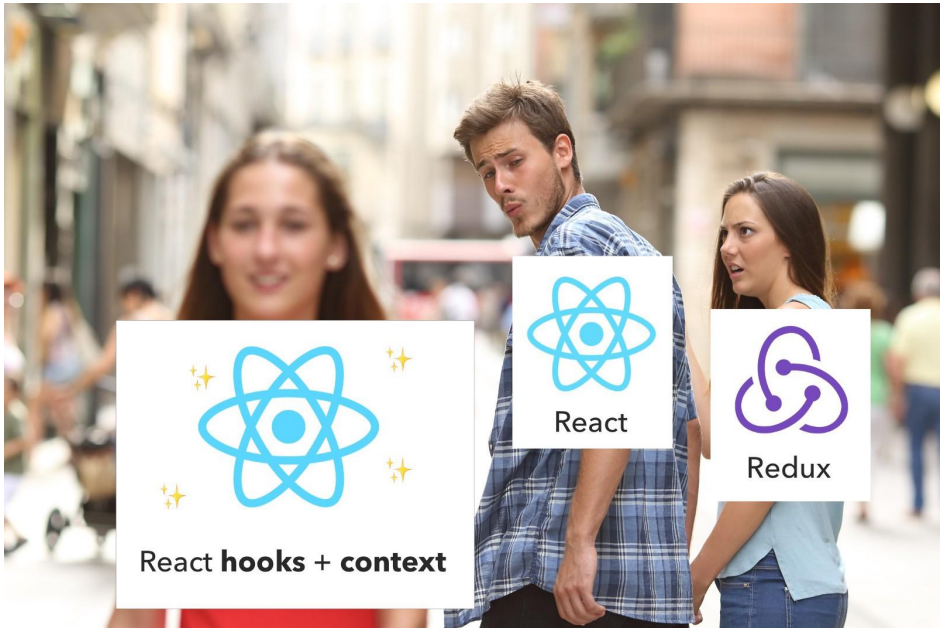
- Jest (jestjs.io) - JavaScript Testing Framework
  - Run tests, prints report / coverage, mocking
- Mocha (mochajs.org) - JavaScript test framework
  - beforeAll / beforeEach / afterAll / afterEach / describe / it
- Chai (chaijs.com) - Expressive assertion library - expect, should
  - `expect([1, 2]).to.be.an('array').that.does.not.include(3);`
  - `let beverages = { tea: [ 'chai', 'matcha', 'oolong' ] };`
  - `beverages.should.have.property('tea').with.lengthOf(3);`
- Enzyme (airbnb.io/enzyme/) - JavaScript Testing utility for React
  - shallow/render/mount React components for individual testing
- Mock: redux-mock-store (store mock), nock (fetch mock), etc


Todo App
localhost:3000
Todo app, example with redux
buy milk
edit
remove
book restaurant table
edit
remove
buy eggs
Add
Remove All
Network
Performance
Console
Elements
Sources
Application
Security
Memory
Audits
React
Elements
Profiler
Search (text or /regex/)
<Provider>
<Context.Provider>
<Connect(Todo)>
<Context.Consumer>
<Todo> == $r

<header className="Todo-header">
<h2>Todo app, example with redux</h2>
</header>

<ul className="Todo-list list-group w-100">
<TodoItem todo="buy milk" index={0} canGoUp={false} canGoDown={true}>…</TodoItem>
<TodoItem todo="book restaurant table" index={1} canGoUp={true} canGoDown={false}>…</TodoItem>
Provider
Context.Provider
Connect(Todo)
Context.Consumer
Todo
Props
actions: {…}
addTodo: fn()
editTodo: fn()
moveTodoDown: fn()
moveTodoUp: fn()
removeAllTodos: fn()
removeTodo: fn()
todos: Array[2]
0: "buy milk"
1: "book restaurant table"
State
newTodo: "buy eggs"

React Hooks API / Context API



Started in React 16.8.

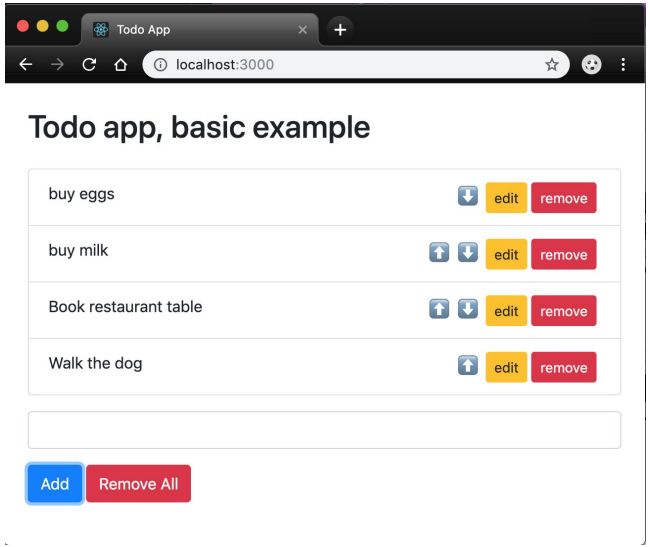
Basic Hooks:

- **useState** - like this.setState(), but for individual variables
- **useEffect** - as componentDidMount() + componentDidUpdate()
- **useContext** - can be used as store's state
- **useReducer**

See more in https://reactjs.org/docs/hooks-reference.html

Tips

- Install bootstrap so you don't have to do CSS (`npm install --save bootstrap`) (getbootstrap.com/)
- Install Chrome React Dev Tools (see Chrome web store)
- Check ToDo app in github.com/navikt/react-introduction-tutorials
  - Unit tests example
  - Redux example
  - Hooks API/Context API example
- Also, check thunk middleware (daveceddia.com/what-is-a-thunk/)



React crash course

Or, why do we need yet another framework for frontend development?

Oslo, 10.04.2019
Kodekata med PenDevCrew - S2, rom 9122