React Crash Course

Introduction to React

Karol Kowalczuk, Łukasz Wiśniewski

October 2019

Agenda

1. JavaScript basics
2. JavaScript ecosystem tools
3. First steps in React
4. Virtual DOM & component lifecycle
5. Untypical DOM manipulation

JavaScript basics 1

JavaScript is a single-threaded scripting programming language.

OOP? Functional programming?

Why not both?!

Execution Environments

Web

Both client- (~~interpreted~~ compiled by browsers to IL nowadays) and server-side (executed by node.js environment)

Non-web

* Embedded scripting language (e.g. browsers extensions)
* General purpose scripting language
* Used as an application platform

JavaScript basics 3

OOP

- Creating objects directly from other objects
- Behaviour delegation
- Composability
- Extending objects dynamically

OLOO* („CAN DO”)

* OBJECTS-LINKED-TO-OTHER-OBJECTS

- Tightly couples base class with inherited ones
- Must know the desired properties of inherited object before instantiation

INHERITANCE („IS A”)

JavaScript basics 4

Behaviour delegation

Include a utility method if needed

```
var Task = {
  setID: function (ID) { this.id = ID },
  outputID: function () { console.log(this.id) },
}

// make 'XYZ' delegate to 'Task'
var XYZ = Object.create(Task)

XYZ.prepareTask = function (ID, label) {
  this.setID(ID)
  this.label = label
}

XYZ.outputDetails = function() {
  this.outputID()
  console.log(this.label)
}
```

Credits to: Kyle Simpson

Functional programming

Pure functions

Given the same input always gives the same output; makes composition easy as pie

No shared state

Do not pass the properties of the object between scopes

No state mutations

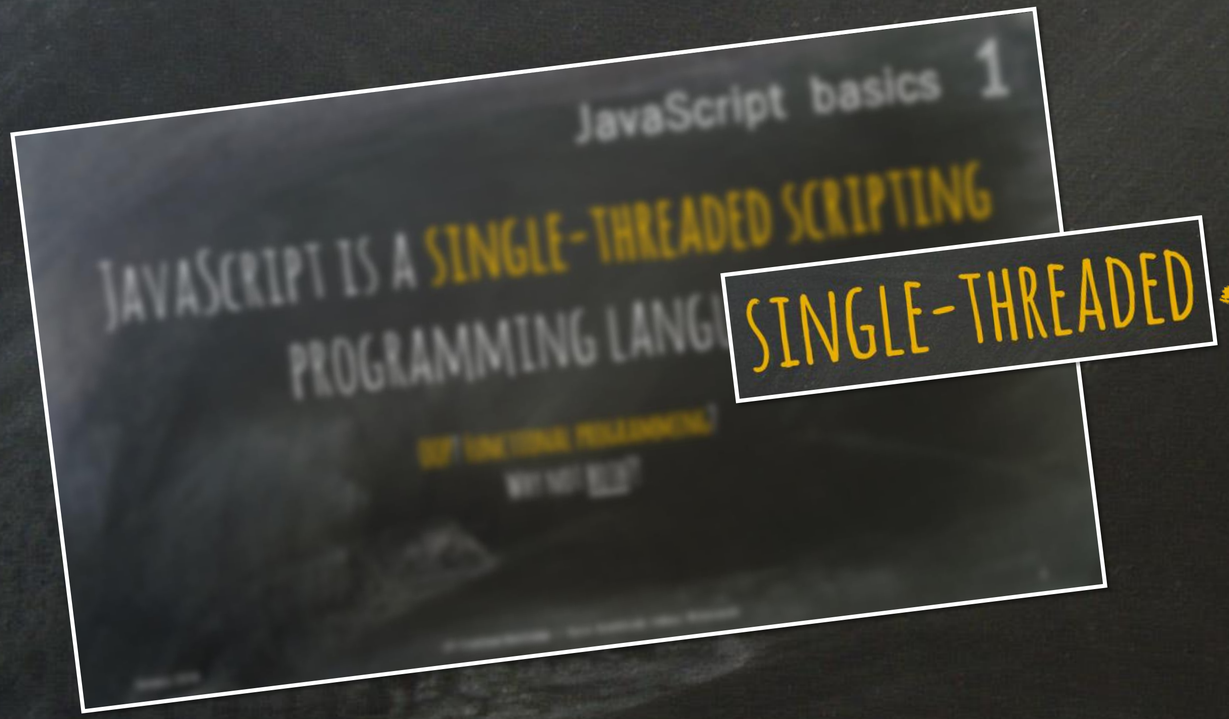
Allows to track the transfomations and do *time-traveling*

No side effects

Just be deterministic

JavaScript basics 6

Asynchronous calls



WAIT, WHAT?

JavaScript basics 7

ONE THREAD == ONE STACK == ONE THING AT A TIME

EXCEPT NOT REALLY.

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

stack

threads (WebAPIs)

console

event loop 

tasks queue

JavaScript basics 8

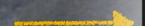
```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

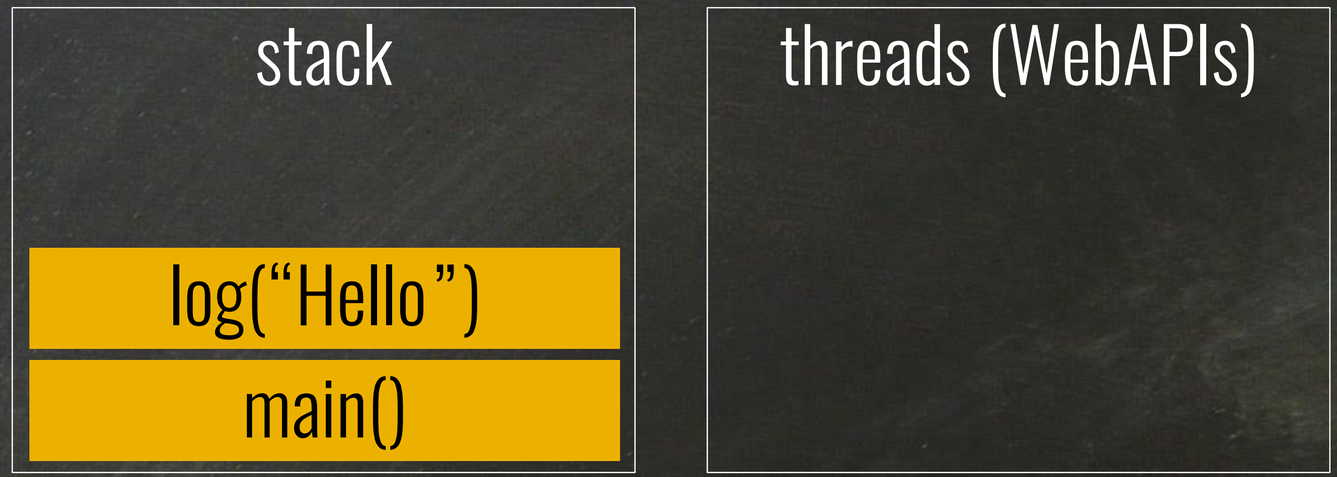
event loop

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```





event loop 

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

event loop ⟳

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

stack

setTimeout(cb)

main()

threads (WebAPIs)

console

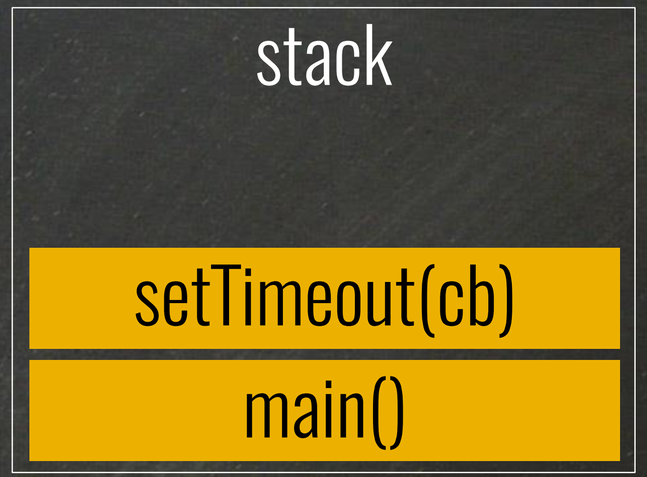
Hello

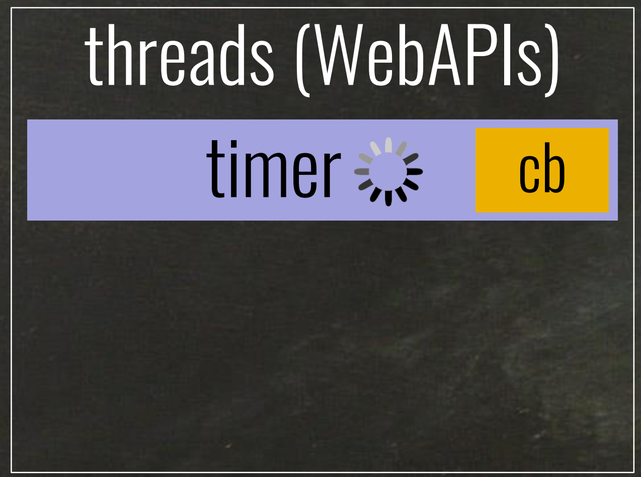
event loop 

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

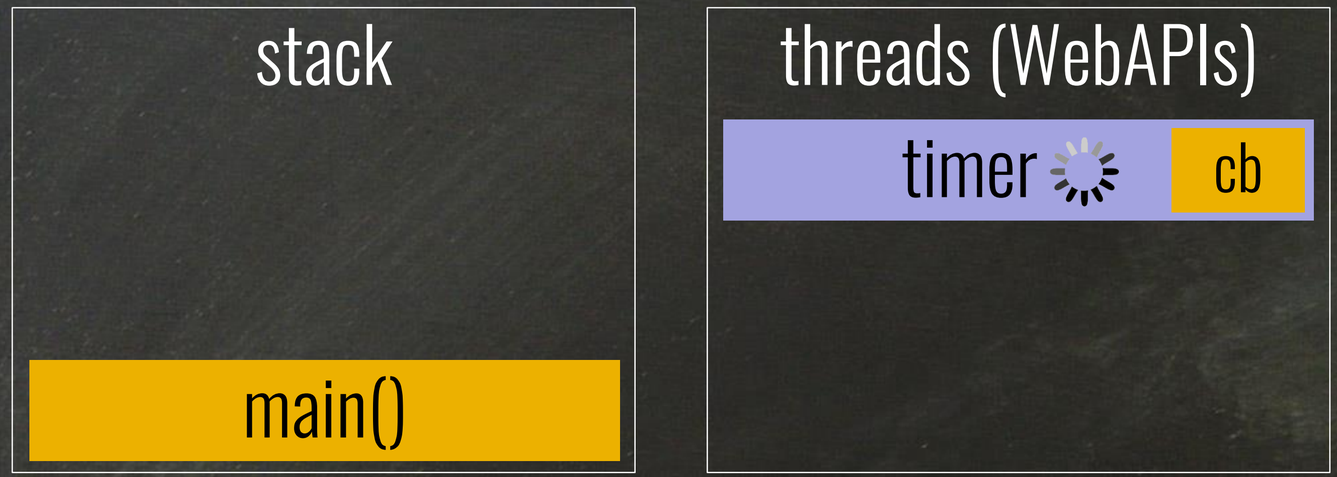




event loop 

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

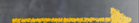


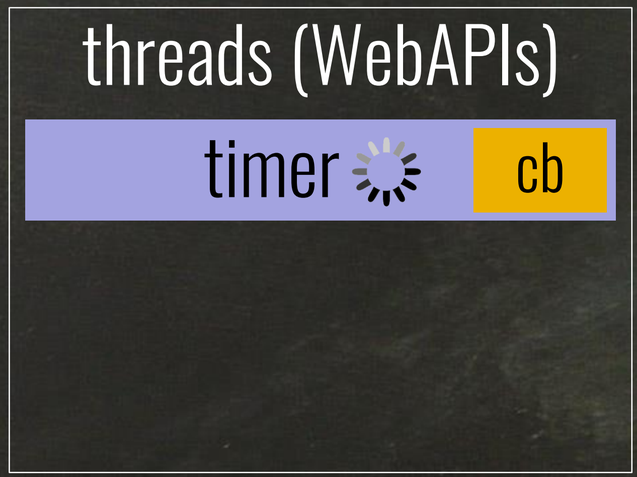
event loop

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

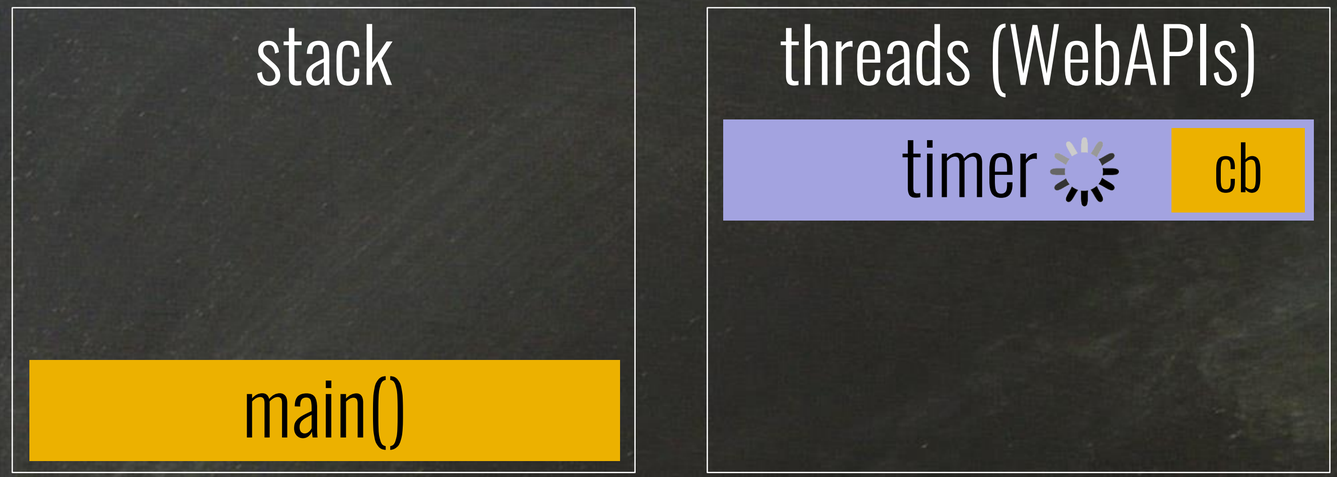




event loop 

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

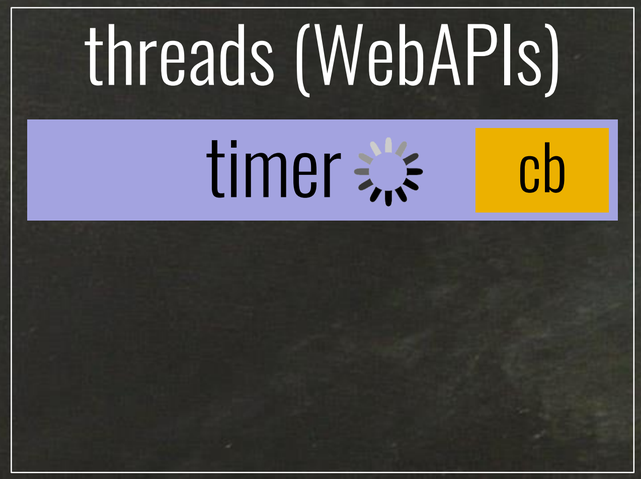


event loop

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```



event loop

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

stack

threads (WebAPIs)

console

Hello
World!

event loop 

tasks queue cb

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

stack

cb

threads (WebAPIs)

console

Hello
World!

event loop 

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

event loop 

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

stack

cb

threads (WebAPIs)

console

Hello
World!
:)

event loop 

tasks queue

JavaScript basics 8

```
console.log("Hello")

setTimeout(function cb() {
  console.log(":)")
}, 5000)

console.log("World")
```

stack

threads (WebAPIs)

console

Hello
World!
:)

event loop 

tasks queue

JavaScript basics 9

Promises

since 2012 (in ES since 2015)

Objects representing async actions result; there are three different states of Promises: fulfilled, rejected, pending

async/await

in ES since 2017

Syntactic sugar for promises

ECMAScript 2015

6th version of standardized scripting-language specification

Initially created to standardize JavaScript.

Arrow functions

No need to bind "this" anymore

```
// Previously
var _this = this
button.click(function (event) {
  _this.sendData()
})

// Now
button.click((event) => {
  this.sendData()
})
```

JavaScript basics 12

```
// Previously
var processData = function (length, width, color) {
  // Be careful with zeros!
  var length = length || 100
  var width = width || 20
  var color = color || "blue"

  // function body here
}

// Now
var processData = function (length = 100, width = 20, color = "blue") {
  // function body here
}
```

Default parameters

Get rid of “or” evaluations

String literals

Put variables in a string directly

```
// Previously
var greetings = "Hello" + firstName + " " + lastName
  + "\n" + "Your cart contains: " + itemsNo + " items."

// Now
var greetings = `Hello ${firstName} ${lastName}
  Your cart contains ${itemsNo} items.`
```

JavaScript basics 14

```
// Previously
var response = callApi(params)
var body = response.body,
  status = response.status

var listContent = [1, 5, 10, 12]
var first = listContent[0],
  third = listContent[2]

// Now
var response = callApi(params)
var { body, status } = response

var listContent = [1, 5, 10, 12]
var [ first, , third ] = listContent
```

Destructuring assignment

It's a kind of magic

Rest/Spread operator

Destructuring made even simpler

Support for objects added in ES9 (2018)

```
// Now (ES6)
var set = ["foo", 14, 1.2, "bar"]

var [ first, second, ...rest ] = set
console.log(first, second) // "foo" 14
console.log(rest) // [ 1.2, "bar" ]

// Now (ES9)
var details = {
  firstName: "John",
  lastName: "Doe",
  age: 39,
}

var { age, ...rest } = details
console.log(rest) // { firstName: "John", lastName: "Doe" }
var copiedDetails = {...details}
```

JavaScript basics 16

```
// Previous
var x = 1000
if (true) {
  var x = 200
  console.log(x) // 200
}
console.log(x) // 200

// Now
let x = 1000
if (true) {
  let x = 200
  console.log(x) // 200
}
console.log(x) // 1000
```

Block-scoped "let"...

Couples variables with the scope

JavaScript basics 17

...and "const"

Disallows reassignments

```
// Previous
var x = 1000
if (true) {
  var x = 200
  console.log(++x) // 201
}
console.log(x) // 201

// Now
const x = 1000
if (true) {
  const x = 200
  console.log(++x) // TypeError: Assignment to constant variable.
}
console.log(x)
```

JavaScript basics 18

```
// Previous
  // module.js
  var port = 3000

  // main.js
  var service = require("module.js")
  console.log(service.port)

// Now
  // module.js
  export var port = 3000

  // main.js
  import { port } from "module"
  console.log(port)
```

Modules

Import only what's really needed

For-of loops

Because every language has them :)

```
// Previous
var iterable = [1, 2, 3, 4, 5, 7]
for (var i = 0; i < iterable.length; ++i) {
  console.log(iterable[i])
}

// Now
var iterable = [1, 2, 3, 4, 5, 7]
for (var i of iterable) {
  console.log(i)
}
```

JavaScript basics 19

```
function* count() {
  yield 0
  yield "first"
  for (let i = 2; i <= 3; ++i) {
    yield i
  }
}

const generator = count()

console.log(generator.next()) // { value: 0, done: false }
console.log(generator.next().value) // "first"
console.log(generator.next()) // { value: 2, done: false }
console.log(generator.next().value) // 3
console.log(generator.next().value) // undefined
console.log(generator.next().done) // true
```

Generators

Return from the function and continue execution when needed

JavaScript ecosystem tools 1

Choice taken to the extreme

(chaos)

JavaScript ecosystem tools 2

Package managers:
NPM vs YARN

JavaScript ecosystem tools 3

NPM

* Package manager
* Public package repository
* Private packages
* self-hosting repository (npm Enterprise)

JavaScript ecosystem tools 4

Yarn

* "just" a Package manager
* Optimized for performance, package caching
* Lock file

JavaScript ecosystem tools 5

Both use

package.json

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "Description here...",
  "main": "index.js",
  "scripts": {
    "test": "echo 1"
  },
  "dependencies": {
    "jquery": "3.2.1"
  },
  "license": "ISC"
}
```

Task Runners:
Grunt, Gulp, DIY scripts

JavaScript ecosystem tools 7

Grunt

* Declarative configuration
* File-based processing
* Relies on plugins

JavaScript ecosystem tools 8

Gulp

* Configuration = Code
* In-memory, stream-based processing
* Still relies on plugins

DIY (NPM) SCRIPTS

* Just your code
* Or a single command line
* Executed by NPM

JavaScript ecosystem tools 10

Compilers/Transpilers:

Millions of them

(but mainly Babel)

JavaScript ecosystem tools 11

Babel

Cutting edge JS

\/

Ordinary ECMAscript

JavaScript ecosystem tools 12

TypeScript

Cutting edge JS (with types)

\/

Ordinary ECMAscript

JavaScript ecosystem tools 13

The rest

* ‘alternative’ JS syntax languages (ex. – CoffeeScript)
* Full blown runtime environments (Scala.js, Elm)

JavaScript ecosystem tools 14

Bundlers:

The 'pack it for the browser' gizmo

JavaScript ecosystem tools 15

Before

* <script>, <link>, <style> managed by hand
* Should A.js be before, or after B.js?
* Global variables

After (Webpack)

* Look Ma, a single JS file!
* And a cryptic Declarative + Code based configuration

JavaScript ecosystem tools 17

Hands-on

Basic webpack configuration

source available: A-00-base

JavaScript ecosystem tools 18

Tests:

need a guide to choose

a tool to **just run them**

JavaScript ecosystem tools 19

Testing

* test runners (mocha, jasmine, jest, karma)
* The 'actual' tests (mocha, jasmine, jest)
* assertions (chai, jasmine, jest)
* snapshot testing (jest, ava)

JavaScript ecosystem tools 20

Testing

* Mocks, spies (sinon, jasmine, jest)
* code coverage (istanbul)
* Browser automation & headless browser environments (PhantomJS, Protractor, testcafe)

Linting & formatting:

Optional, but handy

JavaScript ecosystem tools 22

Linting: ESLint

formatting: Prettier

First steps in React 1

React.js is **NOT** a framework. It is **ONLY** a view library.

First steps in React 2

* No controllers
* No digest loops
* No directives
* No view models

Just Components.

In react.js world every rendered block is a component.

First steps in React 3

Separation of ~~concerns~~ components

(only) if they are designed and built as self-contained entities.

* Composability and flexibility
* Reusability
* Maintainability
* Ability to be tested independently

First steps in React 4

JSX

XML-like syntax extension for ECMAScript

```
return (
  <div className="wrapper s12">
    <p>How the hell is it possible?!</p>
  </div>
)
```

HTML!?

First steps in React 5

Functional components

Define simple and stateless components.

NOT VALID!

```
const SimpleComponent = (props) => {
  return (
    <div className="wrapper">
      <p>Simple, huh?</p>
    </div>
  )
}
```

Class-based components

Old-school way for creating components that may handle errors.

```
const ClassComponent = React.createClass({
  // ...
  render: () => {
    return (
      <div className="wrapper">
        <p>Not that hard too...</p>
      </div>
    )
  },
})
```

Data flow

No library

All-to-all

Any component can communicate directly with any other component.

Angular 2+

Two-way data binding

Compares values representing application state (in all components) before and after browser events.

React.js

One-Way data binding

Model is the *single source of truth*. More deterministic than two-way binding – side effects are not possible.

First steps in React 7

```
const Parent = (props) => {
  return (
    <div><Child name="firstComponent" /></div>
  )
}

const Child = (props) => {
  processName = () => {
    doProcessing(props.name)
  }
  // ...
}
```

Data in child component is accessed via "**props**" object.

props within children are **immutable** (cannot be overwritten).

First steps in React 8

```
const Child = React.createClass({
  // ...
  disableButton: () => {
    this.setState({ disabled: true })
  },

  render: () => {
    return <button type="submit"
      disabled={this.state.disabled} />
  },
})
```

Each component can have its own isolated STATE.

The state can be accessed by its owner only. It CAN BE MUTATED.

First steps in React 9

By default React.js rerenders components whenever their PROPS or STATE change.

Data flows down

```
const Grandparent = React.createClass({
  // ...
  render: () => {
    return (
      <Parent
        onEnabledClick={this.handleItemClick}
        item={this.currentItem}
      />
    )
  },
})
```

```
const Parent = (props) => {
  return (
    <Child
      onEnabledClick={props.onEnabledClick}
      item={props.item}
    />
  )
}
```

```
const Child = (props) => {
  return (
    <div className="itemContainer">
      <label>{props.item.name}</label>
      <input type="check" onClick={props.onEnabledClick} />
    </div>
  )
}
```

First steps in React 10

EVENTS FLOW UP

```
const Grandparent = React.createClass({
  // ...
  render: () => {
    return (
      <Parent
        onEnabledClick={this.handleItemClick}
        item={this.currentItem}
      />
    )
  },
})
```

```
const Parent = (props) => {
  return (
    <Child
      onEnabledClick={props.onEnabledClick}
      item={props.item}
    />
  )
}
```

```
const Child = (props) => {
  return (
    <div className="itemContainer">
      <label>{props.item.name}</label>
      <input type="check" onClick={props.onEnabledClick} />
    </div>
  )
}
```

First steps in React 10

Data flows down*

events flow up*

```
const Grandparent = React.createClass({
  // ...
  render: () => {
    return (
      <Parent
        onEnabledClick={this.handleItemClick}
        item={this.currentItem}
      />
    )
  },
})
```

```
const Parent = (props) => {
  return (
    <Child
      onEnabledClick={props.onEnabledClick}
      item={props.item}
    />
  )
}
```

```
const Child = (props) => {
  return (
    <div className="itemContainer">
      <label>{props.item.name}</label>
      <input type="check" onClick={props.onEnabledClick} />
    </div>
  )
}
```

First steps in React 10

Data flows down

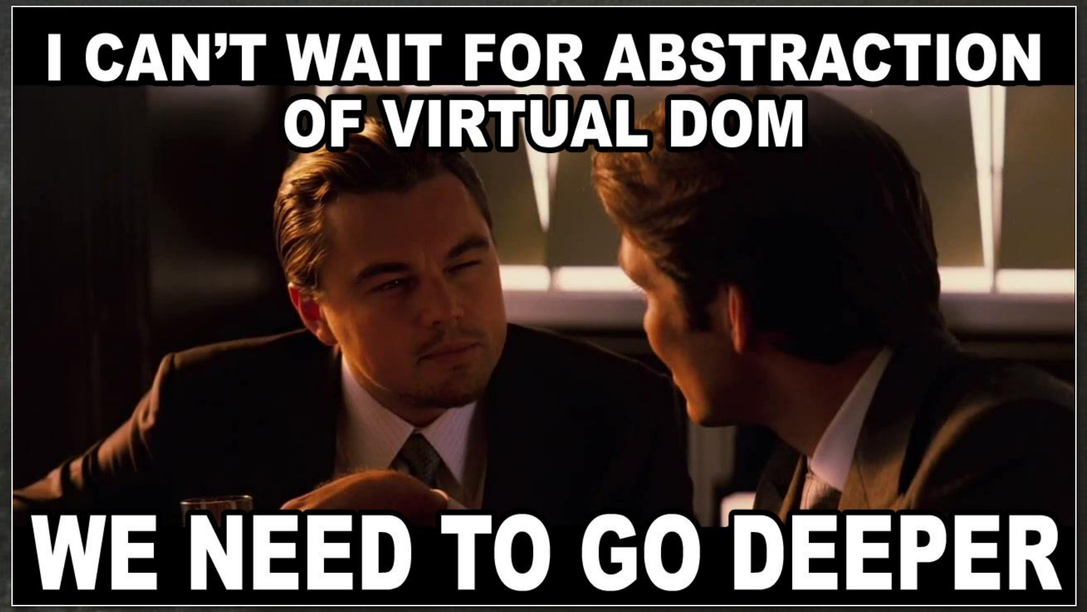
events flow up

Just like in DOM.

Virtual DOM

* Pure JavaScript, in-memory **abstraction of real DOM**
* Whenever render() fires (which means that state or props have been updated), React.js updates the in-memory DOM representation
* If anything has changed, React.js modifies the real DOM to match

Because performance matters.



Hands-on

First react component

source available: A-01-react-base

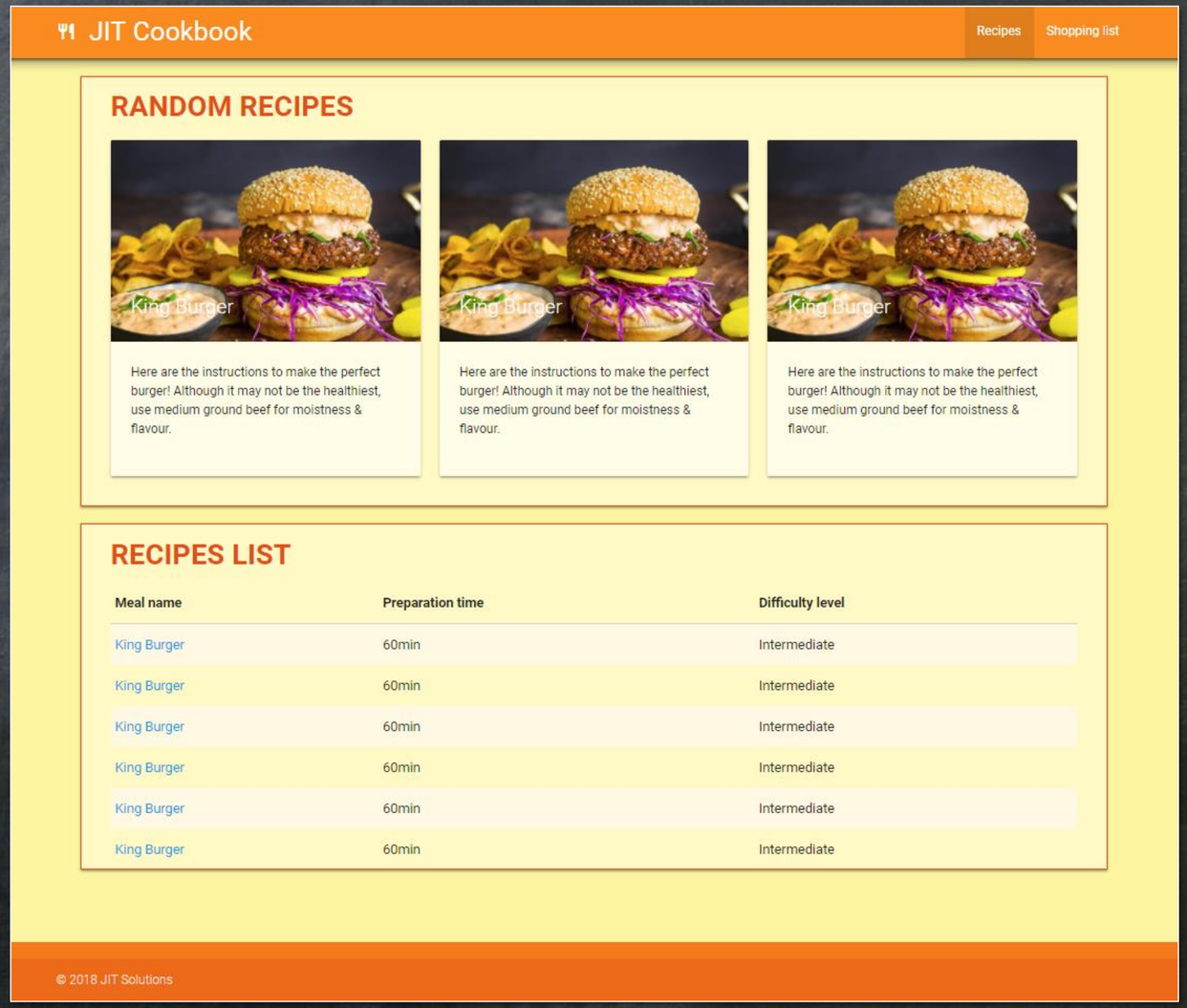
Component split

Has to be done based on the application UI design

&

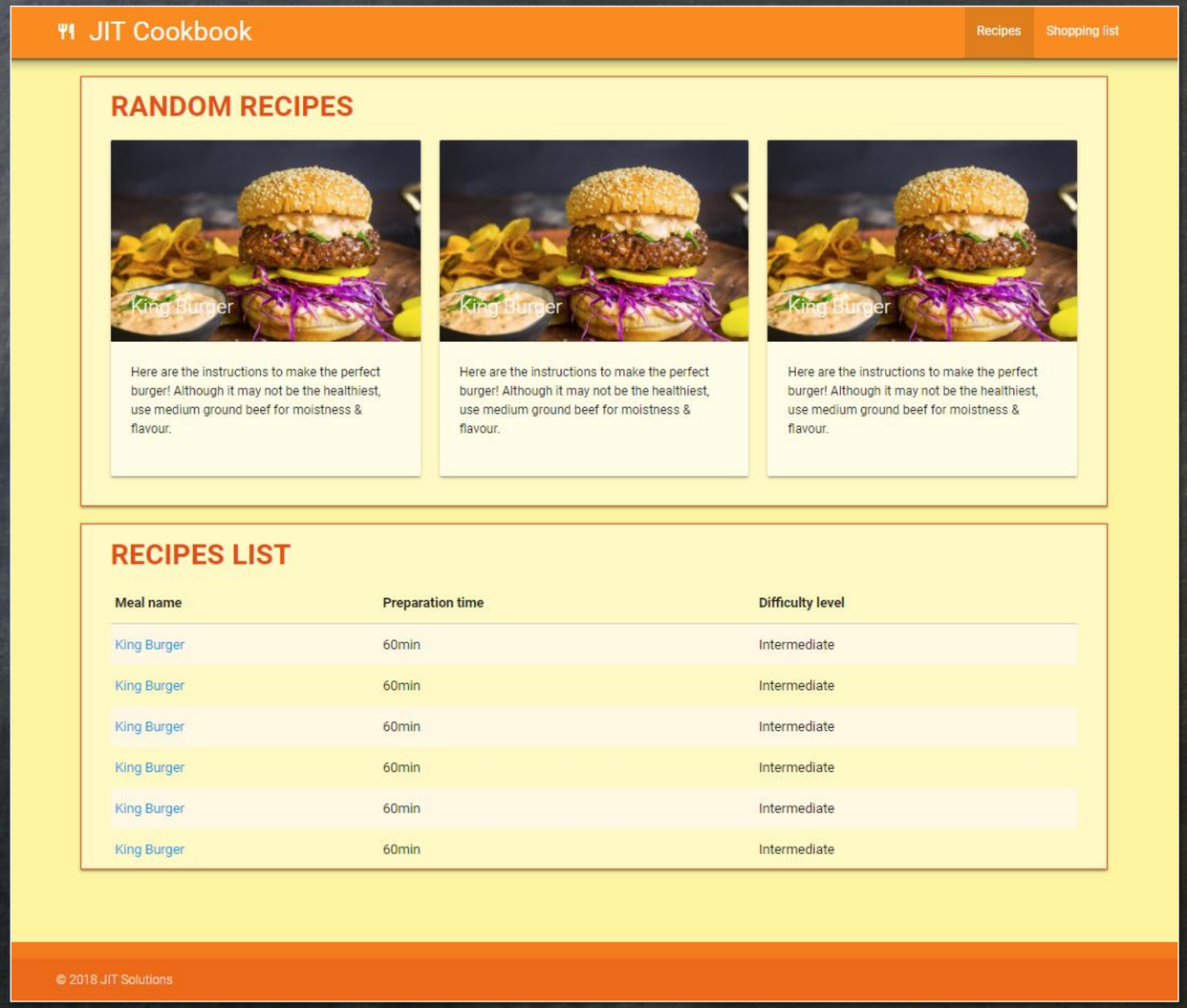
business requirements analysis

First steps in React 15



JIT Cookbook

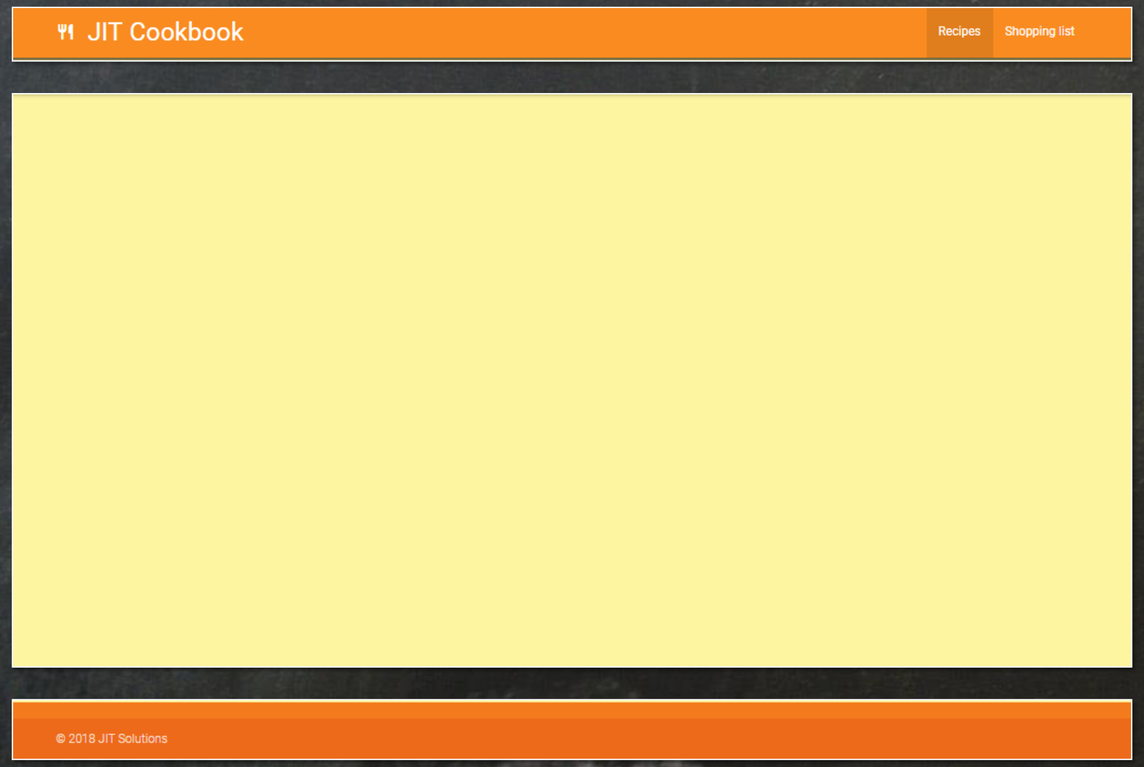
* Navigation bar at the top should show the application name with icon (left side) and navigation panel between dashboards (right side)
* Currently active dashboard tab is indicated with darker color
* User should be able to switch between the dashboards by clicking on particular tabs
* By default Recipes dashboard should be shown
* Footer contains only the copyrights section with company name



JIT Cookbook – recipes dashboard

* Recipes dashboard's main screen is responsible for displaying three random recipes in the top panel and full list of recipes in the bottom panel
* After clicking on any name from the recipes list, the user should see a single recipe view (the whole page should be replaced with page representing single recipe)
* A single recipe view should contain detailed description of the meal and list of ingredients needed in order to prepare it

First steps in React 16

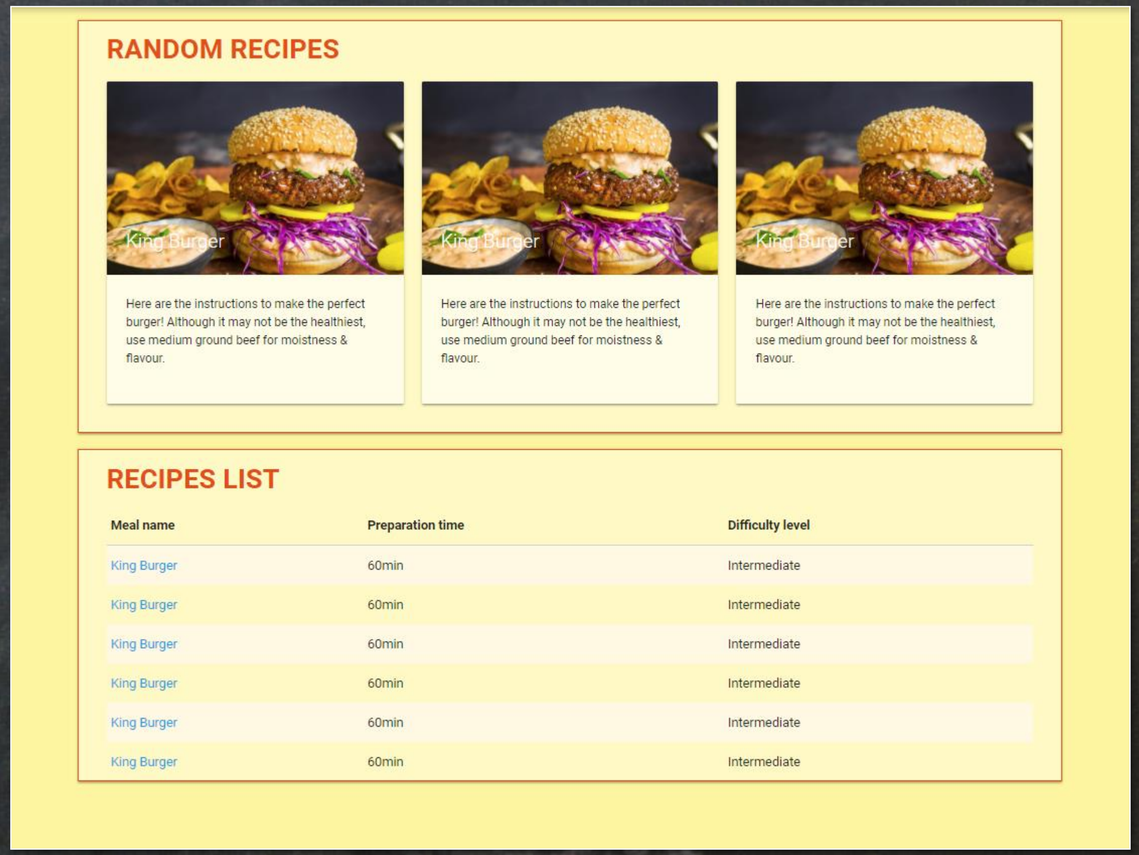


Navbar

Content

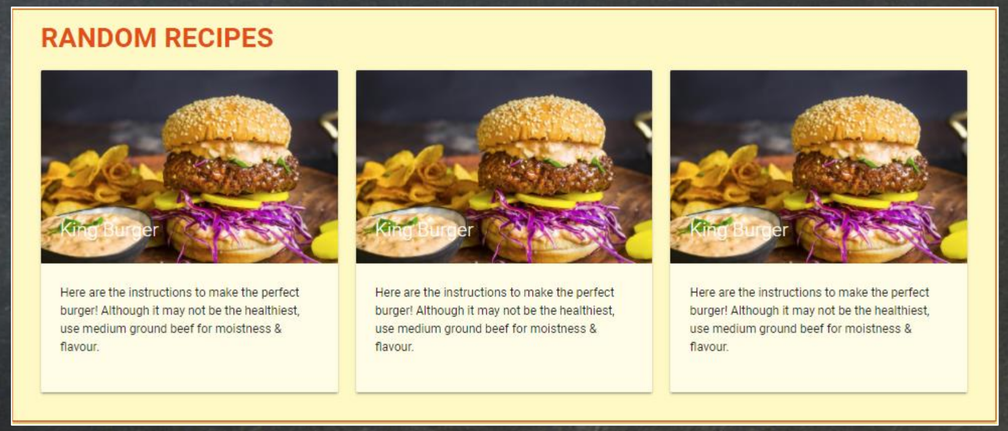
Footer

First steps in React 16

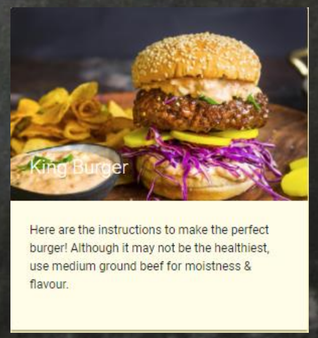


RecipesContainer

First steps in React 16



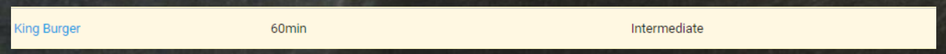
RandomRecipesList



RecipeCard

First steps in React 16

RecipesTable



RecipesTableEntry

Hands-on

Base component split

source available: A-02-react-base-layout

PropTypes

* When the application grows it's quite important to ensure that all objects passed to component as PROPS match the desired types
* PropTypes is a react.js built-in mechanism to run typechecking
* All you have to do is to assign a specific property to a component

Hands-on

Props typechecking

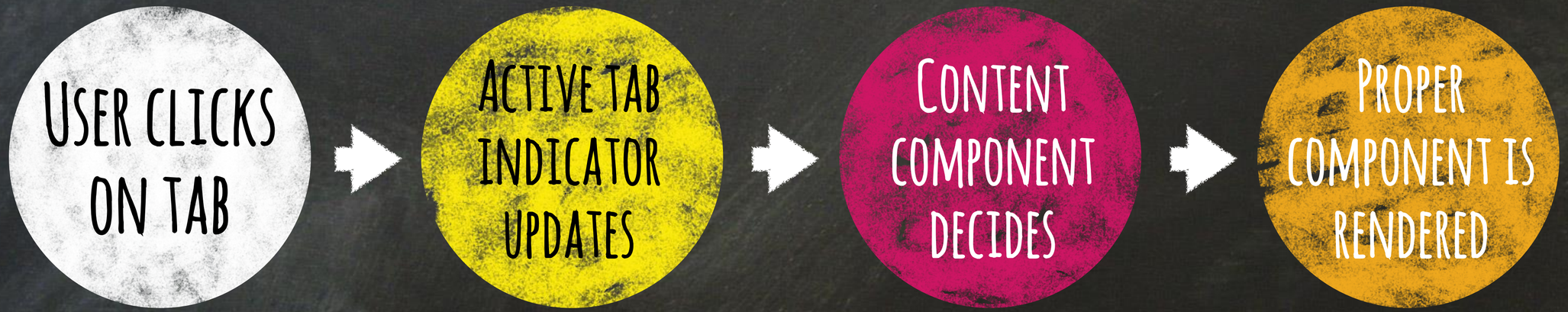
(let's convert some of the components to classes as well)

source available: A-03-class-based-components

Action binding

Making application interactive!

First steps in React 21



Conditional rendering

Decide what to render based on current conditions

```
render = () => {
  userLoggedIn = !isGuest()

  return userLoggedIn
    ? <button name="logOut" type="submit" />
    : <p>Please log in first</p>
}
```

```
render = () => {
  return (
    <div className="profileCard">
      {
        shouldShowAvatar && (
          <img src={this.props.user.imgUrl}
            alt={this.props.user.name} />
        )
      }
      <p>{this.props.user.name}</p>
    </div>
  )
}
```

Hands-on

Bringing some life to the application

source available: A-04-state-action-binding

VDOM and component lifecycle 1

Lists rendering

```
render = () => {
  return (
    <div className="usersList">
      <ul>
        {this.props.users.map((user) => {
          return <li key={user.id}>{user.id}. {user.name}</li>
        })}
      </ul>
    </div>
  )
}
```

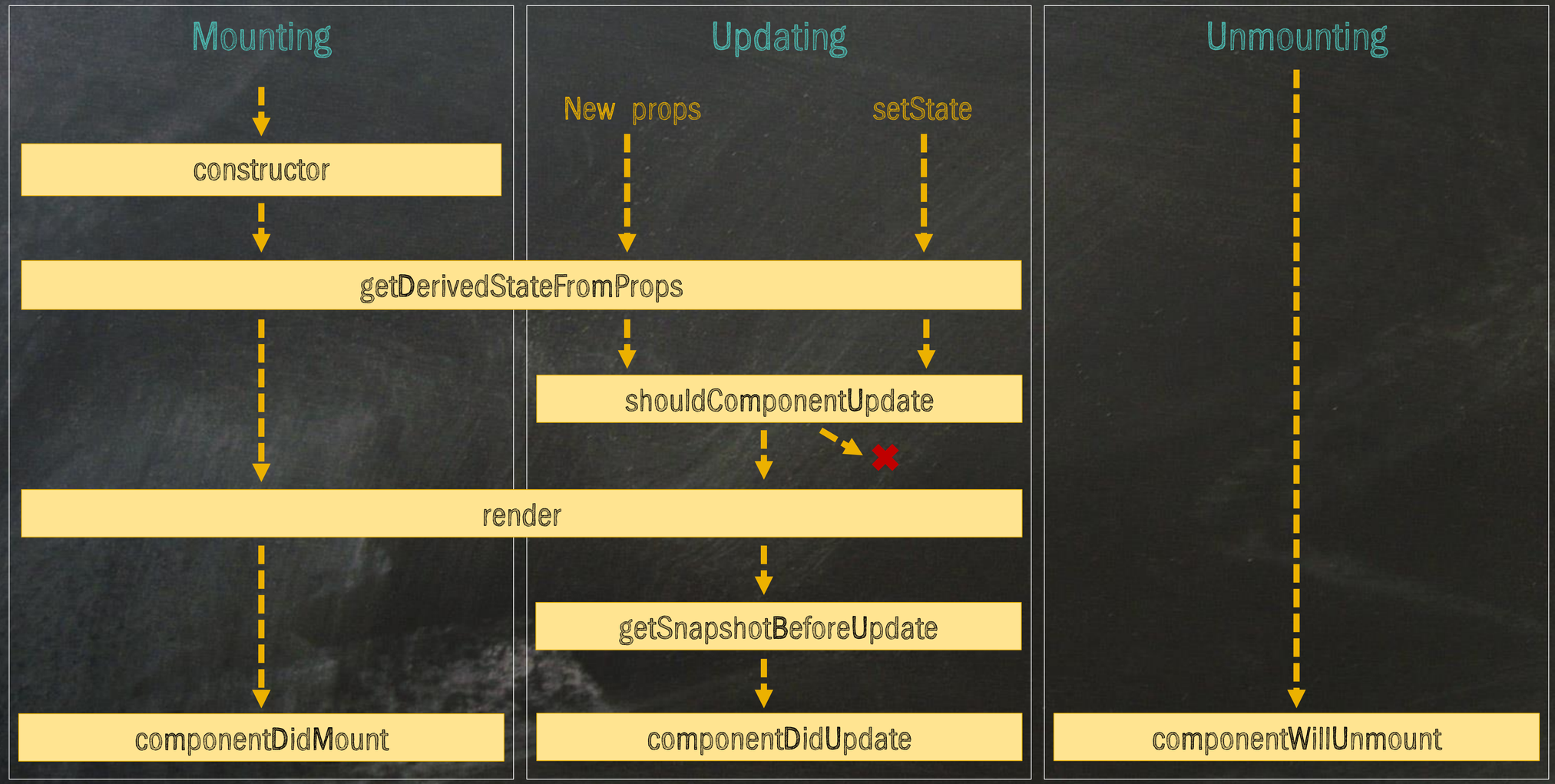
"KEY" PROPERTY IS CRUCIAL.

VDOM and component lifecycle 2

Component lifecycle

* React.js allows us to intercept each phase of components lifecycle
* Useful when we want to update component configuration based on the calculation on new PROPS or the difference between new and old ones
* Hooks may be necessary when you're dealing with 3rd party libraries

VDOM and component lifecycle 3



VDOM and component lifecycle 4

Hands-on

Enhance lists renderings and inspect lifecycle hooks

source available: A-05-list-renderings-and-lifecycle

VDOM and component lifecycle 5

PureComponent

Special subtype of React.js component. Reduces number of wasted renderings.

Functional components equivalent: React.memo(function(props))

VDOM and component lifecycle 6

Hands-on

Optimize number of renderings by inheriting from React.PureComponent

source available: A-06-pure-components

Error handling

During runtime, when exception happens within component code (either constructor, render method or lifecycle methods), the application will crash - the whole component tree will not be rendered.

VDOM and component lifecycle 8

Error handling

We can catch those errors and render fallback UI instead.

Additional lifecycle hooks

componentDidCatch(error, info)

* Called during commit phase
* Side effects are allowed
* Will not work with server-side rendering

static getDerivedStateFromError(error)

* Called during render stage
* State changes affecting the render phase should be placed here

VDOM and component lifecycle 10

Hands-on

Simple error handling for runtime exceptions

source available: A-07-error-handling

Untypical DOM manipulation 1

Portals

* React's Virtual DOM tree matches the structure of the actual DOM by default
* In order to share props with the component located underneath different branch of the DOM tree, we need to go all the way up and preserve the state in the first common ancestor...
* ...what can be really, REALLY frustrating for simple use cases (e.g. modals)

Untypical DOM manipulation 2

Portals

* React 16 came up with a „portal gun” that allows us to modify this behavior so our component hierarchy represent
ation can intentionally break the actual DOM hierarchy

Untypical DOM manipulation 3

Portals

but why?!?!

- Even though the rendered component is located *anywhere*, it still behaves as an ordinary React child
- We can still pass props to the component below in the component hierarchy tree (parent -> child)
- Events triggered by components rendered in Portals bubbles up to the components above in the VDOM tree (child -> parent)

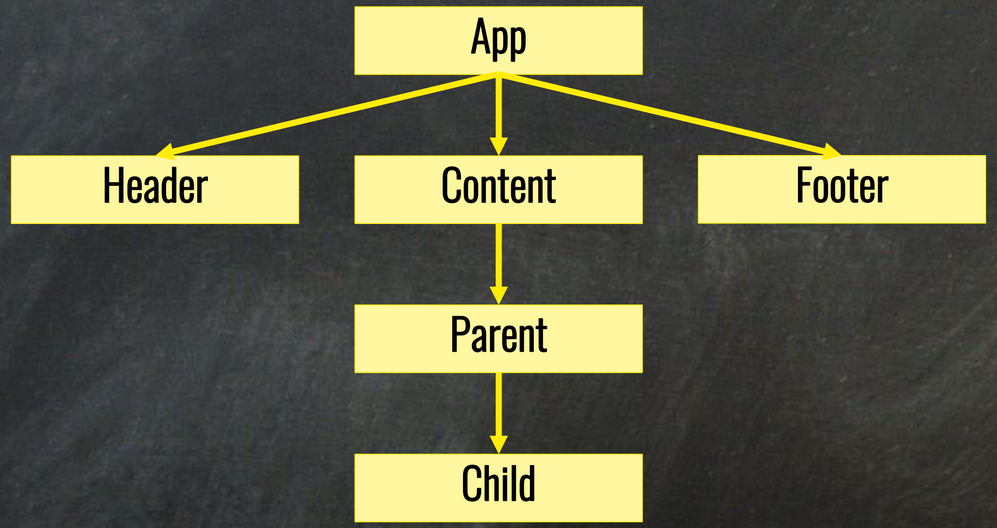
Untypical DOM manipulation 4

```
import ReactDOM from "react-dom"

const Parent = (props) => {
  return ReactDOM.createPortal(
    (
      <div id="cart-entry">
        <Child
          label={props.label}
          onEntryAdd={props.onNewItemAdd}
        />
      </div>
    ),
    document.body,
  )
}
```

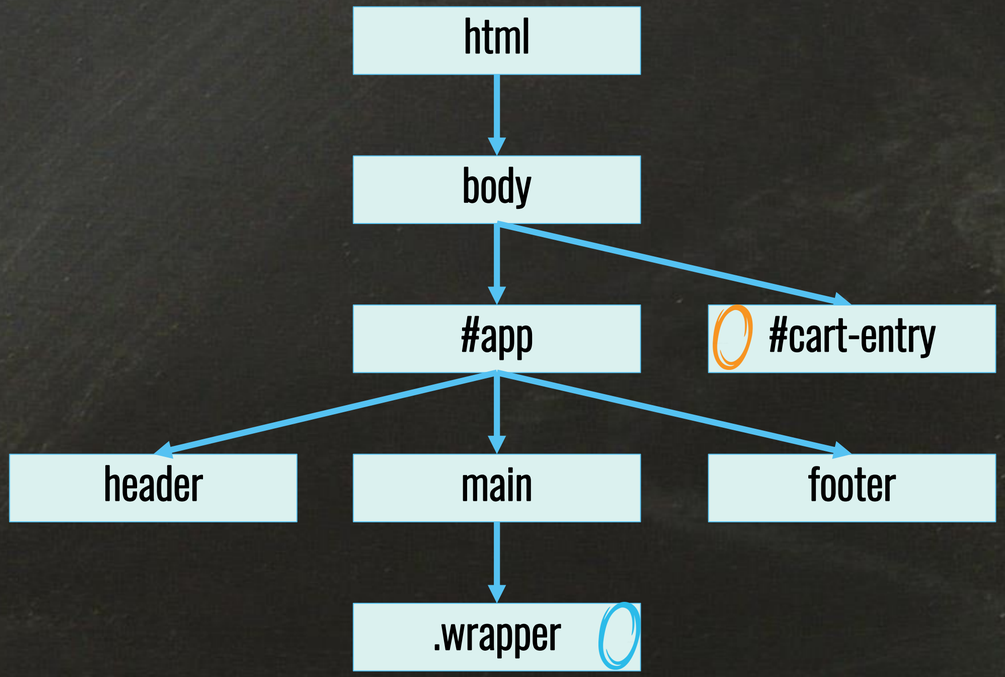
Note: createPortal() is an API from react-dom package

Untypical DOM manipulation 5

Component hierarchy



DOM tree



Untypical DOM manipulation 6

Hands-on

Rendering single recipe view via portal

source available: A-08-single-recipe-view-modal

Untypical DOM manipulation 7

Refs

DO NOT OVERUSE.

Untypical DOM manipulation 8

Refs

* May represent references to raw DOM objects
* As well as handles of instances of React class components
* Useful when you have to deal with focus, selection, scroll-to actions

Untypical DOM manipulation 9

Refs

DO NOT OVERUSE.

Untypical DOM manipulation 10

```
class RefExample extends React.Component {
  constructor(props) {
    super(props)
    this.topContainer = React.createRef()
  }

  render() {
    return (
      <div className="container" ref={this.topContainer}>
        <p>Really long text here (taking more than 100% window height)...</p>
        <button onClick={() => this.topContainer.current.scrollIntoView()}>
          Scroll to top
      </button>
      </div>
    )
  }
}
```

DOM OBJECT AVAILABLE VIA CURRENT PROPERTY OF CREATED REF

Untypical DOM manipulation 11

Refs

DO NOT OVERUSE.

Untypical DOM manipulation 12

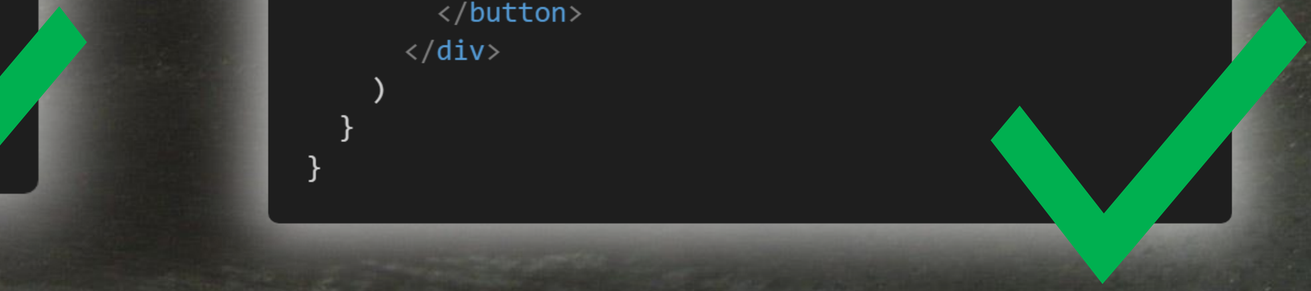
```
class Article extends React.Component {
  constructor(props) {
    super(props)
    this.state({ isExpanded: false })

    this.expand = this.expand.bind(this)
  }

  expand() {
    this.setState({ isExpanded: true })
  }

  render() {
    return (
      <div className="article">
        <h4>Summary of an article...</h4>
        {this.state.isExpanded && (
          <p>Full content of the article</p>
        )}
      </div>
    )
  }
}
```

```
class ArticleWrapper extends React.Component {
  constructor(props) {
    super(props)
    this.articleComponent = React.createRef()

    this.onExpandButtonClick =
      this.onExpandButtonClick.bind(this)
  }

  onExpandButtonClick() {
    this.articleComponent.current.expand()
  }

  render() {
    return (
      <div className="article-wrapper">
        <Article ref={this.articleComponent} />
        <button onClick={this.onExpandButtonClick}>
          Read all...
        </button>
      </div>
    )
  }
}
```

Untypical DOM manipulation 13

```
class Article extends React.Component {
  constructor(props) {
    super(props)
  }

  render() {
    return (
      <div className="article">
        <h4>Summary of an article...</h4>
        {this.props.isExpanded && (
          <p>Full content of the article</p>
        )}
      </div>
    )
  }
}
```

```
class ArticleWrapper extends React.Component {
  constructor(props) {
    super(props)
    this.state = { isExpanded: false }

    this.onExpandButtonClick =
      this.onExpandButtonClick.bind(this)
  }

  onExpandButtonClick() {
    this.setState({ isExpanded: true })
  }

  render() {
    return (
      <div className="article-wrapper">
        <Article isExpanded={this.state.isExpanded} />
        <button onClick={this.onExpandButtonClick}>
          Read all...
        </button>
      </div>
    )
  }
}
```



Untypical DOM manipulation 14

Refs

DO NOT OVERUSE.

Use declarative approach instead.

Untypical DOM manipulation 15

Hands-on

Using ref to manage focus of the input field

source available: A-09-refs

Summary 1

* ES6 is awesome!
* JS ecosystem is a mess, but carefully managed is much more powerful than any other programming languages ecosystems
* In React world everything is a component
* Thanks to virtual DOM React does great job on efficiency; but sometimes we are smarter so we can hint him
* Do not overuse refs :)