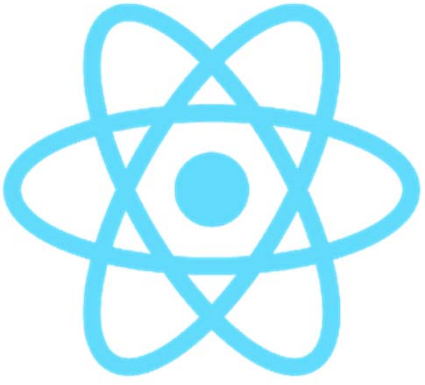
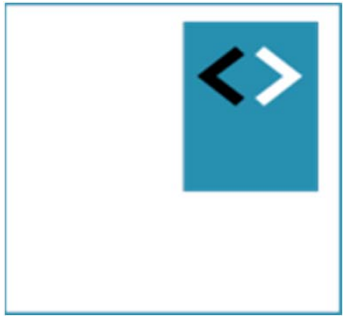
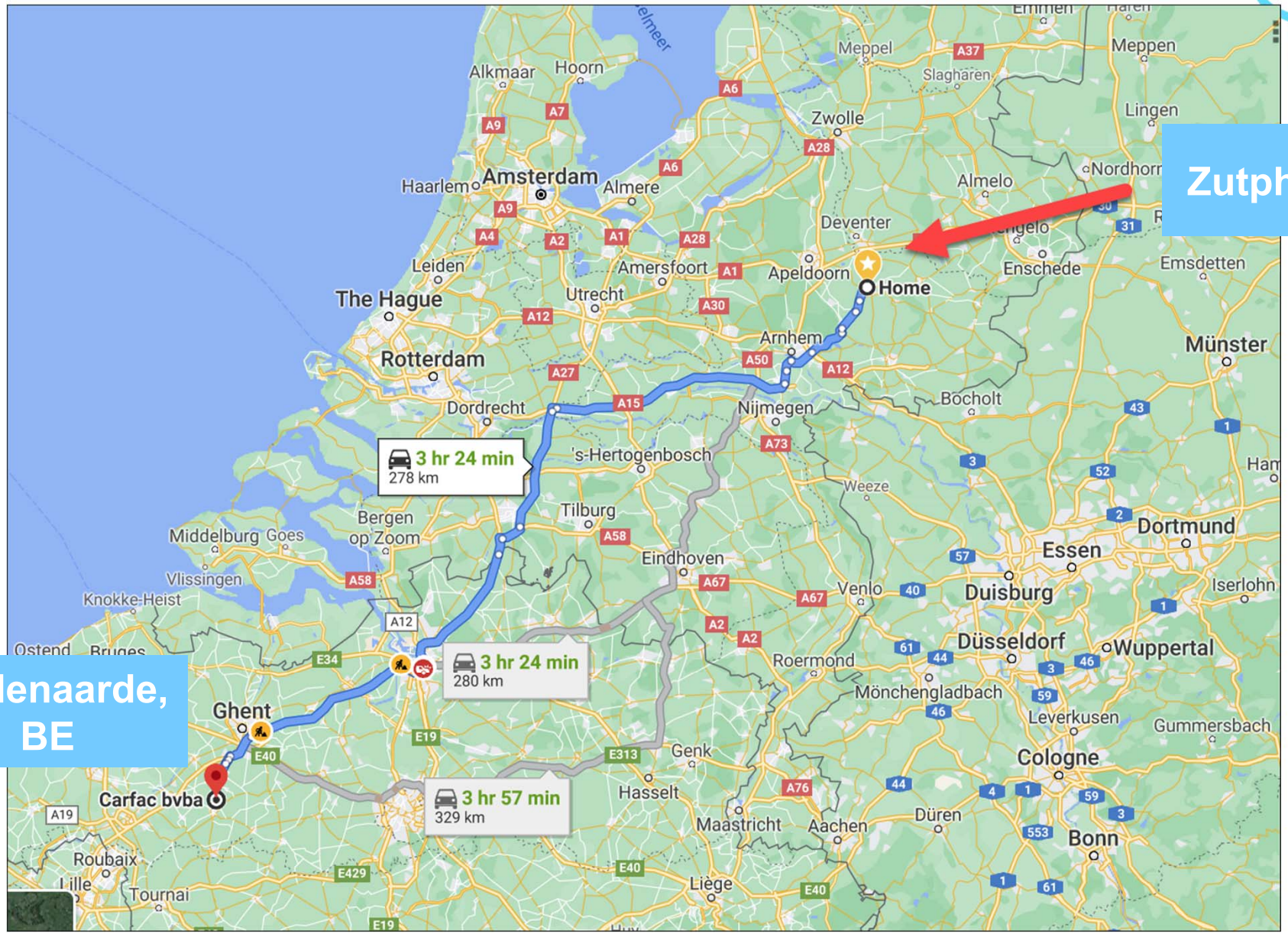


React Fundamentals Module - Introduction



Peter Kassenaar –
info@kassenaar.com


Zutphen, NL
Oudenaarde, BE
Home
Carfac bvba
3 hr 24 min
278 km
3 hr 24 min
280 km
3 hr 57 min
329 km
Amsterdam
Rotterdam
The Hague
Utrecht
Arnhem
Nijmegen
Apeldoorn
Deventer
Zwolle
Amersfoort
Eindhoven
Tilburg
's-Hertogenbosch
Dordrecht
Leiden
Haarlem
Alkmaar
Hoorn
Almere
Meppel
Meppen
Lingen
Almelo
Enschede
Emsdetten
Münster
Bocholt
Weeze
Venlo
Roermond
Mönchengladbach
Düsseldorf
Duisburg
Essen
Dortmund
Wuppertal
Leverkusen
Cologne
Bonn
Gummersbach
Iserlohn
Aachen
Düren
Maastricht
Hasselt
Genk
Liège
Ghent
Bruges
Ostend
Knokke-Heist
Middelburg
Goes
Vlissingen
Bergen op Zoom
Roubaix
Lille
Tournai
A1
A2
A4
A6
A7
A9
A12
A15
A19
A27
A28
A30
A37
A50
A58
A67
A73
A76
E19
E34
E40
E313
E429

Peter Kassenaar

- Trainer, author, developer – since 1996
- Specialty: *“Everything JavaScript”*
- JavaScript, ES6, Angular, NodeJS, TypeScript, jQuery, React, Vue, etc.

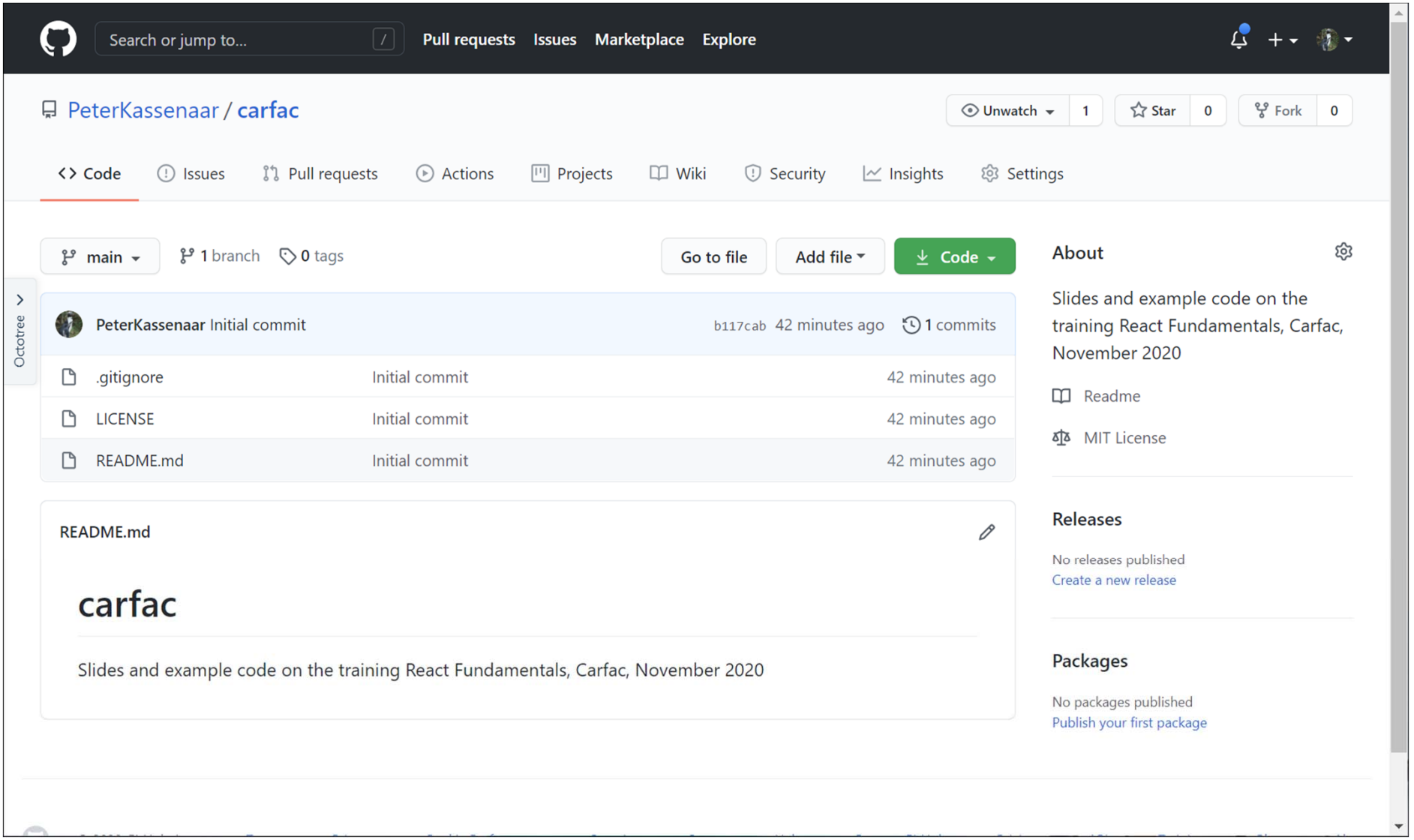
www.kassenaar.com

info@kassenaar.com

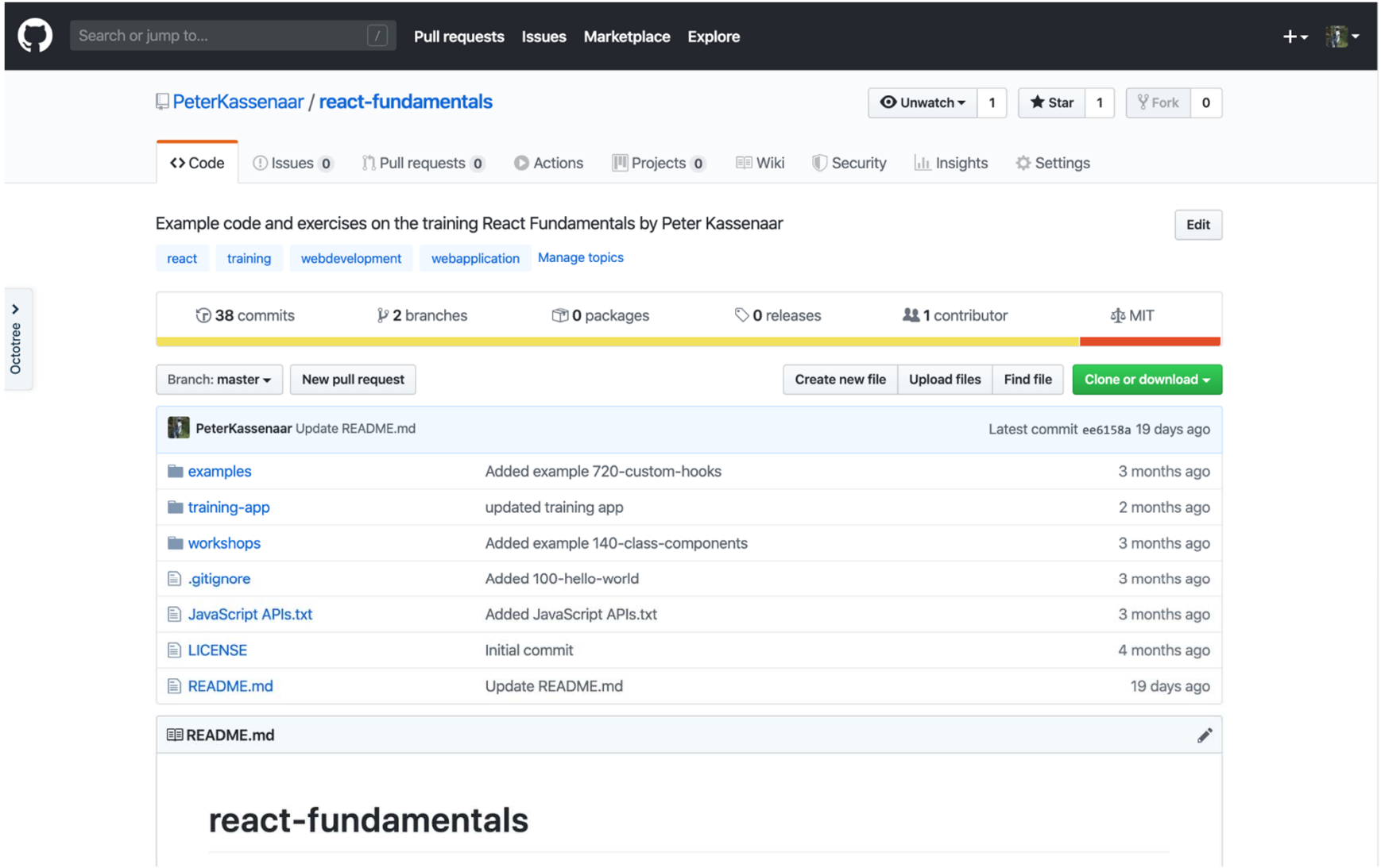
Twitter: @PeterKassenaar

www.reacttraining.nl

https://github.com/PeterKassenaar/carfac

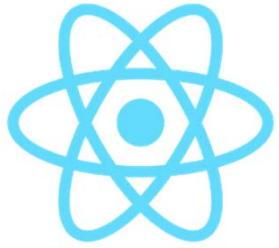


github.com/PeterKassenaar/react-fundamentals



About you…

Tell us a little bit about yourself



Knowledge of React, other front-end frameworks?

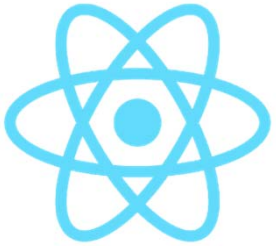
Have you worked with HTML, CSS, JavaScript,

TypeScript and/or other (web)languages?

Tell us a little bit about your projects.

What are your expectations of this course?

Material

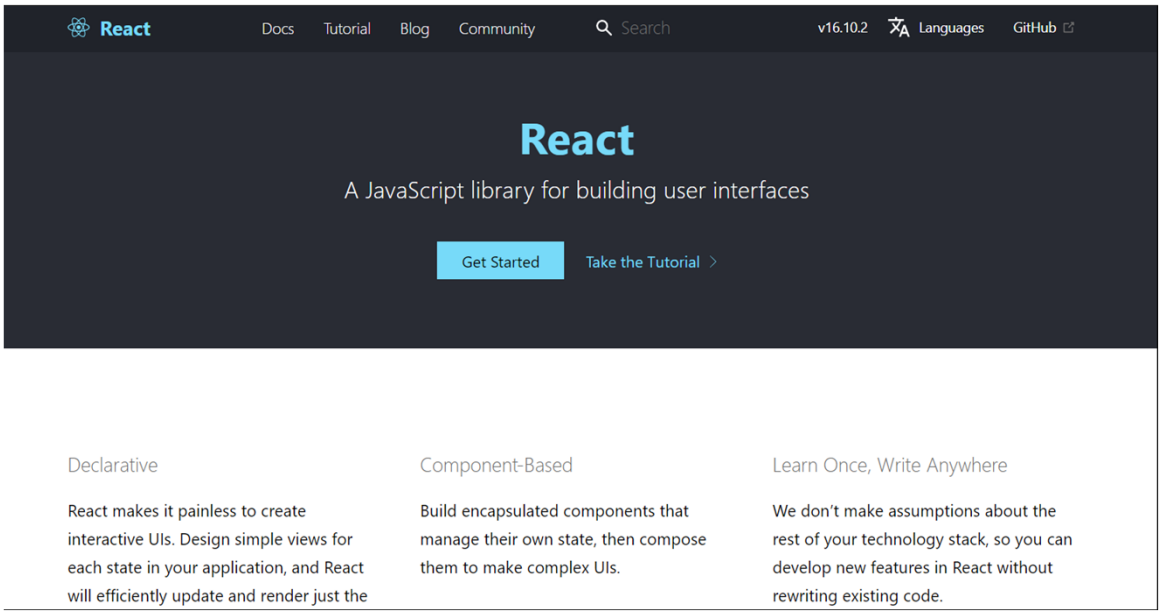


Software (React + libraries)

Handouts (PDF, Github)

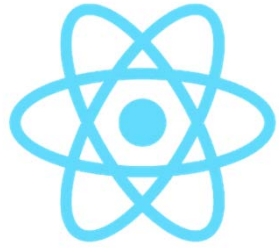
Workshops (In Presentations)

Websites (online)



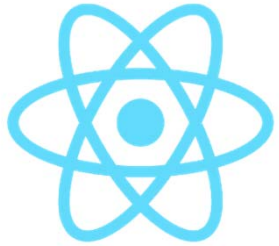
reactjs.org/

Agenda - 3 days



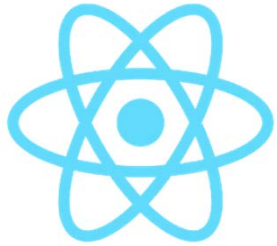
- Introduction – overview of the front-end landscape
- React tooling – installation
- Hello World –the structure and architecture of React apps.
- Zooming in:
  - Components – Class based vs. Function based

Agenda – cont'd



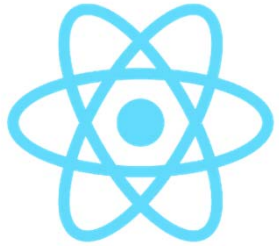
- React hooks - introduction
- Data and state in components
- Styling React components
- Communication between components
  - Props and events
- Lifecycle Hooks
- Handling user input

Agenda – cont'd



- Communicating with external API's
- Using React Router
  - Alternative – Reach Router
- React Forms
- React hooks in depth
  - `useState`, `useEffect`, custom hooks
- Intro in managing state with Redux
- Deploy your app to production

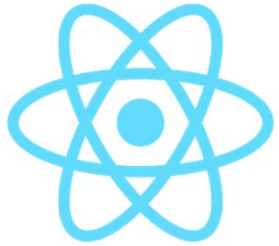
Labs and example code



1. Labs/Exercises
    - In the PDF's in the Github-repo. But: feel free to deviate. Adapt to suit your own needs! (hobby, work, current projects)

2. Example code
    - Executions of the exercises, small projects (`npm install`, `npm run serve`)
    - Work in progress – let me know of additions/errors!
    - github.com/PeterKassenaar/react-fundamentals

Global Agenda



25-26-27 Nov. 2020 – Wed–Thu–Fri

~09:00 start

- ~ 10:30 coffee break

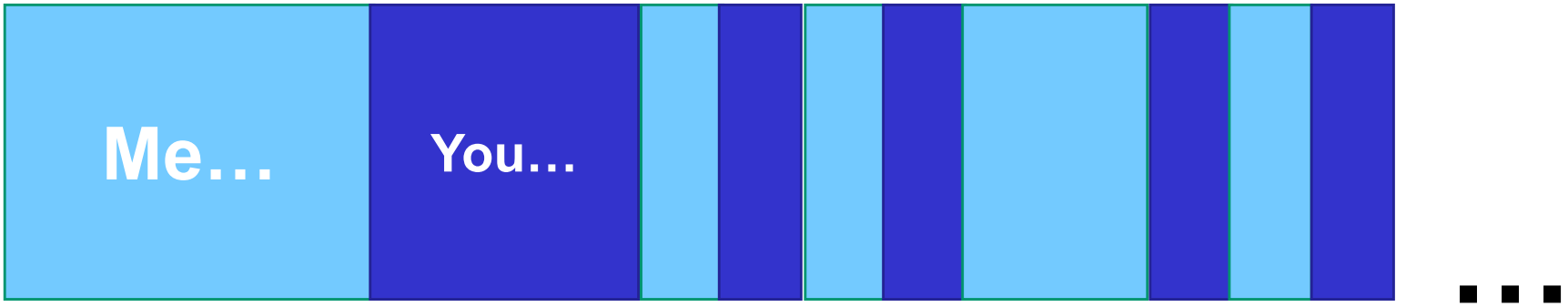
~ 12:00-13:00 Lunch Break

- ~ 14:30 coffee break

~16:15-16:30 wrap-up

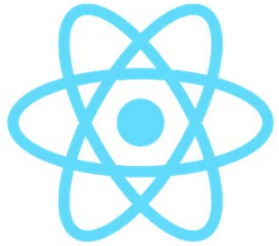
Friday – probably wrap-up a little bit early

Overall process



Questions?

Addressing the “WHY” question!



WHY, would we want to use a frontend framework.

It is all HTML, CSS and JavaScript right?

speed,

not re-inventing

the wheel,

consistency,,

community,

performance,

testing….

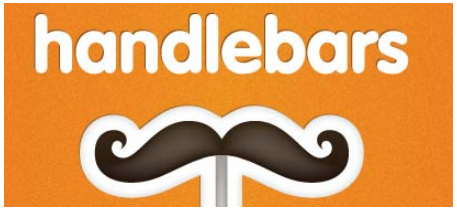
Rethorical question:

“Do we want to go back

to the jQuery days?”

Old school web apps

HTML + templates



Data Binding

Routing

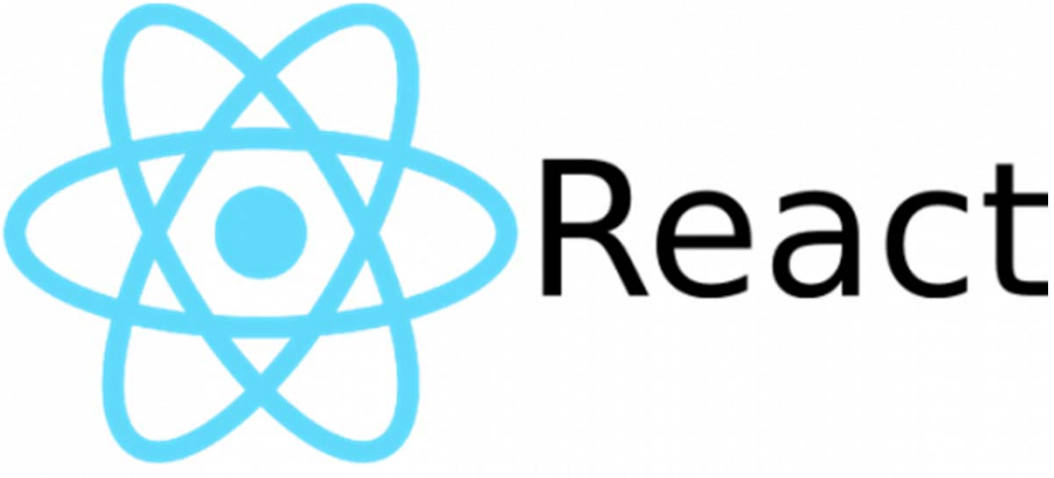
DOM-manipulation

Mobile development

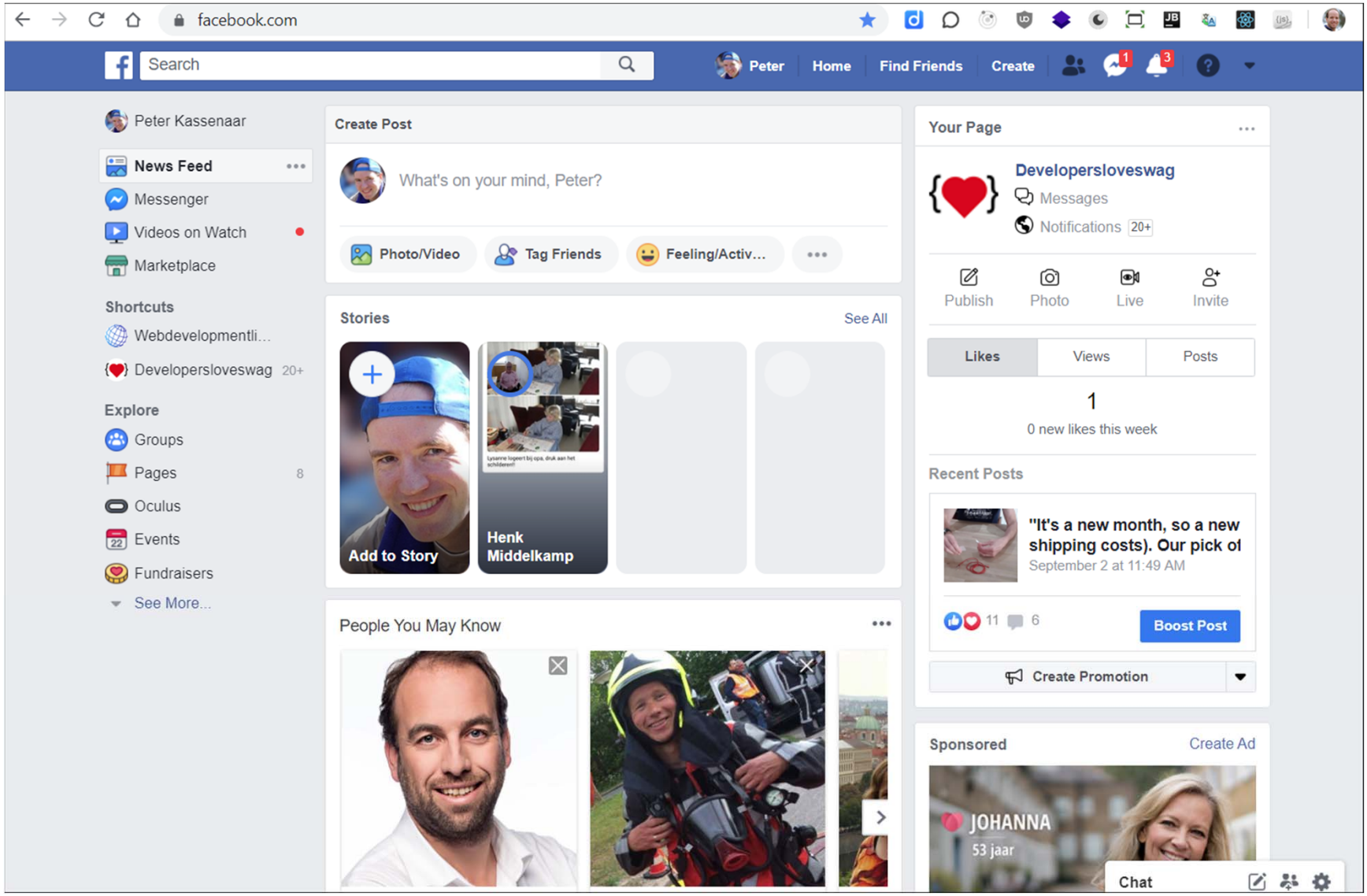
...

“The Frankenstein Framework”

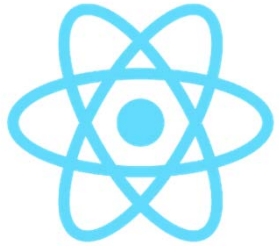
Front-end Frameworks – the big four



React – Created by Jordan Walke (Facebook)



Brief history of React



2011 – Created at Facebook

2012 – Used on Instagram

2013 – Open source (May 29, 2013)

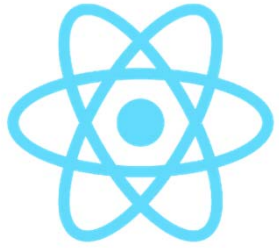
2014 – growth in popularity outside Facebook

2015 – React Native

2016 – React 15 (update to SemVer, from 0.14)

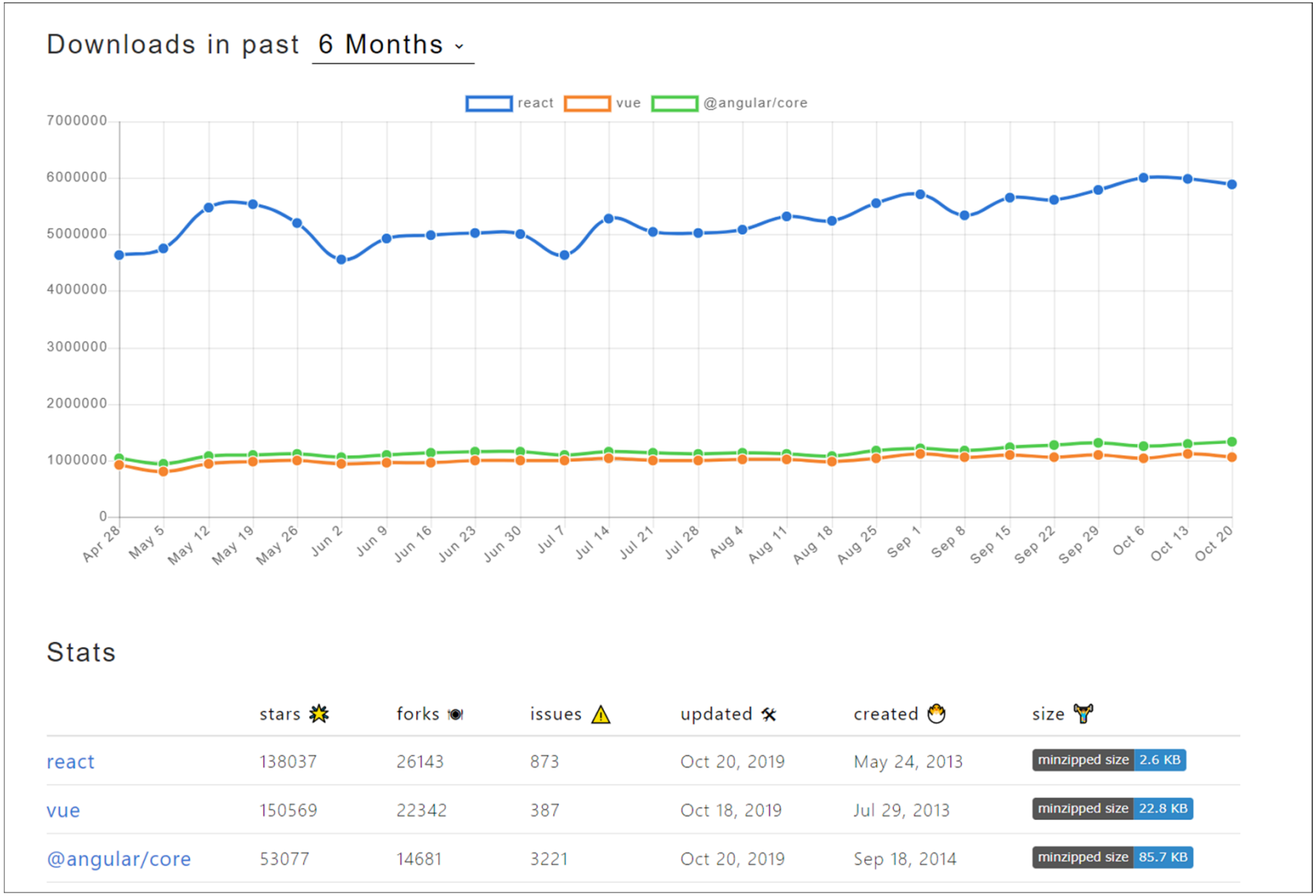
Today – *Facebook uses 30k+ components, employs full-time dev staff, used my many companies*

Why React?



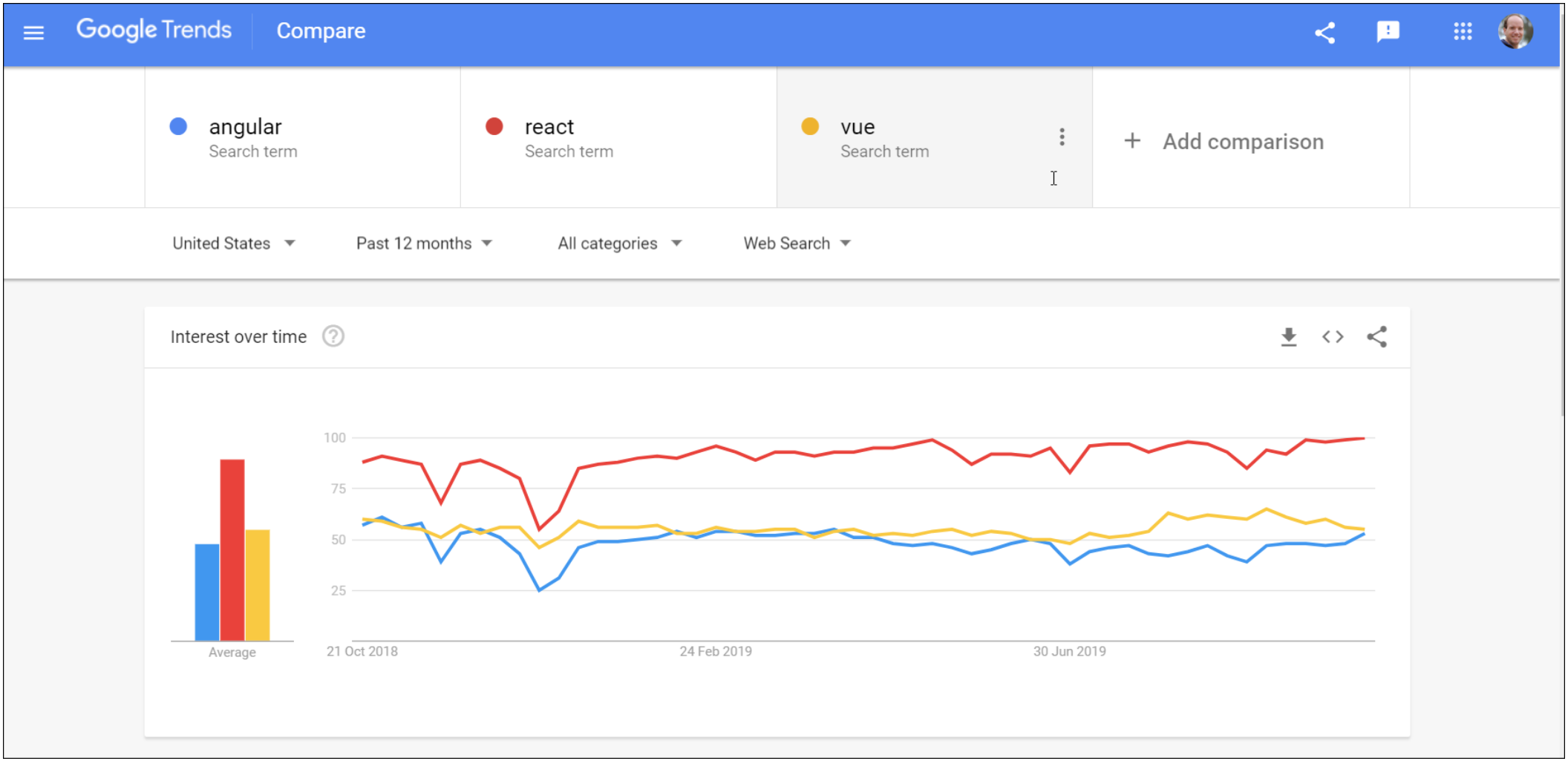
- Flexibility
- Developer experience
- Corporate investment
- Large community (Github, Stack Overflow)
- Performance
- Testability

Npm trends comparison



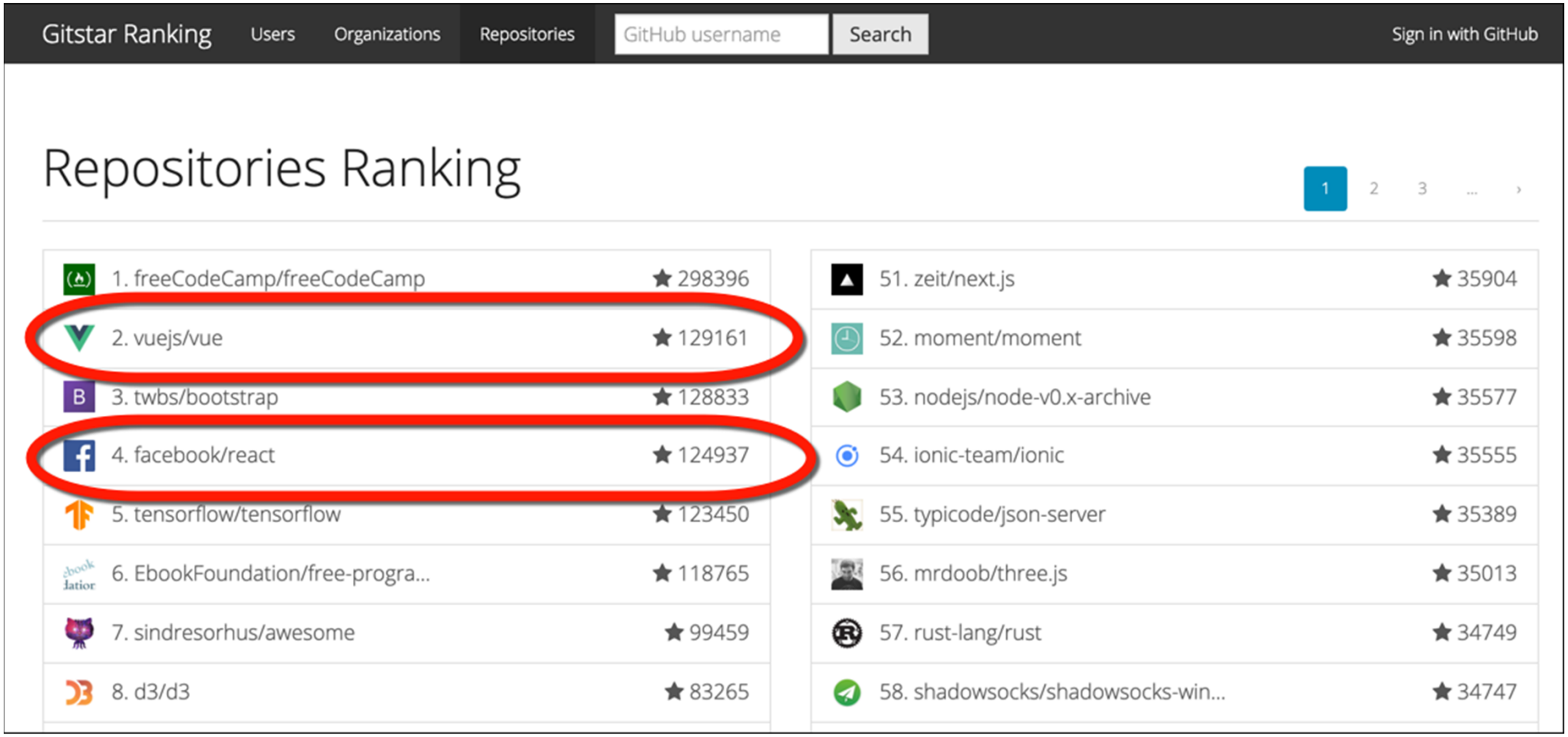
https://www.npmtrends.com/@angular/core-vs-react-vs-vue

Google Trends



https://trends.google.com/trends/explore?geo=US&q=angular,react,vue

Giststar rankings



https://gitstar-ranking.com/repositories

codeinwp

build a website / web hosting / software / wordpress / about us

CodeinWP content is free. When you purchase through referral links on our site, we earn a commission. Learn more

Angular vs React vs Vue: Which Framework to Choose in 2020

by Shaumik Daityari / updated: august 9, 2020 / web & app frameworks

This post is a comprehensive guide on which is perhaps the right solution for you: **Angular vs React vs Vue**.

Just a couple of years ago, developers were mainly debating on whether they should be using Angular vs React for their projects. But over the course of the last couple of years, we saw a growth of interest in a third player called Vue.js.

If you are a developer starting out on a project and cannot decide on which JavaScript framework to use, this guide should help you make a decision.

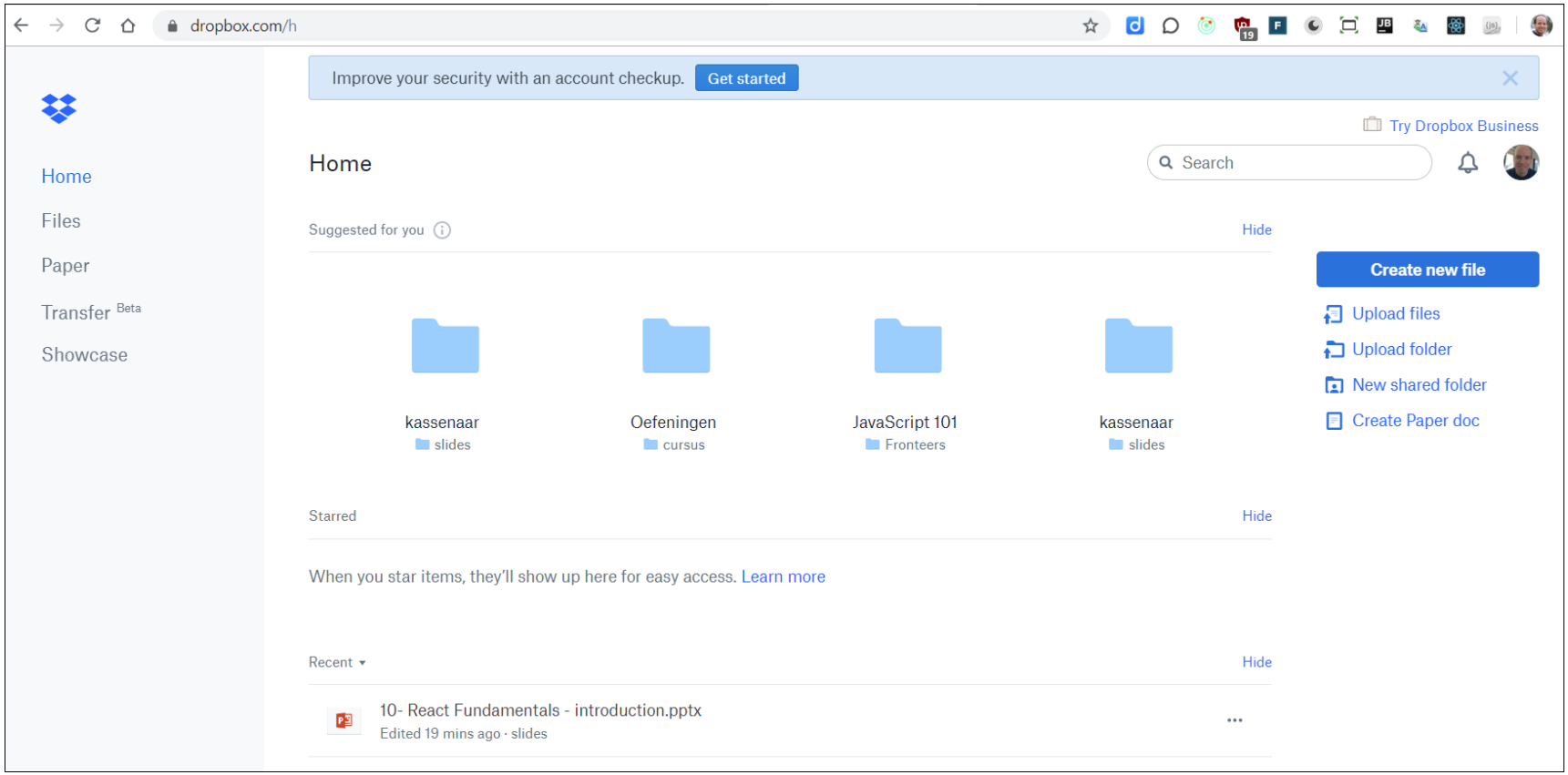
We cover various aspects of Angular, Vue, and React to see how they suit your needs. This post is not just a guide on Angular vs React vs Vue but aims to provide a structure to help judge front-end JavaScript frameworks in general. In case a new framework

Which #javascript framework to choose in 2019: @angular vs @vuejs vs @reactjs

CLICK TO TWEET

https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

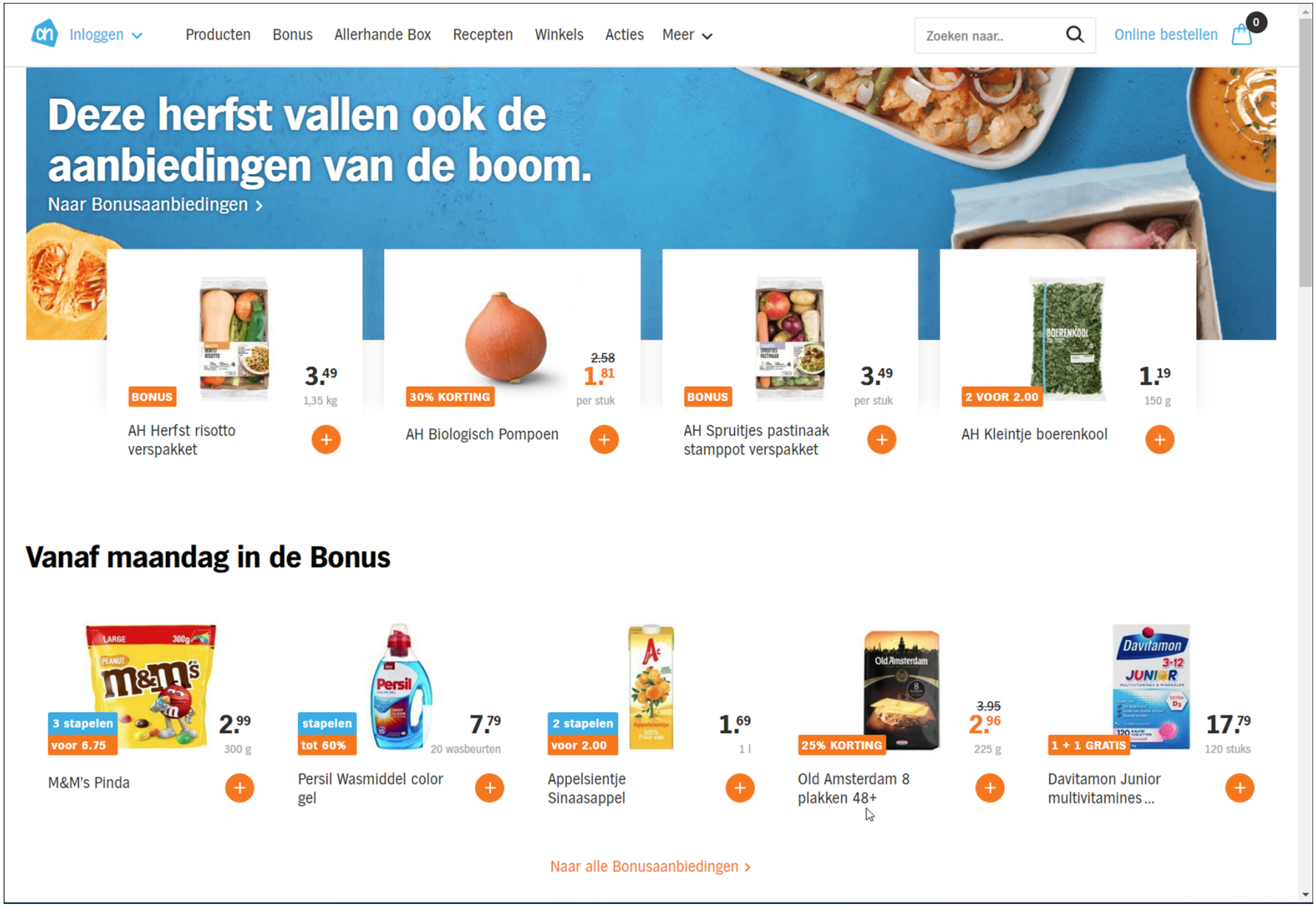
“React is used by many Fortune 500 companies, including Airbnb, BBC, Instagram and Dropbox”



Google for ‘built with react’

https://anyforsoft.com/blog/10-famous-websites-built-react-js

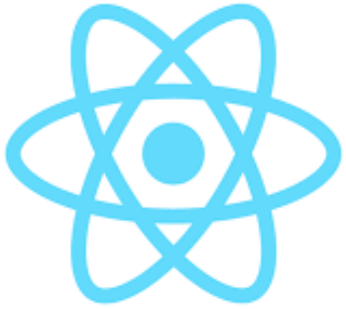
NL:



https://www.ah.nl/

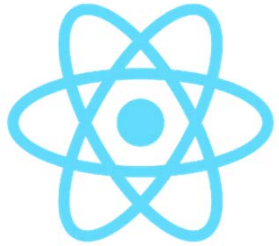
Why *not* React?

- Framework (Angular, Vue)
- Concise (less code)
- Template centered
- Separate files
- Non-standard (html, elements, shadow DOM)
- Community backing (except Angular)



- Library
- Explicit (more coding)
- JavaScript Centered
- Single File
- Non standard (JSX, shadow DOM)
- Corporate backing

More concerns…



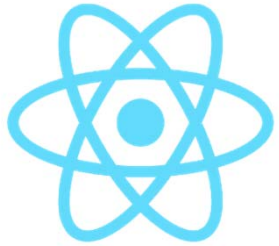
“But JSX is so different from HTML!”

- True, but that’s the choice React made
- Point of departure is JavaScript. Not the HTML-template

```
<h2 color="red">This is a heading</h2>
```

```
React.createElement('h2', { color: 'red'}, 'This is a heading');
```

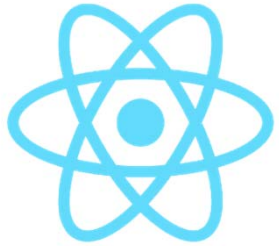
JSX differences



Hoewever - JSX looks a lot like HTML. There are only a few differences:

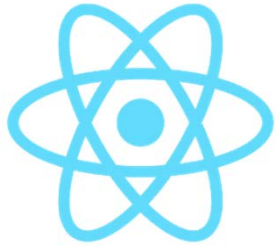
HTML	JSX
`for` (for labels)	`htmlFor`
`class`	`className`
`style="color: red"`	`<style={{color: 'red'}} >`
`<!-- comment -->`	`{/* Comment */}`
Events: `click`, `blur`, `focus` etc	Prefix with `on`+capital: `onClick`, `onBlur`, `onFocus`, etc

Still worried?



- HTML to JSX compiler
- https://magic.reactjs.net/htmltojsx.htm

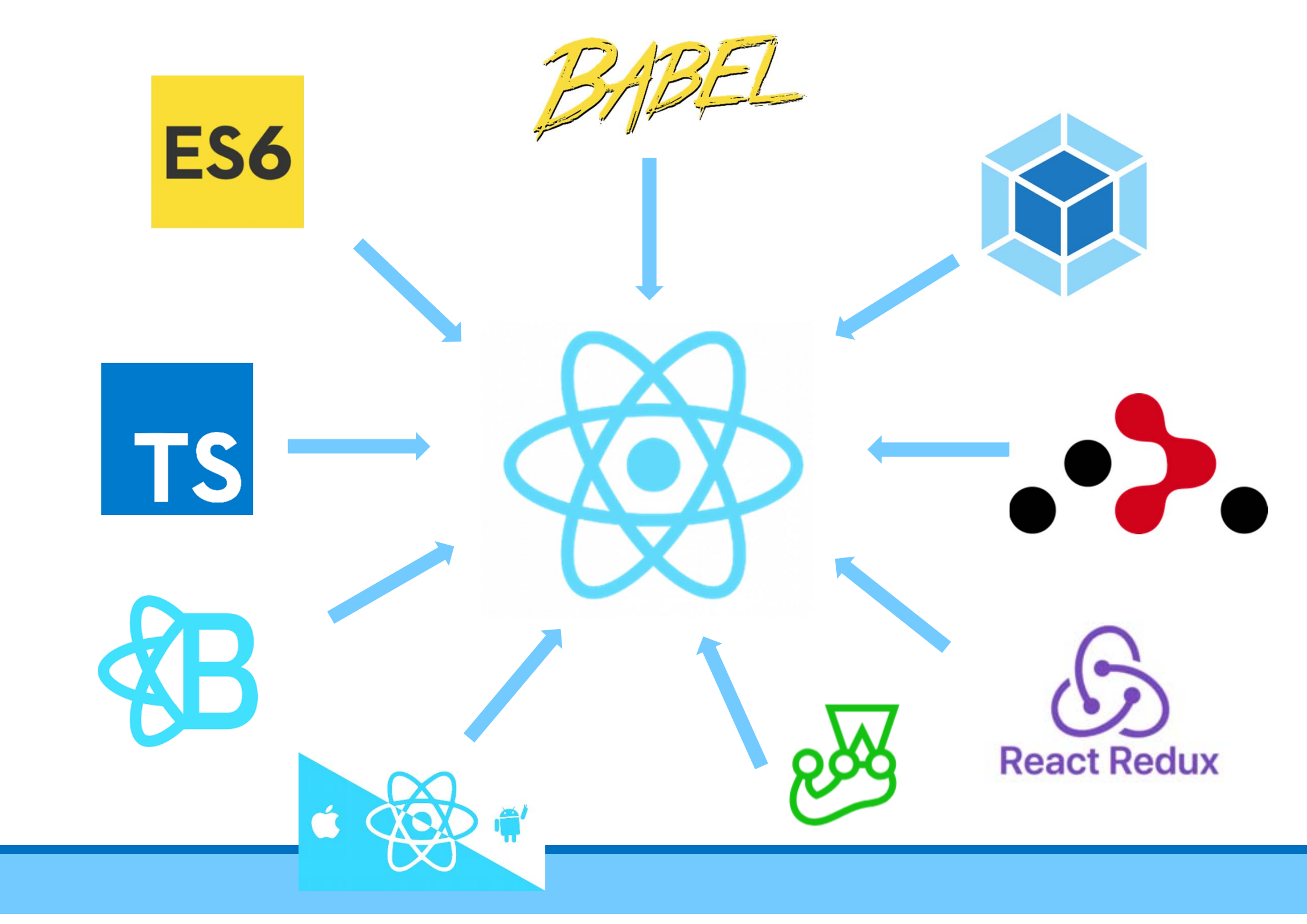
Other concerns...



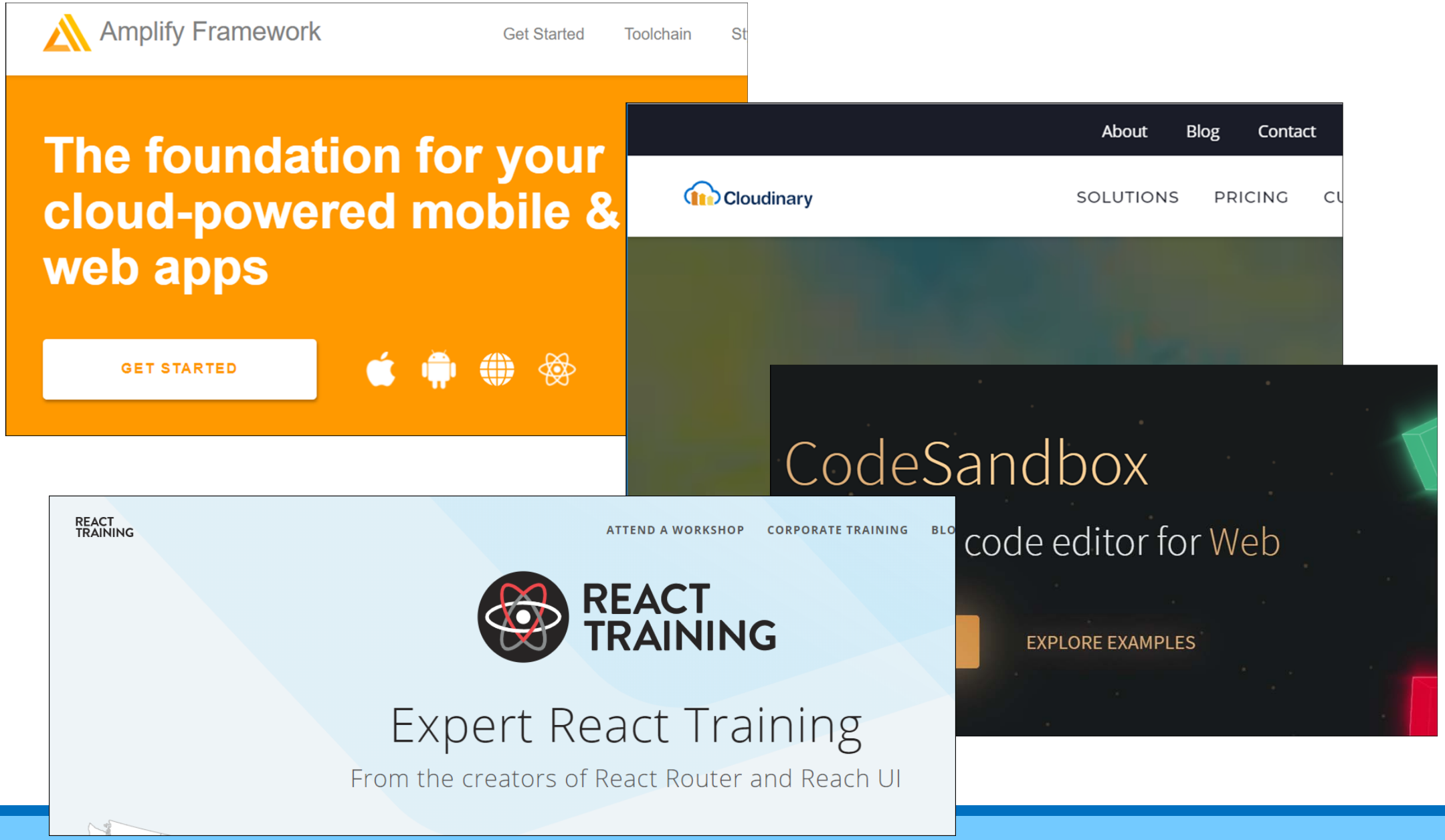
- “React requires a build step”
  - True, but this is also true for all other modern frameworks.
- “There might be version conflicts”
  - Yes, you need to address that and update the code. Just as in other frameworks.
  - Use https://github.com/reactjs/react-codemod to update code
  - Dependencies (router, redux) should be updated separately
- “But there is so much old and outdated stuff online”
  - True. As with other frameworks. Always check date and validity.

The React ecosystem

React is not isolated/not living on an island


ES6
BABEL
TS
React Redux

Additional services, options, etc.

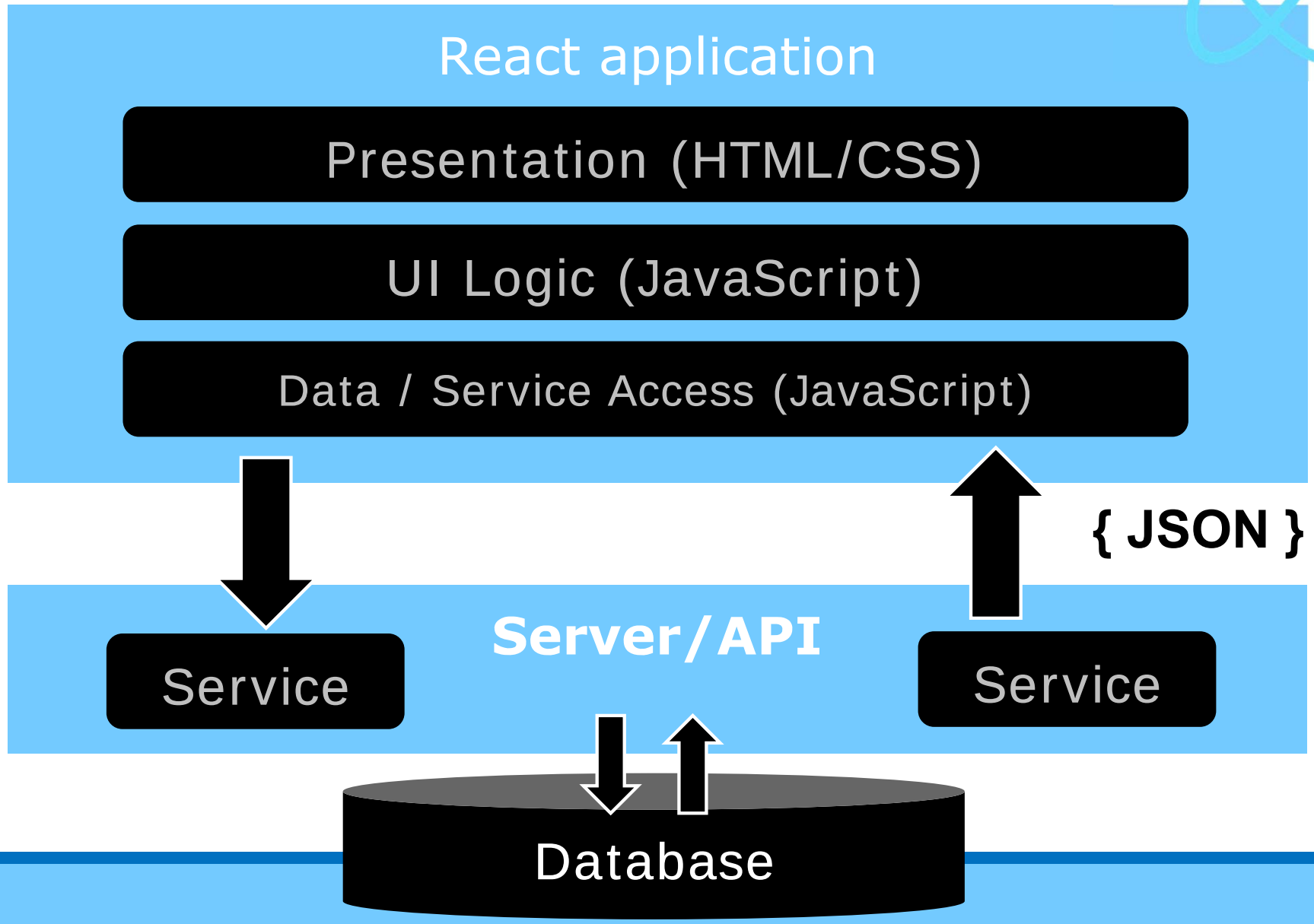


And more….

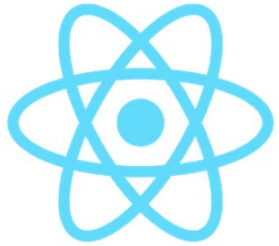
React’s basic concepts

Getting a conceptual understanding of every React app

Single Page Application



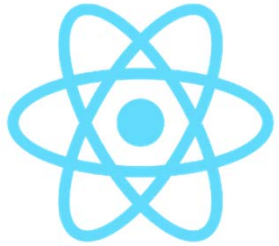
3 Basic principles in every React App



1. Components

- Like functions: `class` based and `function` based components
- Input: `props` and `state`
- Output: UI (via the `render()` function)
- Reusable and composable
- Mentioned via their name, like `<myComponent />`
- Components can have private state

3 Basic principles in every React App



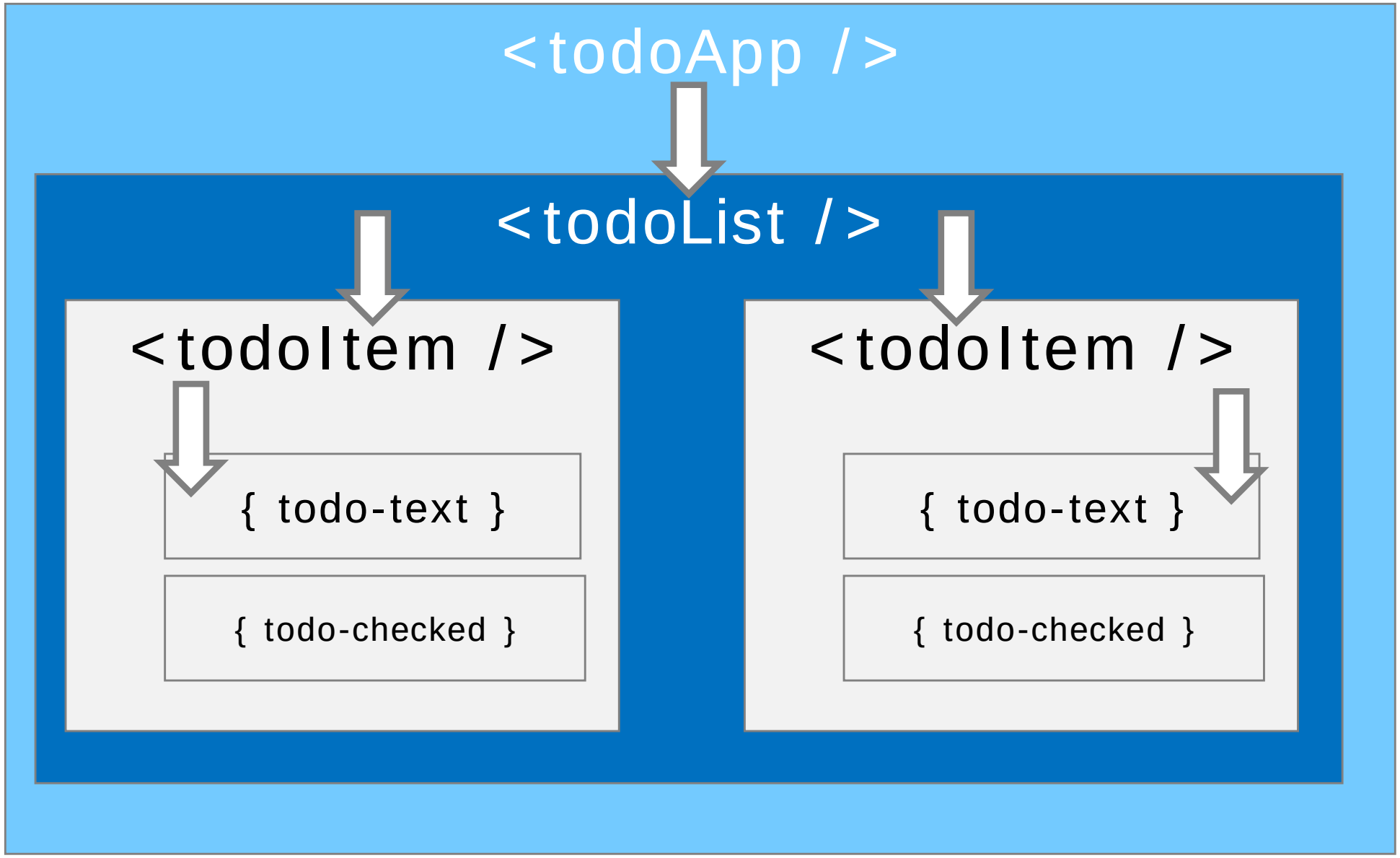
2. Reactive updates

- React will react to the changes it makes or receives
- Update the browser, based on those changes

3. Virtual views in memory

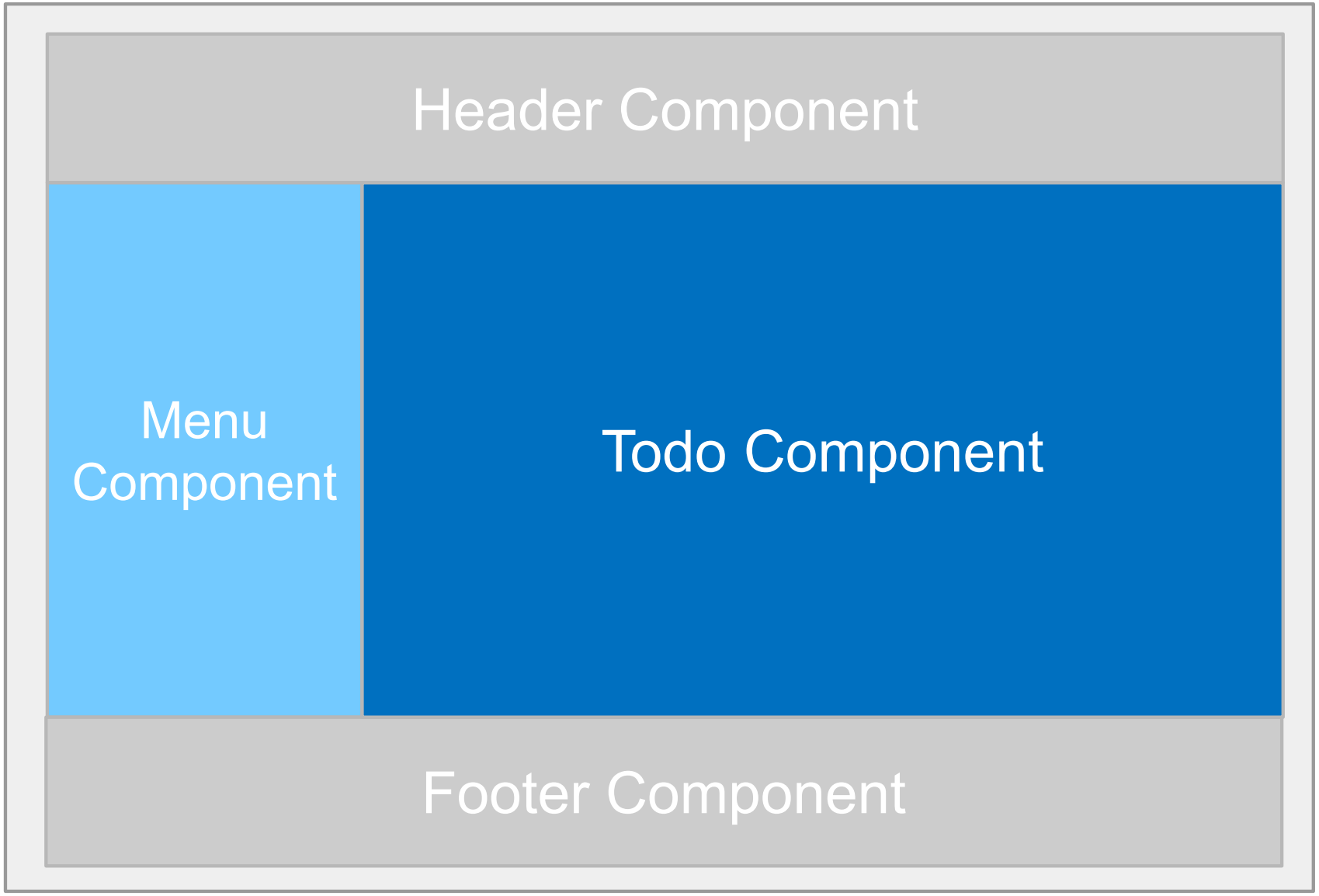
- Don’t write HTML (for instance to handle data)
- Use JavaScript (or JSX) to generate HTML
- No HTML template language
- Virtual DOM or Tree reconciliation
- Build the DOM in-memory before adding it to the UI

React – app- & component structure



“A React app is a tree of components”

Components – visual representation



React components

Function based components

Simple – just write functions that return the UI for the component

Class based components

A bit more complex – but also more powerful (? - opinion)

You will use – and need to learn – both

```
const Counter = props => {
    const [counter, setCounter] = useState(0)
    return (
        <div>
            <h1>A counter component</h1>
        </div>
    )
};
```

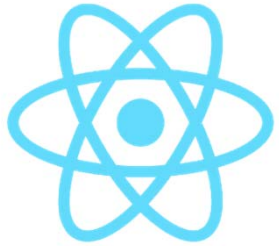
Function component

Class component

```
class Counter extends React.Component {
    state = {
        counter: 0
    };

    render() {
        return (
            <div>
                <h1>A counter component</h1>
            </div>
        )
    }
};
```

Compiling a class component



JSX is compiled to JavaScript

(here: the class component):

```
class Counter extends React.Component {
    state = {
        counter: 0
    };

    render() {
        return (
            React.createElement('div', null,
                React.createElement('h1', null, 'A counter component')
            )
        )
    }
};
```

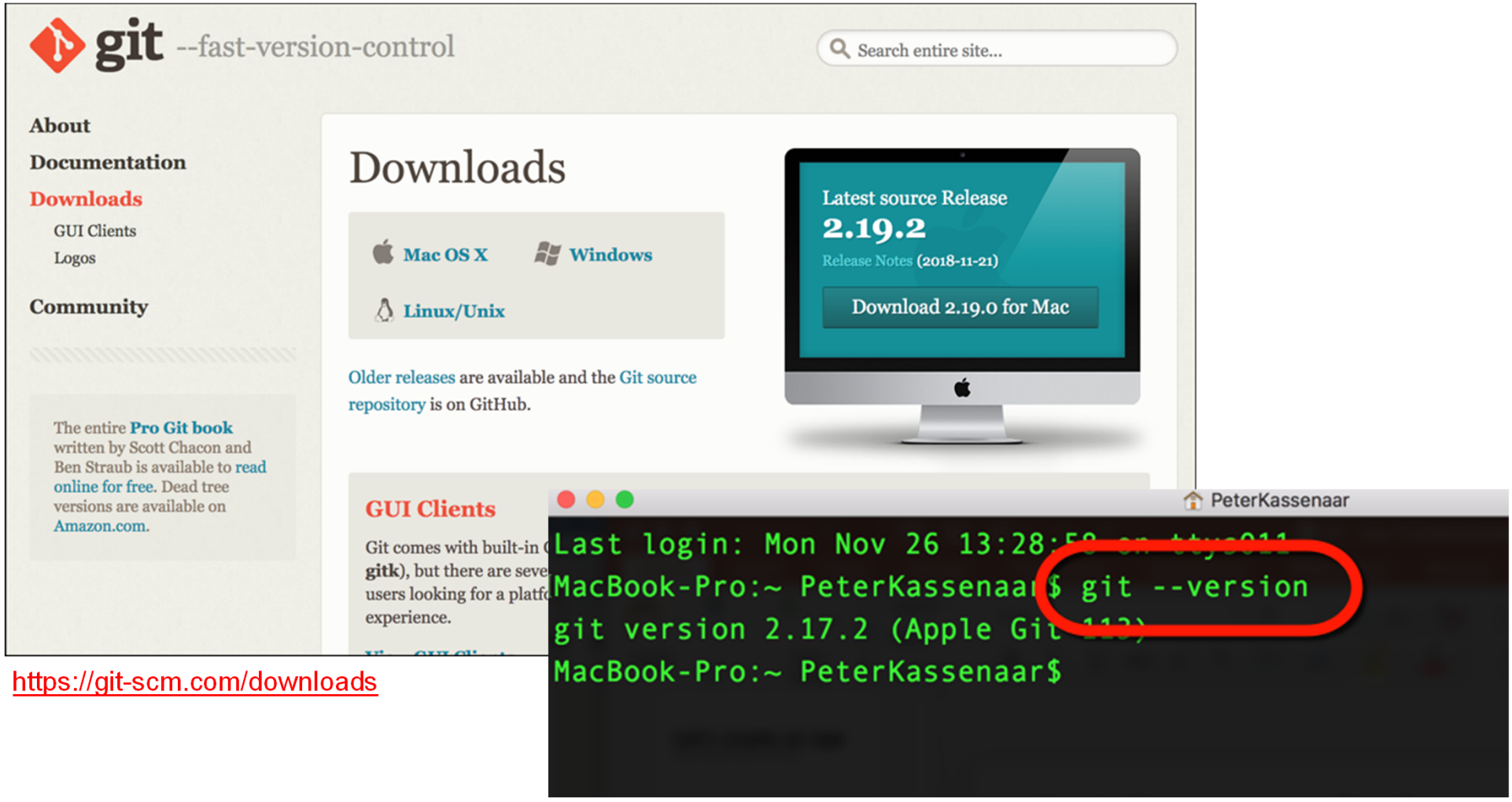
Questions?

Let’s create an app

Install tooling and create your first app

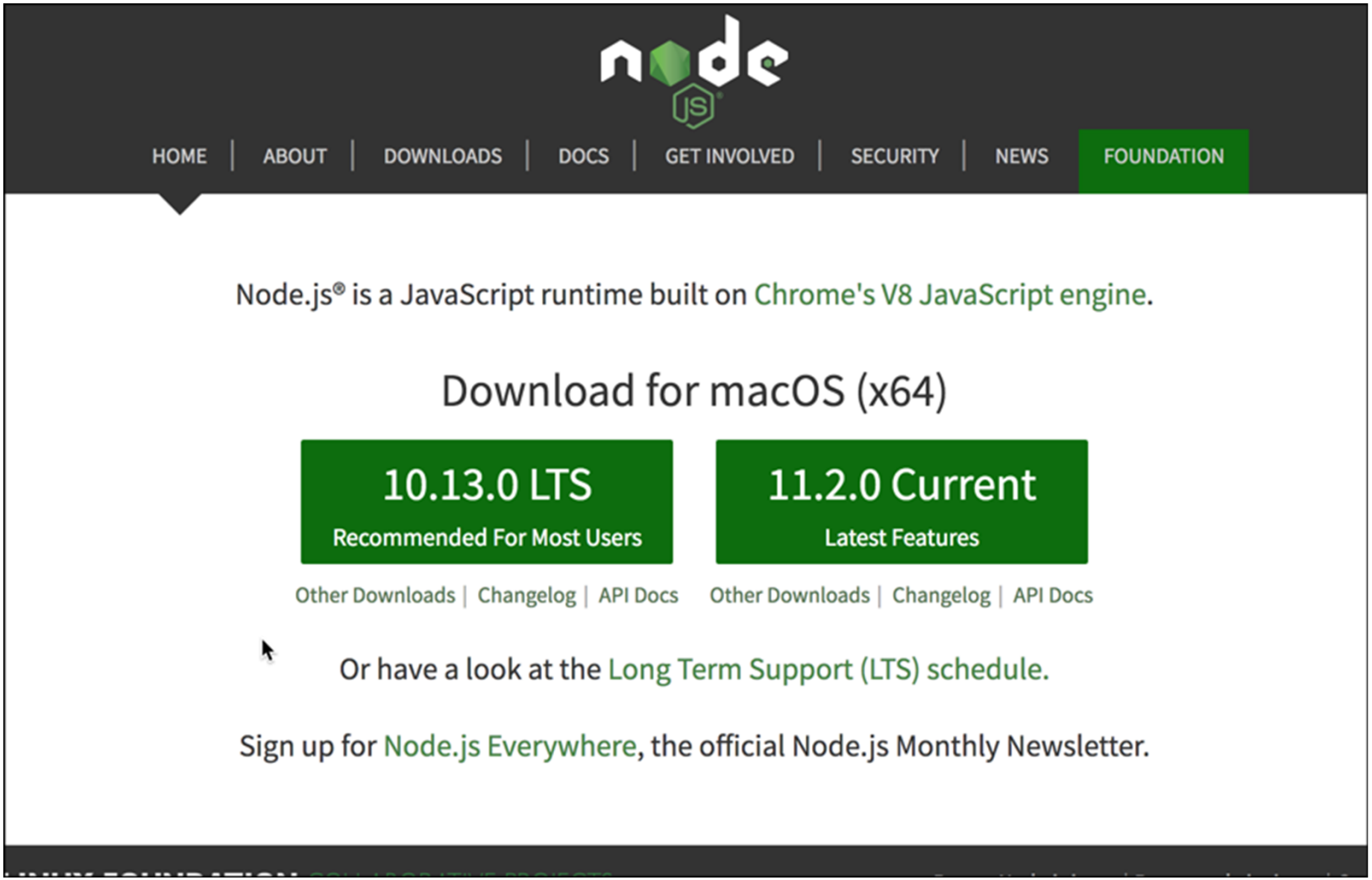
Install necessary Tooling

First – install Git, if you don't have it yet



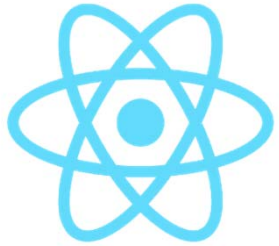
https://git-scm.com/downloads

NodeJS



https://nodejs.org/en/

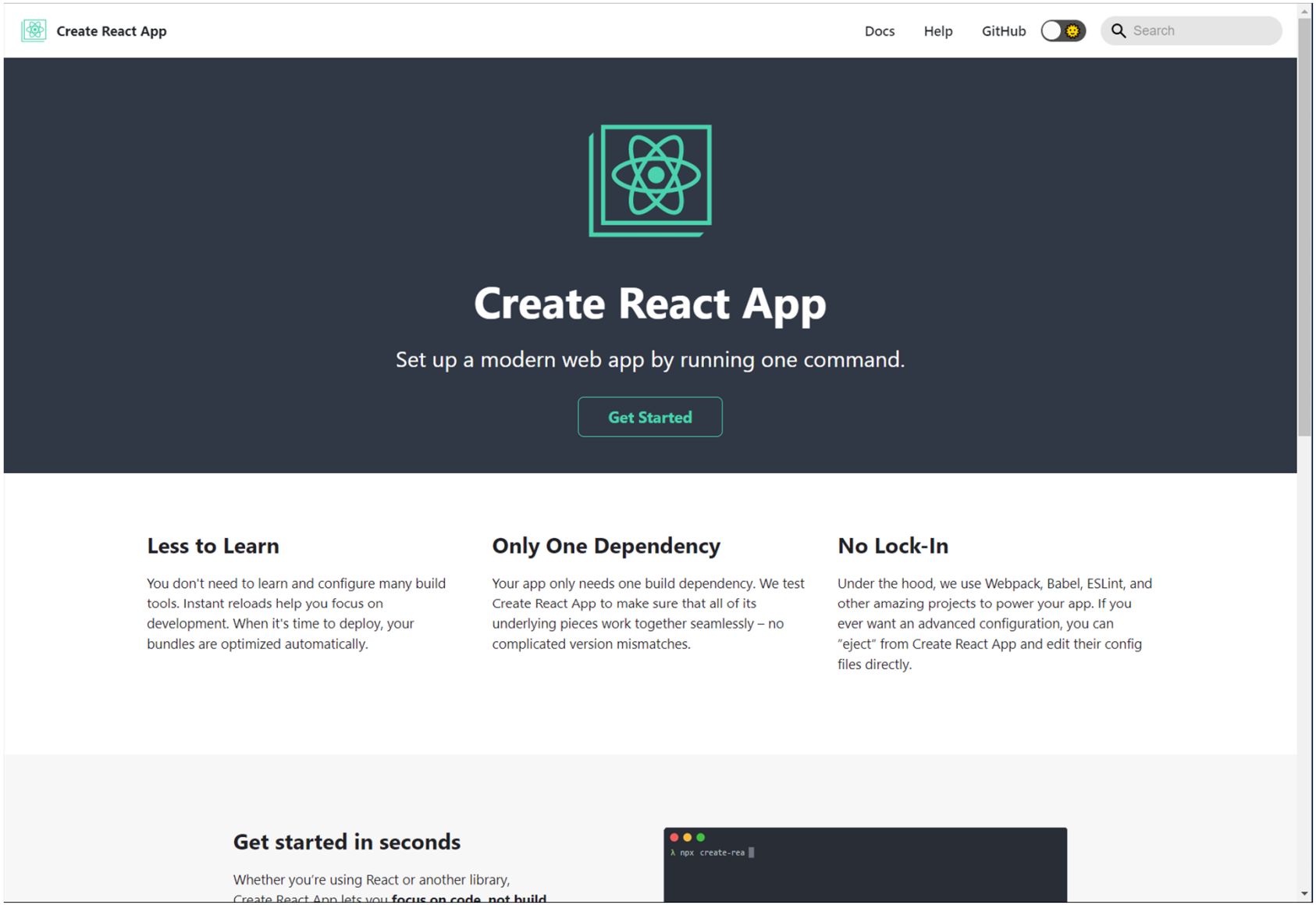
create-react-app



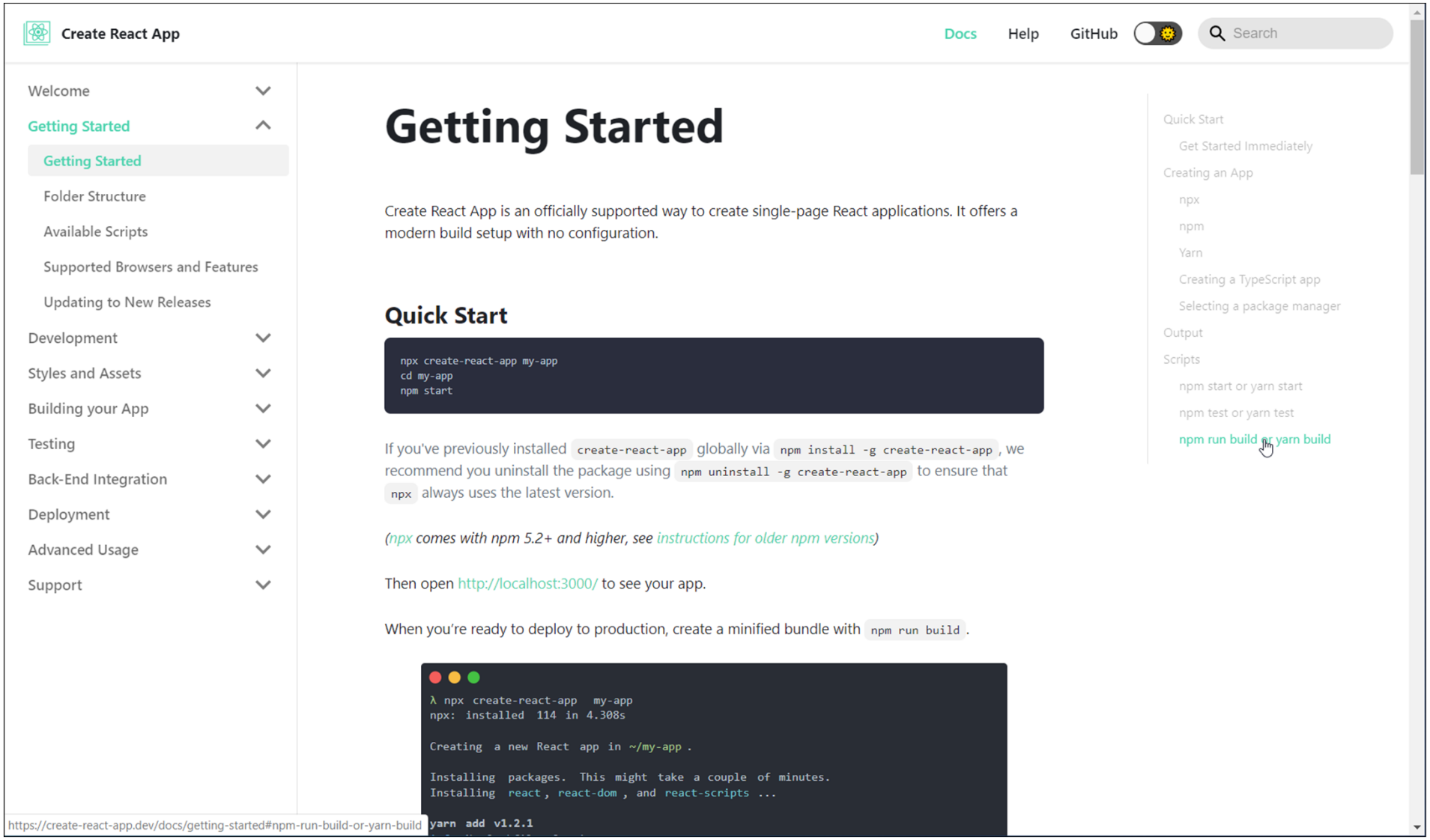
- Create React App (or ‘CRA’ for short)
  - Scaffold new projects
  - Set up JSX compilation by Babel
  - Set up Webpack bundling
  - Set up Linting
  - …
- All this is done by CRA
- https://create-react-app.dev/

```
npx create-react-app my-app
```

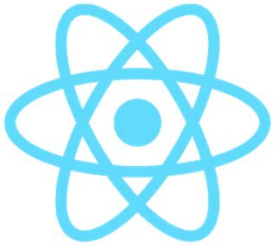
create-react-app.dev



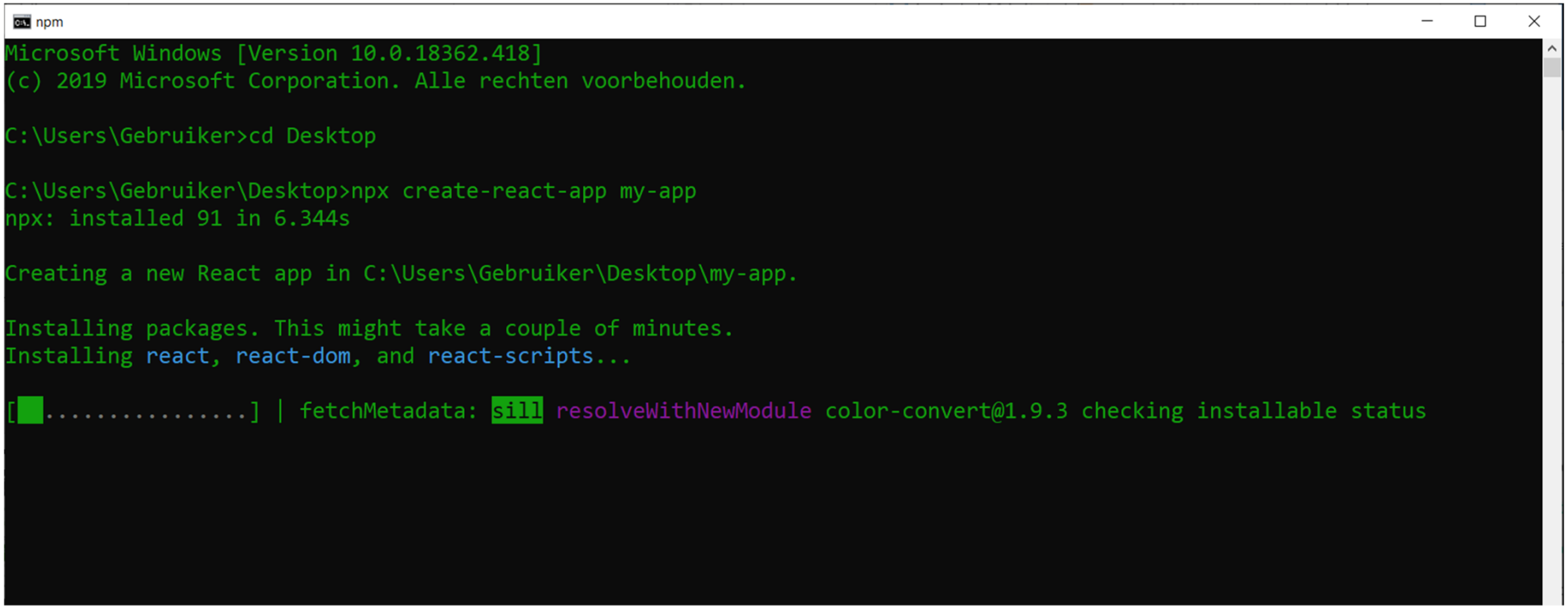
Always check the documentation



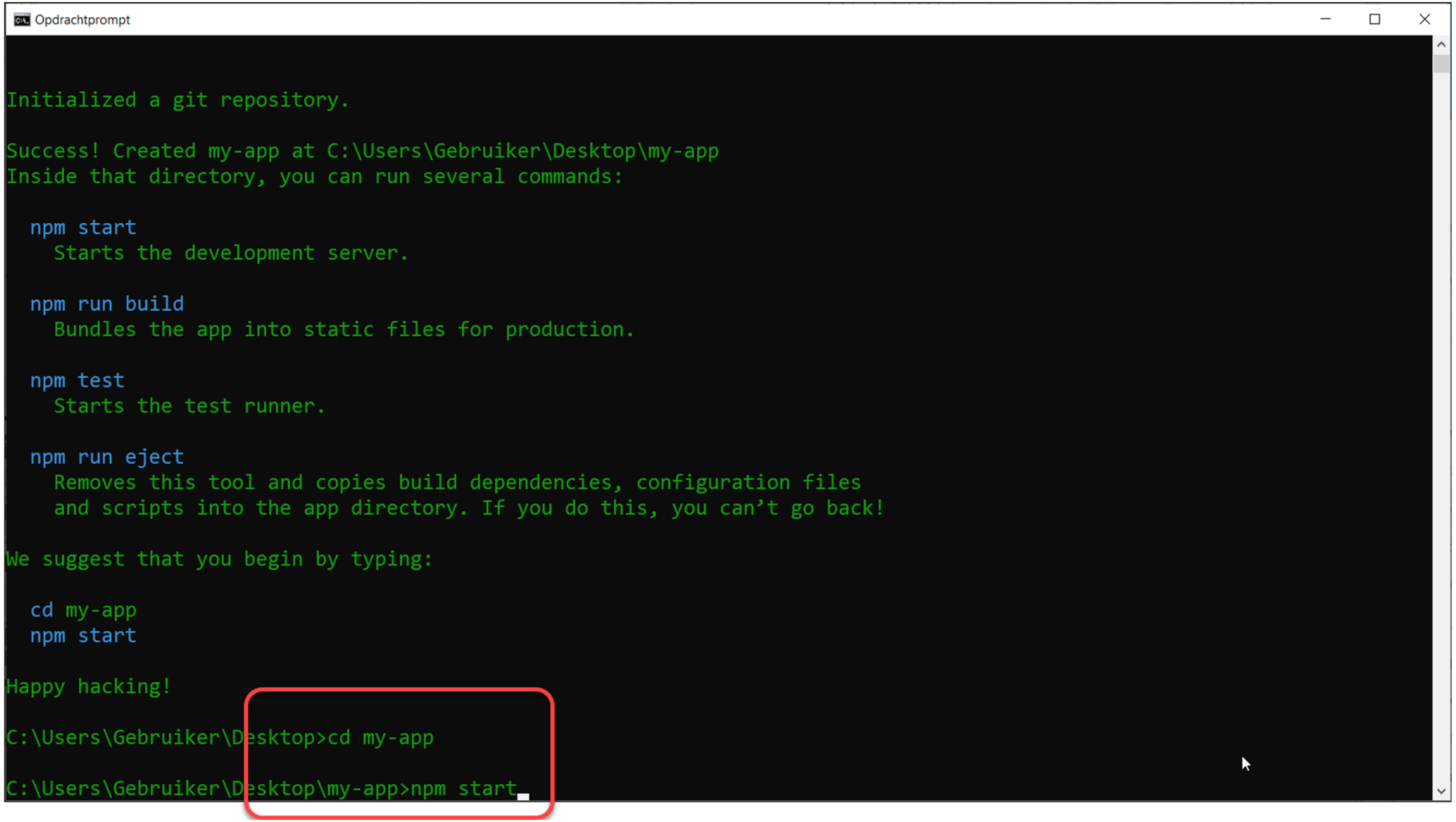
CRA



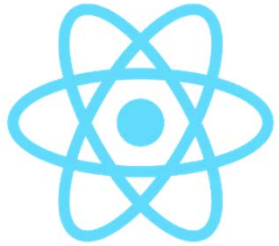
- Use npx to run it. You *can* install CRA locally, but this is not recommended
- You don’t need to set up webpack and babel yourself



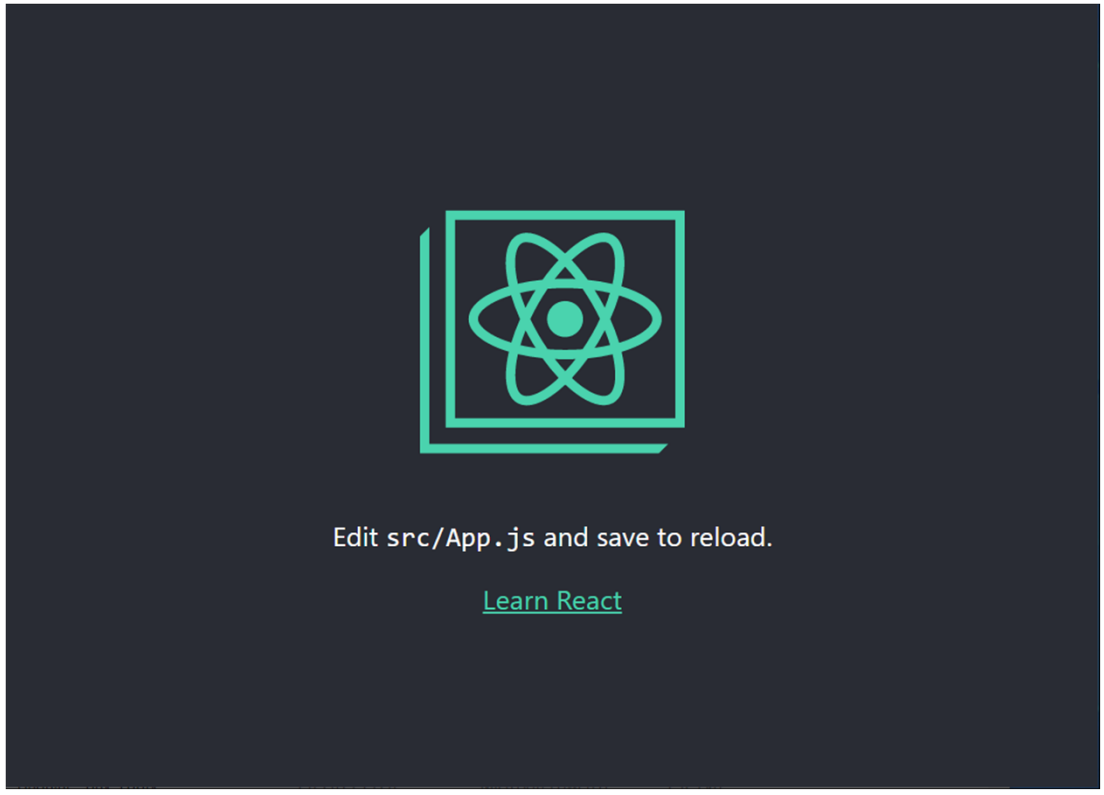
Creating a default application



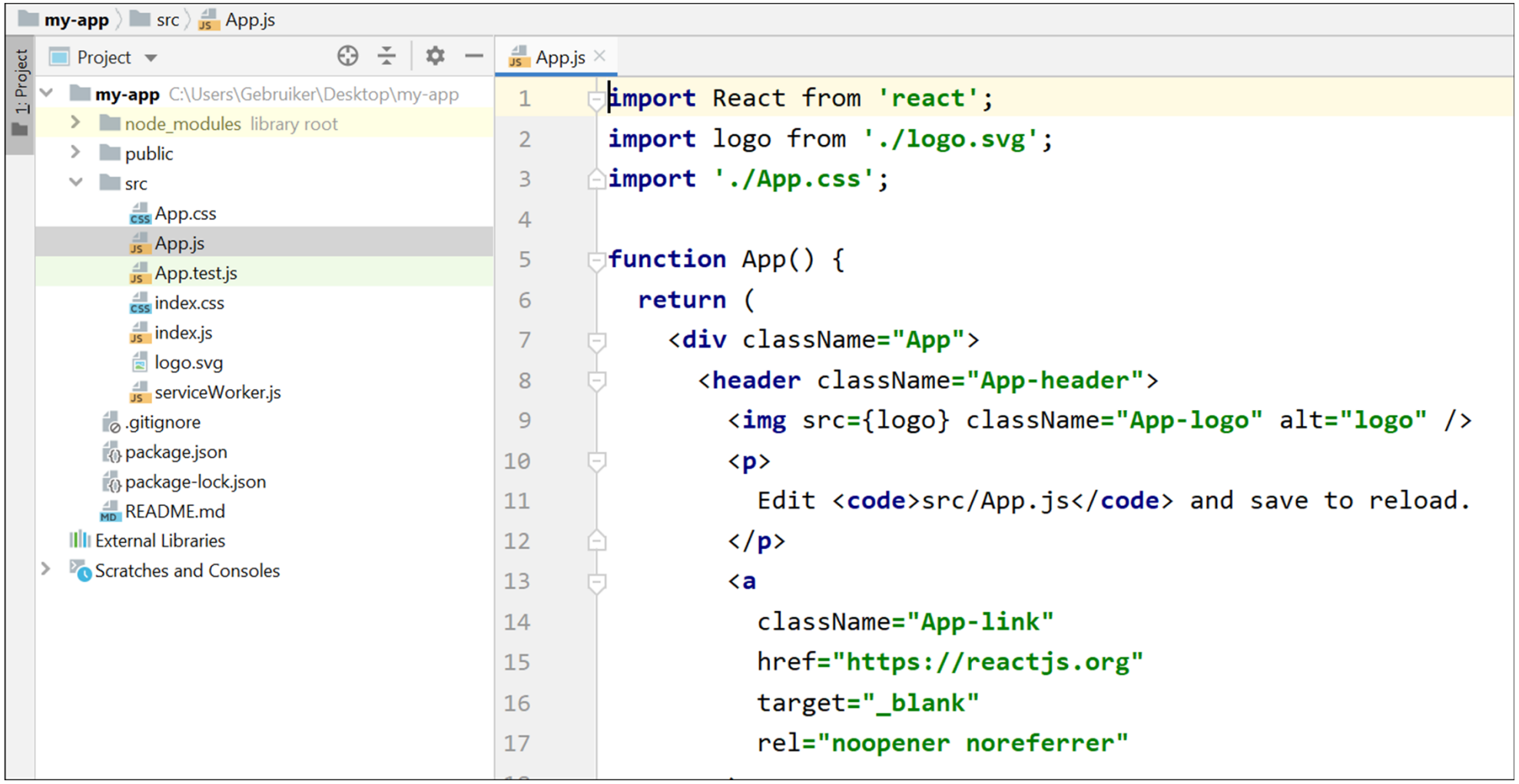
Running the default project



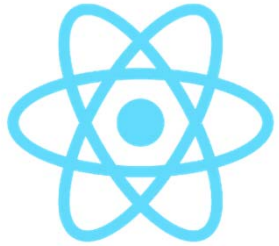
- `npm start`
- http://localhost:3000



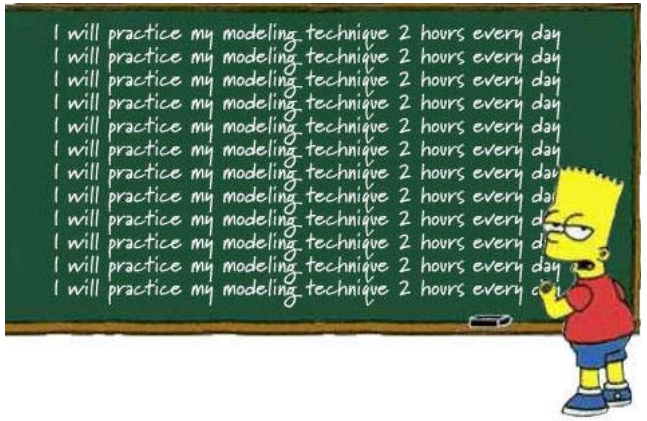
File structure



Workshop



- Create a default project `npx create-react-app my-app`
  - https://create-react-app.dev/docs/getting-started (for background info)
- Create a new, blank React project
- Run it
  - `npm start`
- Replace some content in `src/App.js` component with your own code.
- Code example: ../100-helloworld



Let’s look at the code

A look inside the key files in the generated code

#1. Package.json

package.json

```
{
  "name": "hello-react",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "react": "^16.11.0",
    "react-dom": "^16.11.0",
    "react-scripts": "3.2.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
```

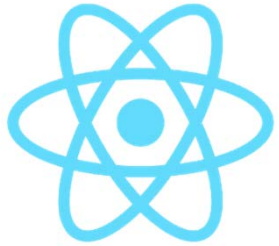
#2 ..src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App />, document.getElementById('root'));

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

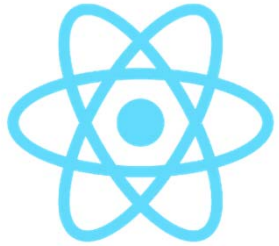
index.js



- Is the root of your application
- Imports starting component
  - Defaults to: <App />
- Adds App to the root element in index.html
- Typically you *never touch* this file again

```
ReactDOM.render(
    <App/>,
        document.getElementById('root')
);
```

#3. App.js



- <App /> is the main component of the app.
  - You typically replace the contents with your own code
  - Main Navigation, routing, other content, footer, etc.

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
```

Import(s)

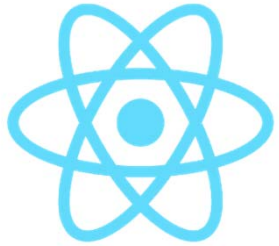
```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        …
      </header>
    </div>
  );
}
```

Component (`function` or `class`)

```
export default App;
```

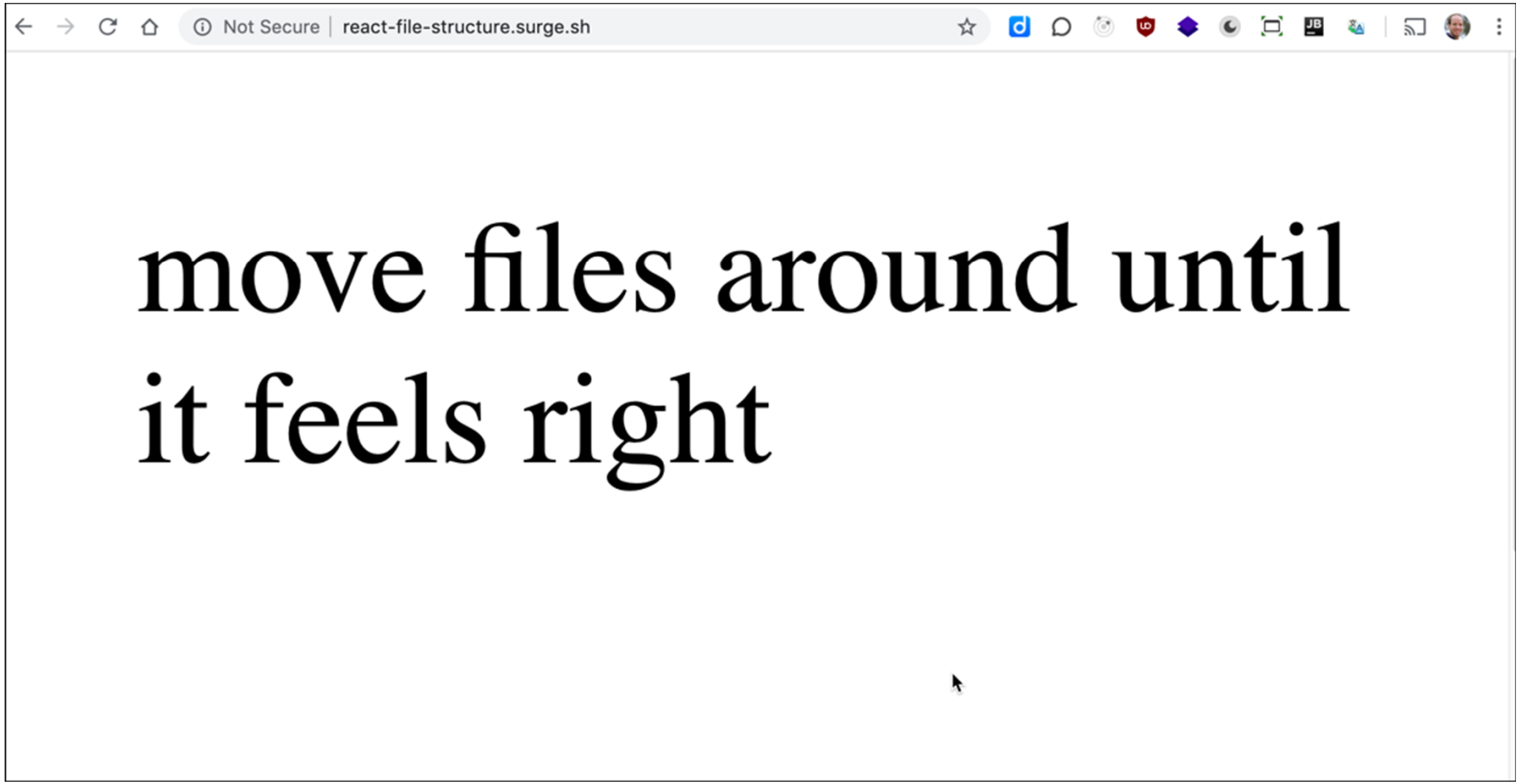
Export

#4. Other files



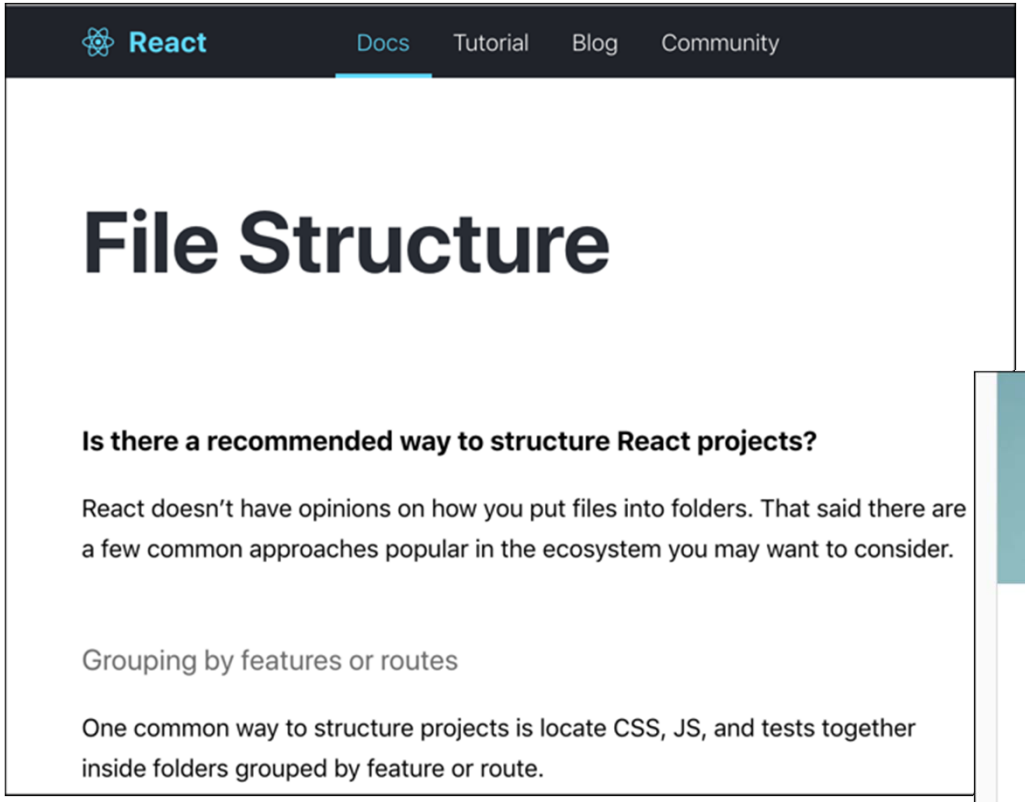
- `*.css` – various stylesheets.
- `*.test.js` – various unit testing files.
- `serviceWorker.js` – to create a PWA. You may remove it.
- Folder `/public` – all files that you want to publish

How to structure your React project?

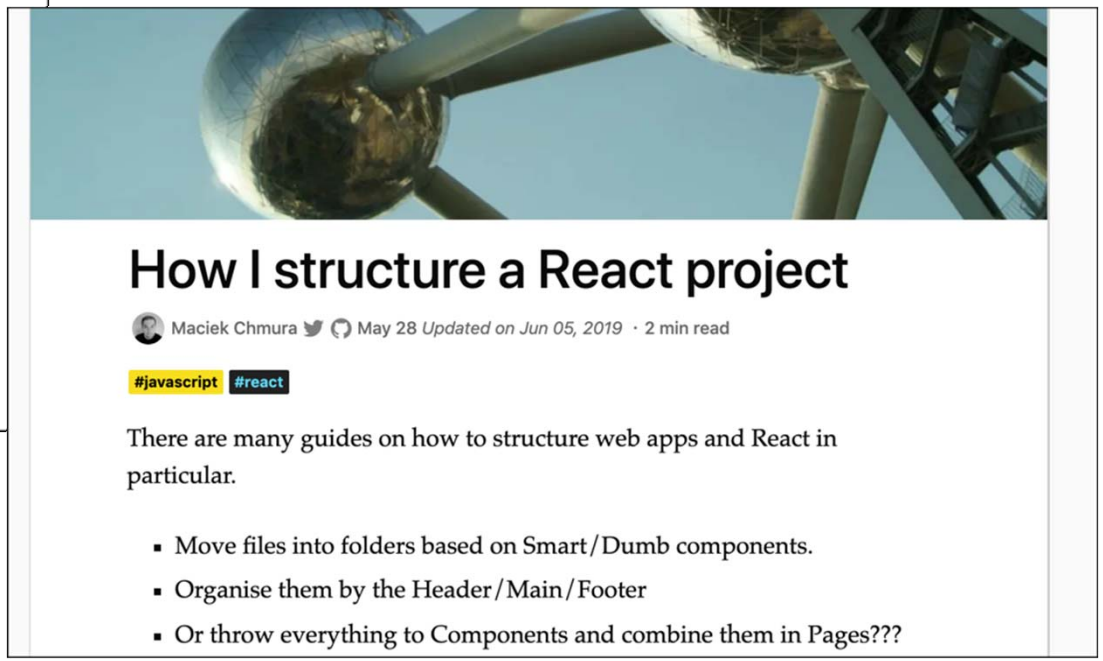


Dan Abramov: http://react-file-structure.surge.sh/

Some other opinions

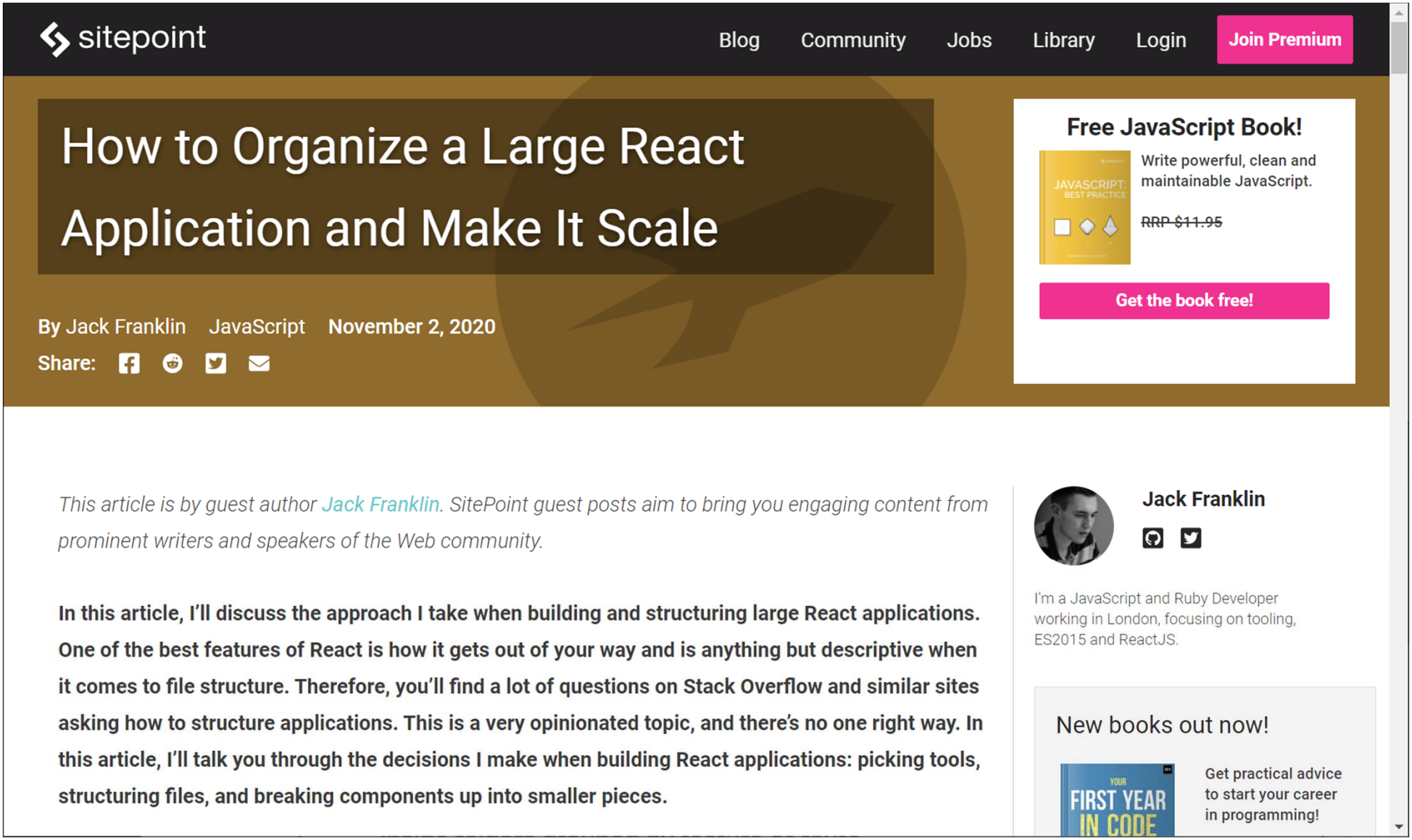


reactjs.org/docs/faq-structure.html



https://dev.to/maciekchmura/how-i-structure-a-react-project-3c2i

Advanced tips on structuring larger applications



https://www.sitepoint.com/organize-large-react-application/

Workshop

- Create a new, blank React project (or go ahead with your previous project
- Identify and study the files discussed in this section
  - Run the project: `npm start`
- Replace some content in App component with your own code.
- Add some images.
- Create your own (new) component. Load it instead of <App />.
- Move component to a new `/components` folder. Make sure project still runs.
- Code example: ../100-helloworld