

INTRODUCTION TO FRONT-END DEVELOPMENT

accenture[>]technology

ABOUT US



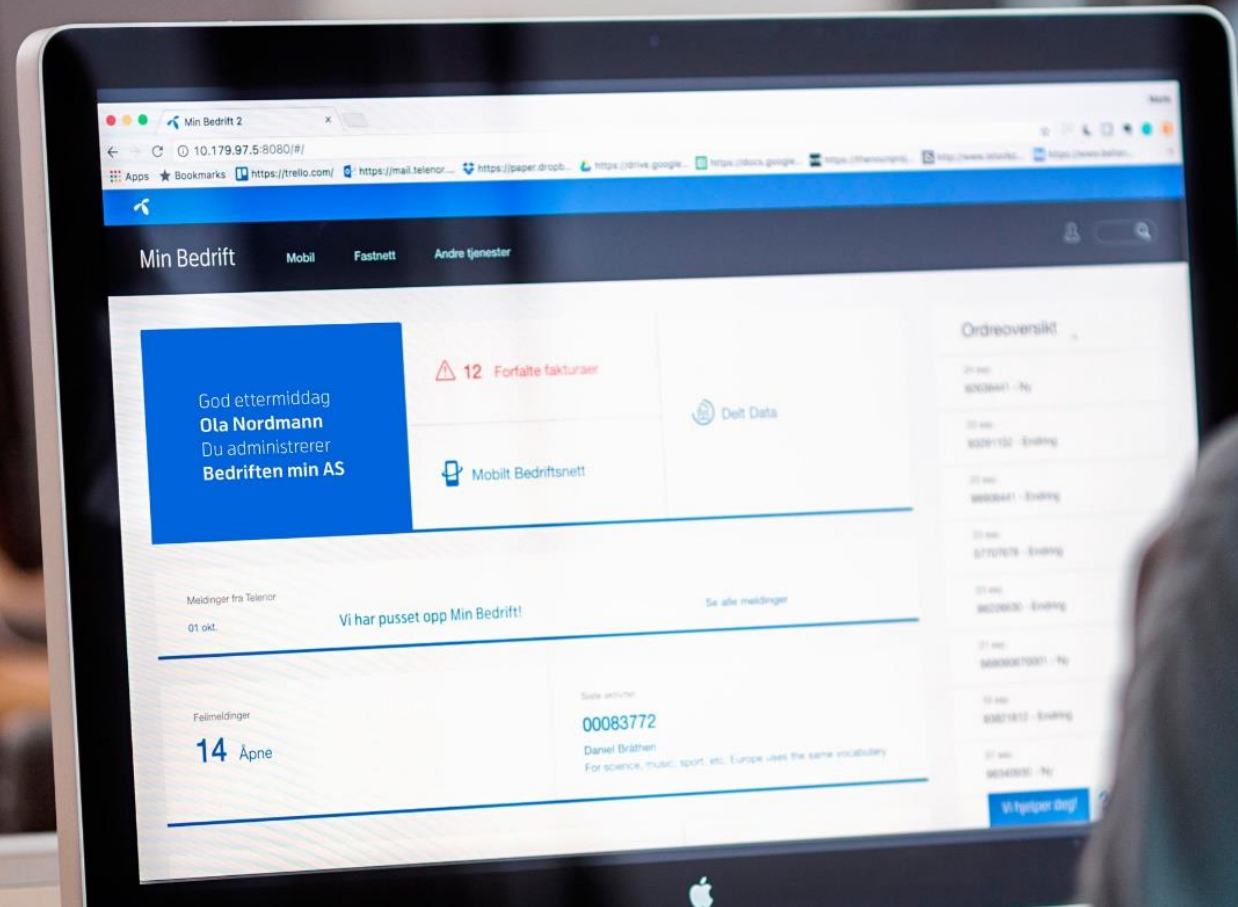
Lars Henrik Nordli

- Worked in Accenture since August 2016
- Front-end Developer
- ACIT



Ekaterina Orlova

- Worked in Accenture since January 2015
- Front-end Developer
- Telenor



A self-service solution for corporate clients for Telenor Norway

ACIT CLOUD STUDIO

ACCENTURE CENTER FOR IBM TECHNOLOGIES (ACIT) IS A MULTI-DISCIPLINARY TEAM THAT MAKES PROTOTYPES FOR CLIENTS IN 4-8 WEEKS.

WE SHOW CLIENTS WHAT IS POSSIBLE WITH NEW TECHNOLOGY



**MULTI-DISCIPLINARY
TEAM**



**USER-CENTERED
DESIGN**



**DELIVERY
INNOVATION**



IBM Bluemix

BUILT ON EMERGING IBM TECHNOLOGIES



IBM Watson



FRONT-END DEVELOPMENT

HTML



CSS



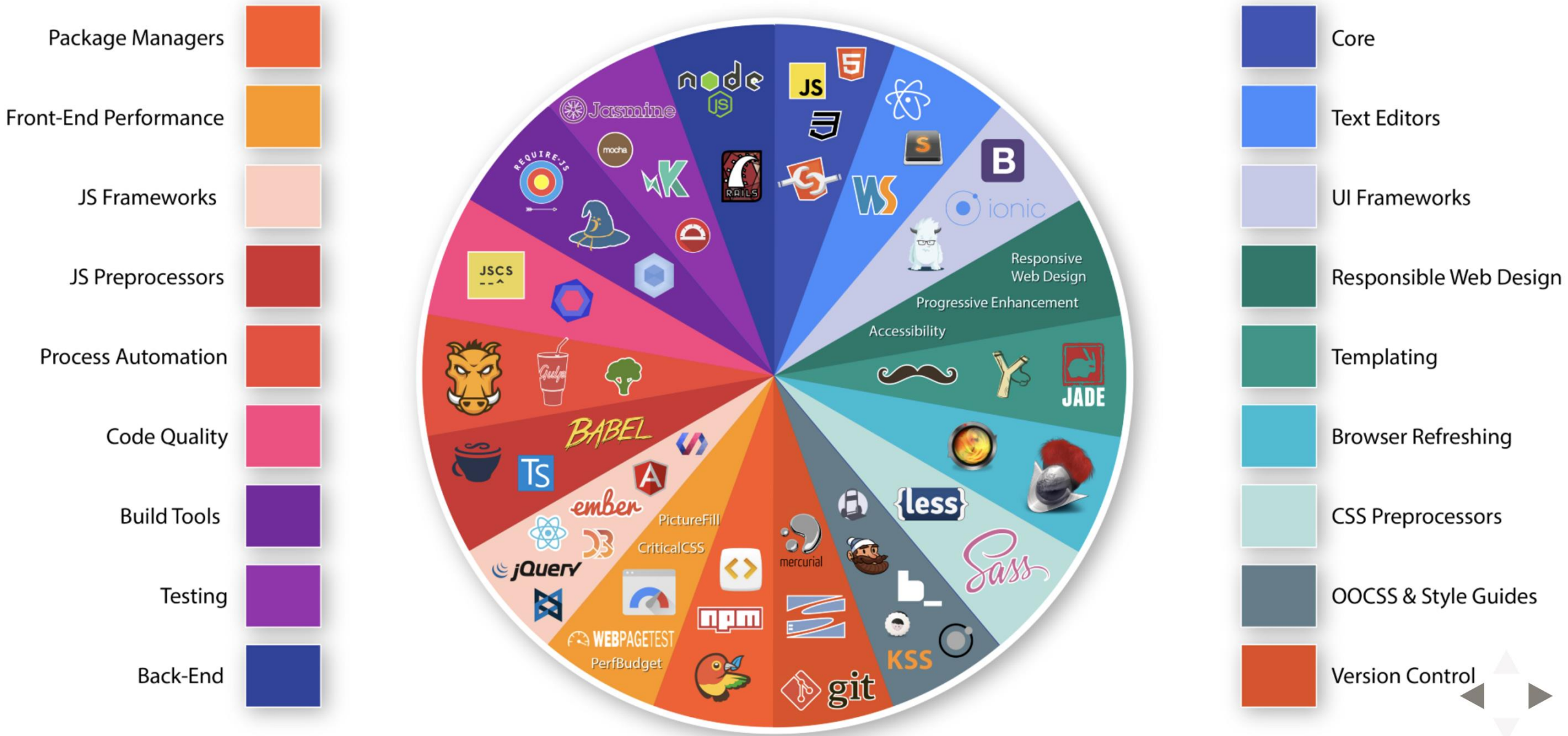
JS

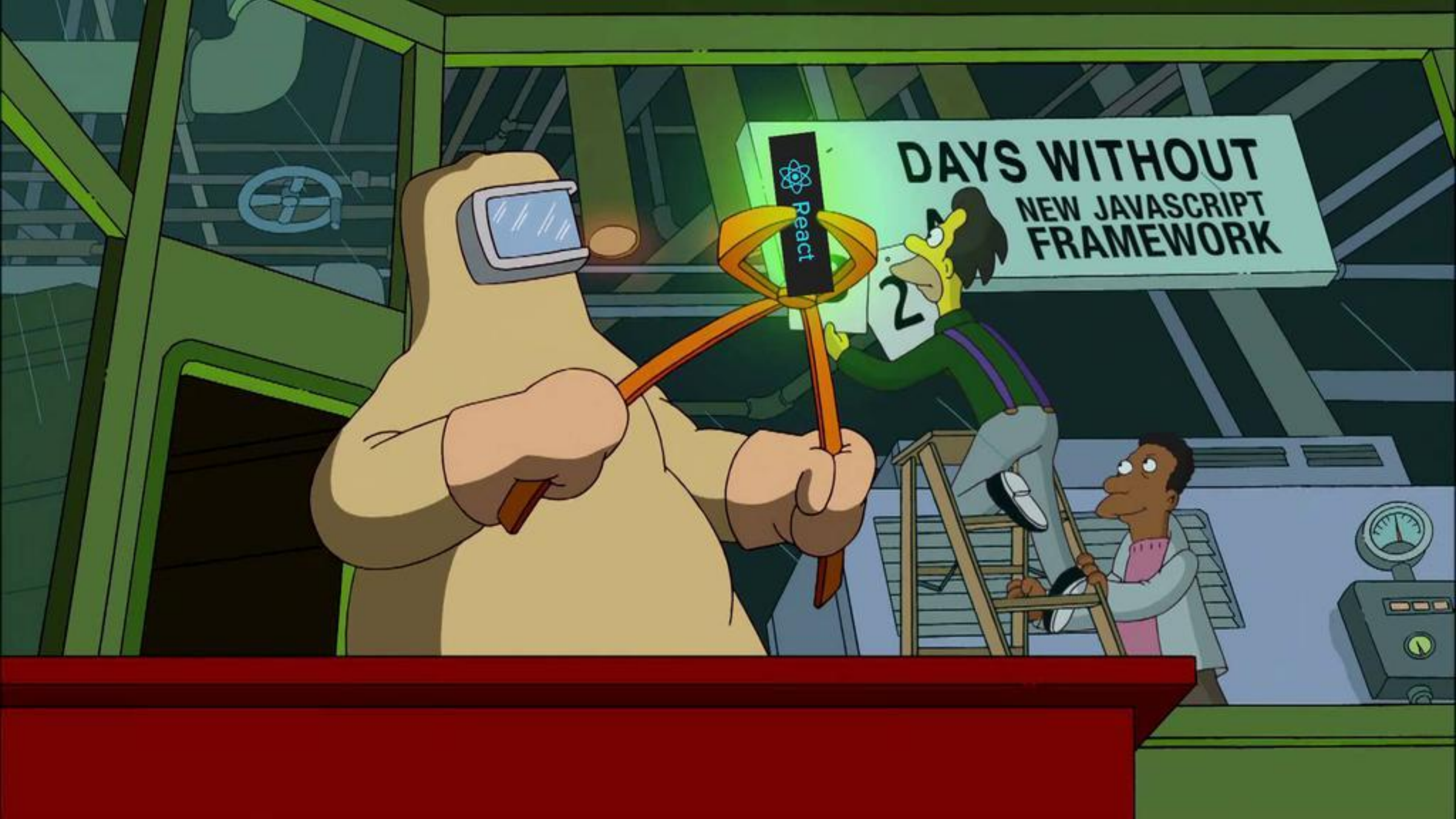


THE WEB



THE FRONT-END SPECTRUM






React

**DAYS WITHOUT
NEW JAVASCRIPT
FRAMEWORK**

2



Jose Aguinaga [Follow](#)

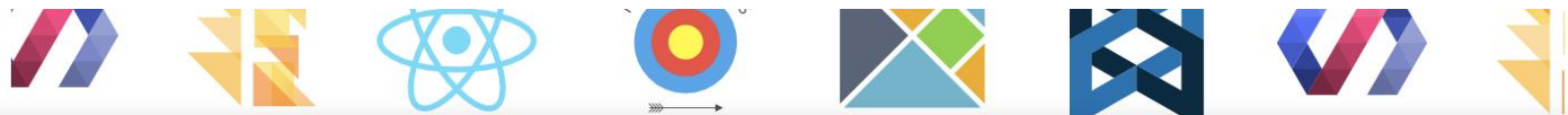
Web Engineer. Previously @numbrs, @plaidhq, currently @getflynt. Javascript, #people, startups, fintech, privacy, blockchain.

Oct 3, 2016 · 13 min read

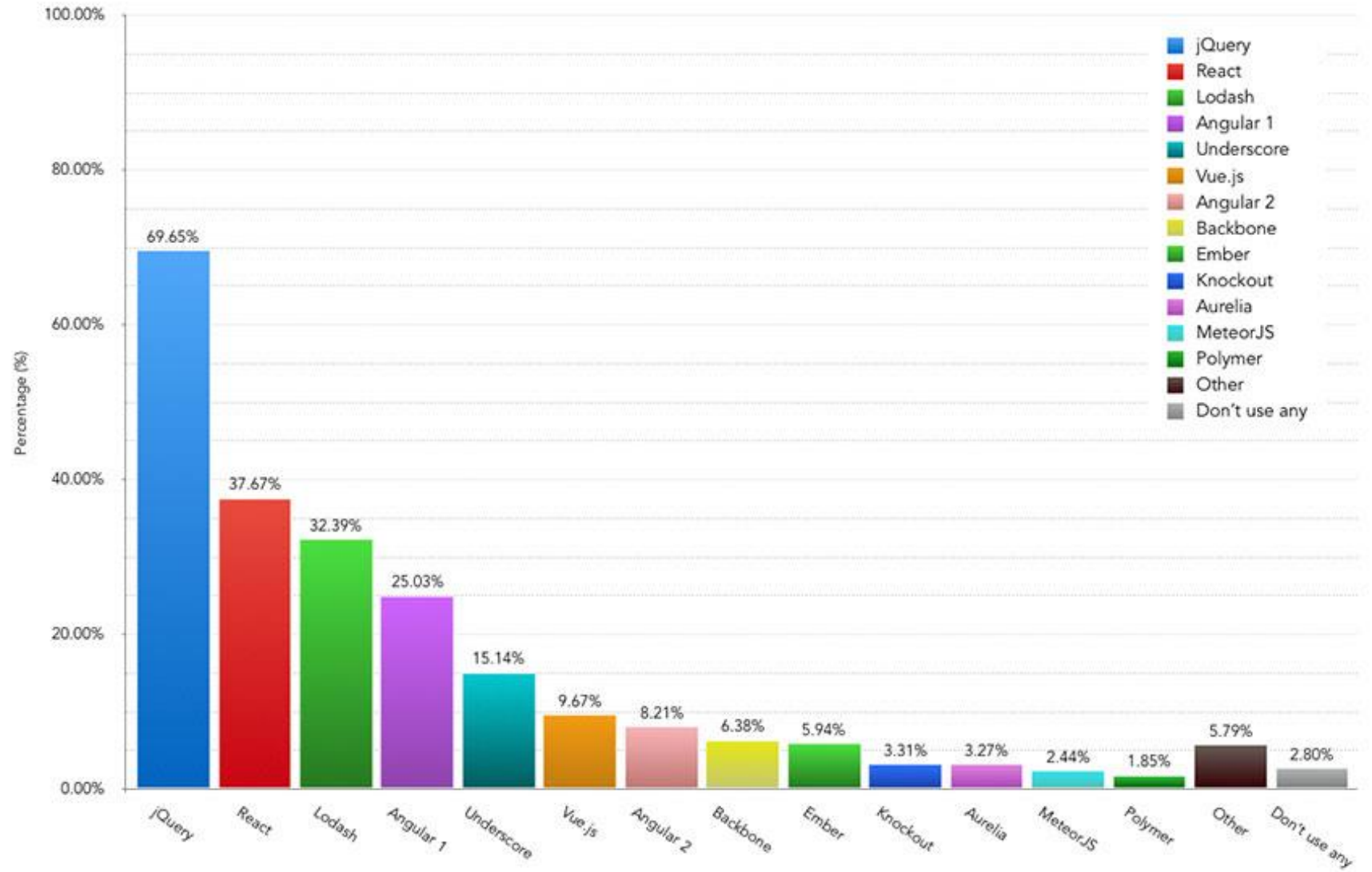
How it feels to learn JavaScript in 2016



<https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f>



Which JavaScript libraries and/or frameworks do you currently use most frequently on projects?



<https://www.webdesignerdepot.com/2017/04/the-state-of-front-end-tooling/>



ANGULAR

ANGULAR INTRODUCTION

ANGULAR IS A PLATFORM FOR BUILDING WEB APPLICATIONS

Created by Google

HTML + TYPESCRIPT

Typed superset of JavaScript – type safety and tooling

MODULAR STRUCTURE AND COMPONENT BASED ARCHITECTURE



ANGULAR CLI

```
npm install -g @angular/cli
```

```
ng new my-app
```

```
cd my-app  
ng serve --open
```

Welcome to app!!



ng generate

class
component
directive
module
pipe
++

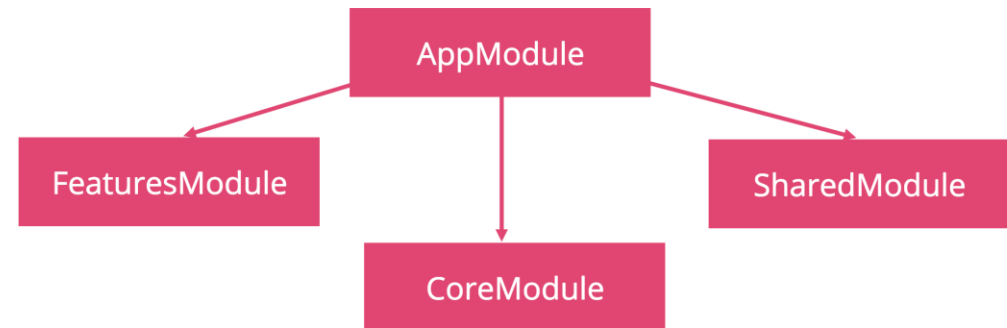
ANGULAR MODULES

AppModule

```
import { NgModule } from '@angular/core';
```

```
@NgModule({  
  imports: [],  
  declarations: [],  
  providers: [],  
  exports: [],  
  bootstrap: []  
})  
class AppModule { }
```

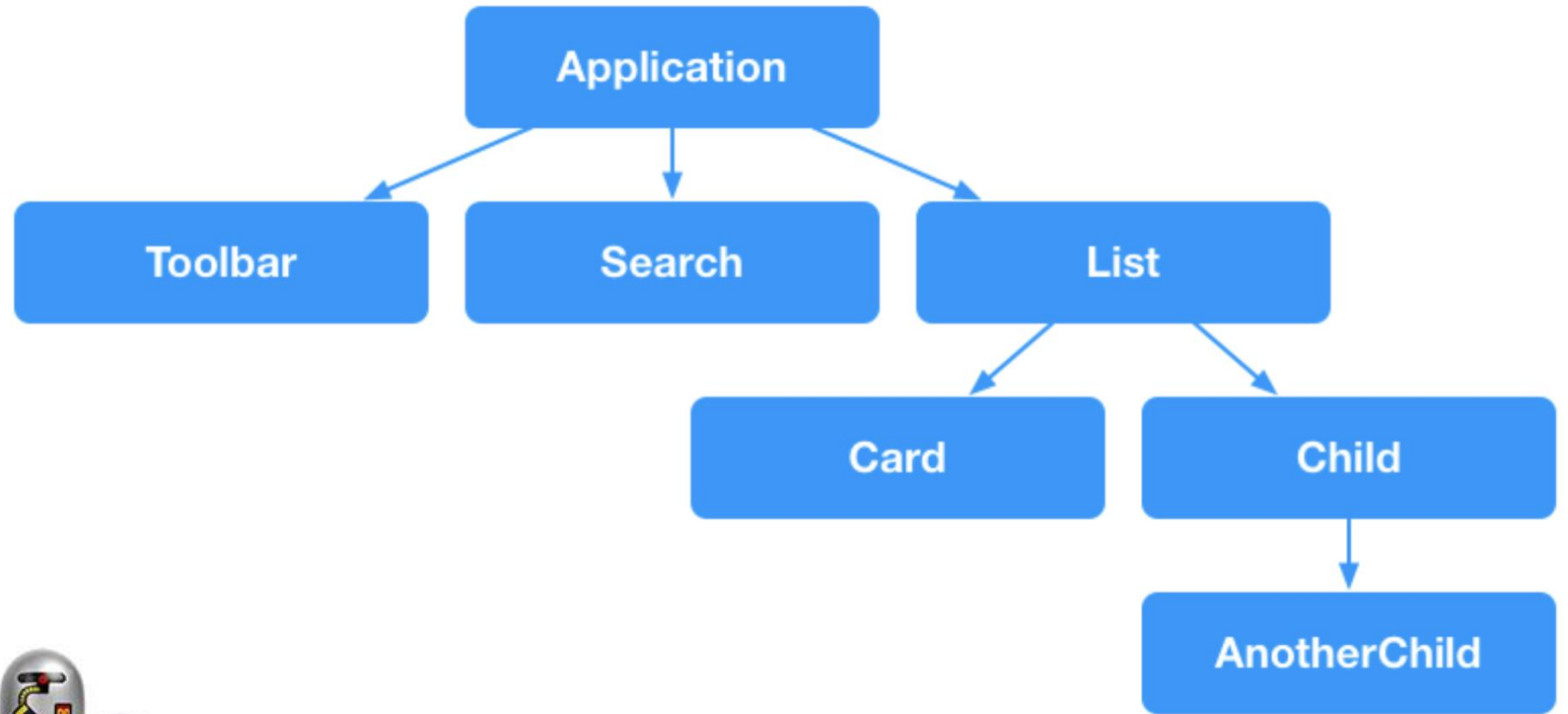
TREE OF MODULES



ANGULAR COMPONENTS

**A MODULE IS A TREE WITH
COMPONENTS**

**THESE CAN HAVE THEIR OWN
CHILD COMPONENTS**



ANGULAR COMPONENTS

A **COMPONENT**
IS IN CHARGE OF ONE PART OF
THE APPLICATION
– A “**VIEW**”

A COMPONENT IS A CLASS WITH
STYLES AND TEMPLATE
CONNECTED TO IT

YOU CAN DEFINE THE TEMPLATE
INLINE OR AS AN EXTERNAL
TEMPLATEURL

ANGULAR AUTOMATICALLY
EXTRACTS VALUES FROM
COMPONENT PROPERTIES AND
UPDATES THEM IN THE BROWSER
IF THEY CHANGE

```
<movie-component></movie-component>
```

```
import { Component } from '@angular/core';

@Component({
  selector: 'movie-component',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite movie is: {{myMovie}}</h2>
  `
})

export class MovieComponent {
  title = 'Movie page';
  myMovie = 'Star wars';
}
```

ANGULAR COMPONENTS

APPLICATION LOGIC IS DEFINED
IN A **CLASS** THAT INTERACTS
WITH THE VIEW THROUGH
PROPERTIES AND **METHODS**

```
export class BookListComponent implements OnInit {  
  
  books: Book[];  
  selectedBook: Book;  
  
  constructor(private service: BookService) {}  
  
  ngOnInit() {  
    this.books = this.service.getBooks();  
  }  
  
  selectBook(book: Book) {  
    this.selectedBook = book;  
  }  
}
```

ANGULAR COMPONENTS

ngOnChanges()	Respond when Angular (re)sets data-bound input properties.
ngOnInit()	Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties. Called <i>once</i> , after the <i>first</i> ngOnChanges().
ngDoCheck()	Detect and act upon changes that Angular can't or won't detect on its own. Called during every change detection run, immediately after ngOnChanges() and ngOnInit().
ngAfterContentInit()	Respond after Angular projects external content into the component's view. Called <i>once</i> after the first ngDoCheck().
ngAfterContentChecked()	Respond after Angular checks the content projected into the component. Called after the ngAfterContentInit() and every subsequent ngDoCheck().
ngAfterViewInit()	Respond after Angular initializes the component's views and child views. Called <i>once</i> after the first ngAfterContentChecked().
ngAfterViewChecked()	Respond after Angular checks the component's views and child views. Called after the ngAfterViewInit and every subsequent ngAfterContentChecked().
ngOnDestroy	Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks. Called <i>just before</i> Angular destroys the directive/component.

SCOPED STYLES

ANGULAR APPLICATIONS IS STYLED WITH TRADITIONAL CSS

- Angular can encapsulate styles for components → a more modular design
- Class names and selectors that are defined for a component does not interfere with other selectors
- Changes made in a component does not interfere with other components

```
@Component({
  selector: 'hero-details',
  template: `
    <h2>{{hero.name}}</h2>
    <hero-team [hero]=hero></hero-team>
    <ng-content></ng-content>
  `,
  styleUrls: ['app/hero-details.component.css']
})
export class HeroDetailsComponent {
  /* ... */
}
```

ANGULAR TEMPLATES: HTML +

INTERPOLATION : {{myValue}}

```
<h3>
    {{title}}
    
</h3>
```

Text in between {{ and }} is called a **template expression** – usually, this is a property of the class that should be evaluated by Angular. Evaluating these expressions should happen without side-effects.

A **template statement** can act on **events** that components emit or broadcast.

```
<button (click)="deleteHero()">
Delete hero
</button>
```

DATA BINDING

Data direction	Syntax	Type
One-way from data source to view	<pre> <hero-detail [hero]="currentHero"> </hero-detail> <div [ngClass]="{'special': isSpecial}"> </div> {{myValue}}</pre>	Interpolation Property Attribute Class Style
One-way from view to data source	<pre><button (click)="onSave()"> Save</button> <hero-detail (deleteRequest)="deleteHero()"> </hero-detail> <div (myClick)="clicked=\$event" clickable>click me</div></pre>	Event
Two-way	<pre><input [(ngModel)]="name"></pre>	Two-way

TWO WAY BINDING

```
<input name="you" [(ngModel)]="you">
```

```
<!-- is equivalent to this -->
```

```
<input name="you" [ngModel]="you"  
      (ngModelChange)="you = $event" >
```

PIPES: DATA TRANSFORMATION

Pipes converts data input to a specified output.

Pipes can also receive input parameters

It is also possible to combine (daisy chain) multiple pipes. Remember that order matters.

```
<p>Today is {{ day | date }}</p>
```

```
<!-- October 18, 2017 -->
```

```
<p>Today is {{ day | date:"dd.MM.yy" }} </p>
```

```
<!-- 18.10.17 -->
```

```
{{ day | date: 'fullDate' | uppercase}}
```

```
<!-- WEDNESDAY, OCTOBER 18, 2017 -->
```


SERVICES

Services are used to calculate or output values or functions that is needed throughout the application.

A class with a concrete and well-defined task.
Services removes implementation details from components = better readability

Import:

```
import { HeroService } from './hero.service';
```

Define a private property in the constructor:

```
constructor(private heroService: HeroService) { }
```

A service should be @Injectable so that it can be used in and injected into a component. The component will create a new instance of the service.

```
@Injectable()  
export class HeroService {  
    getHeroes(): void {} // stub  
}
```

```
providers: [HeroService]
```

ROUTES

Define base URL

```
<base href="/">
```

Define where the components should appear based on the route

```
<router-outlet></router-outlet>  
<!-- Routed views go here -->
```

There are 2 ways to navigate:

```
<a routerLink=["'/quiz', user.id]"></a>
```

```
this.router.navigate([''/quiz', id]);
```

```
import { Routes } from '@angular/router';  
import { RegisterComponent } from './register/register.component';  
import { QuizComponent } from './quiz/quiz.component';  
import { ResultComponent } from './result/result.component';  
import { StartComponent } from './start/start.component';  
  
export const ROUTES: Routes = [  
  {  
    path: '',  
    component: StartComponent  
  },  
  {  
    path: 'register',  
    component: RegisterComponent  
  },  
  {  
    path: 'quiz/:id',  
    component: QuizComponent  
  },  
  {  
    path: 'results/:id',  
    component: ResultComponent  
  },  
];
```

OTHER COOL STUFF

HTTP • ANIMATIONS •
FORMS • MATERIAL
DESIGN • I18N • WEB
WORKER •
RX/OBSERVABLES •
SERVER SIDE RENDERING...

DEMO

ACCENTURE
JAVAZONE 17

QUESTION 3/26

Op
+Op

What does HTTP code 418 stand for?

I am a teapot

Unsupported Media Type

Too Many Requests



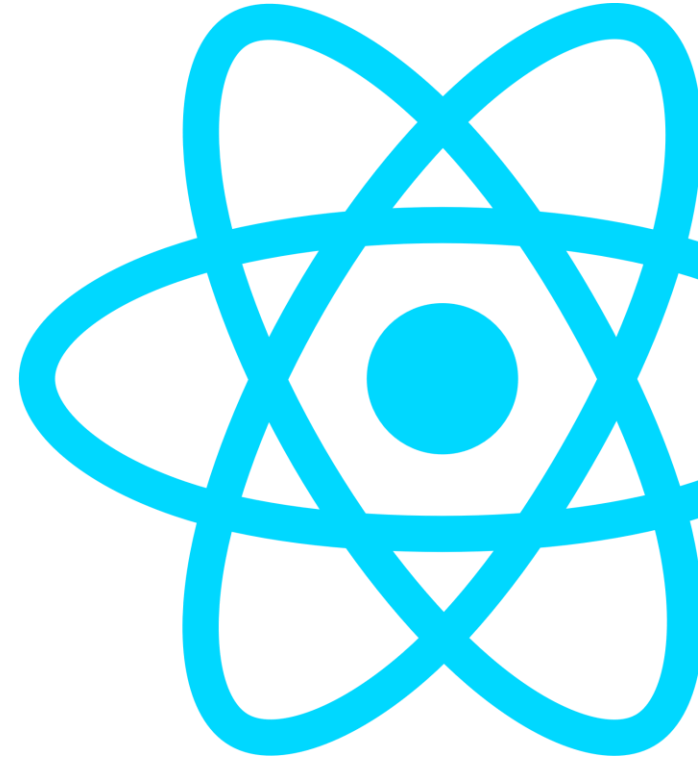
REACT

REACT

INTRODUCTION

React is a JS-library, focused on components and made for user interfaces

- Created by Facebook
 - To better handle micro-tasks and changes on only parts of the web page
- React has quickly become a very popular technology for front-end
- Built for [Yarn](#) Package Manager
 - ...also created by Facebook
- React is usually written with **JSX** (therefore, it needs a transpiler, e.g., Babel)
 - A mix of JS and HTML syntax
 - Babel converts JSX to > JS + HTML so the browser can interpret it

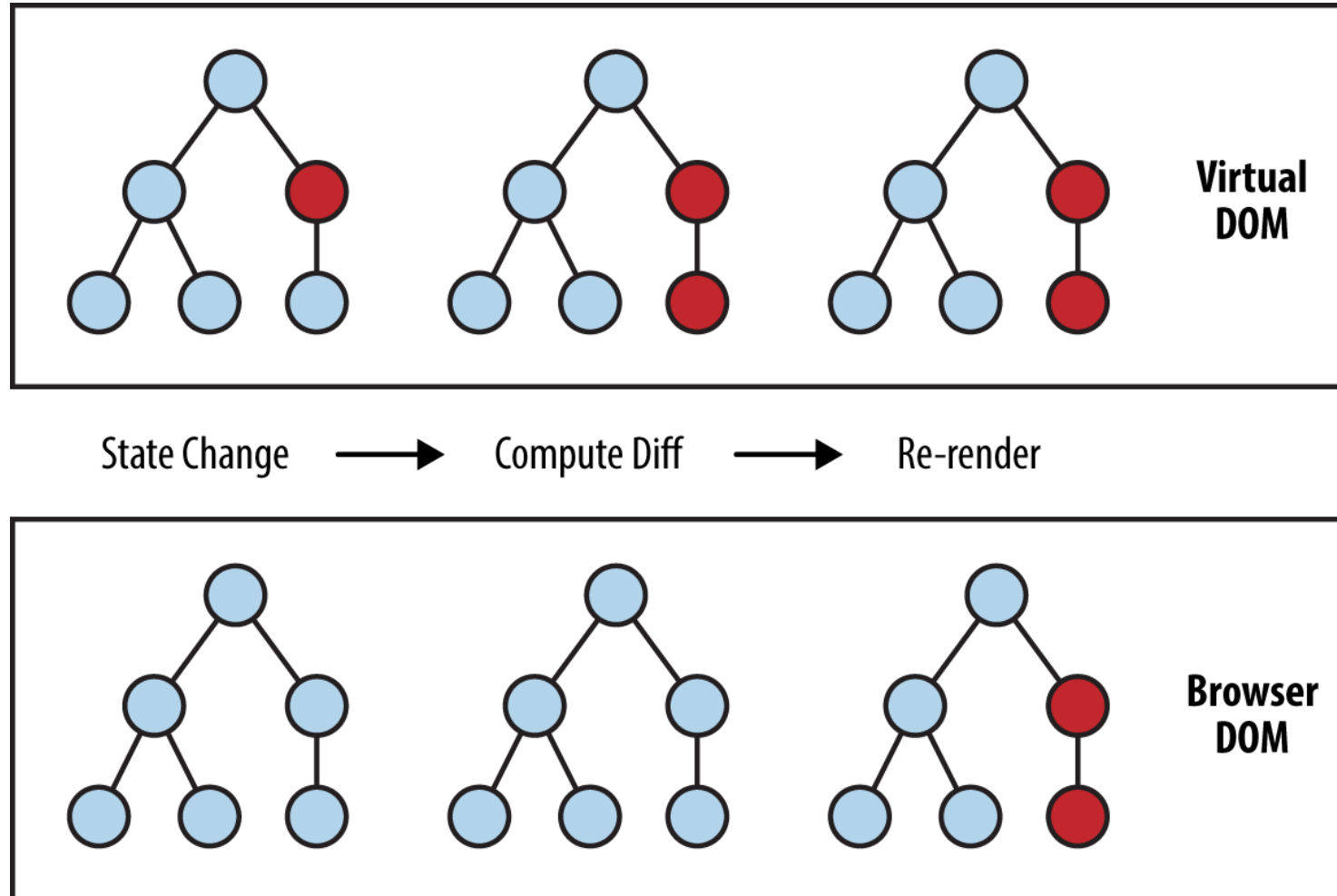


REACT **VIRTUAL DOM**

Virtual DOM makes it easier to manipulate selected and connected parts of a web application

- DOM = Document Object Model
- DOM-manipulation is traditionally quite tedious
 - Programmers are very dependent on events and listeners
- Useful when showing and updating a part of the app dynamically, also to show something based on data in another part of the app
 - Traditional DOM cannot know what happens where, it only renders on page load (except through the use of events and listeners)
- VirtualDOM is layered on top of the DOM and handles changes for you
- `React.renderDOM()`

REACT VIRTUAL DOM



https://www.safaribooksonline.com/library/view/learning-react-native/9781491929049/assets/lrn_0201.png

REACT COMPONENTS

Components are the main building blocks of React

- When using React, the app should be considered as a collection of components (or classes)
 - In other words a “piece”, “element” or “part” of a web app.
- Components can have multiple sub-components
 - Hierarchical, “parent” and “child”
- You can choose how to split the app into components yourself
 - Tradeoff: more components > more difficult to maintain order and cleanliness
 - Avoid tight coupling, make your components as independent as possible
 - You can also consider reuse, for instance buttons, menus, input fields etc.





Studier

Bachelor, master og ph.d.

[Finn studieprogram](#) →

UiO-student? [Se dine studier](#)

→ [Gå til studier](#)

Finn emner

[Søk](#)

Finn pensum, timeplan og eksamen.

[Alle emner](#)

Personer

[Søk](#)

Finn ansatte og studenter. Søk på **navn** eller **fag- og arbeidsområde**.

Tjenester og verktøy

- [Bibliotek](#)
- [Studentweb](#)
- [Fronter](#)
- [Webmail og kalender](#)
- [Mine studier](#)

Forskning



Studier

Bachelor, master og ph.d.

[Finn studieprogram](#) →

UiO-student? [Se dine studier](#)

→ [Gå til studier](#)

Finn emner

Finn pensum, timeplan og eksamen.

[Alle emner](#)

Personer

Finn ansatte og studenter. Søk på **navn** eller **fag- og arbeidsområde**.

Tjenester og verktøy

- [Bibliotek](#)
- [Fronter](#)
- [Mine studier](#)
- [Studentweb](#)
- [Webmail og kalender](#)

Forskning

REACT COMPONENTS

RENDERING AND TAGS

A component is visible in your app through the `render()` function and a custom HTML tag

- Each component must have a `render()` function, which defines what should be “displayed” in your app
- A component is visible through the component’s name in HTML tag format
 - Component: `CustomerList`, JSX: `<CustomerList></CustomerList>`
 - Component: `CustomerListElement`, JSX: `<CustomerListElement/>`
- Tags can either be self-closing or not
 - This depends on whether a component should have sub components or not (sub components must be imported)
 - `<tag />` vs `<tag></tag>`

```
[...]  
<body>  
<main id="app"></main>  
</body>  
[...]
```

```
class App extends React.Component {  
  render() {  
    return(  
      <CustomerList />  
    )  
  }  
}
```

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

```
import Customer from 'Customer'
```

```
class CustomerList extends React.Component {  
  render() {  
    return(  
      <Customer />  
    )  
  }  
}
```

REACT COMPONENTS

FUNCTIONS & PROPS

Every component can receive one or more props (properties). In addition you can make custom functions

- Props defines data that can be passed in as input parameters when you render a component
- Properties can be interpreted as input parameters
- Example: *I want to render a CustomerView component with the user "John Doe"*
 - `<CustomerView user="John Doe" />`
 - Props are available through `this.props.user`

```
class Customer extends React.Component {
  [...]
  render(){
    return(
      <p> {this.props.name} </p>
    )
  }
}
```

```
import Customer from 'Customer'

class CustomerList extends React.Component {
  render(){
    return(
      <Customer name="Bob"/>
    )
  }
}
```

REACT COMPONENTS

STATE & CONSTRUCTOR

Every component has a state

- State describes a state a component has or can be in
- The state should be considered private for the component, and is initialized in the component's constructor `constructor()`
- Available through `this.state.var`
- Can be changed through `this.setState({var: 'value'})`
 - Not `this.state.var = value`, this will not trigger re-rendering of VirtualDOM
- Sometimes it makes sense to lift the state up to a parent component
 - F.ex. If something should be handled based on the output/outcome of multiple components


```
class Customer extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: this.props.name,
      typeOfCustomer: 'default'
    };
  }

  upgrade() {
    this.setState({
      typeOfCustomer: 'premium'
    });
    //NOT this.state.typeOfCustomer = [...]
  }

  render() {
    return(
      <p> {this.props.name} </p>
    )
  }
}
```

```
import Customer from 'Customer'
```

```
class CustomerList extends React.Component {
  render(){
    return(
      <Customer name="Karen"/>
    )
  }
}
```

REACT COMPONENTS

EVENTS

Events are useful to define behavior when a component is clicked on, hovered over etc.

- The most used are maybe onClick, onKeyUp
- Usually used to call a component's function
 - `<CustomerView onClick={ this.showInfo().bind(this) } />`

```

class Customer extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: this.props.name,
      typeOfCustomer: 'default'
    };
  }

  upgrade() {
    this.setState({
      typeOfCustomer: 'premium'
    });
  }

  render() {
    return (
      <p
        onClick={this.upgrade().bind(this)}>
        {this.props.name}
      </p>
    )
  }
}

```

```

import Customer from 'Customer'

class CustomerList extends React.Component {
  render(){
    return(
      <Customer name="Bob"/>
    )
  }
}

```

```
class Customer extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: this.props.name,
      typeOfCustomer: 'default'
    };
  }

  upgrade() {
    this.setState({
      typeOfCustomer: 'premium'
    });
  }

  render() {
    return(
      <p>
        {this.props.name}
      </p>
    )
  }
}
```

```
import Customer from 'Customer'

class CustomerList extends React.Component {
  render(){
    return(
      <Customer onClick={() => this.upgrade()}
name="Bob"/>
      [...]
    )
  }
}
```

REACT TIPS & TRICKS

You may encounter terms and syntax that seems foreign and difficult to understand

- Define your variables correctly to maintain app performance: `const`, `let` and `var`
 - [Read more about variable definition](#)
- The `render()` function can only have **one** element
 - If you want multiple components in the `render()` function, you can, e.g., wrap them in a `<div>`
- The arrow syntax `=>` can seem scary, but it's just another way to declare a anonymous function
 - `() => {}` is the same as `function(){}`
- Spend a few minutes to understand **immutability** to maintain performance and handle state more accurately
- To define a element's class name in JSX, use `className="klasse"`
 - Not `class="klasse"`

```
const a = 'Cannot be changed';  
let a = 'Block-scoped, nearest block-scoped, may not be  
    redeclared';  
var a = 'Global, block-scoped and in global window-object'
```

```
//---
```

```
const f = function(input) {  
    return input;  
}
```

```
const f = (input) => {  
    return input;  
}
```

```
//---
```

```
render(){  
    return(  
        <div className="my-class"></div>  
    );  
}
```



React Native is library made for developing web application for mobile and smart devices

- It has built-in handling for touch-events
 - Tap, drag, navigation between pages and more
- Apps can easily be integrated with camera, microphone, gyrometer, GPS ++
- Wraps a web application to native iOS and Android apps
- Instagram, AirBnB, Skype, and Walmart are some businesses that features parts of the app created with React Native
 - Source: <https://facebook.github.io/react-native/showcase.html>



I WANT TO TRY REACT!

React's home page has a very good introduction to React through making a simple tic-tac-toe application – this one is highly recommended to do!

<https://reactjs.org/tutorial/tutorial.html>

React Docs Tutorial Community Blog Search docs v16.0.0 GitHub

Tutorial: Intro To React

Before We Start

What We're Building

Today, we're going to build an interactive tic-tac-toe game.

If you like, you can check out the final result here: [Final Result](#). Don't worry if the code doesn't make sense to you yet, or if it uses an unfamiliar syntax. We will be learning how to build this game step by step throughout this tutorial.

Try playing the game. You can also click on a button in the move list to go "back in time" and

TUTORIAL ^

- Before We Start
 - What We're Building
 - Prerequisites
 - How to Follow Along
 - Help, I'm Stuck!
- Overview
 - What is React?
 - Getting Started
 - Passing Data Through Props
 - An Interactive Component
 - Developer Tools
- Lifting State Up
 - Why Immutability Is Important
 - Functional Components
 - Taking Turns
 - Declaring a Winner



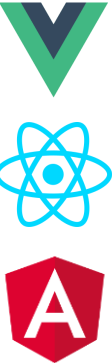




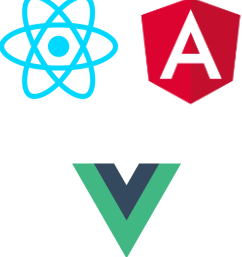
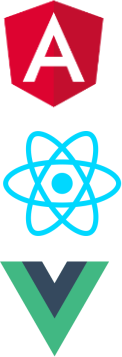

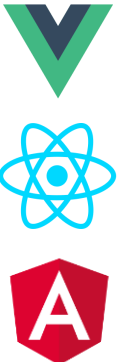
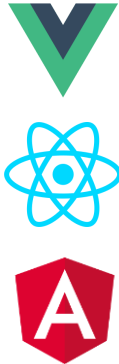
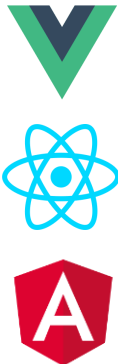
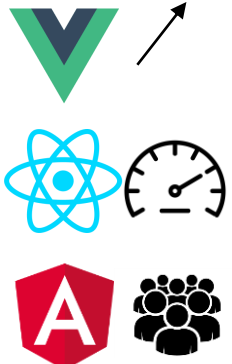
FRAMEWORK COMPARISON

**EVERY LIBRARY HAS
ADVANTAGES AND
DISADVANTAGES**

**“THERE IS A
POINT IN YOUR
PROGRAMMING
CAREER, WHEN
YOU REALISE
THAT THERE
ISN'T A BEST
TOOL”**

- funfunction (YouTuber)

RANKING WITHIN SOME AREAS

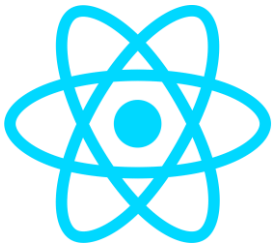
<p>Learning Curve</p> 	<p>Scalability</p> 	<p>3rd party comp.</p> 	<p>Performance</p> 	<p>Community</p> 	<p>Employment</p> 
<p>Companies</p> 	<p>Beyond the Web</p> 	<p>Simplicity (code)</p> 	<p>Development Time</p> 	<p>Size</p> 	<p>Future Outlooks</p> 

You might like...

Du may not like...



Type safety (TypeScript)	Heavy platform
Extensive feature set – Validation, AJAX, Animations, etc.	Performance
Code separation	Some awkward, unintuitive APIs



Almost pure JavaScript	JSX, some awkward syntax
Lightweight	Intermixed View, Style, and Logic code
Brevity	State management



Half-intermixed View, Style, and Logic code	Half-intermixed View, Style, and Logic code
Lightweight	Unintuitive attribute binding
Framework support for filters, computed properties, transitions, etc.	IDE support not great

THANK YOU!

LARS.HENRIK.NORDLI@ACCENTURE.COM

EKATERINA.ORLOVA@ACCENTURE.COM

