WORKSHOP

React and Django for Beginners

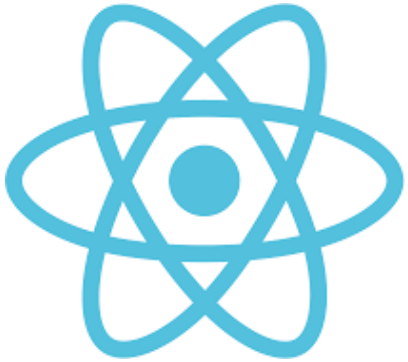
CDTM | Summer 2019

Mihai Babiac, Raul Berganza Gómez, Kai Siebenrock

Version 2.0

Outline

Workshop is split in two sessions for front-end and back-end



Morning

Front-end development

Afternoon

Back-end development

Frameworks to build advanced web applications in MPD

Tutors

Helping you to get started with web development

Mihai Babiac

Fall 2018

Electrical Engineering and Information Technology at TUM

Kai Siebenrock

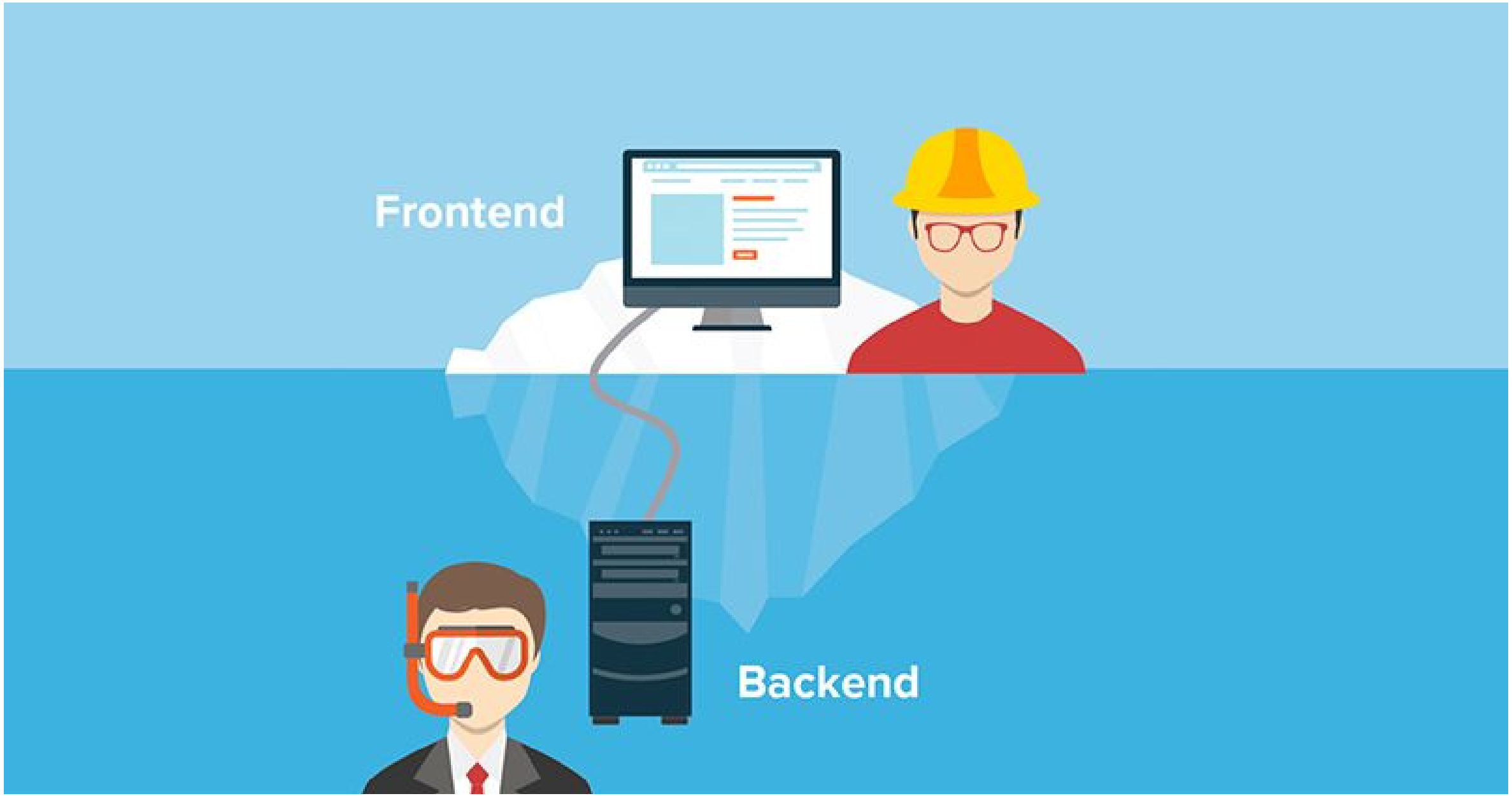
Fall 2018

Management and Technology at TUM

Raul Berganza Gómez

Spring 2019

Computational Science and Engineering at TUM


Frontend
Backend

Front-end and Back-end

Where is the difference?

Front End

- Markup and web languages such as HTML, CSS and Javascript
- Asynchronous requests and Ajax
- Specialized web editing software
- Image editing
- Accessibility
- Cross-browser issues
- Search engine optimisation

Back End

- Programming and scripting such as Python, Ruby and/or Perl
- Server architecture
- Database administration
- Scalability
- Security
- Data transformation
- Backup

Front-end Development

Getting started with React

Afternoon: Back-end development with Django

Agenda

Our schedule for the morning

1. **About React and Why React?**
   About frameworks and alternatives
2. **Introduction to JSX**
   Getting started
3. **Technical Setup**
   Installing Node.js, creating your app
4. **Advanced JSX and ES6**
   About components

Morning Break

5. **Basic React and Practice**
   Building components
6. **Advanced React**
   Composing components
7. **Component Life-Cycle**
   How components are born and die

About React

What is React?

- **JavaScript framework for user interfaces**

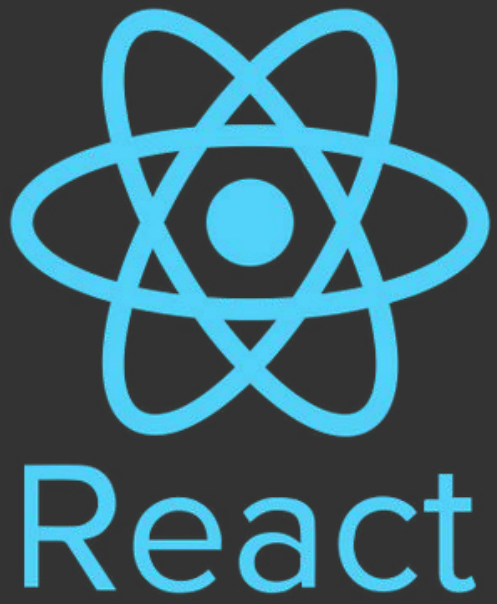
  Great for front-end development of web applications

- **Developed by Facebook in 2011**

  Maintained by a community

- **Component-based**

  Encapsulated components that display the UI when composed

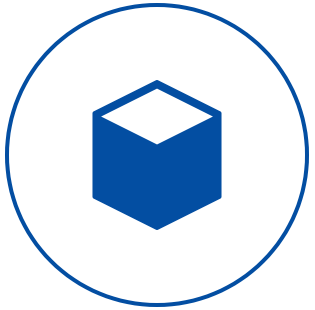


Why React?

What are the advantages of React?

Fast & Scalable

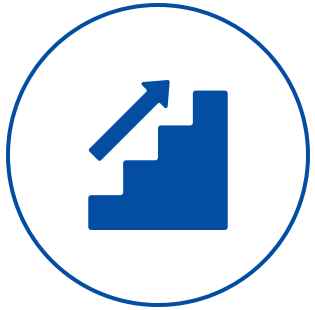
Fast and responsive even for large applications displaying a lot of data



Modular

Many small, clear, and reusable files

Easy to update and extend



Flexible

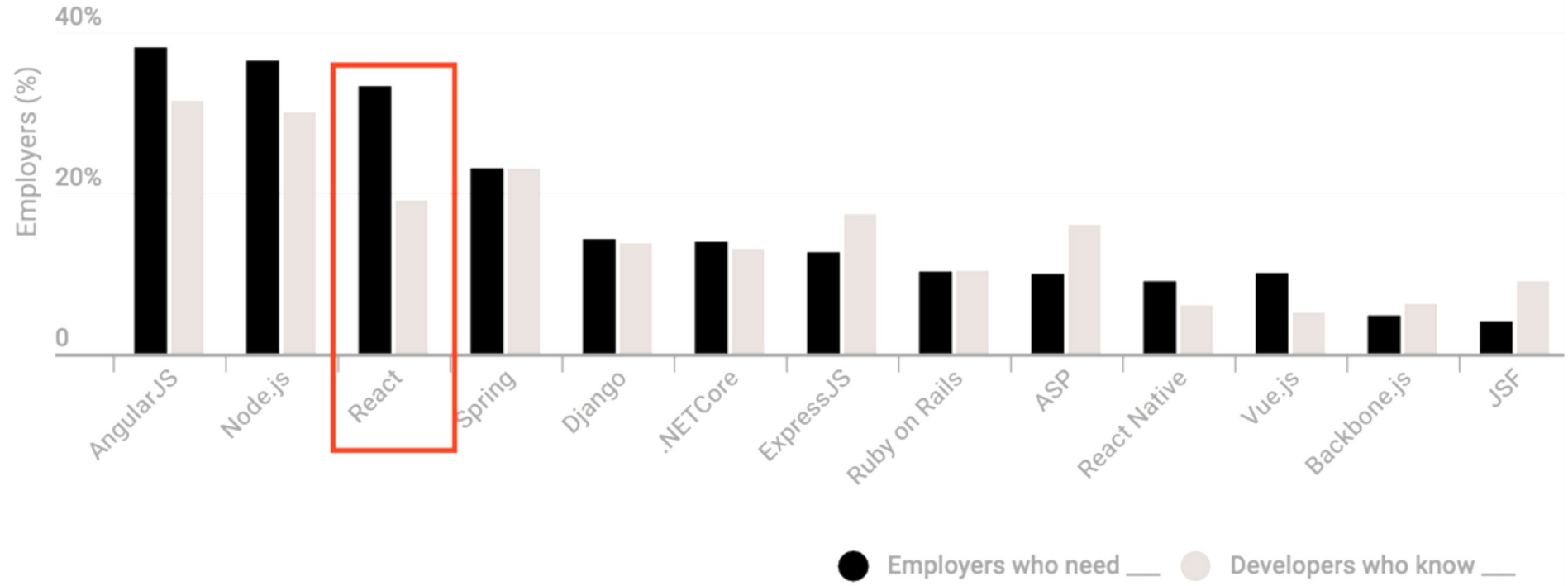
Possibilities beyond web apps e.g. React Native

Popular

Huge increase in popularity in recent years, demanded in the industry

Demand of Developers

Which frameworks do employees need vs. developers know?

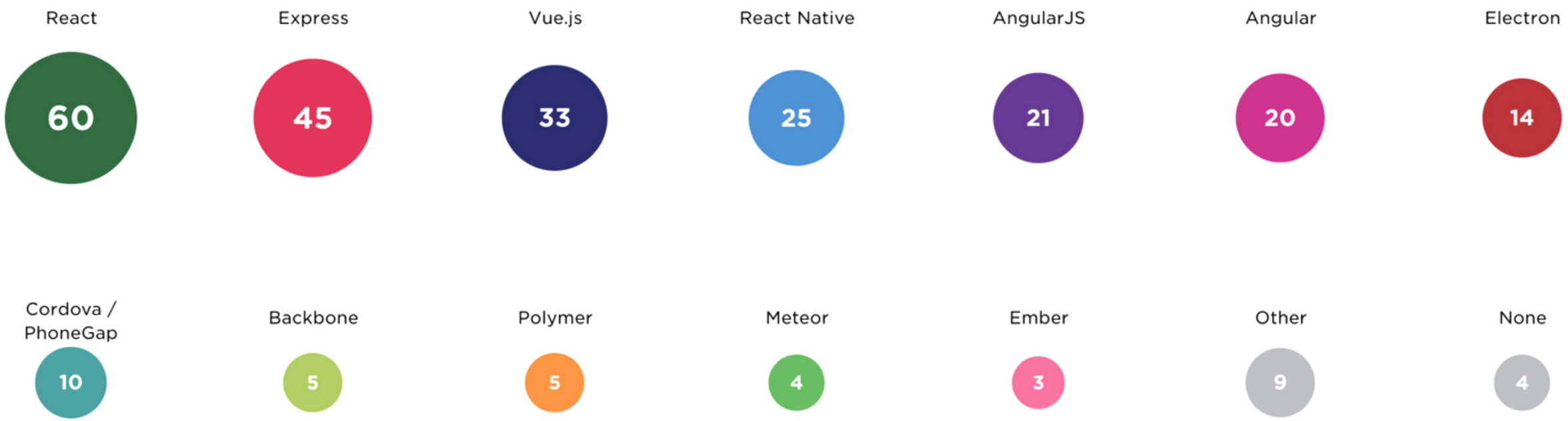


Who is Using React?

Top companies and tools you use daily are build using React

JavaScript Frameworks

React is the number one JavaScript framework for developers



Alternatives to React

Competing frameworks to develop for the web

Angular

Most reknown, more complex

Vue

Intuitive, fast and composable

Ionic

Cross platform development: mobile, web, and desktop

Hello World

Introduction to JSX

```
const title = <h1>Hello world</h1>
```

Hello World

Introduction to JSX

```
const title = <h1>Hello world</h1>
```

- **Hybrid: HTML and JavaScript?**

  JavaScript constant with semicolon but with HTML tags

- **JavaScript file called JSX**

  JSX is extension for JavaScript, written for React

- **Complier needed**

  Can't be read by browser; compiler needs to translate to JavaScript and HTML

- **Store as variable**

  JSX elements are treated as JavaScript expressions, can be stored in variable

Nested Expressions

Can be wrapped in parentheses; can be saved as variables or passed to functions

```
const nested = (
  <a href="https://www.cdtm.de/">
    <h1>
      Apply now!
    </h1>
  </a>
);
```

Outer Elements

Expressions require exactly one wrapping element, *<React.Fragment>* to avoid extra nodes in DOM

```
const courses = (
    <p>Trend Seminar</p>
    <p>Managing Product Development</p>
    <p>Entrepreneurship Laboratory</p>
);
```

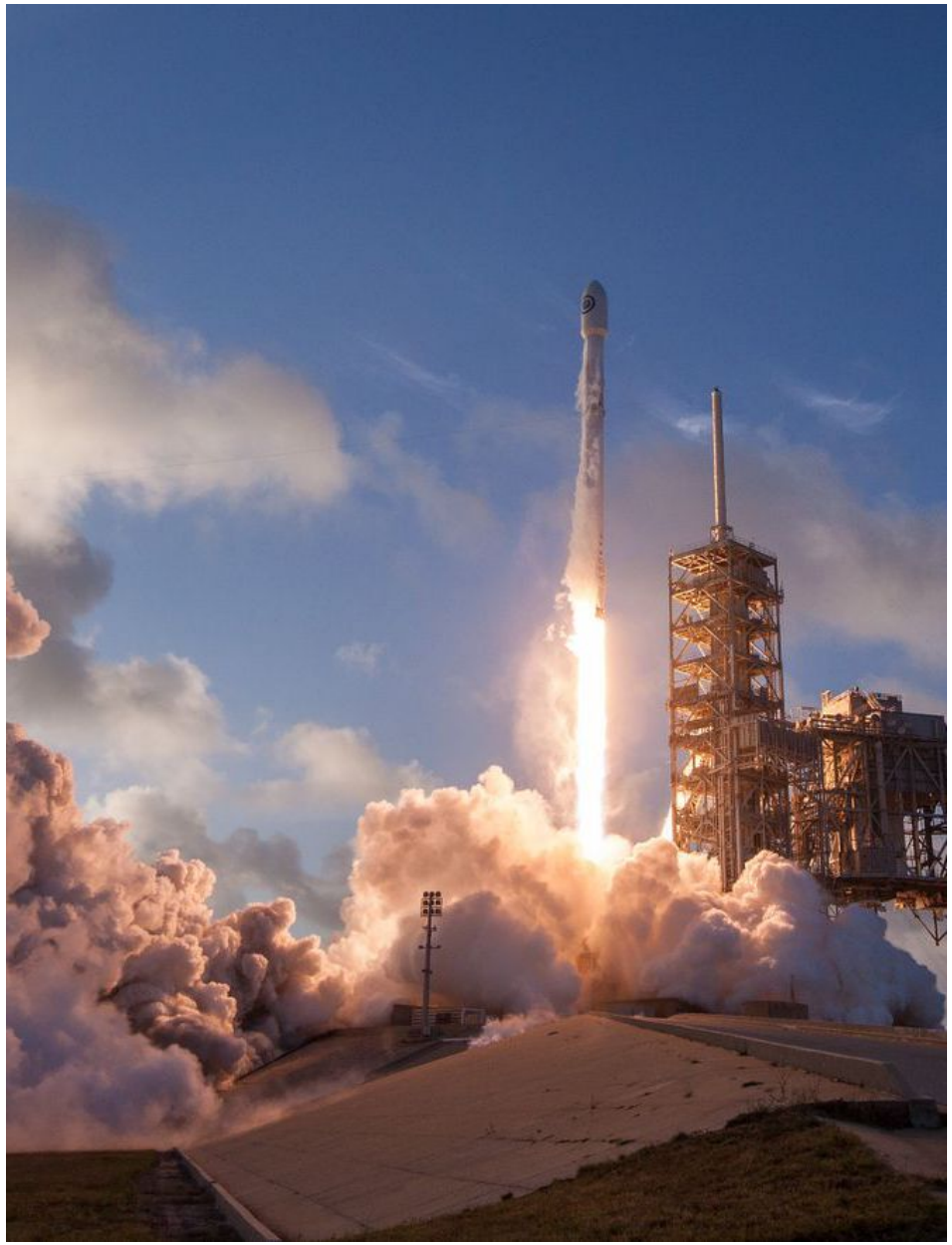
No enclosing element

```
const courses = (
  <React.Fragment>
    <p>Trend Seminar</p>
    <p>Managing Product Development</p>
    <p>Entrepreneurship Laboratory</p>
  </React.Fragment>
)
```

Opening and closing tag of same element

Technical Setup

On your device



Install Node.js

Provides npm, a package manager for JavaScript projects

- **Install Node.js from nodejs.org**

  Use the LTS version

- **Install create-react-app**

  Open a terminal and type the commands to the right

```
npm install -g create-react-app
(OR)
sudo npm install -g create-react-app
```

Install Visual Studio Code

A lightweight source code editor for JS, Python and other languages

- **Get it: code.visualstudio.com**
- **Install Prettier plugin**

  Source code formatter

- **Install Simple React Snippets plugin**

  Code snippets for common tasks

Intro to Node.js

The JavaScript engine running outside of a browser

- **Based on Chrome's V8 engine**

  It's very fast (but can still be 10x slower than C++)

- **Can do almost anything normal programs can**

  Access files? Databases? Open sockets? Use serial ports?

  No problem!

- **Comes bundled with npm**

  It's own dependency management solution

Intro to npm

The Node.js Package Manager (npm)

- **Lets you install software from a registry of 800.000+ packages**

  No need to reinvent the wheel

- **Some packages can be used directly for front-end development**

  They are served to the browser like other assets (JS and CSS files)

  e.g. a calendar widget, the bootstrap UI framework, React components etc.

- **Others are used as tools during development**
  - Minifiers: tools to make JS files smaller
  - Transpilers: tools which convert between programming languages
  - Bundlers: tools which combine multiple files into one
  - Servers

Running Node.js and npm

You installed them. Now what?

- **Node and npm are command line tools**

  They are ran from the terminal/command prompt

- **Node is an interpreter**

  Once run, you can interactively execute JavaScript (similar to running python)

- **npm is a utility**

  It is ran with specific arguments to do various things

```
mihai@laptop:~$ node
> var x = 5
undefined
> 2*x
10
> "My number is " + x
'My number is 5'
> Math.sqrt(x)
2.23606797749979
```

```
mihai@laptop:~$ npm install bootstrap

mihai@laptop:~$ npm uninstall bootstrap

mihai@laptop:~$ npm install -g create-react-app
```

Using npm

How do you deal with packages?

- ***npm install -g <package_name>***

  Installs a package globally

  Used for command line utilities like create-react-app or npm itself

- ***npm install <package_name>***

  Installs a package locally, only for the project in the current folder

- ***npm uninstall <package_name>* and *npm uninstall -g <package_name>***

  Removes a certain package

```
mihai@laptop:~$ npm install bootstrap

mihai@laptop:~$ npm uninstall bootstrap

mihai@laptop:~$ npm install -g create-react-app
```

Create Your First React App

Getting ready for the actual development

- **Run *create-react-app* from the terminal in your project folder***

  Set up all the needed infrastructure for developing with React

  **for Windows, open "Node.js command prompt" to run the command*

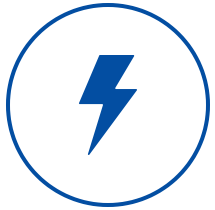
- **Run your app**

  Change directory and run it

- **Curious what was installed?**

  Check out this link

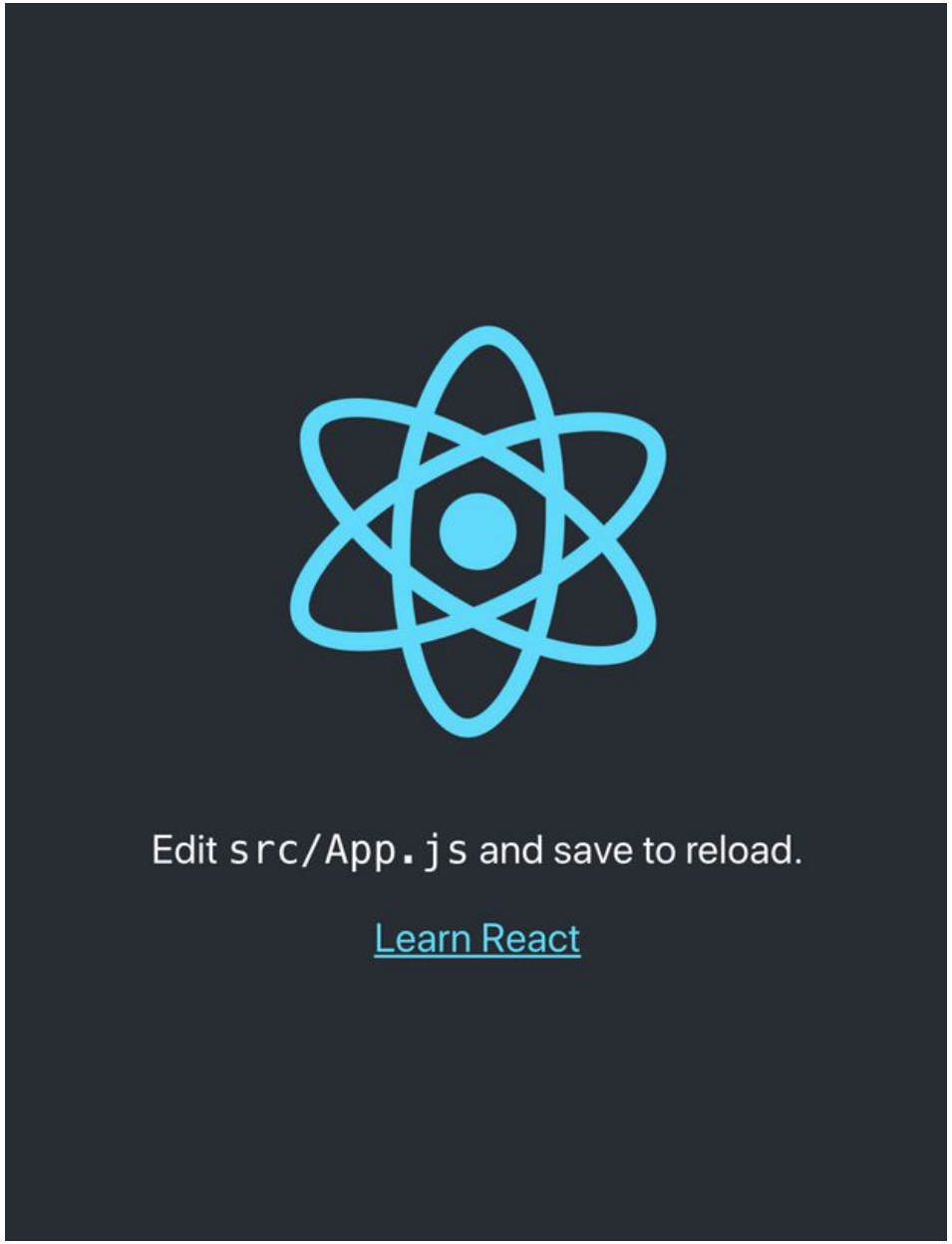
```
create-react-app my-react-app
cd my-react-app
npm start
```



Works?

Go to http://localhost:3000/

Ask us if you have difficulties!



Project Structure

How React projects are organized

- **node_modules**

  Contains all packages install by npm

  e.g. react, webpack (bundler), babel (transpiler), ESlint (linter)

- **public**

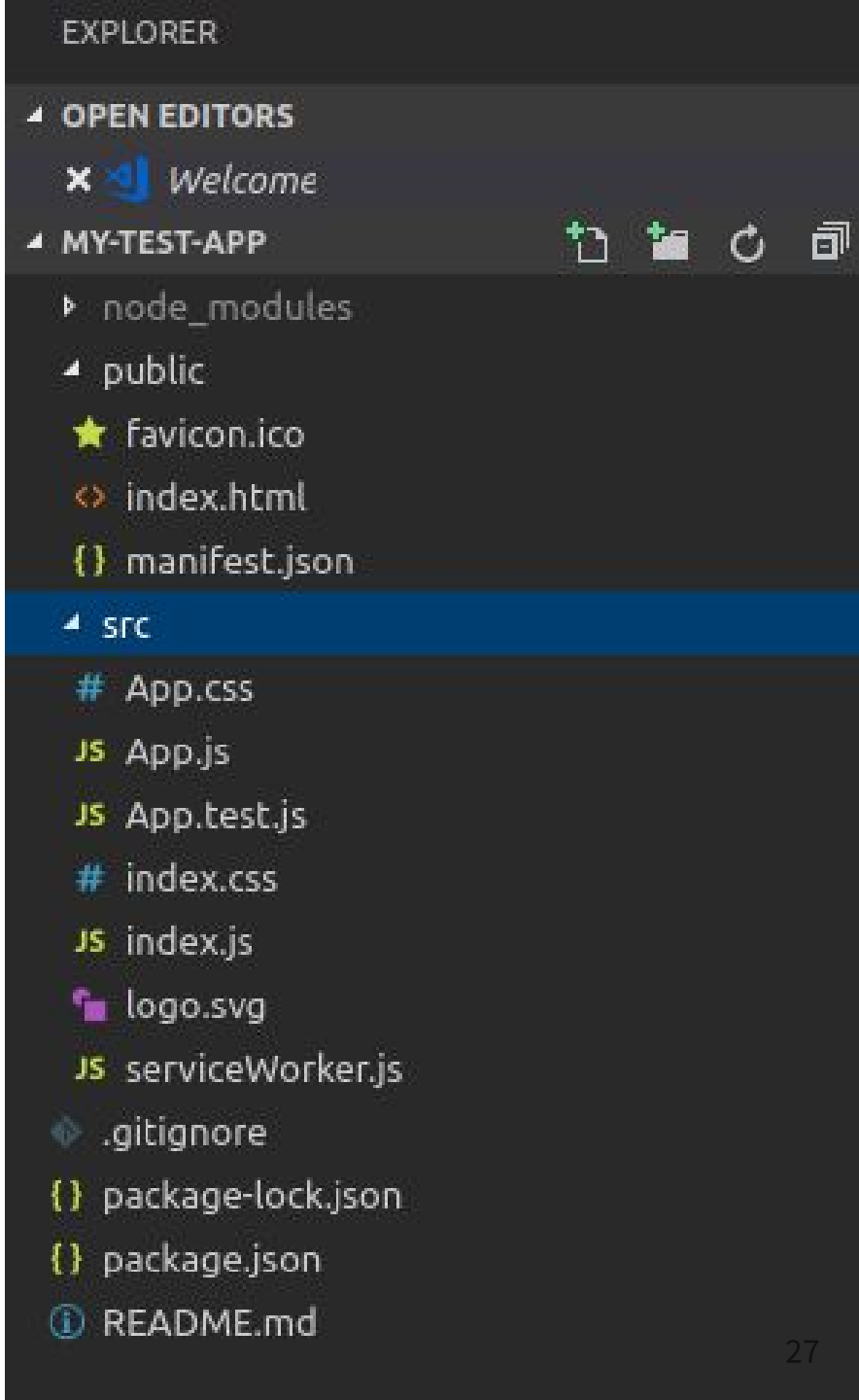
  Contains files that are served as they are

  e.g. index.html (main HTML file), manifest.json and favicon.ico (page icon)

- **src**

  Main source code files for your project

  Most used folder for development



Program Flow

The path from the HTML file to the first React component

- ***index.html* is the HTML file that is served**

  It contains a with the id “root”

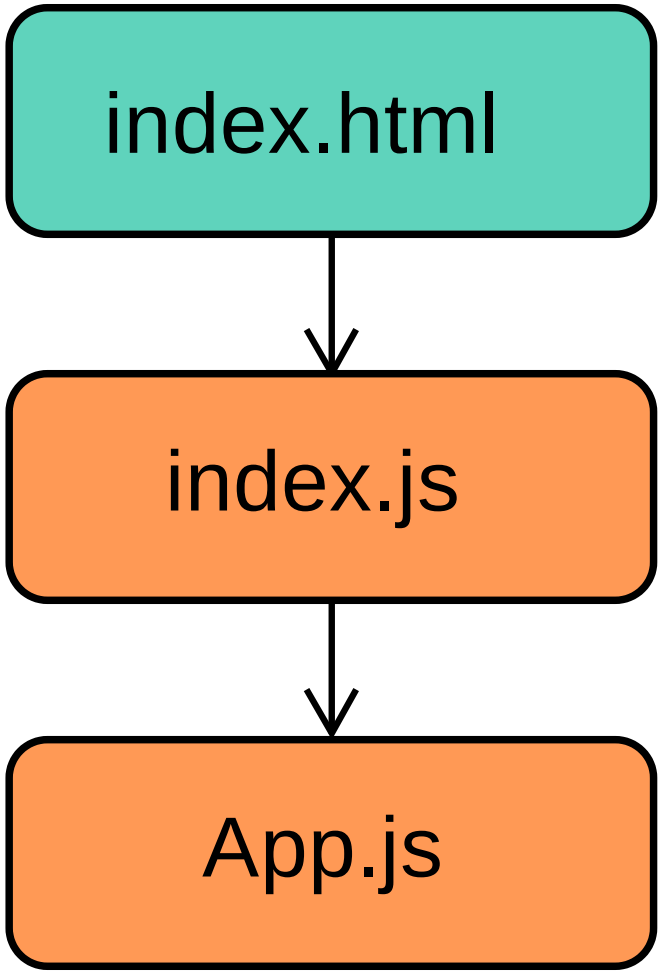
- ***index.js* is the entry point for JS**

  It imports React and places an <App> component in the element with the id “root”.

  First call to ReactDOM.render()

- ***App.js* contains the first displayed React component**

  This is where the high-level structure of the app is usually laid out



Babeljs: Transpiling JSX

JSX is just a shorthand for JavaScript. Try it out at **babeljs.io/repl**

```jsx
const x = <h1>Hello!</h1>;
const y = <ul>
          <li>Intro</li>
          <li><a href="www.google.com">Outro</a></li>
          </ul>;
```

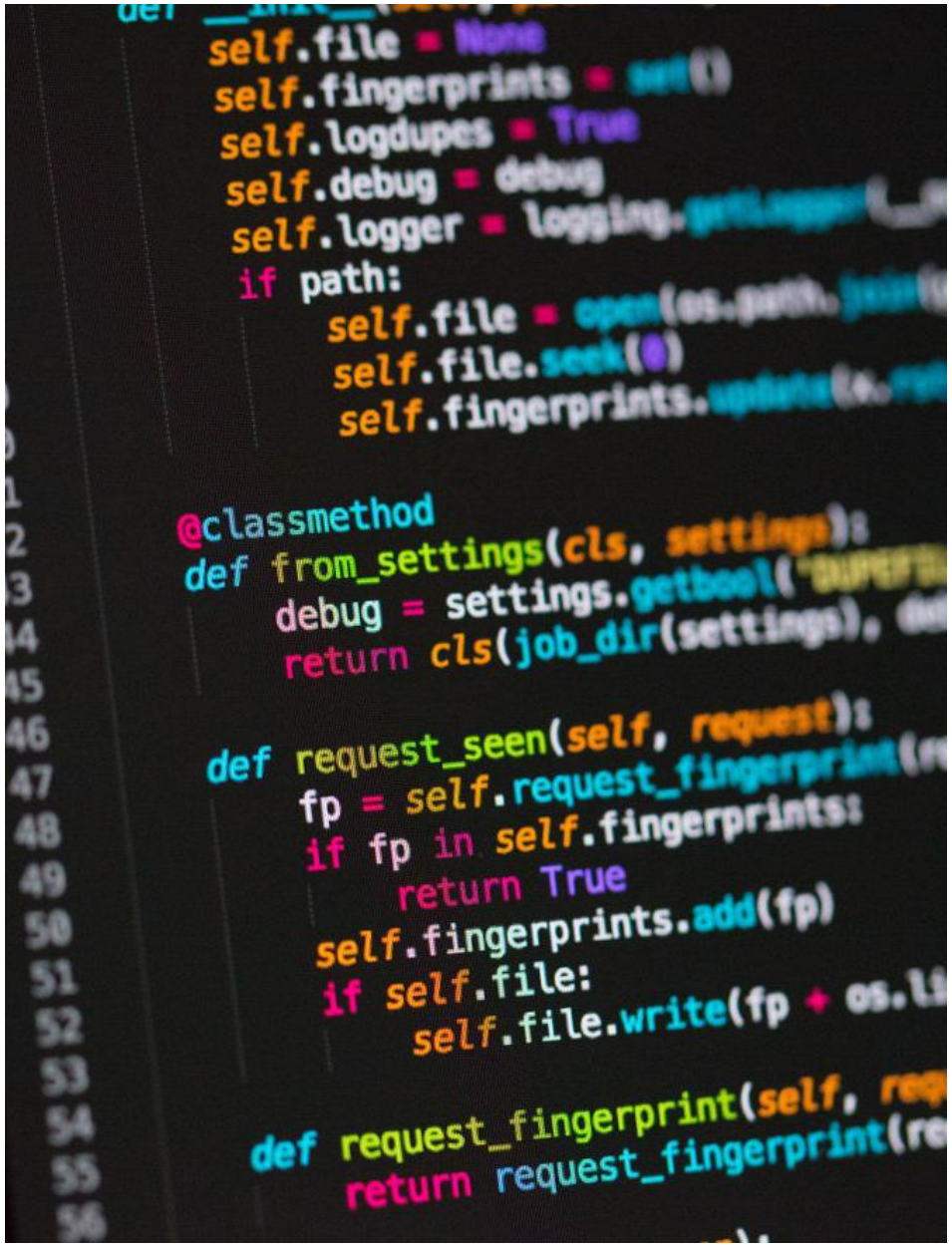
JSX

```javascript
"use strict";

var x = React.createElement("h1", null, "Hello!");
var y = React.createElement("ul", null,
            React.createElement("li", null, "Intro"),
            React.createElement("li", null,
                React.createElement("a", {
                    href: "www.google.com"
                }, "Outro")));
```

Equivalent JavaScript

Coding

Learn about JSX



Starting with index.js

Check out your file; remove the *<App />* component and its import, at least for now

```
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();
```

ReactDOM.render()

Introduction to JSX

- **Import JavaScript libraries**

  Containing React methods to manipulate DOM

- **Render method arguments**

  Arguments are a JSX expression and the HTML container to which the expression should be appended

- **Render JSX code**

  Render element into DOM or update it

- **Make sure to set correct ID!**

  Replace 'app' with 'root'

```
import React from 'react';
import ReactDOM from 'react-dom';

ReactDOM.render(
  <h1>Hello world</h1>,
  document.getElementById('app')
);
```

```
<main id="app">
  <h1>Insert here!</h1>
</main>
```

Variables to ReactDOM.render()

Introduction to JSX

- **Pass variable as argument**

  Possible it evaluates to JSX expression

```
const toDoList = (
  <ol>
    <li>Create Mockup</li>
    <li>Develop Prototype</li>
  </ol>
);

ReactDOM.render(
  toDoList,
  document.getElementById('app')
);
```

Virtual DOM

Introduction to JSX

- **Render only updates if changed**

  Only if DOM elements have changed it is updated

- **Virtual DOM is lightweight copy**

  Update virtual DOM, then compare changes to actual DOM, then update actual DOM

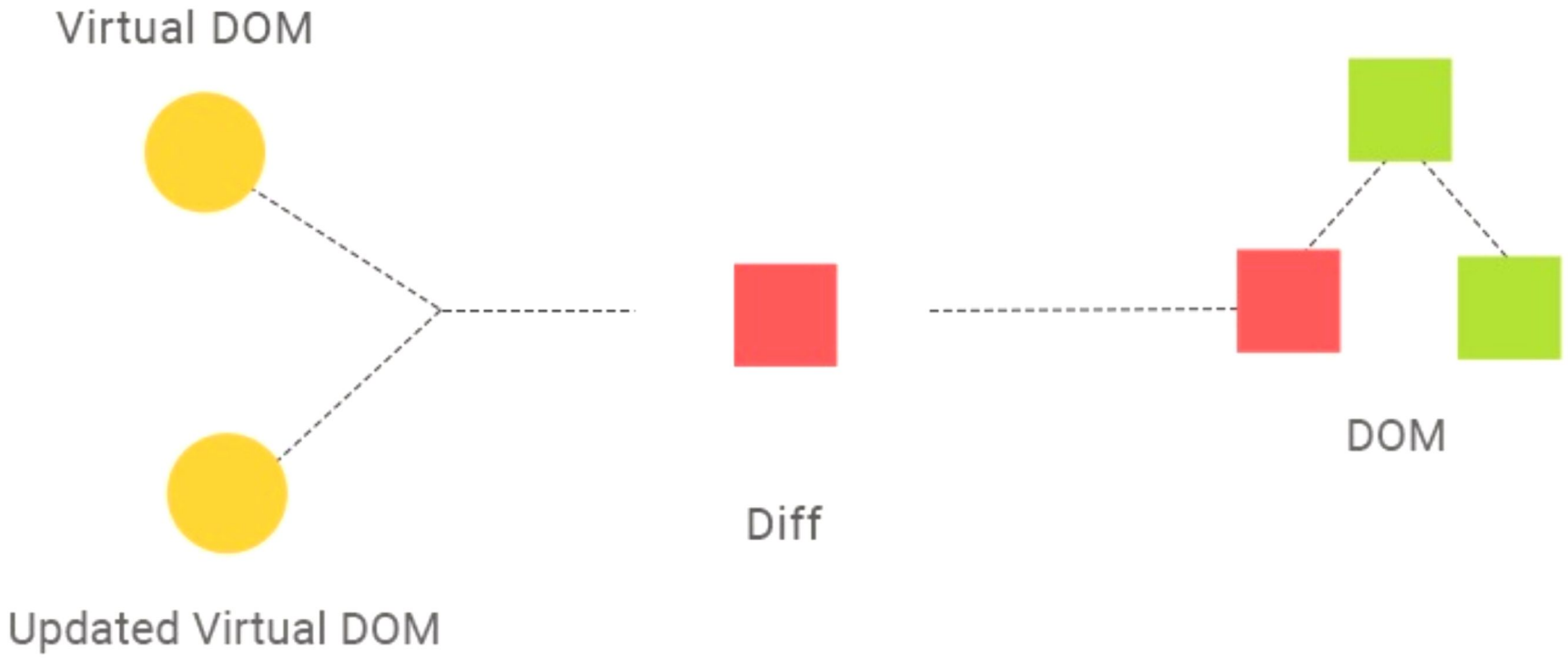
- **React only updated necessary**

  Thereby great performance

```
const title = <h1>Hello world!</h1>;

ReactDOM.render(title, document.getElementById('app'));
// Render "Hello world!"

ReactDOM.render(title, document.getElementById('app'));
// No change in DOM
```

How Does the Virtual Dom Work?

Introduction to JSX



Class vs. ClassName

Advanced JSX

```
<h1 class="card">CDTM</h1>
```

```
<h1 className="card">CDTM</h1>
```

HTML

Use *class* as attribute

JSX

Use *className* as attribute

class is reserved name

Objects for Styling

Advanced JSX

```
<h1 style="color:blue;font-size:46px;">
  My Title
</h1>
```

```
<h1 style={{ color: 'blue', fontSize: '46px' }}>
  My Title
</h1>
```

HTML Inline Style Syntax

Pass a string to attribute

React

Pass an object to attribute

JavaScript in JSX

Advanced JSX

- **Wrap in curly braces**

  If wrapped, it will be evaluated and the result will be inserted at that location

  Heading evaluates to “*2*”

  Paragraph evaluates to “*3.142*”

- **Does this code compile?**

  Find the error!

```
import React from 'react';
import ReactDOM from 'react-dom';

ReactDOM.render(
  <h1>{1 + 1}</h1>
  <p>{Math.PI.toFixed(3)}</p>,
  document.getElementById('app')
);
```

Access Variables in JSX

Advanced JSX

```
import React from 'react';
import ReactDOM from 'react-dom';

const guest = 'Alice';

ReactDOM.render(
  <p>Hello, {guest}!</p>,
  document.getElementById('app')
);
```

Event Listeners

Advanced JSX

```
function notify() {
  alert("Submission successful!");
}

const submit = <button onClick={notify}>Submit</button>;
```

- **Wait for trigger e.g. click**

  Similar to HTML event listeners

- **Special attributes with function**
  - *onClick*
  - *onMouseOver*
  - *onKeyUp*
  - *onSubmit*

Conditionals

Advanced JSX

```
const title = (
  <h1>
    { points >= 80 ? 'Test Passed' : 'Test Failed' }
  </h1>
);
```

```
const performance = (
  <ul>
    { !passed && <li>Repeat test</li> }
    { points > 85 && <li>Good work!</li> }
    { points > 90 && <li>Excellent work!</li> }
    { points > 95 && <li>Outstanding work!</li> }
  </ul>
);
```

- **Breaks with if statements**

  Learn more

  Or place if statement outside of JSX tags

- **Use ternary operator**

  “condition ? a : b”

  Returns a if condition is true, otherwise b

- **Use && operator**

  “{condition && Alice }”

New in ES6: Arrow Function

Advanced JSX

```
function doubleNumber(params) {
  return params * 2
}
```

Function written in ES5 syntax

Much longer

```
var doubleNumber = params => params * 2

// Anonymous function
params => params * 2
```

Same function as arrow function in ES6

Can call anonymously

Omit this. binding

Parameter of Arrow Functions

Advanced JSX

```
// Single parameter, brackets are optional
(singleParam) => { statements }
singleParam => { statements }

// Multiple parameter
(param1, param2, …, paramN) => { statements }

// Without parameter
() => { statements }

// With rest parameter
(param1, param2, ...rest) => { statements }
```

Mapping Function

Advanced JSX

```
const courses = ['TS', 'MPD', 'eLab'];

const list = courses.map((course, i) =>
    <li key={'course_' + i}>{course}</li>
);

ReactDOM.render(
    <ul>{list}</ul>,
    document.getElementById('app')
);

// Evaluates to:
<ul>
  <li key="course_0">TS</li>
  <li key="course_1">MPD</li>
  <li key="course_2">eLab</li>
</ul>
```

- **Create list**

  Call map() on array of strings

  Create HTML list of courses

- **Keys attribute**

  Value must be unique, similar to id attribute

  Used internally to keep track of list

Introduction to Components

Components in React

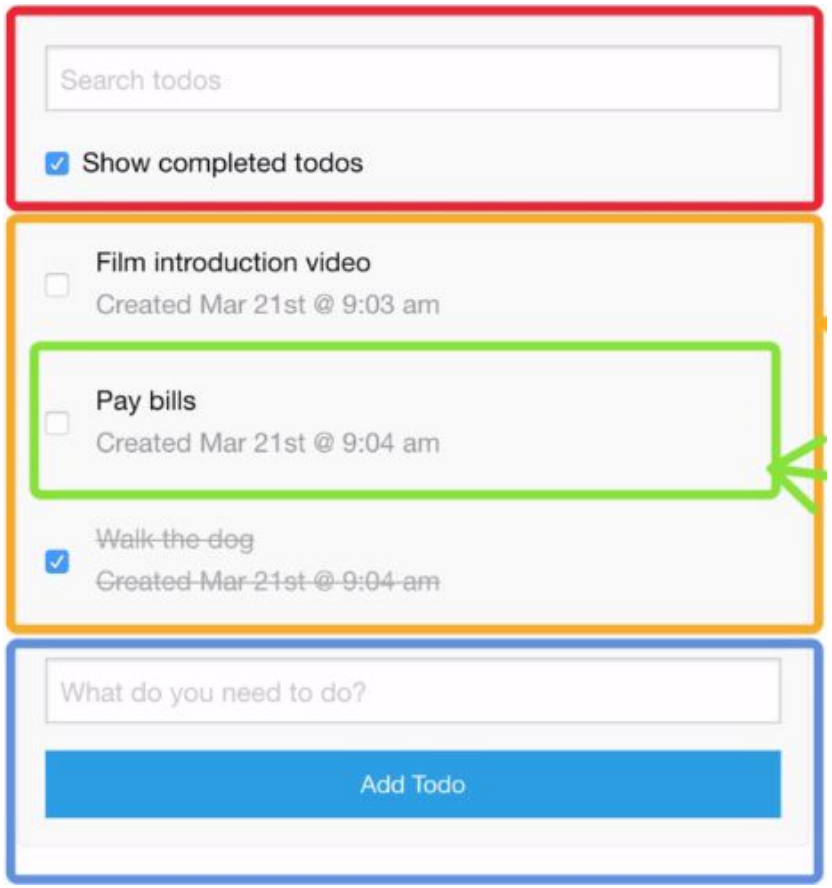
- **What's a component?**

  Reusable chunk of code with one specific purpose

- **Todo list app examples**
  - Search tasks
  - List
  - Task list entry
  - Add task

- **Extend component class**

  Blueprint of component with specific functionality used for new components



Create Components Class

Components in React

- **Components extend component class**

  Class is blueprint for all components with specific functionality

  Learn more about classes and subclasses

- **Name new component**

  Start with capital letter, programming convention

- **Render method in component class**

  Property with value is a function that contains return statement

  When component is called, its render method is executed

```
import React from 'react';
import ReactDOM from 'react-dom';

class NewComponent extends React.Component {
  render() {
    return <h1>Welcome to MPD</h1>;
  }
};
```

Show Components Class

Components in React

- **Show component instance**

  Insert component class in ReactDOM.render()

  JSX can distinguish between HTML and component instances (upper case)

```
import React from 'react';
import ReactDOM from 'react-dom';

class NewComponent extends React.Component {
  render() {
    return <h1>Welcome to MPD</h1>;
  }
};

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

Variables in Components

Components in React

```
const puppySettings = {
  src: 'https://upload.wikimedia.org/wikipedia/commons/6/6e/Golde33443.jpg',
  alt: 'Puppy',
  width:  '200px'
};

class Puppy extends React.Component {
  render() {
    return (
      <div>
        <h1>Cute Puppy</h1>
        <img
          src={puppySettings.src}
          alt={puppySettings.alt}
          width={puppySettings.width} />
      </div>
    );
  }
}

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

Logic in Component Render

Components in React

```
class Generator extends React.Component {
  render() {
    const number = Math.floor(Math.random() * 100)
    return <h1>Random number is {number}!</h1>;
  }
}
```

- **Add methods**
  - Method executed right before component is rendered
  - Must be inside render()

Event Listener and Functions

Components in React

```
class BigButton extends React.Component {
  notify() {
    alert('Submission successful!');
  }

  render() {
    return <button onClick={this.notify}>Submit</button>;
  }
}

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

Coffee Break

See you in 15 minutes

Import and Export

Splitting your code in multiple files

```
import React, { Component } from 'react';

class NewComponent extends Component {
  render() {
    return <h1>Welcome to MPD</h1>;
  }
}

export default NewComponent;
```

Export module **NewComponent**

```
import React from "react";
import ReactDOM from "react-dom";
import NewComponent from "./components/newComponent";

ReactDOM.render(<NewComponent />, document.getElementById("root"));
```

Import module **NewComponent**

CSS Frameworks: Material UI

For beautiful and consistent UI components

```
$ npm install @material-ui/core
$ npm install @material-ui/icons
```

- **Install the packages with npm**

  Use the command line

- **Add <*CSSBaseline* /> to *index.js***

  To set the CSS styling to a better default

- **Add links for fonts and icons to *index.html***

  Add them inside the <*head*> tag

```
import React from "react";
import ReactDOM from "react-dom";
import CssBaseline from "@material-ui/core/CssBaseline";
import App from "./App";

ReactDom.Render(
  <React.Fragment>
    <CssBaseline/>
    <App/>
  </React.Fragment>
  ,document.getElementById("root"))
```

CSS Frameworks: React Bootstrap

Using a UI library gives you default components that help you cut down development time

```
# Install the necessary node modulus:
$ npm install react-bootstrap bootstrap --save

# Add this file yo your index:
import 'bootstrap/dist/css/bootstrap.min.css';
```

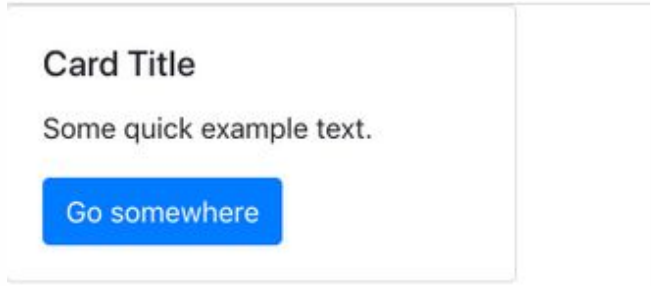
- **Component library**

  react-bootstrap comes with a set of predefined components that can be used to compose your project. Reference of all components can be found here

```
// Each components should be directly imported from the module:
import {Card} from 'react-bootstrap';

class BootstrapComp extends React.Component {
    render() {
        return (
        <div style={parentContainerStyles}>
          /* Using them is as easy as including the right tags
          in your components */
            <Card style={{ width: '18rem' }}>
              <Card.Body>
                  <Card.Title>Card Title</Card.Title>
                  <Card.Text>Some quick example text.
                  </Card.Text>
                  <Button variant="primary">Go somewhere</Button>
              </Card.Body>
            </Card>
        </div>
      );
    }
}
```

- **Grid system**

  Bootstrap provides a flex-box based layout system that eases building responsive apps. For full documentation, access the docs



States in React

The status of a components is defined by a set of states, that evolve through the flow of the app

- **Fully private**

  Only the component itself is capable of accessing it's state

- **Initialization**

  Can be initialized with an explicit constructor or in the body of the class

```
class BootstrapComp extends React.Component {
    // 1st way to define state, in the function body
    state = {
        'state_1': 0,
        'state_2': 0,
        'state_3': 0,
        'state_4': 0,
    }
    // 2nd way to define state, with a constructor
    constructor(props) {
        super(props);
        this.state = {
        'state_1': 0,
        'state_2': 0,
        'state_3': 0,
        'state_4': 0,
        };
    }
};
```

Simple Counter App

The first component for our demo app

- **Create a new component in *src/components/Counter.jsx***

  Has a “value” property in its state* and displays this along with a button

- **Modify the original *App.js* file**

  Has to import the Counter component and display it

** “state” is a special property of any component; you should not store dynamic data in any other properties*

```jsx
import React, { Component } from "react";
import Button from "react-bootstrap/Button";
import Badge from "react-bootstrap/Badge";

class Counter extends Component {
  state = {
    value: 5
  };

  render() {
    return (
      <div>
        <Badge variant="primary"> {this.state.value} </Badge>
        <Button variant="warning" className="m-2">
          Increment
        </Button>
      </div>
    );
  }
}

export default Counter;
```

```jsx
import React from "react";
import "./App.css";
import Counter from "./components/Counter";

function App() {
  return (
    <div className="App">
      <Counter />
    </div>
  );
}

export default App;
```

Change the Display When the Value Is Zero

Show the text “zero” and make the badge gray

- **Use ternary operator**

  Keeps condition and result together making the code quicker to read, use your common sense though ;)

- **Solution 1: two conditions, one component**

  Top picture

- **Solution 2: one condition, two components**

  Bottom picture

```
render() {
  return (
    <div>
      <Badge variant={this.state.value === 0 ? "secondary" : "primary"}>
        {this.state.value === 0 ? "zero" : this.state.value}
      </Badge>
      <Button variant="warning" className="m-2">
        Increment
      </Button>
    </div>
  );
}
```

```
render() {
  return (
    <div>
      {this.state.value === 0 ? (
        <Badge variant="secondary">zero</Badge>
      ) : (
        <Badge variant="primary">{this.state.value}</Badge>
      )}
      <Button variant="warning" className="m-2">
        Increment
      </Button>
    </div>
  );
}
```

&& and Truthy

More logical operators in javascript

- **Falsy**

  Expressions that in a boolean context evaluate to false. Ie. false, 0, “”, null, undefined, and NaN

- **Truthy**

  Those expressions that are not falsy are considered truthy, and thus evaluate to true

- **AND operator, (a && b)**

  If **a** evaluates to true, the logical expression returns value of **b**

```
render() {
  return (
    <div>
      <Badge variant={this.state.value === 0 ? "secondary" : "primary"}>
        {this.state.value === 0 ? "zero" : this.state.value}
      </Badge>
      <Button variant="warning" className="m-2">
        Increment
      </Button>
    </div>
  );
}
```

```
// Falsy, this won't trigger
if (NaN){console.log('alarm!')};
>
// This will trigger
if ("Almonds"){console.log('alarm!')};
> Alarm!

const a = "almonds"
const b = NaN
const c = "I get to be printed!"

a && c
> "I get to be printed!"
b && c
>
```

New in ES6: Destructuring

Saving you from typing the same thing over and over again

- **Extracts data into new variables**

  Works for both objects and arrays

- **Used most often to extract data from this.state or this.props**

  Works as a simple alias for their contents

  More on props later

- **Read more on MDN**

  There are many features for making code shorter and taking care of corner cases

```
let myObj = {
  name: "Bob",
  age: 15
};

// Without destructuring
let name = myObj.name;
let age = myObj.name;

// With destructuring
let {name, age} = myObj;
```

```
var foo = ['one', 'two', 'three'];

var [one, two, three] = foo;
console.log(one); // "one"
console.log(two); // "two"
console.log(three); // "three"
```

Immutability and setState

In React it is considered an anti-patern to manually manipulate state

- ***setState({states_to_change})***

  setState allows to update the value of your components in a safe way. Internally these updates happen asynchronously and react combines them under the hood

- ***setState({func()})***

  Passing a callback function gives you the guarantee that setState will be run immediately, allowing to use previous values of the state for the update

```
handleClick = (e) =>{
    // In this case it is safe to use the previous value of the state
    this.setState((state, props) =>({
        method: state.method + 1
    }));
}

handleClick = (e) =>{
    // This implementation can lead to concurrency errors
    this.setState({method: state.method + 1});
}
```

Array Operations

ES6 introduces a set of operators and methods that ease batch operations and thus help preserve immutability

- ***.filter(criterion)***

  Creates a new array with those elements from the original one that fulfill the criteria

- ***.reduce(fun())***

  Applies a reducing operation in each one of the elements of the original array to produce an output value

```
var num = [1, 2, 3, 4, 5];

const result = num.filter(n => n > 2);
> [3, 4, 5]

// Easy factorial implementation
const fact = (accumulator, currentValue) => accumulator * currentValue;
num.reduce(reducer);
> 120
```

Props

Props are objects that contain the set of values passed to a React component

```
var num = [1, 2, 3, 4, 5];

const result = num.filter(n => n > 2);
> [3, 4, 5]

// Easy factorial implementation
const fact = (accumulator, currentValue) => accumulator * currentValue;
num.reduce(reducer);
> 120
```

- **Immutable**

  You should not explicitly assign values to the props objects inside of the component by any means

- **Top to down communication**

  Data should flow from parent to children components. If you come across the need of propagating them up, time has come to lift state up

```
// Pass some props to your component:
ReactDOM.render(<App name="CDTM" truth="rocks!"/>, document.getElementById('app'));

// In your component definition, feel free to access the props object:
render() {
    console.log(this.props);
    return (
    <div>
        <h1><Card.Title>{this.props.name} {this.props.truth}</Card.Title></h1>
        <h4>Yeah!</h4>
    </div>
  );
}
```

Initializing State from Props

It is not always a bad practice

```
class Child extends React.Component {
    constructor(props) {
        // super() is necessary always when using explicit constructors
        super(props);

        this.state = {
          loggedIn: false,
          currentState: "quite fine",
          // Initializing state with props
          state_1: this.props.initialValue
        }
    }
}
```

- **When it is OK**

  If the value from props is just used to initialize the state and there won't be later updates of the prop

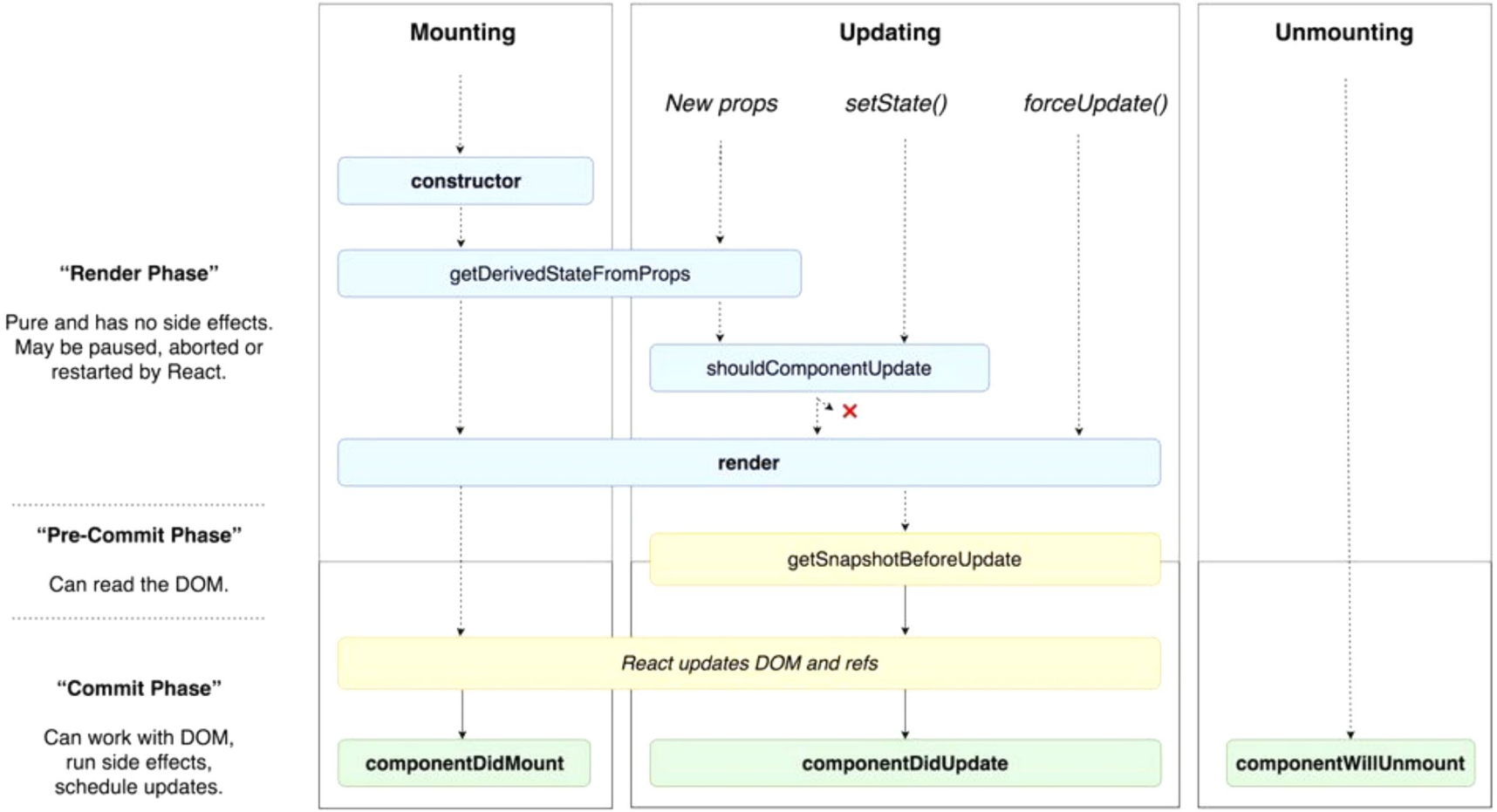
- **When it is not OK**

  If the parent component will update the prop, copying it to a state in the child will create an overhead as it would be necessary to keep them in sync

```
/* If the parent class mutates the value pased as prop, as the prop is only used in
for initializing state, the changes won't be reflected */
class Parent extends React.Component {
    state = {
      'state_1': 1,
    }

    handleClick = (e) =>{
        this.setState((state, props) =>({
            state_1: state.state_1 + 1
        }));
    }

    render() {
        console.log(this.props);
        return (
        <div>
            <Button onClick={this.handleClick} variant="primary">Go somewhere</Button>
            <Child initialValue=this.state.state_1/>
        </div>
      );
    }
}
```

Component Life-cycle Methods

There are three main phases of a component



Resources

Continue learning React after the workshop

AJAX

Http responses are key for communication

Link

Official React Tutorial

All the basics are covered in this set of exercises

Link

PWAs

Upgrade your web app to a progressive one easily

Link

React Native

Now you have the skills, why not making a mobile app?

Link

Deployment

Achieve one click deploy of your React app with Netlify

Link

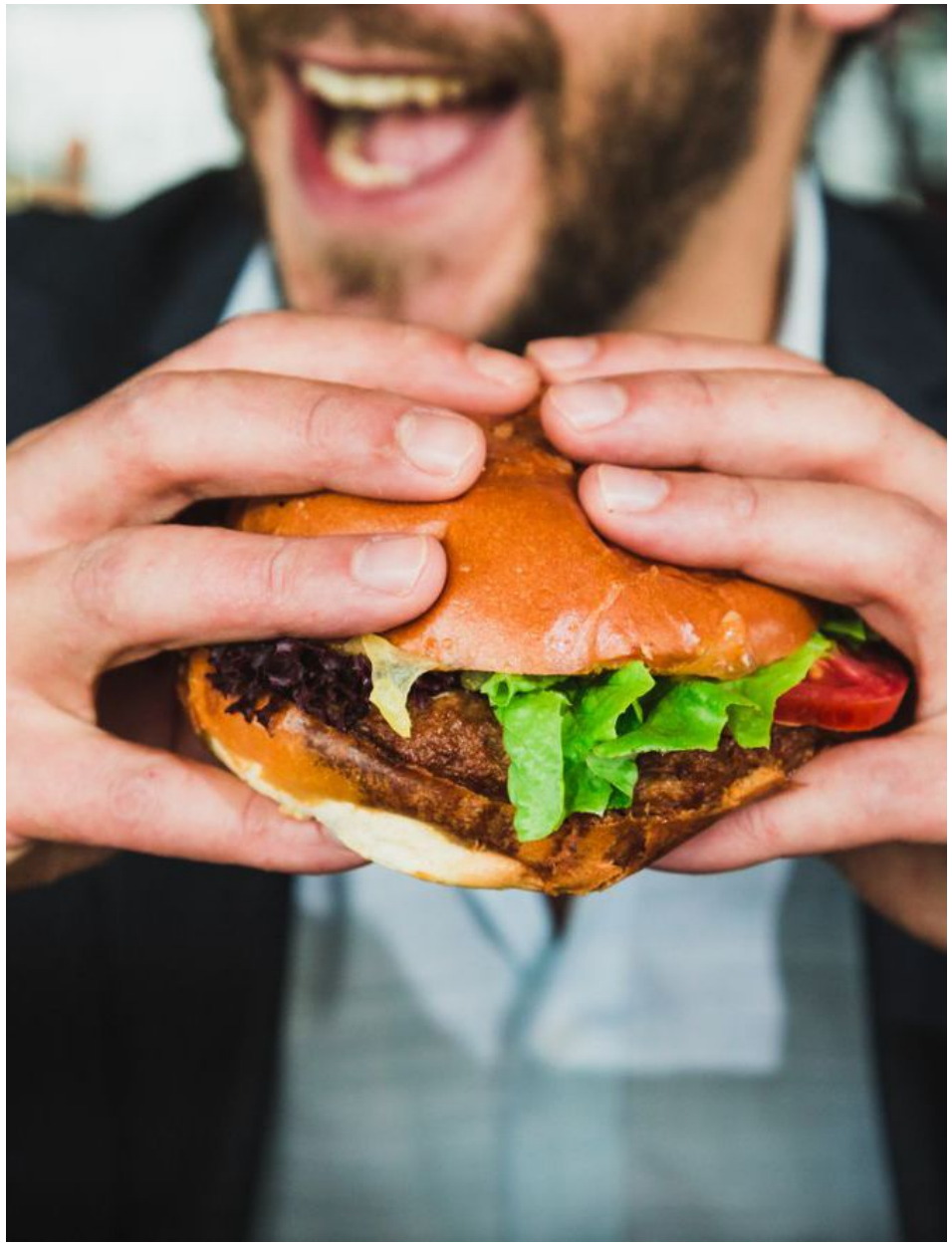
Awesome React

Great resources put together by the community

Link

Lunch Break

See you in 45 minutes



Morning: Front-end development with React

Back-end Development

Getting started with Django

Agenda

Our schedule for the afternoon

1. **Intro to Back-end Development**
   About frameworks and alternatives
2. **Technical Setup**
   Getting started
3. **Folder Structure and MVC**
   The reasoning for Django
4. **Templating Engine**
   How to generate HTML dynamically

Afternoon Break

5. **Bootstrap Implementation**
   How to integrate templates in Django
6. **Django Forms**
   From the form to your DB
7. **REST API**
   Creating an API to interact with other clients
8. **Outlook**

About Django

What is Django?

- **Python framework for web**

  Great for back-end development of web applications

- **Great defaults**

  Default Admin allows setting up a project in minutes

- **ORM makes interaction with DBs easy**

  It's models and migrations simplify interactions with your database

Framework vs. Library

How are they different?

Why Django?

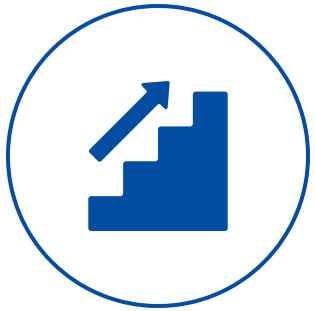
What are the advantages of Django?

Fully loaded

Includes many functionalities out of the box such as an admin interface or authentication

Ship fast

Django reduces the time from idea to production.



Scalable and orthogonal

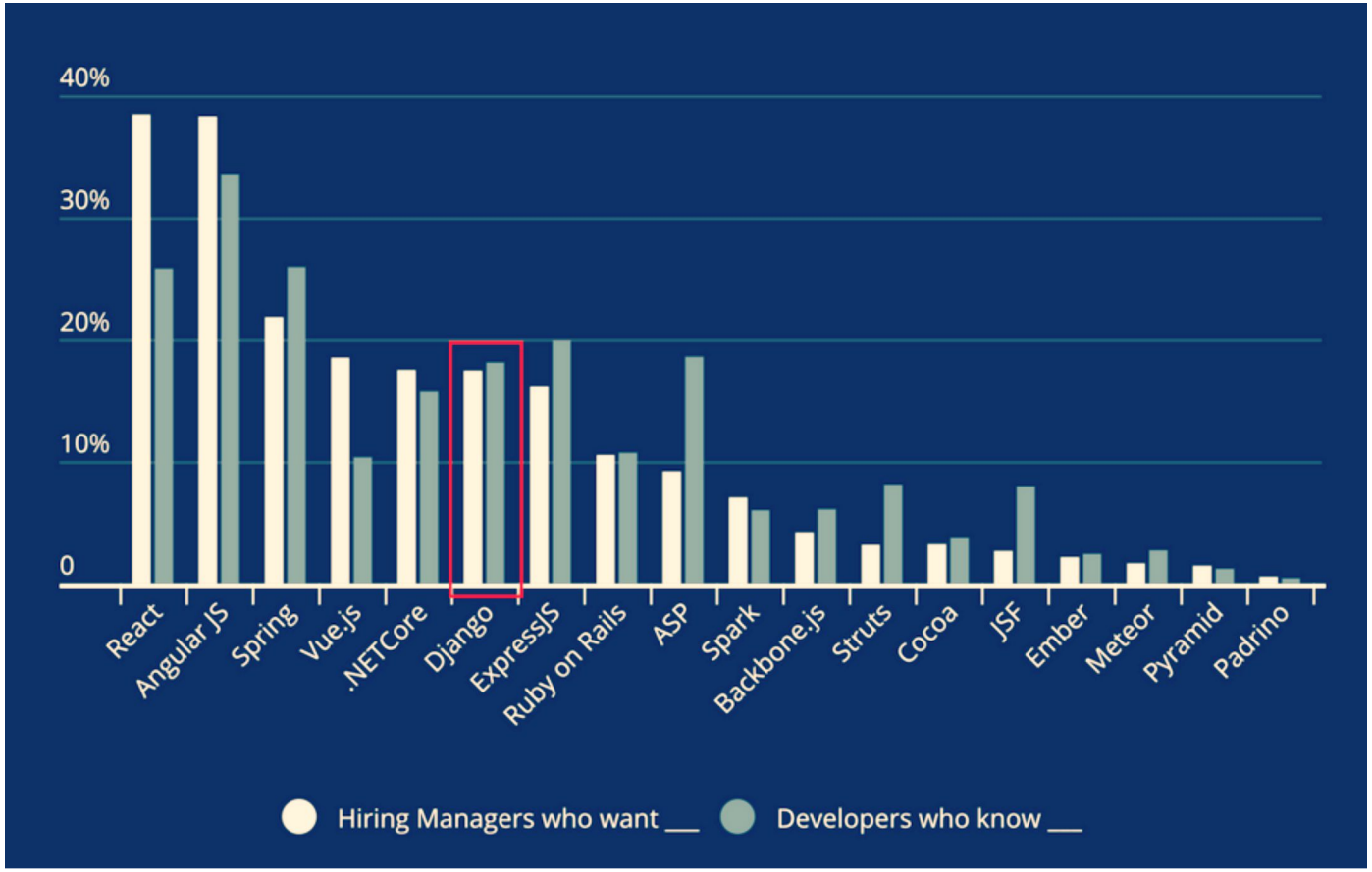
Self contained apps allow deployment on containers that scale on demand

Python powered

A syntax familiar to most of us and a strong community behind

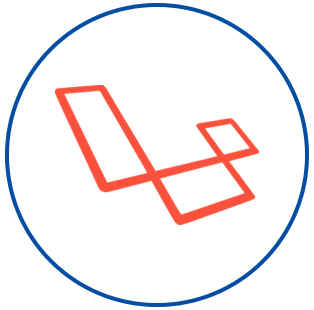
Demand of Developers

Which frameworks do employees need vs. developers know?



Alternatives to Django

Competing frameworks to develop for the web



Laravel

MVC meets PHP

node.js

One language, front and back

Ruby on Rails

Sillicon Valley's most used framework.

Technical Setup

On your device

Technical Setup

The conda package manager keeps your development environment clean

- **We will use Miniconda**

  The most lightweight version

- **Latest stable version can be obtained at**

  https://docs.conda.io/en/latest/miniconda.html

Technical Setup

Starting with conda virtual environments

- **Ease collaboration**

  Using virtual environments makes it easier for developers to work on projects by unifying their toolkit

- **Keep your computer clean**

  Different projects need packages that may conflict with others, reduce complexity by just installing the necessary ones.

```
# Creates a new virtual environment (venv) in Conda with
# Python version 3.6
$ conda create -n {environment_name} python=3.6 django
# Activates the venv
$ conda activate {environment_name}
# Installs django and the REST framework in the venv
$ pip install django djangorestframework
```

Hello World!

First steps in Django

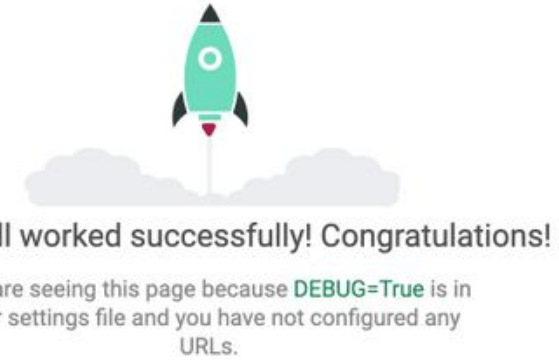
- ***django-admin* creates projects**

  Using this command in your terminal will span the base template for a Django project

- ***manage.py* is control script**

  Once the project is created, control functions such as super admin creation, or running the server, are called from here

```
# Creates a project folder with the indicated name
$ django-admin startproject {project_name}
# Access the project folder
$ cd {project_name}
# Spins up a local server for the Django backend
$ python manage.py runserver
```



Python Interpreter

Each virtual environment can have its own Python binary. For useful tips and recommendations in your IDE, please specify the path of the python binary of your conda environment. To find where your python is:

```
$ python
>>> import os
>>> import sys
>>> os.path.dirname(sys.executable)
>>> /Users/username/miniconda3/envs/cdtm/bin/python
```

Django Structure

It's an opinionated framework

- **Apps enable reusability**

  Splitting your back-end in independent pieces will allow you to recycle functionalities in later projects

- **Understanding the tree**

  It's worth to invest the time in understanding the purpose of each file

Creating an App

Keeping your directories tidy

- **Apps should be 'installed'**

  By default, your project will oversee your app, thus you have to manually add it to the settings page

- **Apps are bare bone**

  The files generated comprise only the ground for building your app. To give it more functionalities we have to add static and template folders, and urls.py file

```
$ python manage.py startapp blog
# Tree utility prints the folder tree bellow the input argument
$ tree blog

blog
├── __init__.py
├── admin.py
├── apps.py
├── migrations
│   └── __init__.py
├── models.py
├── tests.py
└── views.py

$ mkdir templates static
$ touch urls.py
```

settings.py

Global configuration of your Django project

- **One file to rule them all**

  Most of the configuration in Django happens here: DB engine setup, locales, addition of Django applications (potentially made by other devs)

- **Keep an eye on the apps list**

  Add any application you create to the installed apps list, otherwise it's content won't be found by Django

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Your apps
    'blog.apps.BlogConfig',
]
```

Model, View, Controller

Django's architectural pattern

- **Models**

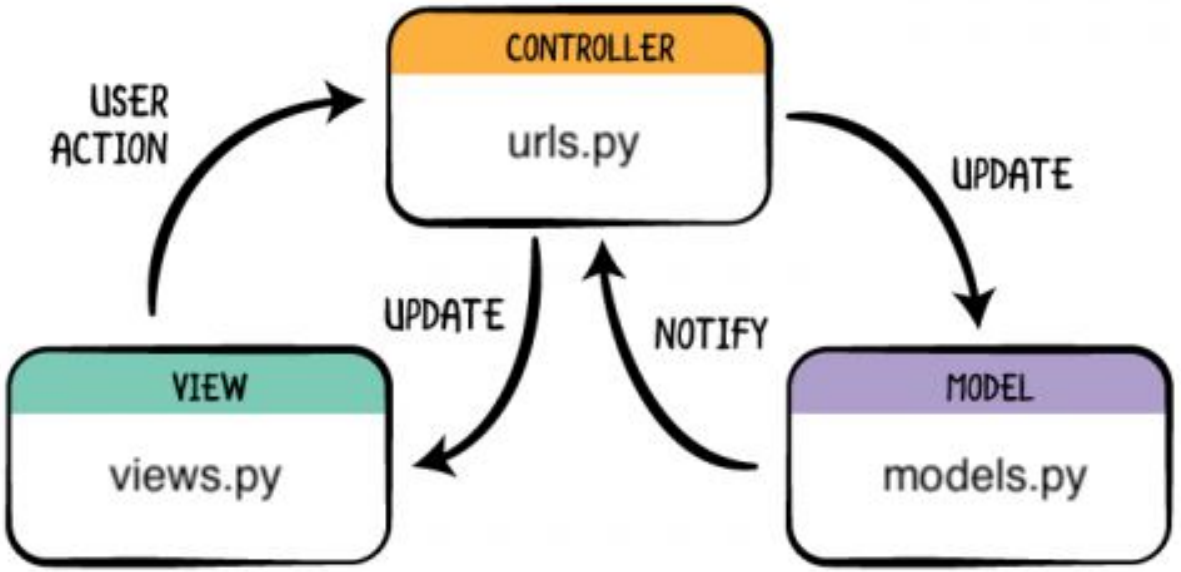
  They normally describe the real world objects that you are trying to represent in your application

- **Views**

  The functions that interact directly with the user. They can display content or trigger internal logic

- **Controller**

  Links the actions of the user with the corresponding view.



URL Structure

Parting from your base domain, Django adds whichever paths we concatenate with the path() directive

urls.py

How your urls affect the logic that is run

- ***include()***

  Django can't see the urls contained in your apps out of the box, thus we use *include()* to tell it do add these routes to a given path, in these case to */blog*

- ***path()***

  Creates the association between a path and a given view. Remember to import the views into *urls.py*

```python
# In the file urls.py of the root folder (the one named like your project)
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    ## Line added!
    path('blog/', include('blog.urls'))
]
```

```python
# In the file urls.py of the app (blog)
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name="index")
]
```

views.py

We will create a view that returns a dummy request

- ***view_name(request)***

  Is standard format for defining views. Return a response, render method or null.

- ***request***

  Request object holds information about client making the request

```python
# In the file views.py of the app (blog)
from django.http import HttpResponse
from django.shortcuts import render

# Create your views here.
def index(request):
    return HttpResponse('I like carrots.')
```

views.py

Rendering HTML with a view

- ***templates***

  In the templates folder add a file named *index.html*. Let your imagination go wild and write some code there

- ***render()***

  The render method is a shorthand to render a given HTML template with the associated context. We'll dig into context once we define our models.

```python
from django.shortcuts import render

# Create your views here.


def index(request):
    # Change the return for render method!
    return render(request, "index.html", {})


#! Content of index.html

<div style="background-color:red">
    <h1> I like carrots a lot!</h1>
</div>
```

models.py

Object oriented and database (DB) friendly

- **Models define DB objects**

  Django model definitions allow you to simply define objects and relationships amongst them in a concise way

- ***__str__()***

  The str method defines how is your object represented by Django. By default objects are represented by <*Object+ pk*>. It's more intuitive for debugging to represent them by their name or other field that allows you to identify them

- ***datetime***

  Datetime is the default Python module for writing time stamps. Will come in handy whenever you wanna create time fields

```
from datetime import datetime
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=50)
    desc = models.CharField(max_length=400)
    date_posted = models.DateTimeField(default=datetime.now))

    def __str__(self):
        return self.title
```

Cardinality

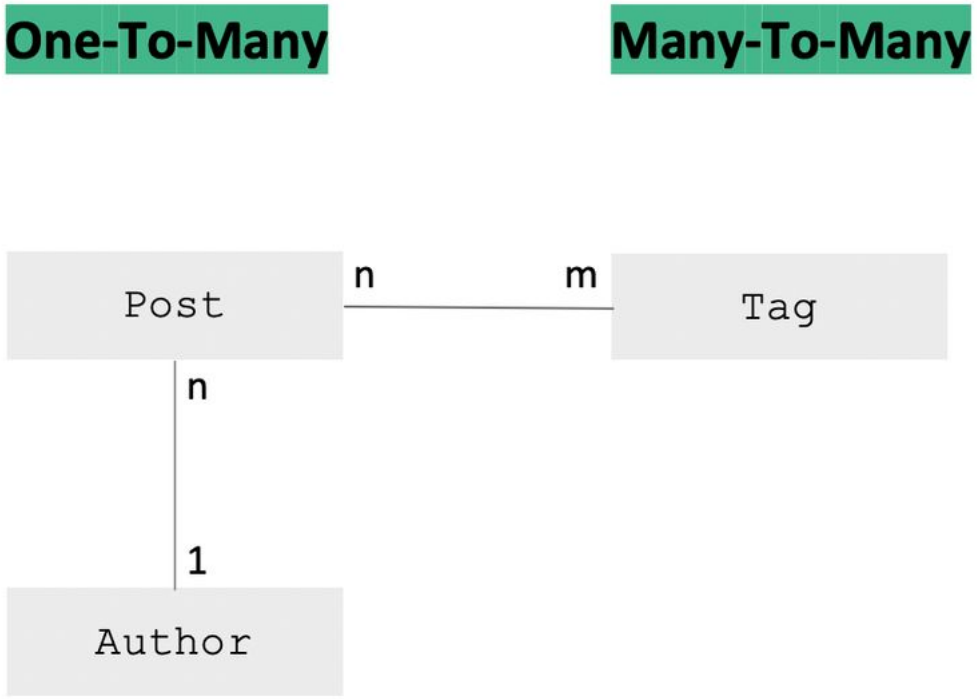
How your objects are related

- **Object Relational Mapper**

  Models are translated to DB tables. Specify the relationships amongst the objects in Django and its ORM will take care of the rest

- **Architecture eases maintainability**

  Effort put into properly modeling the relationships amongst your objects makes database management easier later down the road



One-To-Many

Exactly one object of one class is associated to indefinitely many objects of the other class

```
from django.db import models
from datetime import datetime

# New line!
from django.contrib.auth.models import User

class Post(models.Model):
    title = models.CharField(max_length=20)
    content = models.CharField(max_length=400)
    date_published = models.DateTimeField(default=datetime.now())
    # Establishes the one-to-many relationship on the 'many' end of the relationship
    user = models.ForeignKey(User, on_delete="cascade")

    def __str__(self):
        return self.title
```

Many-To-Many

One object of one class is associated to indefinitely many objects of the other class, and the other way around.

```
# New model created!
class Tag(models.Model):
    name = models.CharField(max_length=20)

    def __str__(self):
        return self.name


class Post(models.Model):
    title = models.CharField(max_length=20)
    content = models.CharField(max_length=400)
    date_published = models.DateTimeField(default=datetime.now())
    # Establishes the one-to-many relationship on the 'many' end of the relationship
    user = models.ForeignKey(User, on_delete="cascade")
    # Establishes the many-to-many relationship
    tags = models.ManyToManyField(Tag)

    def __str__(self):
        return self.title
```

Migrations

They generate or modify the database tables to represent the models described in *models.py*

- ***makemigrations***

  Generates the migration files that contain the set of instructions that accommodate into your database tables the changes implemented in your models since the last migration took place

- ***migrate***

  Applies the changes specified in the migration files created by makemigrations

```
$ python manage.py makemigrations

$ python manage.py migrate
```

Using the Shell

Interact with your database and have fun

```
$ python manage.py shell

>>> from blog.models import Post
>>> from django.contrib.auth.models import User
>>> u = User(username="bugs_bunny")
>>> u.save()
>>> p = Post(title="mytitle", content="tt", user=u)
>>> Post.objects.all()
<QuerySet [<Post: mypost>, <Post: obj>]>
```

Coffee Break

See you in 15 minutes

Django Admin

The secret sauce of Django

- **Out of the box functionalities**

  Django provides you with a full fledge admin for managing your web sites information

- **Highly customizable**

  All the tables and forms can be customized, and dashboards built on top

```
$ python manage.py createsuperuser
$ python manage.py runserver

# Access to http://127.0.0.1:8000/admin
# and use your credentials from the super user
```

admin.py

Allows you to customize the content displayed in the admin

- ***admin.ModelAdmin***

  Allows you to determine which fields will be shown in the admin UI

- ***admin.site.register()***

  Tells Django which models should be editable from the admin interface

```python
from django.contrib import admin

# Register your models here.

from blog.models import Post, Tag

class PostAdmin(admin.ModelAdmin):
    list_display = ("title", "date_published", "user")

admin.site.register(Post, PostAdmin)
admin.site.register(Tag)
```

Templates and View Context

Your views can receive some context that will be used to generate dynamic HTML based on your DB's content

- ***request***

  The request object can contain the data send to a given endpoint or the user object if it has logged in

- ***context***

  The context dictionary contains those objects or strings that we want to make available for the template specified in the render method

```
#! New code
from blog.models import Post

def index(request):
    #! New code
    post = Post.objects.all()
    user = request.user
    context = {"post": post, "user": user}
    return render(request, "index.html", context)
```

Templating Engine

Django generates HTML code dynamically based on your request bodies and back-end

- **Variable interpolation {{ }}**

  The arguments are passed as a context dictionary to the view serving the given .html file

- **Logic {% %}**

  The curly brackets followed by percentage signs allows smarter functionalities like for loops and conditional statements. Click for more information on the Django templating syntax

```
<div style="display:flex;flex-direction: column;align-items: center">
{% if user%}
    <h1>Hello, {{ user }}!</h1>
{% endif %}
{% if posts %}
    {% for post in posts %}
    <div style="background-color: #024EA1; width: 50%; border: solid white 1px">
        <h2 style="color: white">{{ post.title }}</h2>
        <h4 style="color: white">{{ post.content }}</h4>
    </div>
    {% endfor %}
{% endif %}
</div>
```

Django Static Assets

Serving static assets like CSS, JavaScript or images

- *app/static/*

  Django will look for files in the static folder of the app when using {% static /path/ %}. It is a good practice to group files further by type .ie create css subfolder

- **Trash the cache!**

  Remember to disable your browsers cache during development. Otherwise you won't see the changes in static assets.

```
$ mkdir static/css
$ touch styles.css
```

```
:root{
    // This line creates a global css variable;
    --CDTM_blue: #024EA1;
}

h1, h2,h3,h4{color: white;}

div{
    border: solid 1px white;
    background-color: var(--CDTM_blue);
    width: 50%;
    padding-left: 2%;
}
```

Django Static Assets

Serving static assets like CSS, JavaScript or images

- *{% load static %}*

  This line makes Django static assets available in a given template

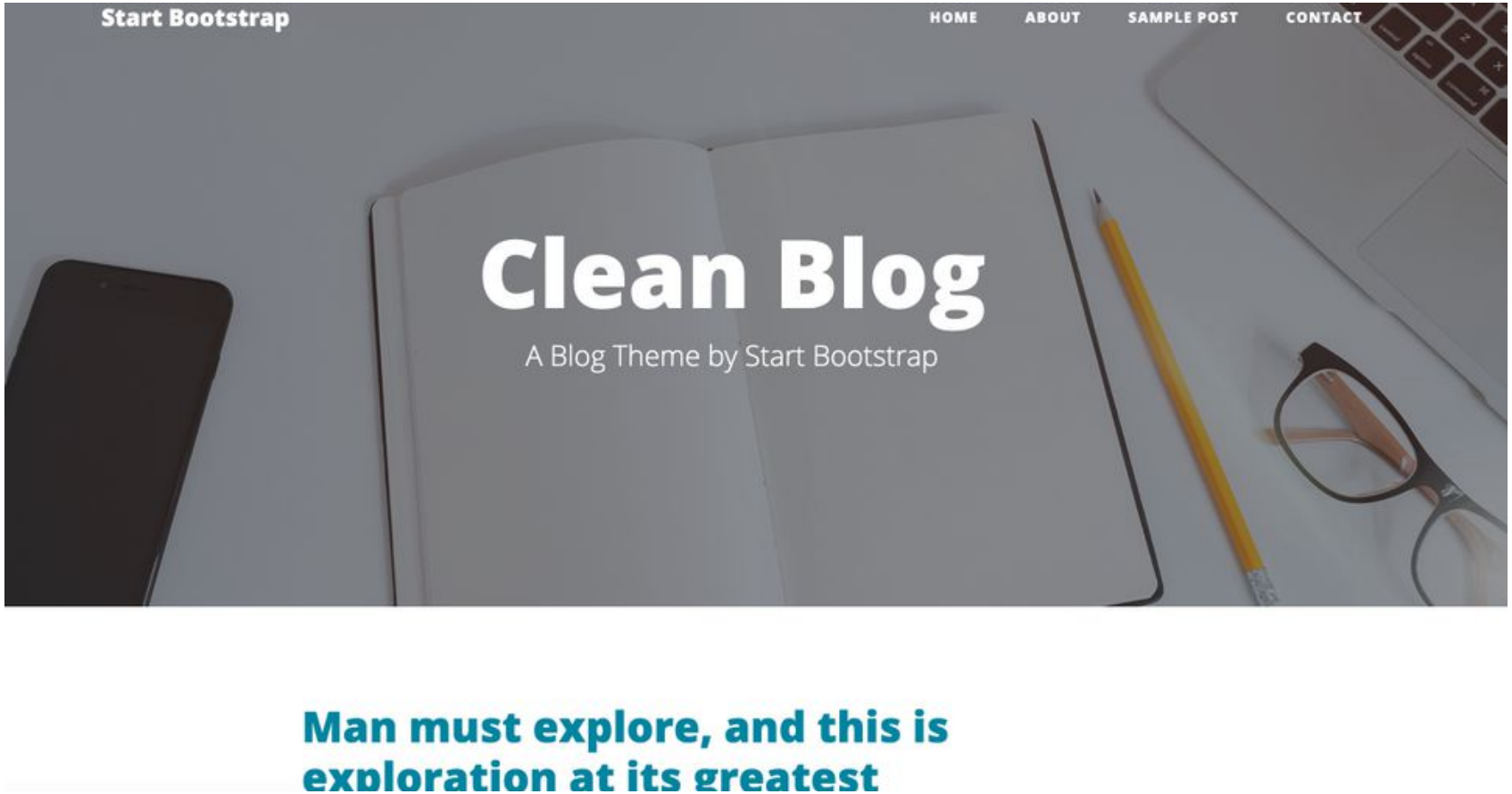
```
{% load static %}
<link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">
```

- **Added personalization**

  The request objects holds details of the logged in user, enabling custom aesthetics or conditional rendering based on their preferences

Let's Make It Flashy!

With the content of the previous slides, incorporate the template from Bootstrap



Solution

How to feed Bootstrap templates into Django

- **Copy all the folders vendor, js, img and css into your static folder**
- **Modify the routing accordingly**

  ie. *href="css/clean-blog.min.css" -> href="{% static 'css/clean-blog.min.css'%}"*

Oh! There Are No Posts

So how do we create content?

- **Django admin**

  It already provides us with a nice user interface to create new objects

- **Django forms**

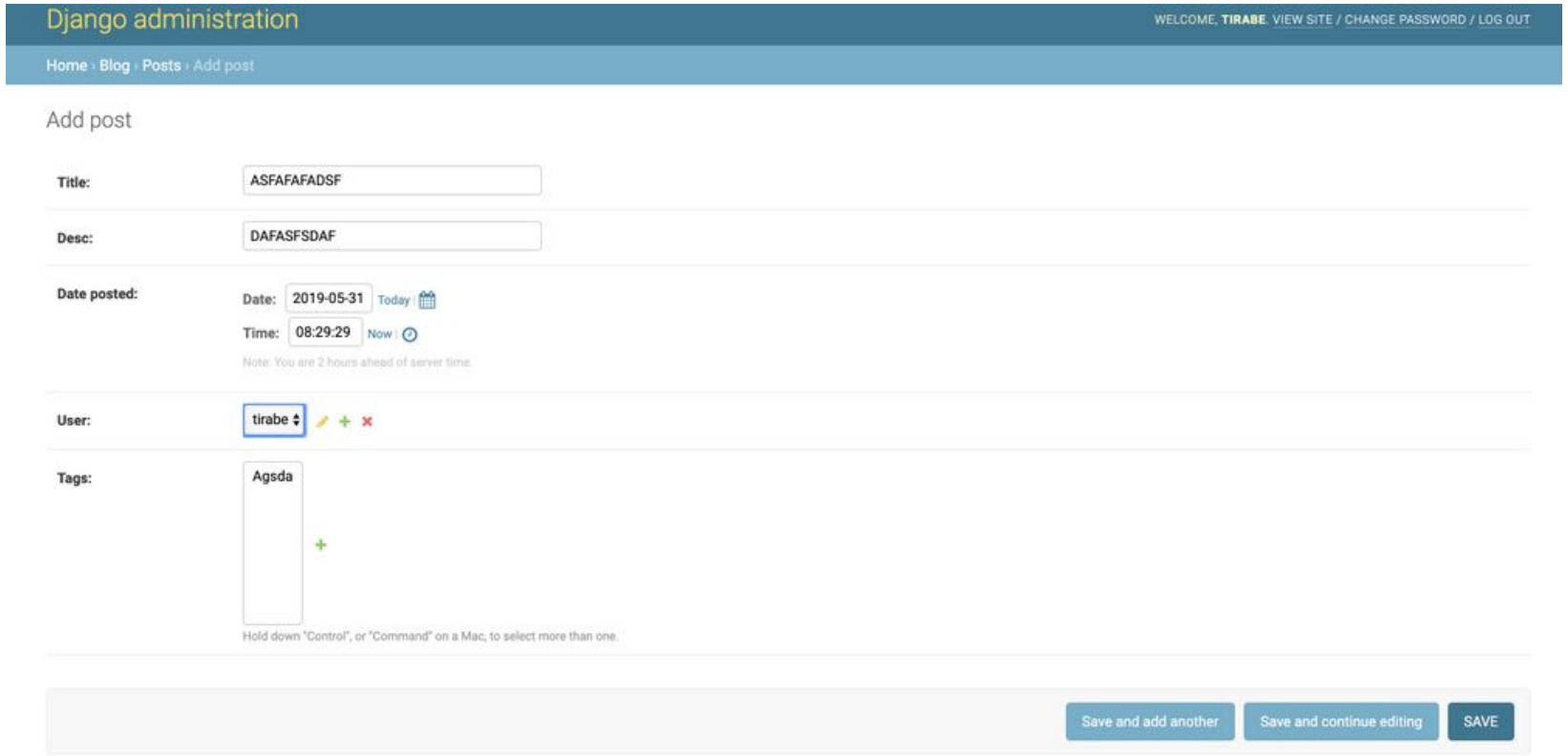
  Using the templating engine to create forms whose content gets recorded in the database

- **Django REST API**

  It is also possible to add elements by means of requests from the client side

Django Admin

The Django admin makes it easy to create new posts



Django Forms

Django provides an interface to generate forms from your models

- ***forms.py***

  In this file it is possible to configure the forms and to define custom validators for each field if necessary

- ***context***

  The context dictionary contains those objects or strings that we want to make available for the template specified in the render method

```
#! New code
from blog.models import Post

def index(request):
    #! New code
    post = Post.objects.all()
    user = request.user
    context = {"post": post, "user": user}
    return render(request, "index.html", context)
```

Django Forms

Django provides an interface to generate forms from your models

- ***forms.py***

  In this file it is possible to configure the forms and to define custom validators for each field if necessary

```
$ touch forms.py
$ cp templates/index.html templates/newpost.html

# We have to add a new path to our url router (urls.py):
urlpatterns = [
    path('', views.index, name="index"),
    # New line!
    path('/new_post', views.new_post, name="newpost")
]
```

- **class *Name(forms.Modelform)***

  Creates the class for a specific form. The model used as a template should be indicated in model and the fields requested can be specified

```
# Content to type in forms.py

from django import forms
from .models import Post

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ("title", "desc", "tags")
```

Forms in Your Templates

Incorporating the form in your site is as easy as mentioning it in your html template

- *{% csrf_token %}*

  To prevent cross site request forgery (CSFR), use this tag in any form that sends a POST request to an address belonging to your website

- **Submit button**

  Form buttons reload your page by default, preventing further logic from being executed. Make sure to add type="submit" class="save btn btn-default" to your form submit button

```
<form method="POST" class="post-form">{% csrf_token %}
    {% csrf_token %}
    <div class="post-form">
    {{ form.as_p }}
    </div>
    <button type="submit" class="save btn btn-default">Save</button>
</form>
```

Storing Content in DB

Views can be used to pass forms to the templating engine or to save them if a POST request happens

- ***http* Methods**

  The default implementation of a view responds to the GET method. By making use of the rest of http methods we can build rich and easier to interpret APIs

- ***save()***

  The save method is used to flush objects into the DB base. By setting the attribute commit to false, it is possible to put the object in a staging state in which it is still possible to modify it

```python
# In views.py of the app
from blog.forms import PostForm

def new_post(request):
    if(request.method == 'POST'):
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.user = request.user
            post.date_posted = datetime.now()
            post.save()
            return redirect("index")
    else:
        form = PostForm()
    return render(request, "newpost.html", {"form": form})
```

Linking Multiple Pages

It is possible to use URL routing in the HTML templates to redirect the user from one view to another. Ie. Sample redirect button

```
<a class="btn btn-primary float-right" href="{% url 'newpost' %}">Create new post! &rarr;</a>
```

Django REST

Allows you to generate a REST API effortlessly from the definition of your models

- **Make it cross-device**

  All your clients can integrate to the same back-end: A mobile app, an embedded system or a desktop browser are some examples

- **Powerful defaults**

  Django Rest generates endpoints for authentication and several other functionalities out of the box. Less hassle, more functionality

Initial Setup

To get the REST framework up and running it is required to modify the config.py file

- **Django packages**

  It is possible to add aditional functionalities to Django by simply pip installing and adding them to the INSTALLED_APPS array. One of the best sources is djangopackages.org

- **Default permissions**

  settings.py allows to set defaults to who can access the API endpoints. However, global permission defaults are encouraged only for testing purposes and these permissions should be defined in a per-view basis.

```
# Installing the corresponding python module
$ pipenv install djangorestframework
# Creating a new app to keep your project clean
$ python manage.py startapp api
# Creating new necessary files
$ touch api/serializers.py api/urls.py

# New content to add to your config.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Add line!
    'rest_framework',

    # Your apps
    'blog.apps.BlogConfig',
    # Add line!
    'api.apps.ApiConfig',
]

# Add block!
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ]
}
```

urls.py

In order to accommodate the router for the API

- **DRF router**

  When using it together with ViewSets, the DRF url router will take care of the path definitions for every endpoint of the API

- **Browsable API**

  DRF generates a default user interface for browsing your API. If the previous steps went through and your server is running, you should be able to visit it at localhost/api/v1

```
# In the urls.py of the whole project
urlpatterns = [
    path('admin/', admin.site.urls),
    path('blog/', include('blog.urls')),
    # New line!
    path('api/v1/', include('api.urls'))
]

# In the urls.py file of the api app
from django.urls import path, include
from rest_framework import routers

router = routers.DefaultRouter()

urlpatterns = [
    path("", include(router.urls))
]
```

serializers.py

Convert your python objects into JSON format

- **Base serializer**

  Create a serializer by inheriting from the serializers.ModelSerializer class. It is possible to define the desired fields and the model that the serializer will expect

- **Composition**

  It is possible to model the relationship amongst our objects in our JSON responses too. It is possible to nest serializers when including an object as a field of other. If we want to keep just some attribute of the nested object, such as the name, it is enough to use a SlugRelated field

- **Validation**

  DRF can also include validators in the serializers, for further reference visit the official docs

```python
# api/serializers.py
from django.contrib.auth.models import User
from rest_framework import serializers
from blog import models


class UserSerializer(serializers.ModelSerializer):
    class Meta:
        fields = ("username", "first_name", "last_name", "email")

        model = User


class PostSerializer(serializers.ModelSerializer):
    user = UserSerializer(read_only=True)
    tags = serializers.SlugRelatedField(many=True, read_only=True,
                                        slug_field="name")
    date_posted = serializers.DateTimeField(read_only=True)

    class Meta:
        fields = ("id", "title", "date_posted", "user", "content",
                  "tags")
        model = models.Post
```

views.py

Using ViewSets can save us time and nerves

- **ViewSets**

  Django allows to group the logic associated to a set of related views under a ViewSet. Their main advantages are the reduction of code duplication and the automatic URL routing

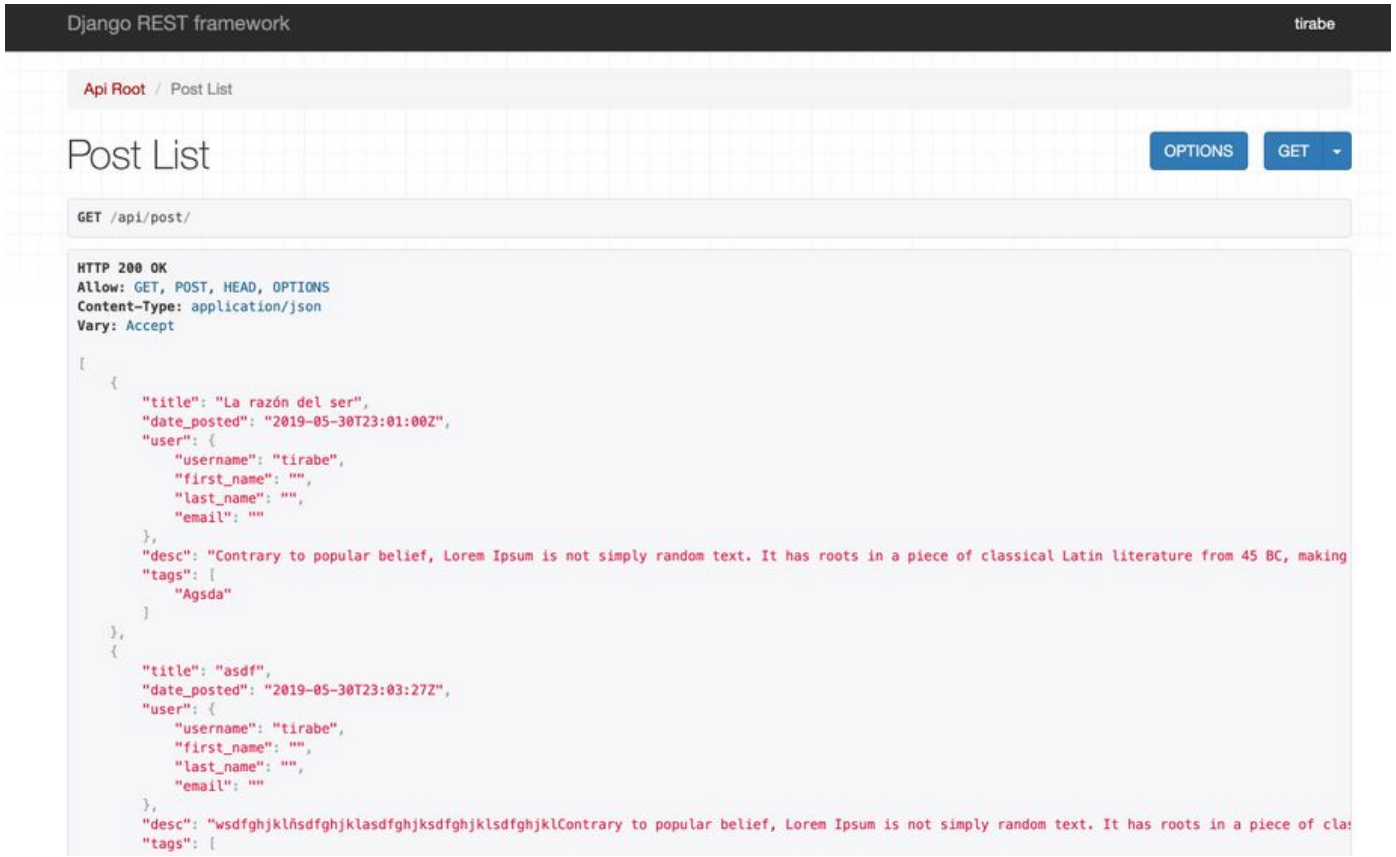
- ***viewsets.ModelViewSet***

  To create a new ViewSet, indicate the query to be displayed and the serializer that applies to the objects retrieved. The query can be narrowed down by using the Django QuerySet API

```python
# Add to views.py of the API
from rest_framework import viewsets

from api.serializers import PostSerializer
from blog.models import Post


class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

Testing

Now our *api/v1/post/* endpoint should respond to GET responses



POST Method

By default the ViewSet endpoint is read only, we should specify the expected behaviour on POST requests

- ## ViewSets

  Django allows to group the logic associated to a set of related views under a Viewset. Their main advantages are the reduction of code duplication and the automatic url routing

- ## ViewSet methods

  Contrary to APIViews, where you define the type of request received, Viewsets rely on a set of methods, that can be found here. When dealing with complex models (with ForeignKeys or Many to many relationships) it's necessary to redefine the methods create() and update().

```python
class PostSerializer(serializers.ModelSerializer):
    user = UserSerializer(read_only=True)
    tags = serializers.SlugRelatedField(many=True, slug_field="name", read_only=True)
    date_posted = serializers.DateTimeField(read_only=True)

    # New block!
    def create(self, context):
        # **operator includes the key/value pairs as arguments instead of
        # the dictionary object
        post = Post(user=self.context["request"].user, **context)
        post.save()
        return post
```

Testing POST Creation

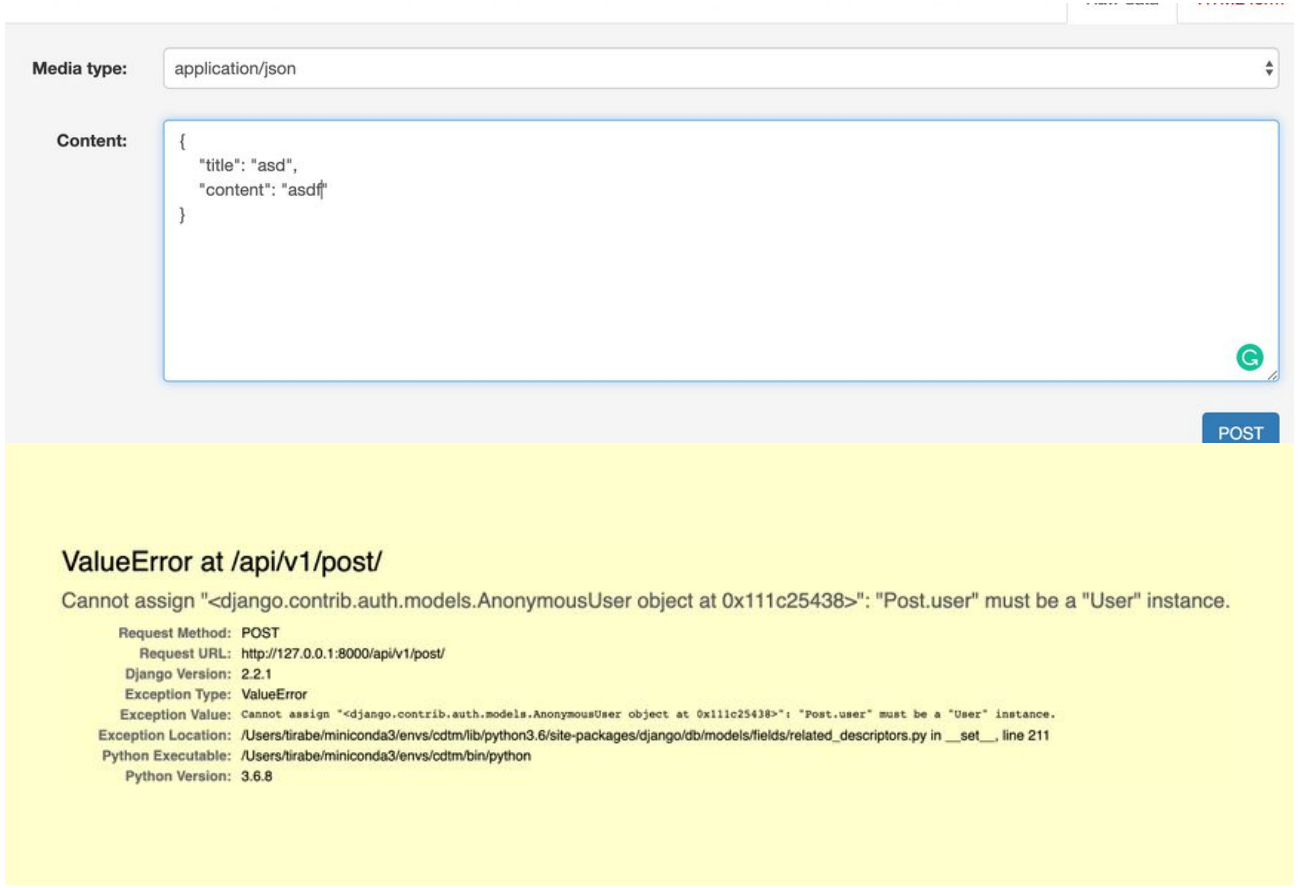
The visual API explorer provides an interface to make requests

- **The API browser**

  Apart from displating queries, the browser allows to perform different kinds of requests as an user will when using your application

- **Need for authentication**

  We defined the previous view in a way that every Post is associated to the user created it. When no user is logged in, the user attached is an anonymous object



Token Authentication

By using a third party app *django-rest-auth*, obtention of tokens becomes straightforward

- ***django-rest-auth***

  This packages provides by default login and logout API endpoints

- **Point to the urls**

  Include the urls directory of the package and it will take care of the routing

```
# To install the python module, introduce in bash:
$ pip install django-rest-auth

# In the settings.py, add the lines:
INSTALLED_APPS = [
    'rest_framework.authtoken', # New line!
    'rest_auth' # New line!
]
```

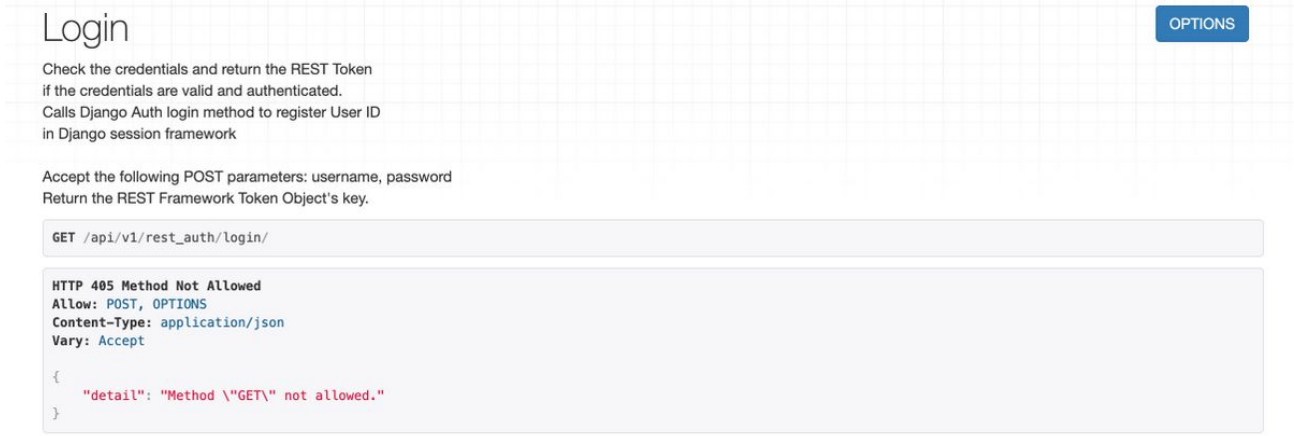
```
from django.urls import path, include
from rest_framework import routers
from . import views

router = routers.DefaultRouter()
router.register("post", views.PostViewSet)

urlpatterns = [
    path("", include(router.urls)),
    # New line!
    path('rest-auth/', include('rest_auth.urls')),
]
```

Token Authentication

By using a third party app *django-rest-auth*, obtention of tokens becomes straightforward

- **Login and logout**

  The new endpoints include login and logout ones, together with a password change endpoints, for more info visit the official site



- **Admin token management**

  After generation, the tokens will be stored and accessible through the admin interface

Setting Permissions Straight

Authentication allows to give a more fine grained access to the API

- **Global permissions**

  In the settings.py file we should set permissions to IsAuthenticated

- **ViewSet permissions**

  We can also specify the permissions on a viewset basis. Moreover, customized permission configurations can be created. Click here for more info

```
# In settings.py of your project

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
      # Line changed!
        'rest_framework.permissions.AllowAny',
    ]
}
```

```
# In views.py of the API, add these lines:
from rest_framework.permissions import IsAuthenticated, IsAdminUser


class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
    # Add this method:
    def get_permissions(self):
        """
        Instantiates and returns the list of permissions that this view requires.
        """
        if self.action == "list":
            permission_classes = [IsAuthenticated]
        else:
            permission_classes = [IsAdminUser]
        return [permission() for permission in permission_classes]
```

Outlook

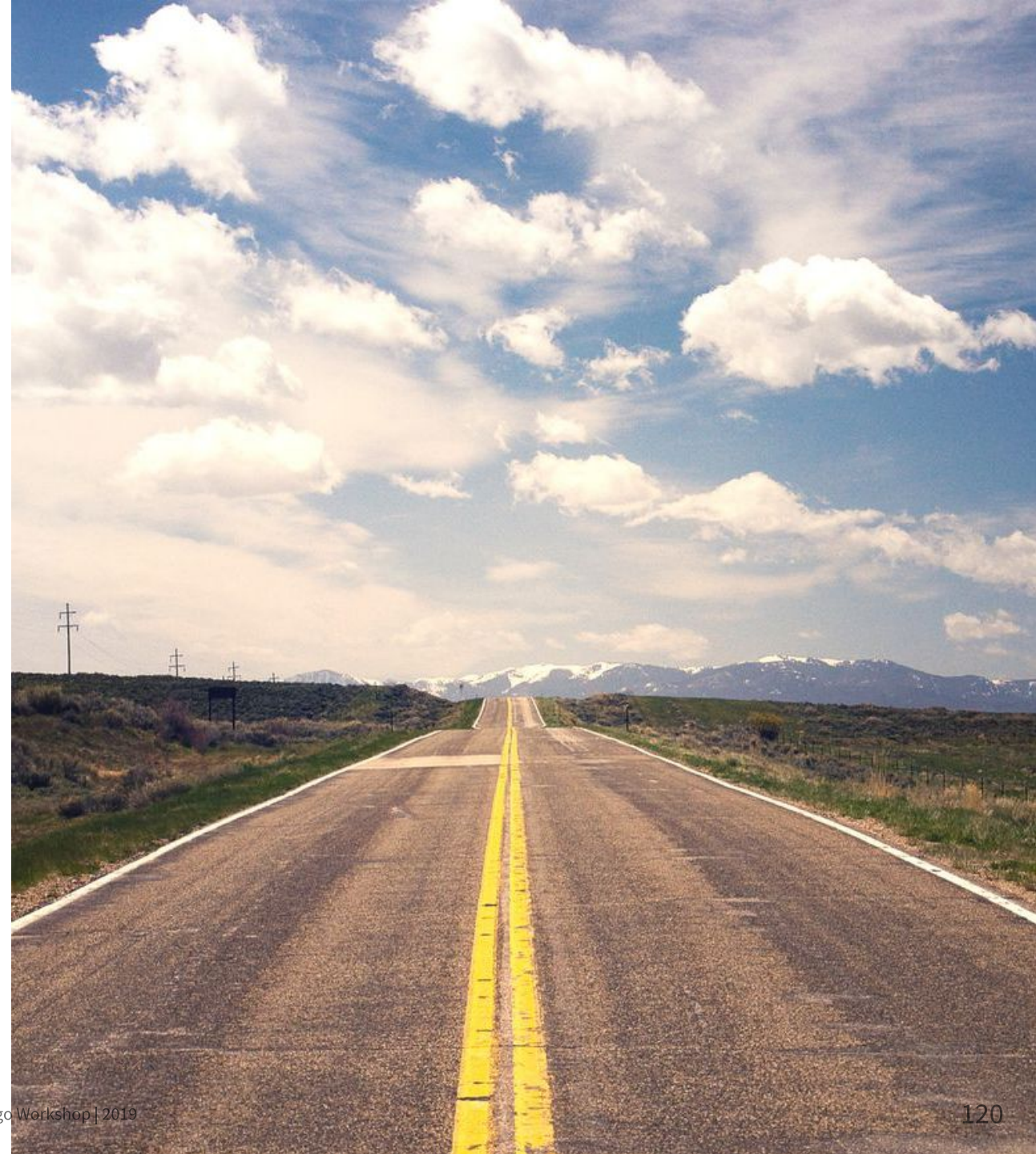
Now what?

- ## Connecting React and Django

  Django REST api will respond to requests initiated in the client side by a React.js powered SPA

- ## Deployment

  Make your back-end available to the world. The most straightforward solution is using Heroku. More personalization and server administration can be achieved by using GCloud or Digital Ocean



Resources

Continue learning Django after the workshop

Try Django Series

Exceptional YouTube channel, clean explanations

Link

Bootstrap

Awesome layouting and templates

Link

PostgreSQL

Database for production environments

Link

Deployment

Achieve one click deployment with Heroku

Link

Micro Services Arch

Splitting your app in docker containers makes it reproducible

Link

Awesome Django

Great resources put together by the community

Link

Even More Resources

Additional content to keep on learning after the workshop

Curated Resources

We created a file with additional resource

Link

Django Udemy

Full Udemy course on Django available on CDTM Home

Link

React Udemy

Full Udemy course on React available on CDTM Home

Link

Thank You!

Feel free to reach us out via Slack or e-mail

Mihai Babiac

Fall 2018

Electrical Engineering and Information Technology at TUM

mihai.babiac@cdtm.de

Kai Siebenrock

Fall 2018

Management and Technology at TUM

kai.siebenrock@cdtm.de

Raul Berganza Gómez

Spring 2019

Computational Science and Engineering at TUM

raul.berganza@cdtm.de

Create Code Snippets

To show source code images in this presentation

Carbon

Use this link to create images of code snippets as used throughout this presentation

Set correct language

Styling is pre-defined, export as .png

Documentation

Congratulations

The journey just started, excited to see what amazing tech you build!

Thanks for taking part!