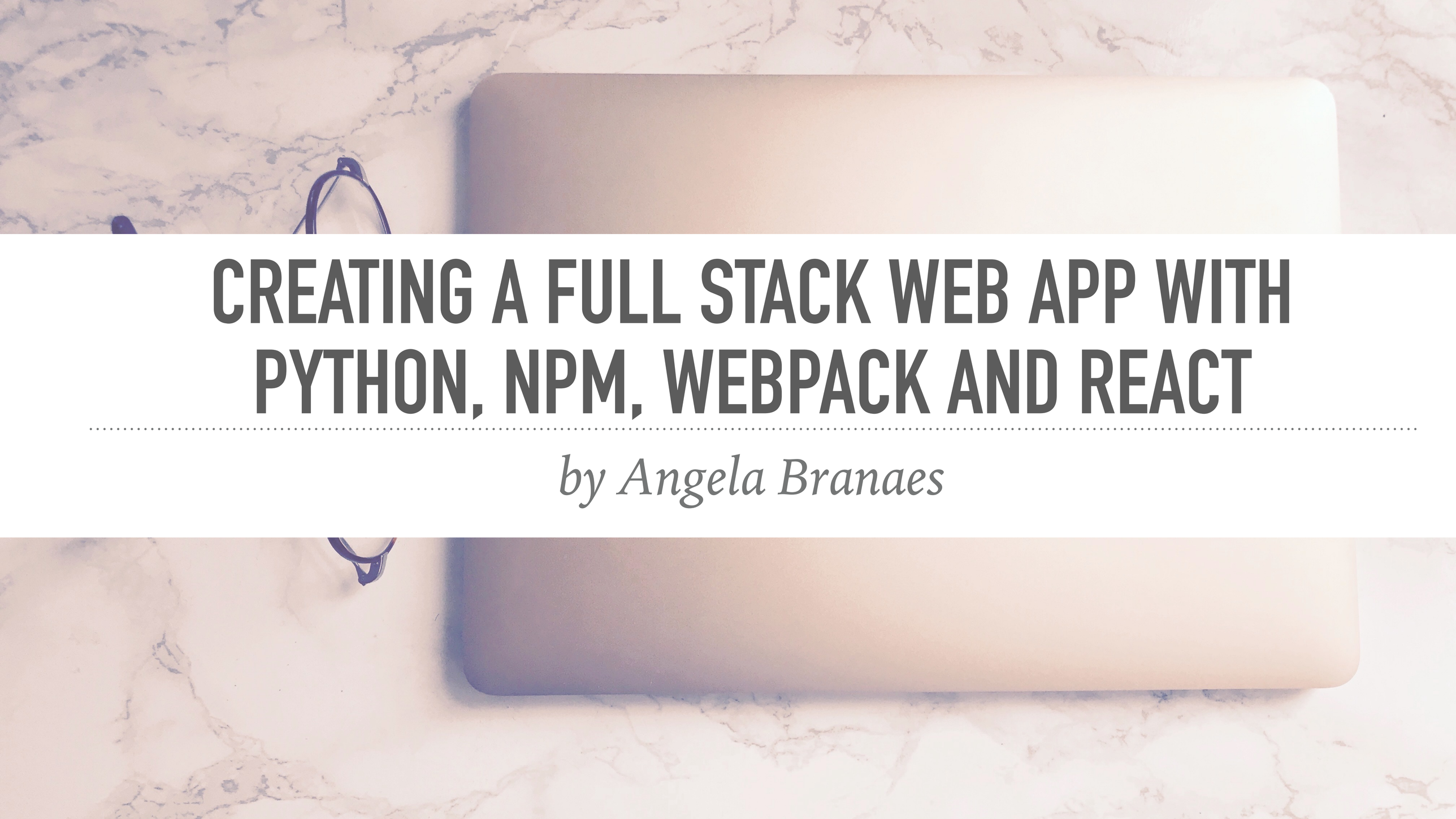


CREATING A FULL STACK WEB APP WITH PYTHON, NPM, WEBPACK AND REACT

by Angela Branaes

Hello Rimini

Say Hello!

Salut Rimini!

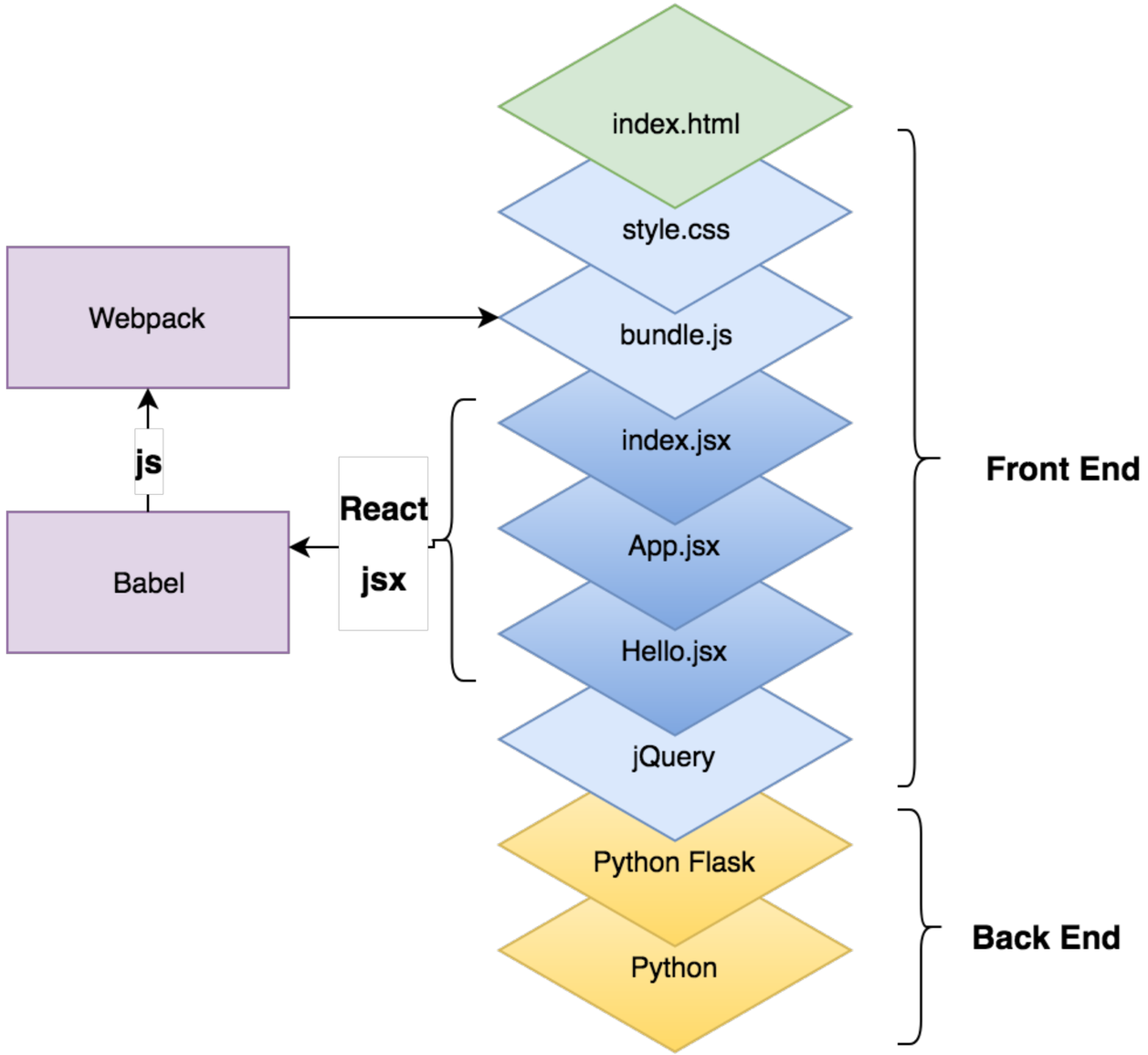
Say Hello!

Hei Rimini!

Say Hello!

THE STRUCTURE OF A WEB APP

- Separated into front and back end.
- Multiple technologies.
- Front end is:
  - Look and feel.
  - User interaction.
- Back end is:
  - Long running operations.
  - Data, and data processing.
  - Application logic.



INITIAL PROJECT SETUP

- Create the following directory structure

```
.
├── README.md
└── fullstack_template/
    ├── server/
    └── static/
        ├── css/
        ├── dist/
        ├── images/
        └── js/
```

```
$ cd fullstack_template/static
```

WE'RE GOING TO NEED A JAVASCRIPT PACKAGE MANAGER

- Using a package manager makes it painless to:
  - Keep your project dependencies up to date.
  - Fetch and install new packages.
- Similar to brew, pip, pacman etc.

NPM

NPM = NODE PACKAGE MANAGER

- Automatically included with Node.js
- Easy to use
- well documented
- nearly 500 000 packages available

- Initialise NPM:

```
$ npm init
```

- This creates a file called package.json

OUR UPDATED DIRECTORY TREE

```
.
├── README.md
└── fullstack_template/
    ├── server/
    └── static/
        ├── css/
        ├── dist/
        ├── images/
        ├── js/
        └── package.json
```

THE PURPOSE OF PACKAGE.JSON

1. Keeps track of dependencies and versions.

2. Informs other developers about your project.

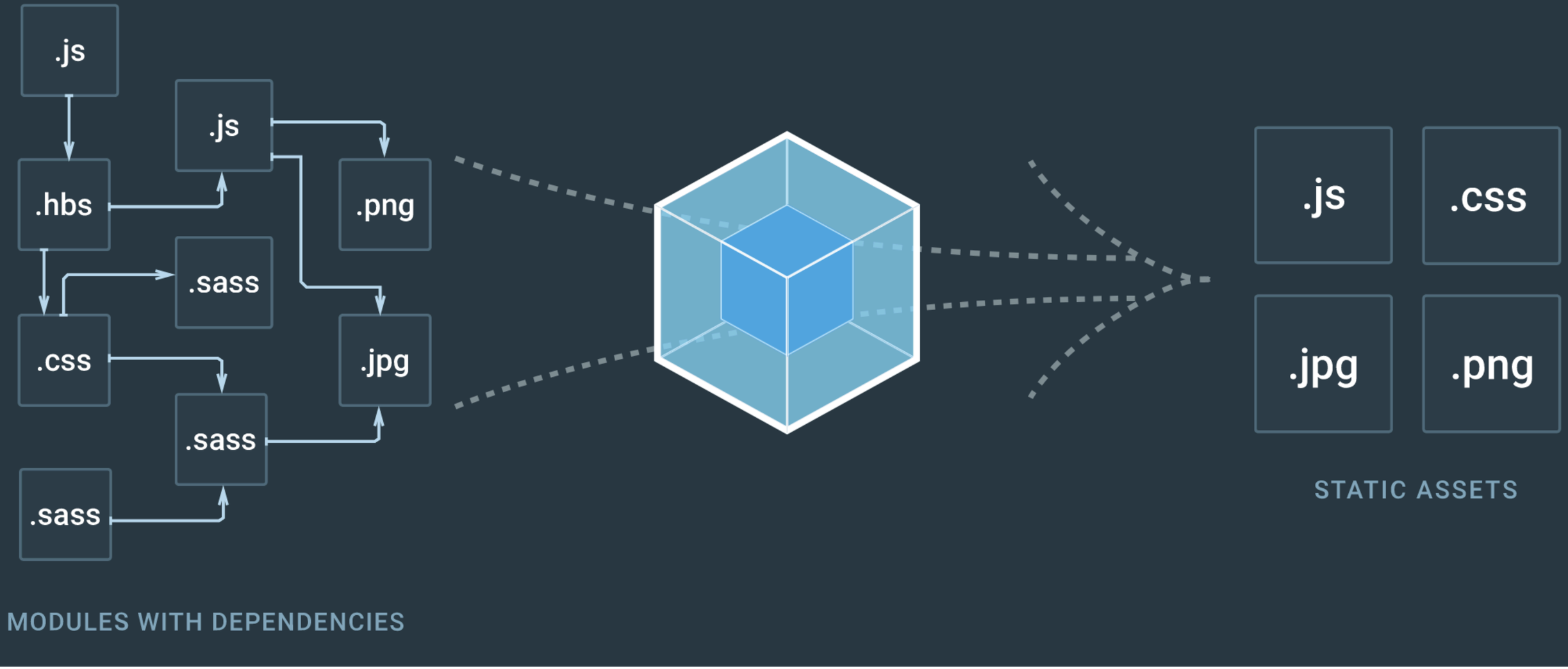
3. Makes installing, running and updating a project automatic and reproducible

THIS IS MY PACKAGE.JSON

```
{
  "name": "FullStackTemplate",
  "version": "1.0.0",
  "description": "Fullstack Template",
  "main": "index.jsx",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
  },
  "keywords": [
    "python",
    "react",
    "npm",
    "webpack"
  ],
  "author": "Angela Branaes",
  "license": "MIT",
  "devDependencies": {
    "webpack": "^3.0.0"
  }
}
```

WHY WE NEED A MODULE BUNDLER

- Minimises the number of <script> tags in your HTML
  - This means faster loading, and less hassle.
- Don’t need to worry about bundle import ordering!
- Don’t have to use globals or namespacing.
- Lazy module loading.
- We will be using Webpack.


.js
.js
.hbs
.png
.sass
.css
.jpg
.sass
.sass
MODULES WITH DEPENDENCIES
.js
.css
.jpg
.png
STATIC ASSETS

WEBPACK

WEBPACK

- ***Only*** understands JavaScript.
  - Everything else needs a loader or plugin.
- Everything is a module.
  - You can require() JavaScript, React, CSS or images.
- Creates a dependency tree of all your required modules, and packages those into bundles.
- Easy to get started with, if you only need JavaScript.
  - Minor learning curve for css/less and image loading.
- Lets you split your bundles and load them asynchronously and lazily.
- Integrates seamlessly with npm.

INSTALL & CONFIGURE WEBPACK

- Install webpack:

```
$ npm i webpack --save-dev
```

- Add a file named webpack.config.js:

```
const webpack = require('webpack');

const config = {
    entry:  __dirname + '/js/index.jsx',
    output: {
        path: __dirname + '/dist',
        filename: 'bundle.js',
    },
    resolve: {
        extensions: ['.js', '.jsx', '.css']
    },
};
module.exports = config;
```

OUR UPDATED DIRECTORY TREE

```
.
├── README.md
└── fullstack_template/
    ├── server/
    └── static/
        ├── css/
        ├── dist/
        ├── images/
        ├── js/
        ├── package.json
        └── webpack.config.js
```

SINGLE VS MULTI PAGE APPLICATIONS

- I mostly use Webpack for *single* page applications.
- You can also use webpack for *multi* page applications:

```
entry: {
    “indexPage”: __dirname + “js/index.jsx”,
    “aboutPage”: __dirname + “js/about.jsx”
},
output: {
    path: __dirname + “/dist”,
    filename: “[name].js” //e.g. aboutPage.js
}
```

LET'S DISPLAY AN ALERT!

OUR UPDATED DIRECTORY TREE

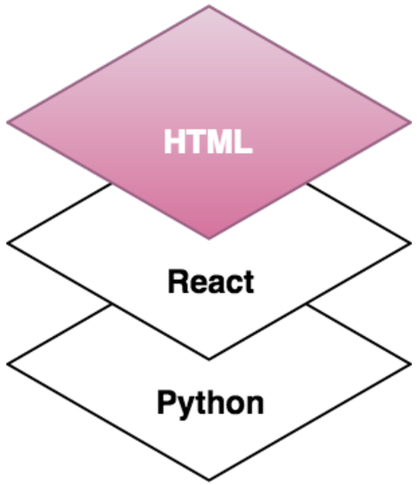
```
.
├── README.md
└── fullstack_template/
    ├── server/
    └── static/
        ├── css/
        ├── dist/
        │   └── bundle.js
        ├── images/
        ├── js/
        │   └── index.jsx
        ├── index.html
        ├── package.json
        └── webpack.config.js
```

CREATE INDEX.HTML

- A really simple index.html is all you need.
- The index.html just loads your JavaScript bundle

```
  <body>
    <div id="content" />
    <script src="dist/bundle.js" type="text/javascript">
    </script>
  </body>
```

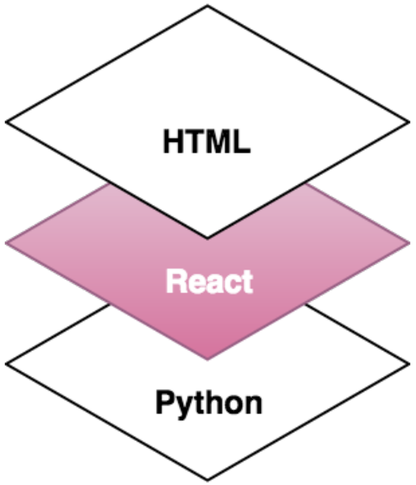
- *All the layout and behaviour live in the JS/React files!*



CREATE INDEX.JSX

- Just 1 line of Plain Old JavaScript

```
alert(“Hello World!”);
```



BUILDING YOUR APPLICATION

- Create run commands to simplify building your code.
- Run commands are just aliases.
- Example from my package.json:

```
“scripts": {
  "watch": "webpack --progress -d --config webpack.config.js -watch"
}
```

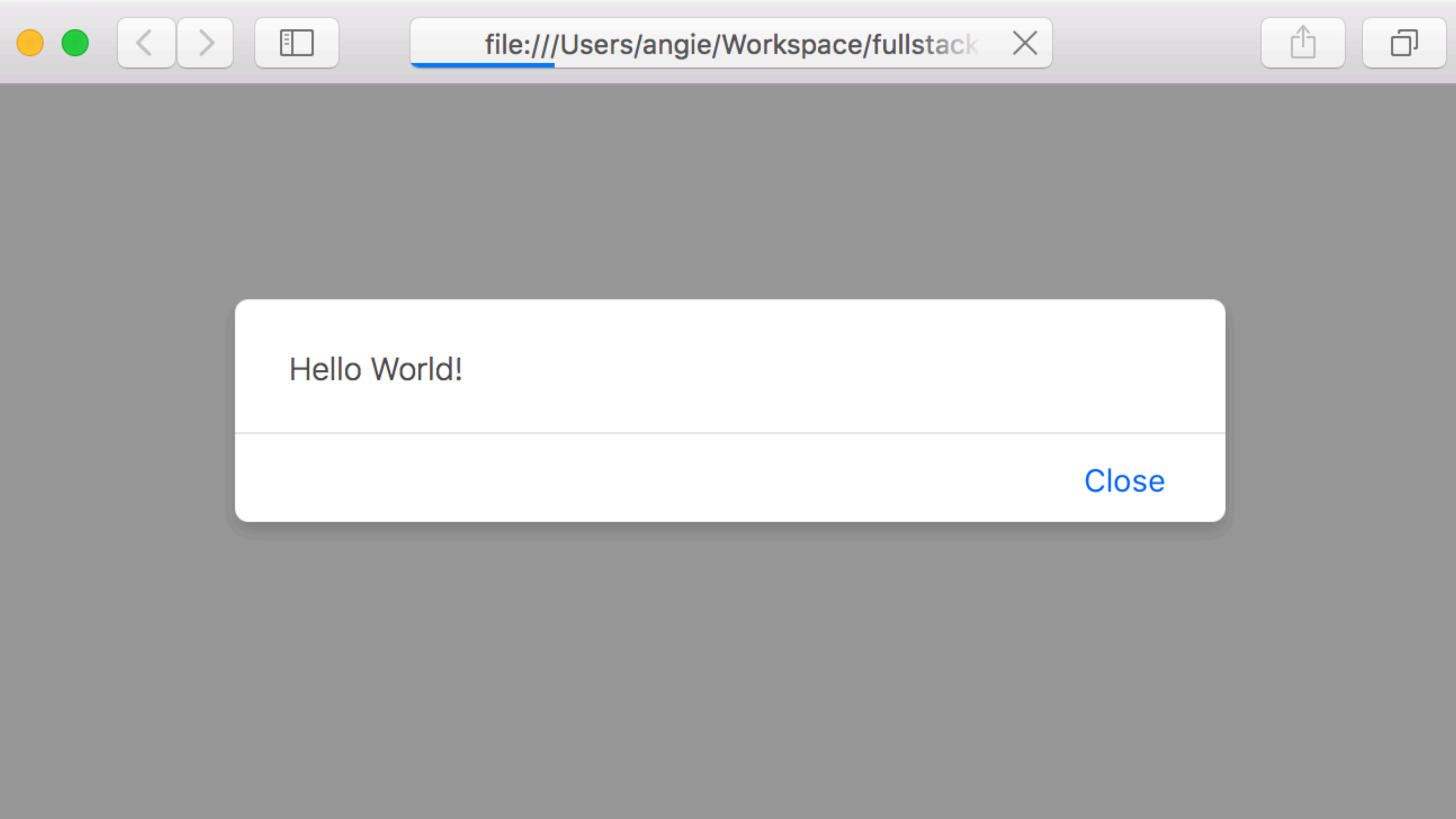
- Makes the dev process more fluid.
- I always add the following:
  - build
  - dev-build
  - watch

START THE WATCH COMMAND

- Any changes get built automatically

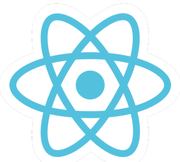
```
$ npm run watch
```

- Open index.html….


file:///Users/angie/Workspace/fullstack
Hello World!
Close

CREATING A SIMPLE REACT APP

THE BENEFITS OF USING REACT



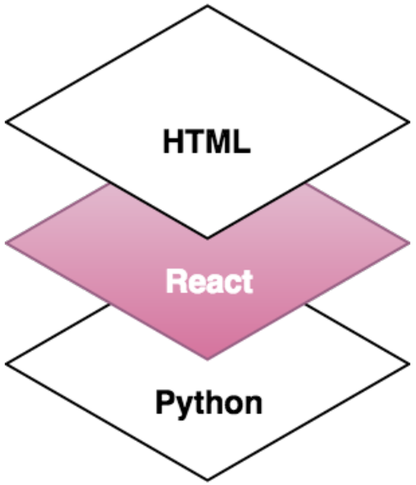
- Easy to build UIs composed of small, distinct components.
- Encourages design of reusable UI components.
- Easier to understand what's going on when markup lives with the code (JSX).
- **Automatic re-rendering of components on change.**
- Easy to maintain.

INSTALL REACT

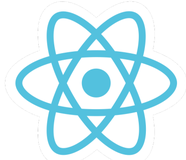
```
$ npm i react react-dom –save-dev
```

REACTIFY INDEX.JSX

```
import React from "react";
import ReactDOM from "react-dom";
import App from “./App";

ReactDOM.render(<App />, document.getElementById("content"));
```

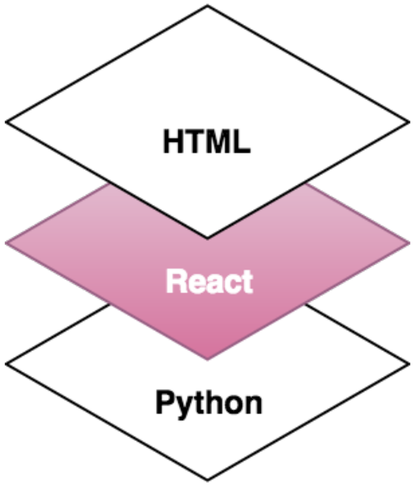


ADD A REACT “APP” CLASS

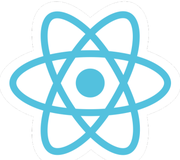


- Remember to export your react classes!

```
// App.jsx
import React from “react”;

export default class App extends React.Component {
   render () {
      return <p> Hello React!</p>;
   }
}
```

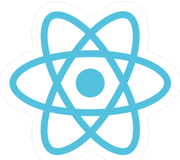


OUR UPDATED DIRECTORY TREE



```
.
├── README.md
└── fullstack_template/
    ├── server/
    └── static/
        ├── css/
        ├── dist/
        ├── images/
        ├── js/
        │   ├── index.jsx
        │   └── App.jsx
        ├── index.html
        ├── package.json
        └── webpack.config.js
```

WHAT IS JSX?



- Syntax extension to JavaScript.
- Optimised on compilation, so faster than JavaScript.
- Statically typed and mostly type-safe. JavaScript is not.
- Lets you write HTML tags in your JavaScript functions:

```
<Hello name=“Rimini” />
```

instead of

```
React.createElement(Hello, {name: “Rimini}, null)
```

- Recognises upper-case element-types as written in React. E.g. <Hello />
- How do we make our browser understand JSX?

INTRODUCING BABEL

Put in next-gen JavaScript

```
[1, 2, 3].map(n => n ** 2);
```

Get browser-compatible JavaScript out

```
[1, 2, 3].map(function (n) {
  return Math.pow(n, 2);
});
```

Write next-generation javascript *right now!*

Transform JSX to JS.

ADD BABEL

- Install Babel
- Add the Babel presets to the package.json:

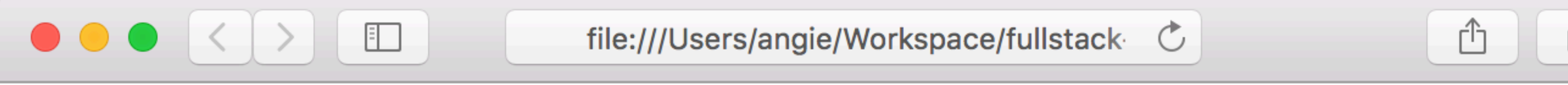
```
“babel”: {
  “presets”: [
    “es2015”,
    “react”
  ]
},
```

ADD A BABEL-LOADER RULE TO THE WEBPACK CONFIG:

```
module: {
  rules: [
    {
      test: /\.jsx?/,
      exclude: /node_modules/,
      use: 'babel-loader'
    }
  ]
}
```

OPEN INDEX.HTML IN YOUR BROWSER

- This should now show the “Hello React” paragraph we added in our new React App.jsx file.



Hello React!

PYTHON FLASK SERVER

PYTHON

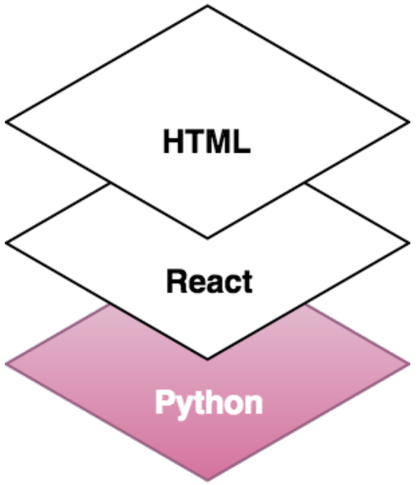
- Go to the server folder
- Ensure you're in a Python virtualenv

```
$ mkvirtualenv fullstack
```

- Install Flask

```
(fullstack)$ pip install flask
```

- Create a file called server.py in the server folder

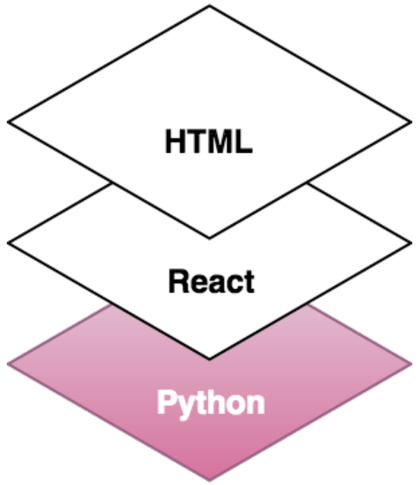


OUR UPDATED DIRECTORY TREE

```
.
├── README.md
└── fullstack_template/
    ├── server/
    │   └──  server.py
    └── static/
        ├── css/
        ├── dist/
        ├── images/
        ├── js/
        │   ├── index.jsx
        │   └── App.jsx
        ├── index.html
        ├── package.json
        └── webpack.config.js
```

SERVER.PY

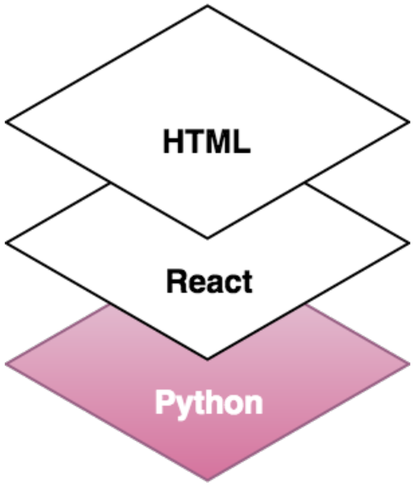
```
from flask import Flask, render_template


app = Flask(__name__, static_folder="../static/dist",
            template_folder="../static")


@app.route("/")
def index():
    return render_template("index.html")


@app.route("/hello")
def hello():
    return "Hello World!"


if __name__ == "__main__":
    app.run()
```



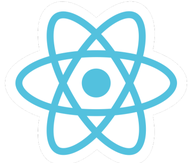
START THE SERVER

```
$ python server.py
```

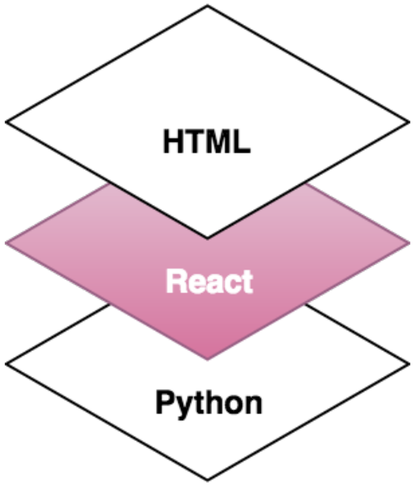
- Go to: http://localhost:5000/



REQUEST INFO FROM THE SERVER



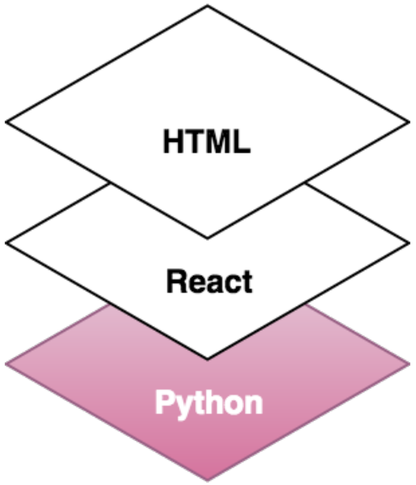
```
var $ = require('jquery');


getPythonHello() {
        $.get(window.location.href + 'hello', (data) => {
            console.log(data);
            this.personaliseGreeting(data);
        });
 }
```



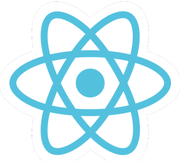
MAKE PYTHON DO SOMETHING MORE INTERESTING

- We call get_hello() whenever we hit the /hello endpoint

```
def get_hello():
    greeting_list = ['Ciao', 'Hei', 'Salut', 'Hola', 'Hallo', 'Hej']
    return random.choice(greeting_list)
```

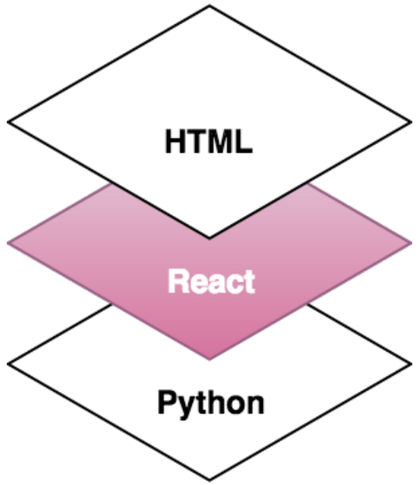


SAY HI TO SOMEONE SPECIAL

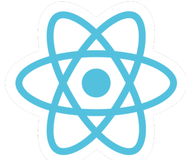


- There should be a class for that!
- Change the App.jsx render method to the following:

```
render () {
        return (
            <PageHeader>
                <div className='header-contents'>
                <Hello name='Rimini' />
                </div>
            </PageHeader>
        );
)
```

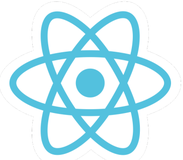


OUR UPDATED DIRECTORY TREE

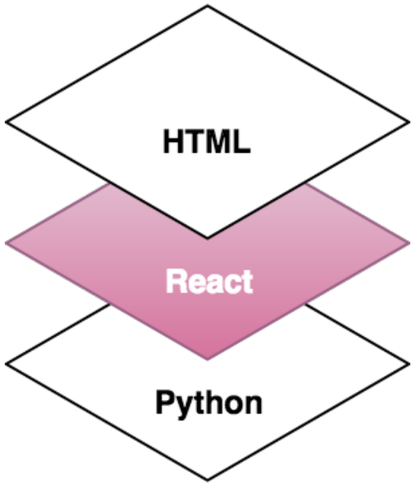


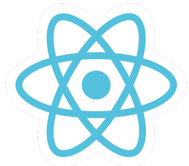
```
.
├── README.md
└── fullstack_template/
    ├── server/
    │   └──  server.py
    └── static/
        ├── css/
        ├── dist/
        ├── images/
        ├── js/
        │   ├── index.jsx
        │   ├── App.jsx
        │   └── Hello.jsx
        ├── index.html
        ├── package.json
        └── webpack.config.js
```

CREATE THE HELLO CLASS



```
export default class Hello extends React.Component {
    constructor(props) {
        super(props);
        // greeting is now “Hello Rimini”
        this.state = {greeting: 'Hello ' + this.props.name};
        // This binding is necessary to make `this`
        // work in the button callback
        this.getPythonHello = this.getPythonHello.bind(this);
    }
}
```

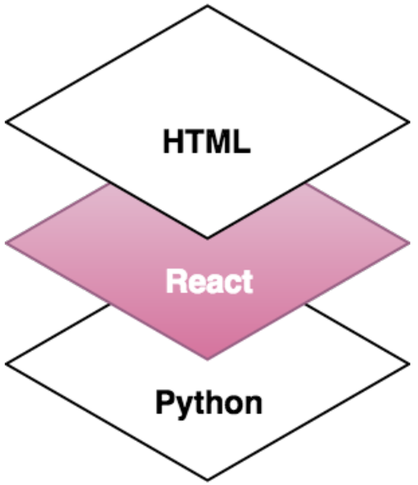




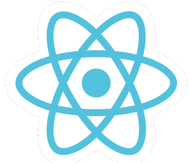
- Add this function to the Hello class:

```
personaliseGreeting(greeting) {
    this.setState({greeting: greeting + ' ' + this.props.name + '!'});
}
```

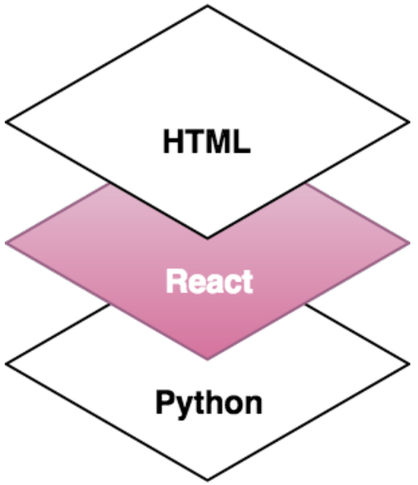
- This will re-render the greeting on our website to a new one when called.



LET'S FINALLY RENDER OUR HELLO!



```
render () {
        return (
                <h1>{this.state.greeting}</h1>
                <hr/>
                <Button onClick={this.getPythonHello}>
                    Say Hello!
                </Button>
        );
}
```



Hello Rimini

Say Hello!

Ciao Rimini!

Say Hello!

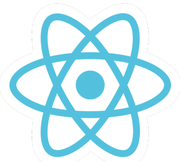
CREATE A NICE LAYOUT WITH CSS

- Webpack ONLY understands JavaScript.
- Install the following loaders and plugins:
  - style-loader
  - css-loader
  - extract-text-webpack-plugin
- Add a plugin to the webpack config:

```
    plugins: [
        new ExtractTextPlugin('styles.css'),
    ]
```

OUR UPDATED DIRECTORY TREE

```
.
├── README.md
└── fullstack_template/
    ├── server/
    │   └──  server.py
    └── static/
        ├── css/
        │   └── fullstack.css
        ├── dist/
        │   ├── bundle.js
        │   └── styles.css
        ├── images/
        ├── js/
        │   ├── index.jsx
        │   ├── App.jsx
        │   └── Hello.jsx
        ├── index.html
        ├── package.json
        └── webpack.config.js
```



- Add fullstack.css to the css folder
  - Add a few css rules for our header

- require fullstack.css in **App.jsx**:

```
require('../css/fullstack.css');
```

- Add the bundled css to the **index.html**:

```
  <link rel="stylesheet" href="dist/styles.css">
```

Hello Rimini

Say Hello!

OUR UPDATED DIRECTORY TREE

```
.
├── README.md
└── fullstack_template/
    ├── server/
    │   └──  server.py
    └── static/
        ├── css/
        │   └── fullstack.css
        ├── dist/
        │   ├── bundle.js
        │   └── styles.css
        ├── images/
        │   └── header.png
        ├── js/
        │   ├── index.jsx
        │   ├── App.jsx
        │   └── Hello.jsx
        ├── index.html
        ├── package.json
        └── webpack.config.js
```

ADD A NICE BACKGROUND IMAGE

1. Add the file-loader loader
2. Add an image to your images/ folder
3. Set the image to be your header background in the fullstack.css file

```
    background-image: url('../images/header.jpg');
```

4. **NOTE: you *have* to load the image in your React app for it to show up!**

IN APP.JSX

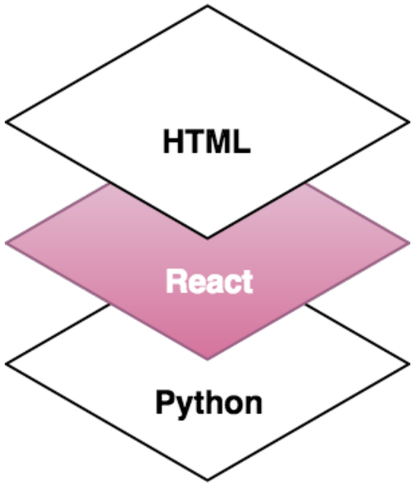
```
import HeaderBackgroundImage from '../images/header.jpg';
```

- Add this fn to your class:

```
addHeaderImg() {
        let headerBg = new Image();
        headerBg.src = HeaderBackgroundImage;
}
```

- And this to your render():

```
{this.addHeaderImg()}
```



Hello Rimini

Say Hello!

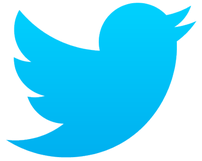
Salut Rimini!

Say Hello!

Hei Rimini!

Say Hello!

THANK YOU! ANY QUESTIONS?

 @angineering

 @angineering - The code is on GitHub: https://github.com/angineering/FullStackTemplate

 @angineering - This talk is also a blogpost