Front-End

ReactJs Overview and Walkthrough

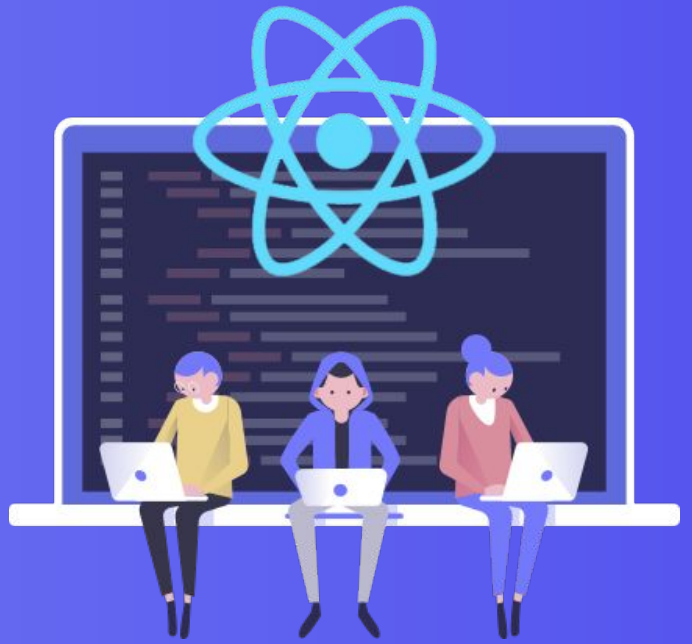
Hello World!

I am **Yacine Kharoubi**, final year student at **ESI-SBA** and **front-end developer** at Lasting Dynamics.

01.

About the session

Quick introduction to today’s session.



Prerequisites

There are a few things you should know in advance before you start playing around with React. If you've never used JavaScript or the DOM at all before, for example, I would get more familiar with those before trying to tackle React.

Here are what I consider to be React prerequisites:

- Basic familiarity with **HTML & CSS**.
- Basic knowledge of JavaScript and programming.
- Basic understanding of the **DOM**.
- Familiarity with **ES6 syntax and features**.
- **Node.js and npm** installed globally.

What’s React ?

- **React is a JavaScript library - one of the most popular ones, with over 200,000 stars on GitHub.**
- **React is not a framework (unlike Angular, which is more opinionated).**
- **React is an open-source project created by Facebook.**
- **React is used to build user interfaces (UI) on the front end.**
- **React is the view layer of an MVC application (Model View Controller)**

One of the most important aspects of React is the fact that you can create **components**, which are like custom, reusable HTML elements, to quickly and efficiently build user interfaces. React also streamlines how data is stored and handled, using **state** and **props**.

02.

Getting Started

Setup and Installation

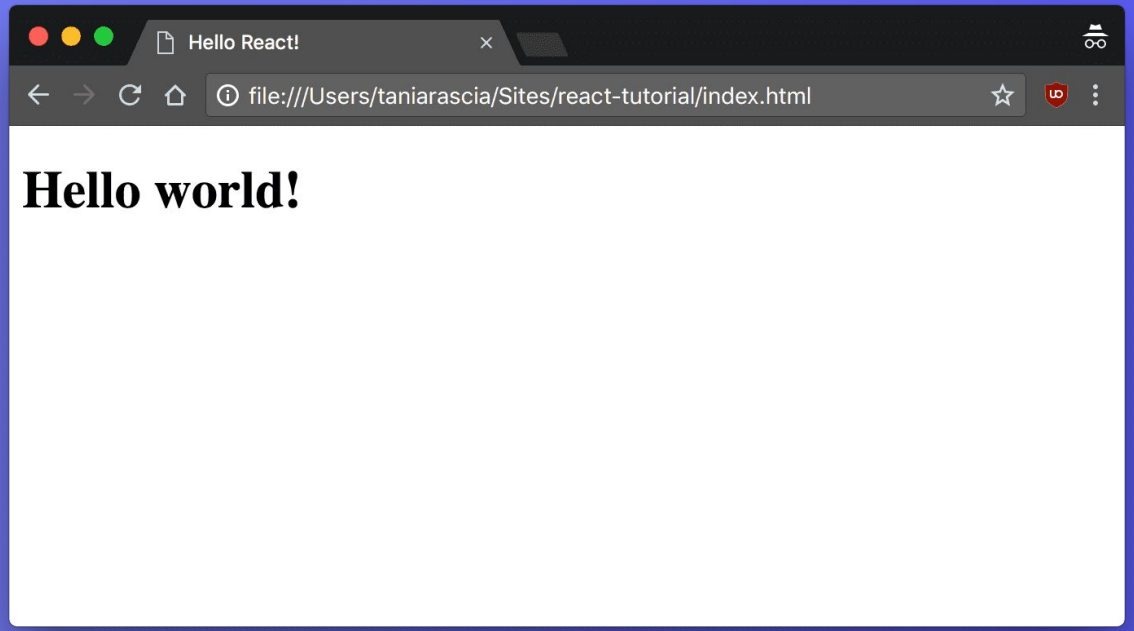
Static HTML file

This first method is not a popular way to set up React and is not how we'll be doing the rest of our tutorial.

Let's start by making a basic **index.html** file. We're going to load in three **CDNs** in the head - **React**, **React DOM**, and **Babel**. We're also going to make a **div** with an **id called root**, and finally we'll create a **script** tag where your custom code will live.

```
//Static HTML File
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />

    <title>Hello React!</title>

    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
    <script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>
  </head>

  <body>
    <div id="root"></div>

    <script type="text/babel">
      class App extends React.Component {
        render() {
          return <h1>Hello world!</h1>
        }
      }

      ReactDOM.render(<App />, document.getElementById('root'))
    </script>
  </body>
</html>
```

Static HTML file

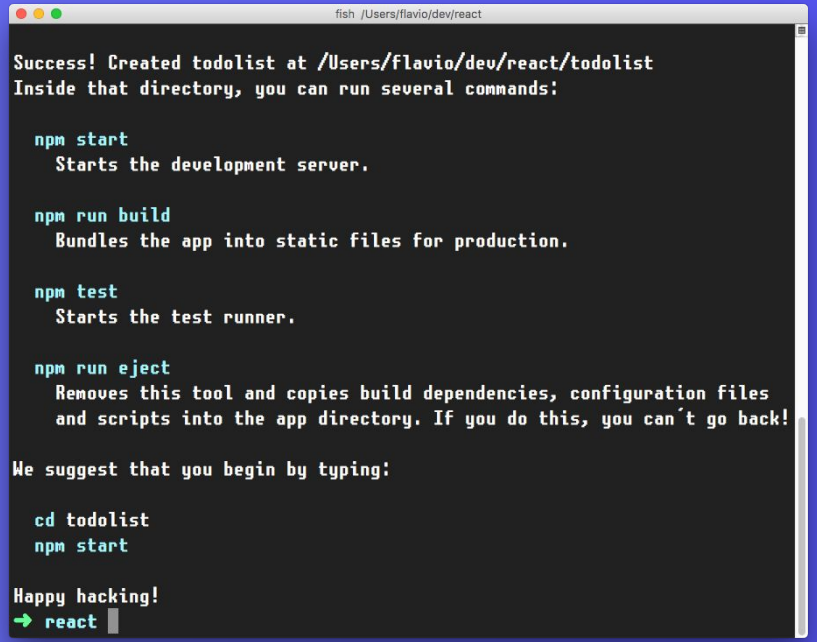


Create React App

create-react-app is a project aimed at getting you up to speed with React in no time. It provides a ready-made React application starter, so you can dive into building your app without having to deal with Webpack and Babel configurations.

To set up **create-react-app**, run the following code in your terminal, one directory up from where you want the project to live.

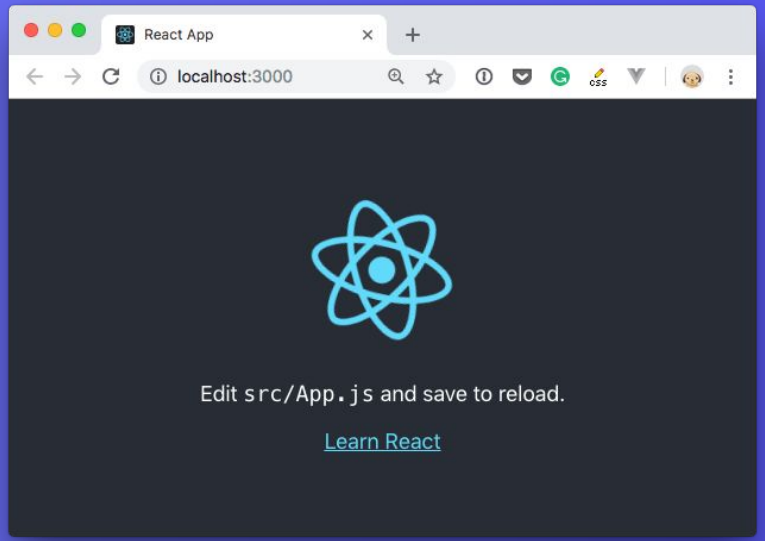
$ npx create-react-app my-app



Create React App

Once that finishes installing, move to the newly created directory and start the project.

$ cd react-tutorial && npm start



Create React App

And now to create our Hello World application we just need to update the **src/App.js** file.

If you go back to **localhost:3000**, you'll see "Hello, React!" just like before. We have the beginnings of a React app now.

```
import React, { Component } from 'react'
import ReactDOM from 'react-dom'
import './index.css'

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>Hello, React!</h1>
      </div>
    )
  }
}

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

Create React App is very good for getting started for beginners as well as large-scale enterprise applications, but it's not perfect for every workflow. You can also create your own Webpack setup for React.

Keep in mind!

03.

React JSX

Introduction to React JSX syntax

Introduction to JSX

JSX is a technology that was introduced by React.

Although React can work completely fine without using JSX, it's an ideal technology to work with components, so React benefits a lot from JSX.

It looks like a strange mix of JavaScript and HTML, but in reality it's all JavaScript.
What looks like HTML, is actually syntactic sugar for defining components and their logic inside the markup.

```
const element = <h1>Hello, world!</h1>;
```

```
const myId = 'test'
const element = <h1 id={myId}>Hello, world!</h1>
```

HTML in JSX

JSX resembles HTML a lot, but it's actually XML syntax.

In the end you render HTML, so you need to know a few differences between how you would define some things in HTML, and how you define them in JSX.

- **class** becomes **className**
- **for** becomes **htmlFor**
- **value** becomes **defaultValue**
- **camelCase** is the new standard (**onchange** => **onChange**)

```jsx
import React from "react";

const MyComponent = () => {
  const handleChange = (e) => {
    console.log(e.target.value);
  };

  const handleSubmit = () => {
    // Do something ...
  };
  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="myInput">This is the label</label>
      <input id="myInput" type="text" defaultValue="..." onChange={handleChange} />
      <button type="submit">Submit</button>
    </form>
  );
};

export default MyComponent;
```

04.

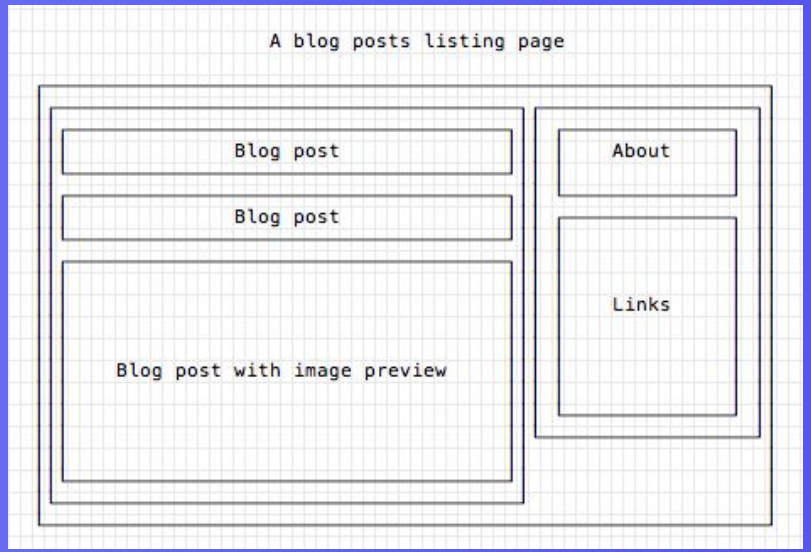
Components and Props

Introduction to React Components and Props

Components

A component is one isolated piece of interface. For example in a typical blog homepage you might find the Sidebar component, and the Blog Posts List component. They are in turn composed of components themselves, so you could have a list of Blog post components, each for every blog post, and each with its own peculiar properties.

React makes it very simple: **everything is a component.**



Components

```
import React, { Component } from 'react'
import BlogPost from "./BlogPost"

class Blog extends Component {
  render() {
    return (
      <div>
        <BlogPost />
        <BlogPost />
        <BlogPost />
      </div>
    )
  }
}
export default Blog;
```

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

class BlogPost extends Component {
  render() {
    return (
      <div>
        <h1>Title</h1>
        <p>Description</p>
      </div>
    )
  }
}
export default BlogPost;
```

Function and Class Components

```
// Class component
class BlogPost extends Component {
  render() {
    return (
      <div>
        <h1>Title</h1>
        <p>Description</p>
      </div>
    )
  }
}
```

```
// Functional component
const BlogPost = () => {
  return (
    <div>
      <h1>Title</h1>
      <p>Description</p>
    </div>
  )
}
```

React Props

Just like properties in functions, we can pass data to Components through **Props**. Starting from the top component, every child component gets its props from the parent. In a function component, props is all it gets passed, and they are available by adding props as the function argument:

```
const BlogPost = (props) => {
  return (
    <div>
      <h1>{props.title}</h1>
      <p>{props.description}</p>
    </div>
  )
}
```

React Props

```
const Blog = () => {
  return (
    <div>
      <BlogPost title="title1" description="description1" />
      <BlogPost title="title2" description="description2" />
    </div>
  )
}
```

```
const BlogPost = (props) => {
  return (
    <div>
      <h1>{props.title}</h1>
      <p>{props.description}</p>
    </div>
  )
}
```

05.

React State

Introduction to React state

React State

State is a place to store data within react components.

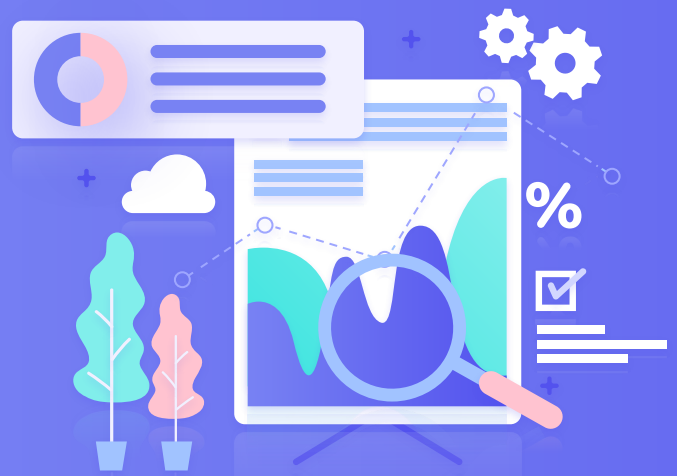
You can think of state as any data that should be saved and modified without necessarily being added to a database - for example, adding and removing items from a shopping cart before confirming your purchase.

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

const Example = () => {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

06.

Conditional Rendering

Introduction to React Conditional Rendering



Conditional Rendering

In React, you can create distinct components that encapsulate behavior you need. Then, you can render only some of them, depending on the state of your application.

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like **if-else** or the **conditional operator** to create elements representing the current state, and let React update the UI to match them.

```
const UserGreeting = () => {
  return <h1>Welcome back!</h1>;
}

const GuestGreeting = () => {
  return <h1>Please sign up.</h1>;
}

const Greeting = (props) => {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
```

```jsx
const LoginControl = () => {
  const [isLoggedIn, setLoggedIn] = useState(false)

  const handleLoginClick = () => {
    setLoggedIn(true);
  }

  const handleLogoutClick = () => {
    setLoggedIn(false);
  }

  return (
    <div>
      <Greeting isLoggedIn={isLoggedIn} />
      {isLoggedIn
        ? <LogoutButton onClick={this.handleLogoutClick} />
        : <LogInButton onClick={this.handleLoginClick} />
      }
    </div>
  );
}
```

07.

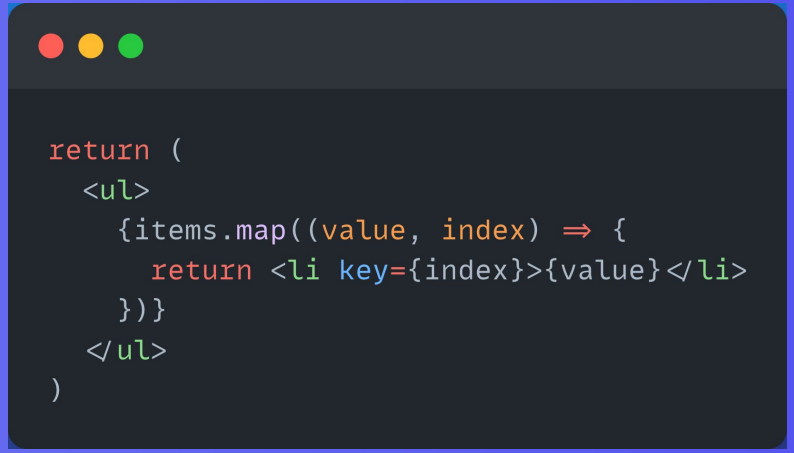
Lists and keys

How to loop inside React JSX

Lists and keys

Suppose you have a React component and an **items** array you want to loop over, to print all the ”items“ you have. Here’s how you can do it.

Inside this list, we add a JavaScript snippet using curly brackets **{}** and inside it we call **items.map()** to iterate over the items.
We pass to the **map()** method a callback function that is called for every item of the list.
Inside this function we return a <**li**> (list item) with the value contained in the array, and with a **key** prop that is set to the index of the item in the array. This is needed by React.



```
return (
  <ul>
    {items.map((value, index) => {
      return <li key={index}>{value}</li>
    })}
  </ul>
)
```

08.

React Thinking

React thinking

10.

React Hooks

Introduction to react hooks

React hooks

Hooks is a feature that was introduced in React 16.7, and changed how we write React apps.

Before Hooks appeared, some key things in components were only possible using class components: having their own state, and using lifecycle events. Function components, lighter and more flexible, were limited in functionality.

Hooks allow function components to have state and to respond to lifecycle events too. They also allow function components to have a good way to handle events.

```
import { useState, useEffect } from 'react'

const Counter = () => {
  const [count, setCount] = useState(0)

  useEffect(() => {
    console.log(`You clicked ${count} times`)
  }, [count])

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  )
}
```

Phases of a component's lifecycle

A React component undergoes three different phases in its lifecycle, including **mounting**, **updating**, and **unmounting**. Each phase has specific methods responsible for a particular stage in a component's lifecycle.

1. The ***mounting phase*** is when a new component is created and inserted into the DOM or, in other words, when the life of a component begins. This can only happen once, and is often called "initial render."
2. The ***updating phase*** is when the component updates or re-renders. This reaction is triggered when the props are updated or when the state is updated. This phase can occur multiple times, which is kind of the point of React.
3. The last phase within a component's lifecycle is the ***unmounting phase***, when the component is removed from the DOM.

Effect Hook ⚡

Another very important feature of Hooks is allowing function components to have access to the lifecycle hooks.

Hooks provide the **useEffect()** API.

The function runs when the component is first rendered, and on every subsequent re-render/update. React first updates the DOM, then calls any function passed to useEffect(). All without blocking the UI rendering even on blocking code.

```
useEffect(() => {
  console.log(`Component mounted`)
}, [])
```

```
useEffect(
  () => {
    console.log(`Hi ${name} you clicked ${count} times`)
  },
  [name, count]
)
```

11.

Show Time 🥊

Finally, let's get our hands dirty !

Thanks!

Do you have any questions?

m.abdelkaderkharoubi@esi-sba.dz
neoxs.github.com


slidesgo