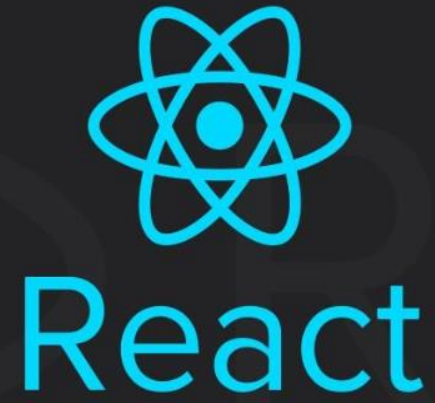




## Saptamana 14

### Partea 2

# Programare Front-End



# React benefits

- **Gasim foarte multe resurse pentru a invata react.** Comunitatea de react este foarte dezvoltata online si putem gasi raspuns la cam orice problema putem avea. Pe langa asta avem si foarte multe cursuri de react free.
- **Putem integra componente de react foarte usor in cod vechi.** E mai usor sa facem update la cod vechi folosind react.
- **Virtual DOM** - eficienta randarii aplicatiilor creste
- **React CLI ne ajuta sa incepem sa scriem cod foarte repede pentru o aplicatie stabila**
- **Este usor de testat codul pe care-l scriem.** Din moment ce componentele sunt modulare e usor sa facem teste pe ele
- **Componentele pot fi reutilizate pe intreaga aplicatie.** Asta inseamna ca avem mai putin cod de scris si mai eficient.
- **Stilizarea componentelor e 'scoped'.** Rezulta ca fiecare componeta poate avea css-ul ei fara sa interfereze cu o clasa asemanatoare din alta componenta.
- **Componentele au 'state'-ul lor.** Inseamna ca fiecare componenta are state-ul ei separat fara sa interfereze cu alte state-uri ale altor componente.

# Getting Started

```
// Create A React Application In A New Folder
```

```
npm create-react-app my-app
```

```
cd my-app
```

```
npm start
```

```
1 //Create A React Application In An Existing Folder
```

```
2
```

```
3 cd my-app
```

```
4 npm create-react-app|
```

```
5 npm start
```

# Breakdown the code

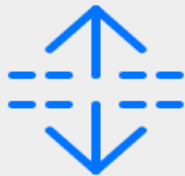
## Index.js

- Linia 1 si 2 : importam React si ReactDOM sa le putem folosi in aplicatie. ReactDOM - ne ofera metode specifice DOM-ului pe care le putem folosi pe componente
- Linia **3** importam css-ul pentru index.js
- Linia **4** importam **<App />** pentru a putea fi atasat mai tarziu pe DOM
- Linia **5** importam registerServiceWorker. Acest serviciu este pentru ca aplicatia noastra sa poata functiona si offline - nu intram acum in detalii dar puteti citi aici > [Service workers](#)
- Linia **7** folosind ReactDOM putem lega componenta **App** de DOM. Aceasta metoda de **render()** primeste 2 parametri mandatori si unul optional :
  - Componenta ce trebuie randata - `<App />`
  - Node-ul parinte pe care o atasam : `rootElement`
  - O functie de callback ce se va executa dupa ce componenta a fost randata
- Linia 12 putem modifica in **register()** pentru a folosi aplicatia offline.

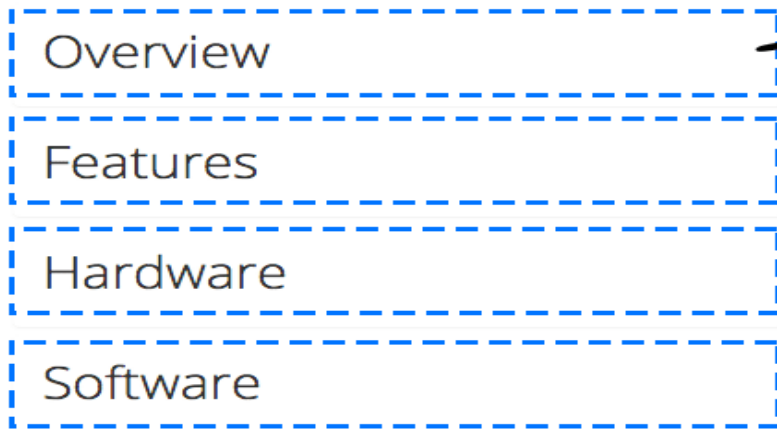
# Breakdown the code

## App.js

- Linia 1 - importam React.
- Liniile 2 si 3 - importam logo si css-ul.
- Linia 5 - construim functia App ce returneaza ceva ce arata ca HTML-ul dar defapt este **JSX**. Important este pentru aceasta functie ca are doar un singur outer element. (Un parinte ce n-are alte elemente pe aceeasi linie cu el, adica un root - un div singur)
- Sunt 2 tipuri de componente in React : **functionale si clase**.
  - **Functional components** : sunt declarate cu keyword-ul **function** si accepta ca si argument *props* si returneaza un element React valid
  - **Class Components**: sunt declarate cu keyword-ul **class**. *Putem importa si { Component }.* Folosim {} pentru 'destructuring' - importam module specifice ce au fost exportate din 'react' in cazul de fata. Putem schimba **Component** cu **React.Component** - aceste 2 sunt echivalente si functioneaza la fel.
  - Ambele metode sunt aproape la fel, vedem mai jos diferentele
- Note: toate elementele de jsx(care arata ca html) trebuie returnate in paranteze adica **return ( jsx code...)**
- Linia 26 exportam componenta **App**.



## Collapsible Content



## Collapsible Component with props

```
<Collapsible title="Overview" />  
  
<Collapsible title="Features" />  
  
<Collapsible title="Hardware" />  
  
<Collapsible title="Software" />
```

# Let's get in !

**Fiecare aplicatie REACT este construita din componente.** Putem avea o singura componenta intr-o aplicatie simpla sau mai multe, in functie de complexitate.

## Componente simple:

- Poate randa html static(hardcodat)
- Poate randa dinamic bazat pe un json local
- Pot fi nested una in alta pentru un layout mai complex
- Pot fi randate in functie de datele(props) date de la parinte spre copii (mai jos o sa avem props explicate)

## Componente complexe:

- Pot include logica avansata ce defineste cum va arata HTML-ul la final
- Pot contine propriul state
- Pot contine *lifecycle methods*
- Pot contine metode custom ce vor fi executate cand un user da click pe un buton de exemplu





## Stateless Components vs Class Components

```
const Intro = () => {  
  return <p>Hi there...</p>  
}
```

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

# Let's get in!

## Simple(stateless) components

```
const Headline = () => {  
  return <h1>React Cheat Sheet</h1>  
}
```

```
1 // Component must only return ONE element (eg. DIV)  
2 * const Intro = () => {  
3   return <div>  
4     <Headline />  
5     <p>Welcome to the React world!</p>  
6   </div>  
7 }
```

```
1 * const Intro = () => {  
2   return (  
3     <div>  
4       <Headline />  
5       <p>Welcome to the React world!</p>  
6     </div>  
7   )  
8 }  
9  
10 const Intro = () => (  
11   <div>  
12     <Headline />  
13     <p>Welcome to the React world!</p>  
14   </div>  
15 )
```

Avem nevoie sa returnam un singur element pt ca asta este cerinta de JSX - fiecare componenta poate returna un singur element  
Din acest motiv trebuie returnat un singur element HTML - div, ul etc

# Let's get in!

**Advanced components (contin clasa ES6)** (mai sunt numite si smart components)

```
class App extends React.Component {  
  render() {  
    return (  
      <h1>React smart component</h1>  
    )  
  }  
}
```

Vom folosi aceasta metoda doar atunci cand avem nevoie de componente smart (de obicei componente care au state pe ele)

# Let's get in!

```
class App extends React.Component {  
  // fires before component is mounted  
  constructor(props) {  
    // makes this refer to this component  
    super(props);  
    // set local state  
    this.state = {  
      date: new Date()  
    };  
  }  
  
  render() {  
    return (  
      <h1>  
        It is {this.state.date.toLocaleTimeString()}.  
      </h1>  
    )  
  }  
}
```

1. Am inclus **constructor** cu **super()** si am setat starea **initiala** a aplicatiei in **this.state**
2. Acum putem accesa data din state inaintur metodei **render()** astfel - **{this.state.date}**

## IMPORTANT!!

- Nu putem modifica state-ul direct adica : **this.setState( {data: newDate() } )** - corect  
**This.state.date = newDate() - gresit!**
- **Modificarile state-ului pot fi asincrone.** Nu ne bazam pe valorile state-ului de acum pentru state-ul viitor.



## Props vs State



✓ props are read-only

✓ props can not be modified

✓ state changes can be asynchronous

✓ state can be modified using `this.setState`

# Let's get in !

Putem 'pasa' ce valori din state-ul parintelui avem catre componentele copii ca si **props**

```
1 • const Greetings = (props) => {  
2     return <p>You will love it {props.name}</p>  
3 }  
4 // <Greetings name={Ovi}>
```

```
• const Greetings = ({name}) => {  
    return <p>You will love it {name}</p>  
}  
// <Greetings name={Ovi}>
```

Principala diferenta dintre **props** si **state** in React este ca props-urile sunt **read-only** si nu pot fi modificate din interiorul componentei

Daca in exemplul de mai sus vrem sa modificam **name** ce este returnata in componenta Greetings trebuie sa mergem in parinte si sa modificam numele apoi sa-l pasam din nou spre **copil**.



## ES6 Destructuring



```
const name = this.props.name;  
const age = this.props.age;
```



```
const {name, age} = this.props;
```

# Destructuring

```
const name = this.props.name;  
const age = this.props.age;  
const isLoggedIn = this.state.isLoggedIn;  
const username = this.state.username;  
  
const {name, age} = this.props;  
const {isLoggedIn, username} = this.state;
```

## REACT CHEAT SHEET



**PRACTICE: React**  
**<http://bit.do/reactQuiz>**  
**<http://bit.do/reactSocialCard>**

