Laboratorul 07: Introducere in ReactJS

1. Ce este Single Page Application (SPA)?

Un SPA sau o aplicatie cu o singura pagina, este o aplicatie (sau un website) care interactioneaza cu browserul web prin rescrierea dinamica, a DOM-ului current, cu date noi generate fie din client, fie din server. In acest caz, browserul nu este nevoit sa incarce o pagina noua.

2. Ce este JSX ?

Considerati urmatoarea linie

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

Aceasta sintaxa se numeste JSX, iar denumirea a pornit de la Syntax Extension to JavaScript. Aceasta sintaxa este folosita in React pentru a descrie cum va arata o componenta. JSX se poate sa va duca cu gandul la HTML, diferenta este ca JSX vine si cu toate functionalitatile JavaScript. In exemplul de mai jos, declaram o variabila name, iar ulterior o folosim in interiorul JSX fiind pusa intre paranteze acolade.

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

De asemenea, puteti sa puneti orice cod JavaScript valid in interiorul parantezelor acolade in JSX.

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

JSX poate sa fie folosit si in contextul unor expresii, precum

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

Tag-urile JSX pot contine si mai multi copii.

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

De asemenea, putem folosi JSX pentru a ne proteja si de eventuale atacuri.

```
const title = response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

JSX reprezinta obiecte. Babel se ocupa de compilarea JSX in apeluri React.createElement()

```jsx
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```jsx
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

3. Ce este React ?

React este o biblioteca JavaScript pentru construirea de interfete utilizator. Este creat si intretinut de Facebook si de o comunitate de dezvoltatori (open source) si desigur si de alte companii individuale.

4. Crearea unei aplicatii React

Pentru a putea folosi tool-ul oferit de Facebook, avem nevoie sa indeplinim urmatoarele conditii: Node >= 8.10 si npm >= 5.6 [https://nodejs.org/en/].

Pentru a genera un proiect trebuie sa rulati urmatoarele:

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

Dupa rularea script-ului *npm start* puteti accesa aplicatia accesand link-ul din consola

```
http://localhost:3000
```

```
Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://10.0.75.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
```

5. Componente

Componentele va permit sa va impartiti interfata utilizator in “piese” reutilizabile, iar in acelasi timp va puteti gandi la fiecare piesa ca fiind izolata.

Conceptual, componentele sunt asemanatoare cu functiile JavaScript. Acestea accepta intrari (denumite "props") si returneaza elemente React, care descriu ceea ce ar trebui sa apara in interfata.

5.1. Functional Components

Cea mai simpla cale de a defini o componenta este aceea de a scrie o functie JavaScript

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Aceasta functie este o componenta valida React deoarece accepta un singur obiect "props" (proprietati) cu date si returneaza un element React. Aceste componente poarta denumirea de **function components** deoarece ele reprezinta niste functii JavaScript.

5.2. Class Components

De asemenea, puteti folosi si o clasa ES6 [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes] pentru a defini o componenta:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Cele doua componente definite mai sus, sunt echivalente din punct de vedere React.

5.3 Adaugarea unei componente in DOM (Rendering a Component)

Pana acum, am intampinat elemente React care reprezentau tag-uri DOM, precum:

```
const element = <div />;
```

De asemenea, elementele pot reprezenta si componente definite de utilizator:

```
const element = <Welcome name="Sara" />;
```

Cand React descopera un element care reprezinta o componenta definita de utilizator, acesta trimite atributele JSX si copiii catre aceasta componenta folosind un singur obiect. Acest obiect poarta numele de "props".

In exemplul urmator, codul "randeaza", in pagina, mesajul *Hello, Sara*.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Componentele create pot fi reutilizate de cate ori dorim. Spre exemplu, putem crea o componenta *App* care afiseaza *Welcome* de mai multe ori, folosind alt nume.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

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

5.4 Stare (State)

State [https://reactjs.org/docs/state-and-lifecycle.html] reprezinta un obiect JavaScript care pastreaza informatia influentand aspectul sau starea componentei compilate si randate. Diferenta fata de **props** este aceea ca starea este mentinuta in contextul componentei - similar cu definirea unei variabile in interioriul unei functii.

5.4.1 Functional Components

In cazul componentelor functionale, o variabila de stare este declarata cu ajutorul useState (React Hooks [https://reactjs.org/docs/hooks-state.html]). O variabila de stare poate lua mai multe forme: string, numar sau obiect. Functia useState primeste ca paramentru valoarea initiala pe care vrem sa o atribuim variabilei si returneaza variabila de stare impreuna cu functia de actualizare, similara cu this.setState din componentele de tip clasa.

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

const initialState = {
  message: 'Hello world!'
};

const Welcome = () => {
  const [state, setState] = useState(initialState);

  const onButtonClicked = () => setState({ message:
'Why did you change me?' });

  return (
    <>
      <div>{state.message}</div>
      <button onClick={onButtonClicked}>Change text</button>
    </>
  );
};

export default Welcome;
```

5.4.2 Class Components

```jsx
import React from 'react';

class Welcome extends React.Component {
  state = {
    message: 'Hello world!'
  }

  onButtonClicked = () => this.setState({ message:
'Why did you change me?' });

  render() {
    return (
      <>
        <div>{this.state.message}</div>
        <button onClick={this.onButtonClicked}>Change text</button>
      </>
    )
  }
};

export default Welcome;
```

Cele doua componente prezentate mai sus sunt echivalente, insa exista o mica diferenta:

Class Components fac referire la state sau props folosind **this**.

```
Class components

this.state
this.props
```

```
Functional Components

state
props
```

5.5 Event Handling

Gestionarea evenimentelor cu elemente React este foarte asemanatoare cu gestionarea evenimentelor pe elemente DOM. Totusi, exista cateva diferente de sintaxa:

- evenimentele din React sunt camelCase
- pentru gestionarea evenimentelor in React sunt folosite functii, nu string-uri

HTML

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

React

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

HTML

```
<a href="#" onclick="console.log('The link was clicked.'); return false">
  Click me
</a>
```

React

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

5.6 Conditional Render

Randarea bazata pe conditii, in React, functioneaza in acelasi mod precum conditionarea in JavaScript. Folositi operatori JavaScript precum if [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else] sau conditional operator [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator] pentru a crea elemente reprezentand starea curenta.

Considerati urmatoarele doua componente:

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

Vom implementa o componenta *Greeting* care afiseaza una dintre aceste componente, daca un utilizator este autentificat sau nu.

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

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

5.7 Liste si chei (key)

In primul rand, sa ne amintim cum transformam listele in JavaScript.

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map((number) => number * 2);
console.log(doubled);
```

Acest cod afiseaza urmatorul rezultat in consola

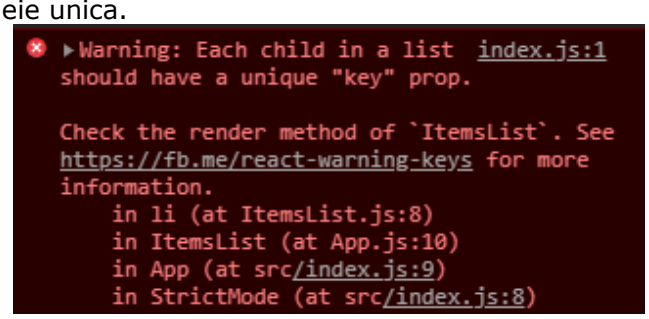
```
[2, 4, 6, 8, 10]
```

Afisarea mai multor componente

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li>{number}</li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Cand rulam acest cod, in consola va fi afisat un mesaj de atentionare spunandu-ne ca pentru fiecare *li* (list item) trebuie sa furnizam o cheie unica.



Pentru a rezolva aceasta problema, trebuie sa introducem urmatoarea modificare:

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Aceste **key**s (chei) ajuta React-ul la identificarea elementelor care s-au schimbat, adaugat si eliminat. Putem privi aceste chei precum identitatea unui element.

5.8 Compozitie

React are un model puternic de compozitie si va recomandam sa folositi compozitia in loc de mostenire pentru a reutiliza codul intre componente

Unele componente nu stiu din timp ce copii vor avea. Aceste componente sunt de obicei folosite cand ne definini **Layout**-ul aplicatiei sau orice element care reprezinta o 'cutie' pentru elementele noastre.

Fiecare componenta React, are o proprietare rezervata in obiectul **props** numita **children**. Aceasta proprietate poate fi accesata in felul urmator:

```
Class components

this.props.children
```

```
Functional components

props.children
```

```jsx
function FancyBorder(props) {
  return (
    <div className={'FancyBorder FancyBorder-' + props.color}>
      {props.children}
    </div>
  );
}
```

```jsx
function WelcomeDialog() {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        Welcome
      </h1>
      <p className="Dialog-message">
        Thank you for visiting our spacecraft!
      </p>
    </FancyBorder>
  );
}
```

6. Sass (Syntactically Awesome Style Sheets)

6.1 Ce este Sass si de ce este util?

Sass [https://sass-lang.com/guide] este un preprocesor de stil compilat in CSS. Va permite sa utilizati variabile, nested rules, mixins, functii si multe altele, toate cu o sintaxa complet compatibila CSS. SASS ajuta la mentinerea bine organizata a fisierelor mari de stil si faciliteaza partajarea designului in cadrul proiectelor.

6.2 Variabile, mixins, nesting

Ganditi-va la variabile ca la o modalitate de a stoca informatiile pe care doriti sa le reutilizati in intreaga aplicatie (stiluri). Puteti stoca lucruri precum culori, stive de fonturi sau orice valoare CSS pe care considerati ca veti dori sa o reutilizati. SASS foloseste simbolul **$** pentru a face ceva variabil.

Exemplu folosire variabile

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Cand scrieti HTML, ati observat probabil ca are o ierarhizare clara si vizuala. CSS, in schimb, nu. SASS va permite sa va ierarhizati selectoarele CSS intr-un mod care sa urmeze aceeasi ierarhie vizuala a HTML-ului. Fiti constienti de faptul ca regulile exagerat de imbibate vor duce la CSS supra-calificate, care s-ar putea dovedi greu de intretinut si nu este, in general, considerata o practica buna.

Exemplu folosire nesting

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

Unele lucruri din CSS sunt destul de complicate de scris, in special cu CSS3 si cu numeroasele prefixuri ale furnizorilor care exista. Un mixin va permite sa faceti grupuri de declaratii CSS pe care doriti sa le reutilizati pe site-ul vostru.

Exemplu folosire mixins

```
@mixin transform($property) {
  -webkit-transform: $property;
  -ms-transform: $property;
  transform: $property;
}
.box {
  @include transform(rotate(30deg));
}
```

6.3 Modules

Pentru o organizare buna a codului, putem folosi module in care sa tinem culorile, fonturile, dimensiunile folosite in website.

```
// _base.scss
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

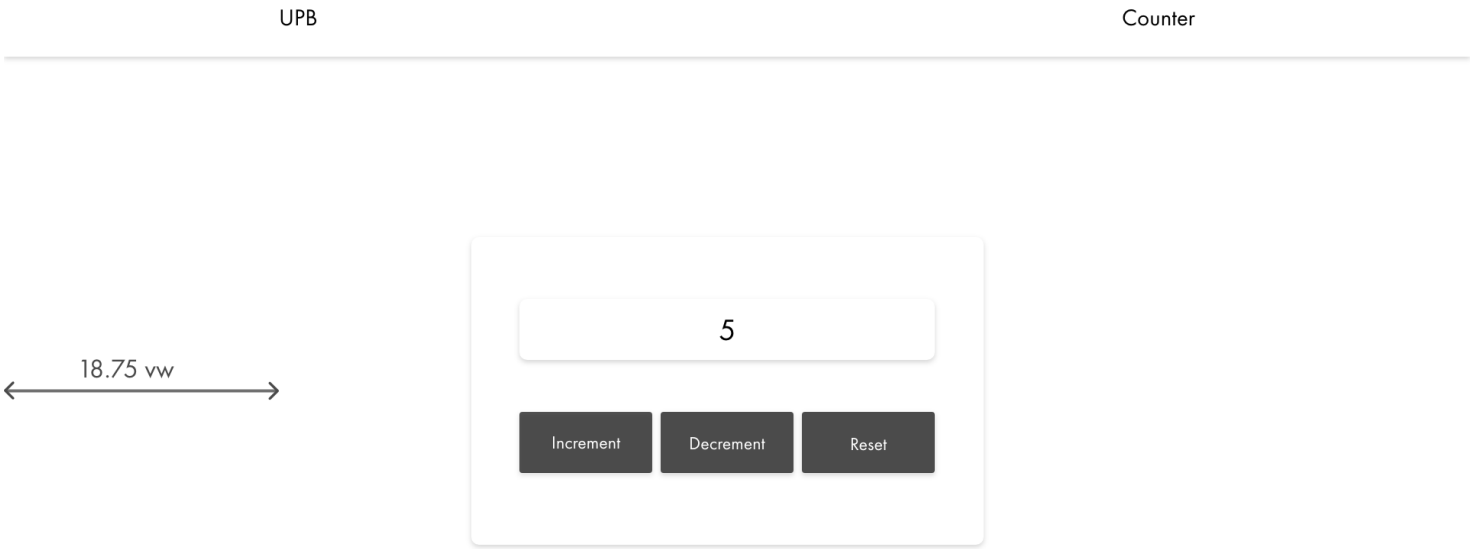
```
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

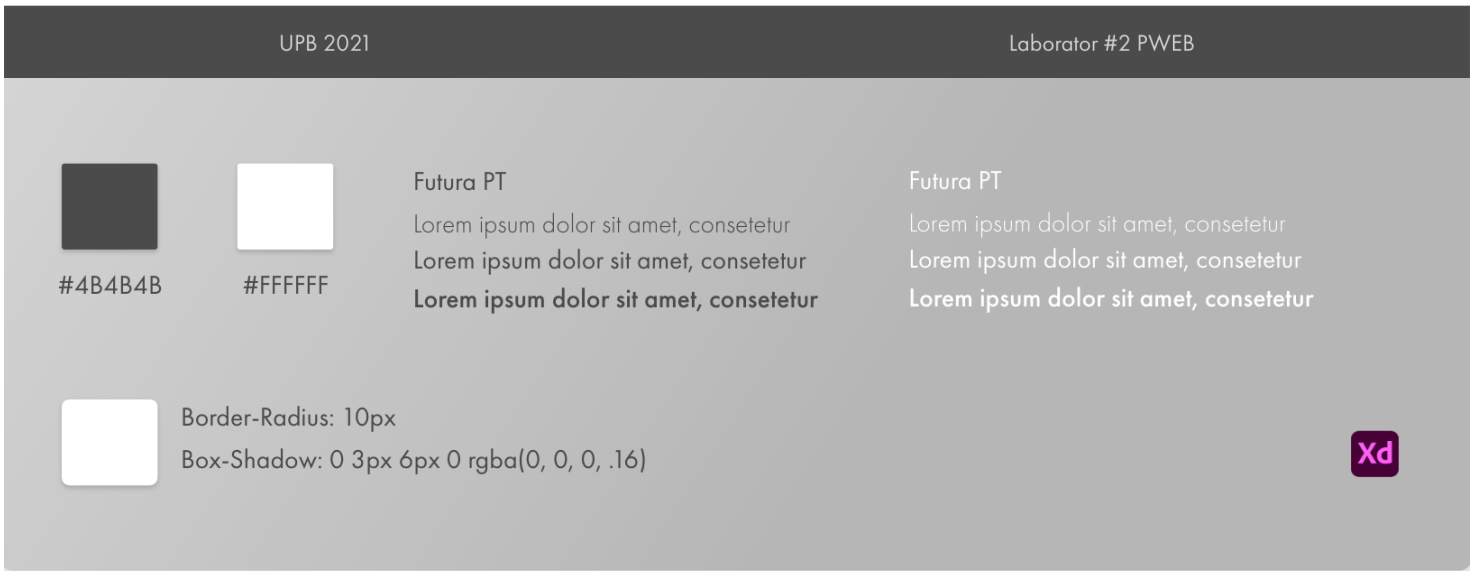
6.4 Extend

Poate una dintre cele mai utile feature-uri din Sass, cu @extend putem sa distribuim proprietati CSS de la un selector la altul.

```
/* This CSS will print because %message-shared is extended. */
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

// This CSS won't print because %equal-heights is never extended.
%equal-heights {
  display: flex;
  flex-wrap: wrap;
}

.message {
  @extend %message-shared;
}

.success {
  @extend %message-shared;
  border-color: green;
}

.error {
  @extend %message-shared;
  border-color: red;
}

.warning {
  @extend %message-shared;
  border-color: yellow;
}
```

Exercitii





Realizati urmatoarele exercitii astfel incat sa respecte design-ul de mai sus. Informatiile legate de font si culori le puteti gasi sub design. Pentru font, puteti folosi Google Fonts [https://fonts.google.com], iar pentru partea de layout folositi Flexbox [https://flexboxfroggy.com].

1. Generati o aplicatie React folosind **create-react-app**.
2. Creati o componenta **Counter** care sa afiseze un numar si 3 butoane **Increment**, **Decrement** si **Reset**.
3. Adaugati functionalitatea necesara pentru a incrementa, decrementa si reseta starea componentei **Counter**.
4. Creati 3 componente (Class sau Functional) **Header**, **Footer** si **Layout**.
5. Creati o sectiune in componenta **Layout** care sa reprezinte continutul afisat pe pagina, iar in interiorul acestui continut sa putem adauga copii folosind compozitia.
6. Adaugati **Header** si **Footer** in Layout pentru a defini structura paginii.
7. Afisati componenta **Counter** in **Layout**ul pe care l-ati creat anterior.