ReactJS

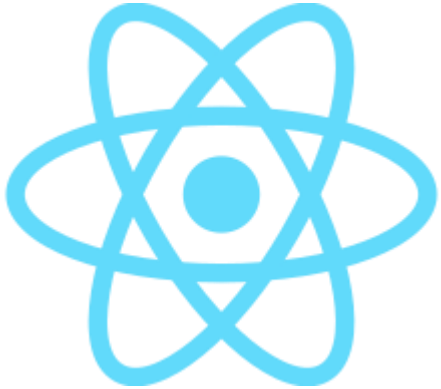
09/06/2023

ReactJS

09/06/2023

Introduction à React

React (aussi appelé React.js ou ReactJS) est une bibliothèque Javascript open source pour créer des interfaces utilisateur (UI). Il a été initialement développé en 2013 par Facebook et est maintenant maintenu par tous les contributeurs open source, y compris FB.



Introduction à React

React fonctionne en changeant le DOM de la page et rend toutes les modifications apportées au DOM lors de l'interaction / mise à jour de la page. Ces modifications du DOM peuvent être effectuées par un utilisateur ou automatiquement par le système. Il détecte les modifications apportées au DOM et ne met à jour que ces modifications spécifiques, ce qui le rend rapide pour les sites Web dynamiques car seule une petite partie du code HTML est modifiée sans recharger la page Web complète.

Introduction à React

Facebook est un site Web dynamique, et pour charger du nouveau contenu; il n'est pas possible de rendre le DOM entier à plusieurs reprises pour apporter de petites modifications à la page Web car cela ralentira l'ensemble du site Web. React aborde ce problème d'une manière unique; il conserve un «DOM virtuel», qui est une copie du DOM réel qui est affiché à l'utilisateur.

Chaque fois qu'une modification est apportée au DOM réel, React modifie d'abord le DOM virtuel, puis vérifie la différence entre le DOM réel et le DOM virtuel. Cela aide à identifier les éléments qui doivent être restitués à l'écran. Il ne met donc à jour que les éléments requis, ce qui le rend beaucoup plus rapide.

React le révolutionnaire !

React a popularisé une toute nouvelle architecture d'application Web appelée Single Page Application. Auparavant, la page Web était chargée à partir du serveur, et tout ce que vous cliquez entraînait une nouvelle demande adressée au serveur et le navigateur chargeait une nouvelle page.

Les applications à page unique, en revanche, ne chargent la page Web (HTML, CSS, JS) qu'une seule fois, et toute autre interaction avec l'application ne charge que les données requises ou effectue une action sur le serveur. Cela ne recharge jamais l'ensemble de l'application, ce qui la rend plus légère sur le serveur et plus rapide.

Gmail, Facebook et Twitter sont tous des exemples de SPA.

Javascript everyware….

Javascript, qui est le langage utilisé pour développement frontend ainsi que React, est le langage le plus connu parmi les développeurs et devient de plus en plus populaire depuis 2018.

Javascript everyware….

Javascript, qui est le langage utilisé pour développement frontend ainsi que React, est le langage le plus connu parmi les développeurs et devient de plus en plus populaire depuis 2018.

Best known languages: 2018-2020

		2020	2019	2018
JavaScript		1	1	2
Java		2	2	1
C		3	3	3
Python		4	4	5
C++		5	5	4
C#	▲	6	7	6
PHP	▼	7	6	7
TypeScript		8	8	8
Pascal		9	9	9
R		10	10	10

Source: HackerRank 2020 Developer Skills Report

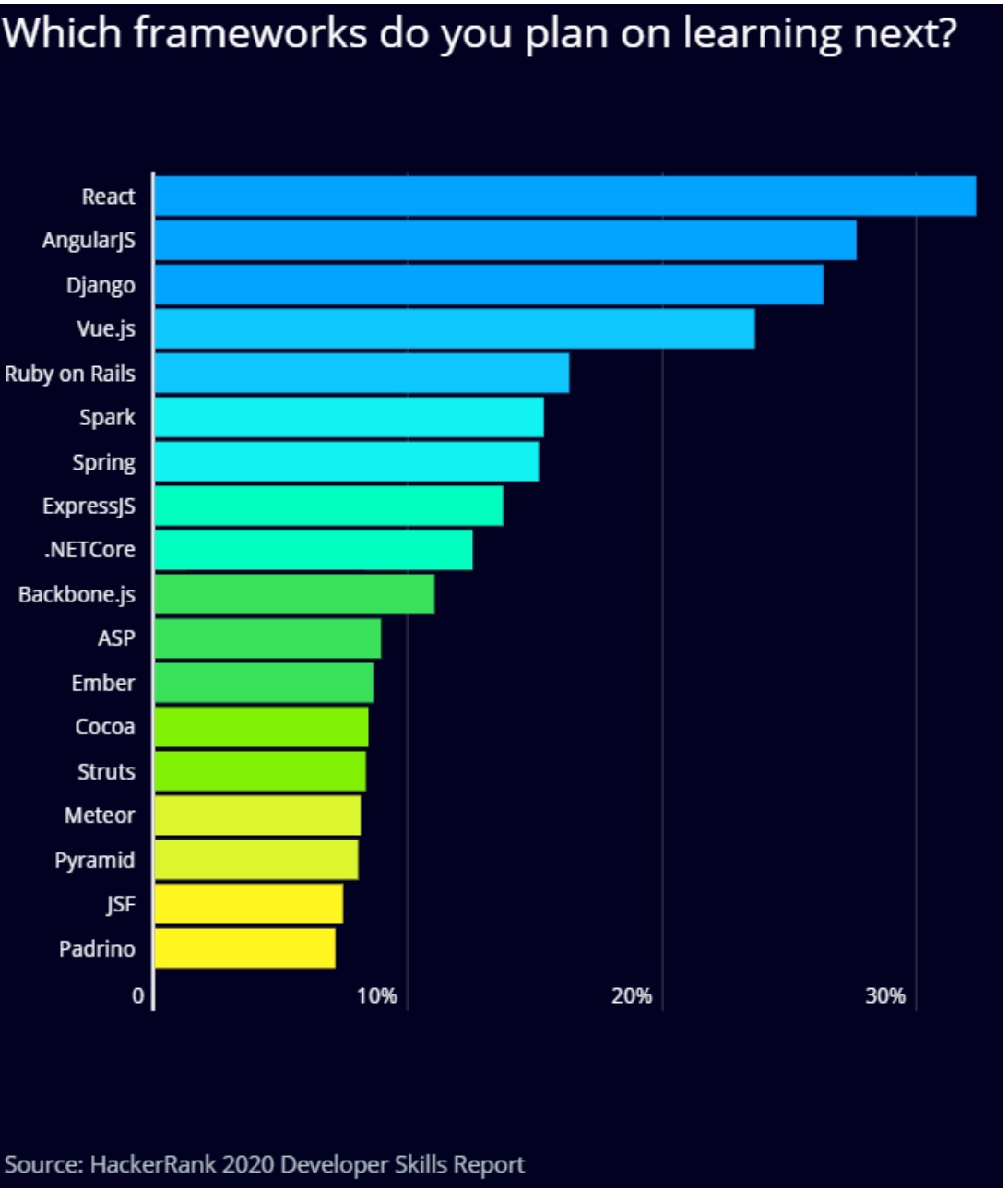
La bataille avec AngularJS

La popularité de React augmente d'année en année et devrait prendre le dessus bientôt. La facilité d'apprentissage et les avantages techniques qu'elle offre sont les raisons de cette tendance.

Best known frameworks: 2018-2020

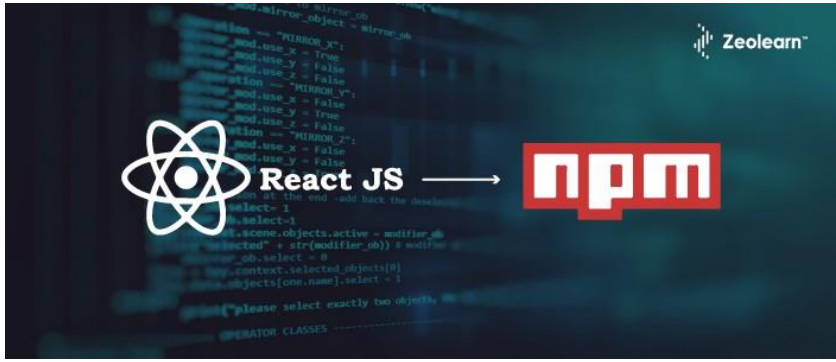
		2020	2019	2018
AngularJS		1	1	1
React		2	2	3
Spring		3	3	2
Django	▲	4	6	6
ExpressJS	▼	5	4	4
ASP	▼	6	5	5
.NETCore		7	7	7
Vue.js	▲	8	9	10
Ruby on Rails	▼	9	8	8
JSF		10	10	9

Source: HackerRank 2020 Developer Skills Report



React la base ?

React bat tous les autres frameworks frontend par une énorme marge lorsqu'il s'agit de l'apprendre. De nombreux développeurs aiment monter dans le train React, qui offre un tout nouveau monde d'opportunités.



Installation de React

Installation de React

React est une bibliothèque JavaScript déclarative basée sur des composants, utilisée pour créer des interfaces utilisateur.

Pour atteindre les fonctionnalités du framework MVC dans React, les développeurs l'utilisent en conjonction avec des architectures d'application pour la création d'interface utilisateur comme par exemple Flux et Redux.

https://www.npmjs.com/package/react

https://react.dev/learn

Installation / configuration

ReactJS est une bibliothèque JavaScript contenue dans un seul fichier react-<version>.js pouvant être inclus dans n'importe quelle page HTML. Les gens installent également généralement la bibliothèque React DOM react-dom-<version>.js avec le fichier principal React:

Inclusion de base

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
    <script type="text/javascript" src="/path/to/react.js"></script>
    <script type="text/javascript" src="/path/to/react-dom.js"></script>
    <script type="text/javascript">
        // Use react JavaScript code here or in a separate file
    </script>
    </body>
</html>
```

<script src="https://unpkg.com/react@18/umd/react.development.js"></script>

<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

Installation / configuration

React prend également en charge la syntaxe JSX . JSX est une extension créée par Facebook qui ajoute une syntaxe XML à JavaScript. Pour utiliser JSX, vous devez inclure la bibliothèque Babel et changer <script type="text/javascript"> en <script type="text/babel"> afin de traduire JSX en code Javascript.

```
<body>

<script type="text/javascript" src="/path/to/react.js"></script>

<script type="text/javascript" src="/path/to/react-dom.js"></script>

<script src="https://npmcdn.com/babel-core@5.8.38/browser.min.js"></script>

<script type="text/babel">

// Use react JSX code here or in a separate file

</script>

</body>
```

Installation / configuration

Nous pouvons l’utiliser via npm. C’est cette version de react que nous utiliserons.

Initialisation de node dans le dossier

```
npm init
```

Installation / configuration

Nous pouvons l'utiliser via npm. C'est cette version de react que nous utiliserons.

Installation via npm

```
npm install --save react react-dom
```

Pour utiliser React dans votre projet JavaScript, vous pouvez effectuer les opérations suivantes:

var React = require('react');

var ReactDOM = require('react-dom');

ReactDOM.render(<App />, ...);

Installation / configuration

Installation via fil

Facebook a publié son propre gestionnaire de paquets nommé Yarn , qui peut également être utilisé pour installer React. Après avoir installé Yarn, il vous suffit d'exécuter cette commande :

```
yarn add react react-dom
```

Vous pouvez ensuite utiliser React dans votre projet exactement comme si vous aviez installé React via npm.

Installation / configuration

Création d’une application react :

create-react-app est un générateur de réactions créé par Facebook. Il fournit un environnement de développement configuré pour une facilité d'utilisation avec une configuration minimale, notamment:

- Transpilation ES6 et JSX
- Serveur de développement avec rechargement de module à chaud
- Linting code
- Préfixe CSS
- Créer un script avec JS, CSS et regroupement d'images, et des sourcemaps
- Cadre de test Jest

Installation / configuration

Installation

```
npm install -g create-react-app
```

Ensuite, lancez le générateur dans le répertoire choisi.

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

Accédez au répertoire nouvellement créé et exécutez le script de démarrage.

```
cd my-app/
```

```
npm start
```

Installation / configuration

Pour créer votre application pour la production prête, exécutez la commande suivante

```
npm run build
```

Composants et accessoires

Comme React ne concerne que le point de vue d'une application, l'essentiel du développement dans React sera la création de composants. Un composant représente une partie de la vue de votre application. "Props" sont simplement les attributs utilisés sur un nœud JSX (par exemple, `<SomeComponent someProp="some prop's value" />`),

et sont la principale manière dont notre application interagit avec nos composants. Dans l'extrait ci-dessus, à l'intérieur de SomeComponent, nous aurions accès à `this.props` , dont la valeur serait l'objet `{someProp: "some prop's value"}` .

Composants et accessoires

Il peut être utile de considérer les composants de React comme des fonctions simples: ils prennent en compte les «accessoires» et produisent une sortie sous forme de balisage. Beaucoup de composants simples vont plus loin en se faisant des "fonctions pures", ce qui signifie qu'ils ne génèrent pas d'effets secondaires et sont idempotents (étant donné un ensemble d'entrées, le composant produira toujours la même sortie). Cet objectif peut être formellement appliqué en créant des composants en tant que fonctions, plutôt que des "classes".

Composants et accessoires

Composants et accessoires

- Créons un dossier components
- Nous allons créer un fichier Greet.js. **Attention de bien respecter la nomenclature des fichiers**.
- Premiere chose importons react :
- import React from "react";
- Puis écrivons la fonction de notre premier component :

```
function Greet(){
    return <h1>Hello Fred !</h1>
}
export default Greet;
```

Composants et accessoires

- Pour afficher ce composant retour dans App.js :

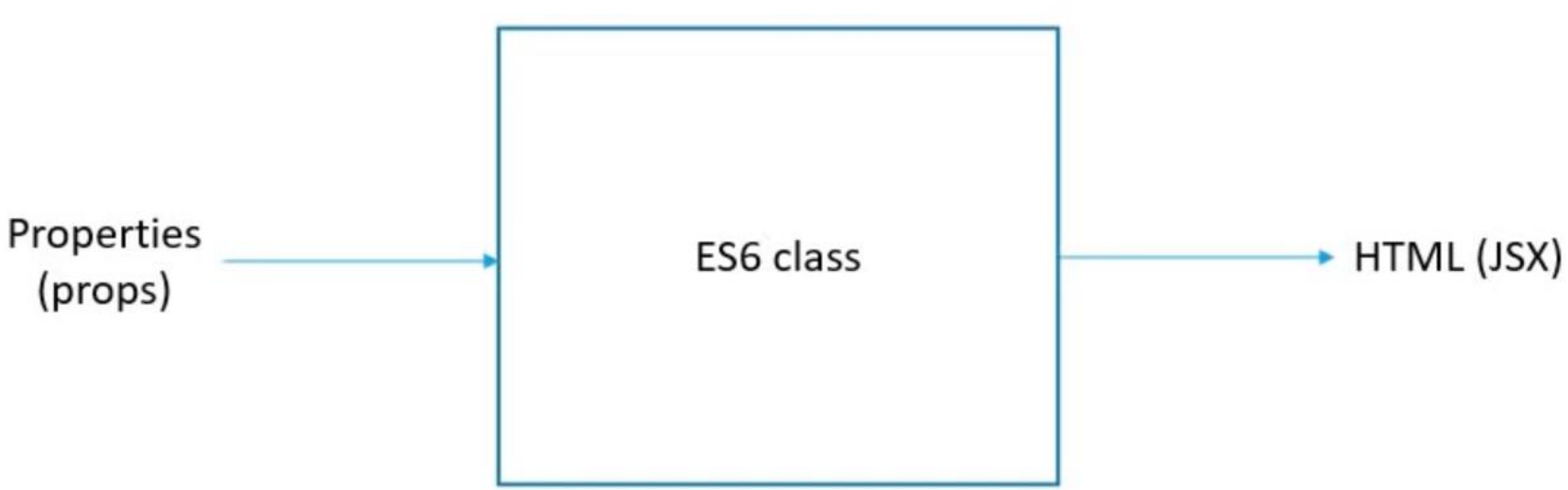
```
import Greet from './components/Greet'

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

Composants et accessoires

Une autre façon de créer un component avec des fonctions fléchées :

```
const Greet = () => <h1>Hello Fred !</h1>
```

Component par class



Component par class

Component par class

Réalisons la même fonctionnalité mais avec les class.

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

class Welcome extends Component {
    render() {
        return <h1>Class component</h1>
    }
}

export default Welcome
```

Component par class

Dans App.js

```
import Welcome from './components/Welcome';
function App() {
  return (
    <div className="App">
      <Greet />
      <Welcome />
    </div>
  );
}
```

Composant par fonction VS par Class

Par fonction :

Fonctionalité simple

Permet d'utiliser les fonctions du composants assez simplement

Pas de state donc moins complexe

Plus compréhensible pour l'UI

Class

Pour les fonctionnalités plus lourde et complexe

Propre variable et propre state

UI complexe

JSX

JavaScript XML (JSX) est une extension du langage JavaScript.

L’ecriture du code peut s’apparenter à de l’XML

JSX ont des tags name, des attribues et des enfants

JSX n’est pas vraiment necessaire pour coder des applications React.

Mais l’utilisation de JSX va rendre votre code plus simple et plus élégant.

Props

Essayons de personnaliser l'application :

App.js

```
<Greet name="Thomas" />
<Greet name="Toto" />
<Greet name="John" />
```

Props

Essayons de regarder ce que contient props

Greet.js

```
const Greet = (props) => {
    console.log(props);
    return <h1>Hello Fred</h1>
}
```

C’est un objet. Pour acceder au nom il faut donc :

Props

Essayons de regarder ce que contient props

Greet.js

```
const Greet = (props) => {
    console.log(props);
    return <h1>Hello {props.name} !</h1>
}
```

Props

Nous pouvons ajouter d’autres propriétés :

App.js

```
<Greet name="Thomas" age="23 ans"  />
<Greet name="Toto" age="90 ans" />
<Greet name="John" age="33 ans" />
```

Props

Nous pouvons ajouter d’autres propriétés :

Greets.js

```
const Greet = (props) => {
    console.log(props);
    return <h1>Hello {props.name} vous avez {props.age}!</h1>
}
```

Props

Nous pouvons ajouter des enfants aux props :

```
<Greet name="Thomas" age="23 ans">
    Thomas est un super gars parce qu'il adore React !!
</Greet>
```

Props

Nous pouvons ajouter des enfants aux props :

```
const Greet = (props) => {
    console.log(props);
    return (
        <div>
            <h1>Hello {props.name} vous avez {props.age}!</h1>
            {props.children}
        </div>
    )
}
```

Exercice

Reproduire cet affichage:

Props par class

Pour les class cela change un peu :

App.js :

```
<Welcome name="Thomas" age="23 ans">
    ceci est un test
    </Welcome>
    <Welcome name="Toto" age="90 ans" />
    <Welcome name="John" age="33 ans" />
```

Props par class

Pour les class cela change un peu :

Welcome.js :

```
class Welcome extends Component {
    render() {
        return <h1> Welcome {this.props.name} vous avez {this.props.age} !</h1>
    }
}
```

Props VS state

Props VS state

Les props ne sont pas modifiable et ne peuvent pas etre transisent hors de leurs composants. C'est la ou les states rentrent en jeux.

Créons un nouveau component :

Message.js

```
import React, {Component} from "react";
class Message extends Component {
    render() {
        return <h1> Welcome visitor !</h1>
    }
}
export default Message
```

Props VS state

Les props ne sont pas modifiable et ne peuvent pas etre transisent hors de leurs composants. C'est la ou les states rentrent en jeux.

Créons un nouveau component :

Message.js

```
import React, {Component} from "react";
class Message extends Component {
    render() {
        return <h1> Welcome visitor !</h1>
    }
}
export default Message
```

Props VS state

App.js

```
import Message from './components/Message';

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

Props VS state

L’objectif est de créer un bouton et de changer l’affichage du message lors du clique sur ce bouton.

Pour cela nous allons créer une state car une props ne peut être modifiée.

Nous devons dans un premier temps créer un constructeur dans Message.js

Props VS state

```
constructor(){
    super();
    this.state = {
        message : "Welcome visitor"
    }
}
render() {
    return <h1> {this.state.message}</h1>
}
```

Props VS state

On ne voit donc pas vraiment de changement car on lit directement la state présente dans le constructeur.

Codons maintenant le changement de texte lors du clique sur le bouton :

```
return (
    <div>
        <h1> {this.state.message}</h1>
        <button onClick={()=> this.changeMessage()}>Subscribe</button>
    </div>
)
```

Props VS state

```
changeMessage(){
    this.setState({
        message:'Thank you for subscribing !'
    })
}
```

Props VS state

Nous allons installer une extension :

https://marketplace.visualstudio.com/items?itemName=dsznajder.es7-react-js-snippets

Props VS state

Pour mieux comprendre les states, nous allons créer un compteur avec un bouton permettant d'incrémenter une valeur.

Nous allons créer un nouveau component Counter.js.

Grace à l'extension juste en tapant rce je peux crée une composant par class.

Penser à bien retirer le mot export avant la class Counter

Props VS state

Pour mieux comprendre les states, nous allons créer un compteur avec un bouton permettant d'incrémenter une valeur.

Nous allons créer un nouveau component Counter.js.

Grace à l'extension juste en tapant rce je peux crée une composant par class.

Penser à bien retirer le mot export avant la class Counter

App.js

```
import Counter from './components/Counter';

<Counter />
```

Props VS state

Créons maintenant le constructeur :

rconst

```
 constructor(props) {
     super(props)


     this.state = {
        count: 0
     }
   }
```

Props VS state

Pour initialiser la valeur, la logique voudrais que l'on écrivent ce bout de code

```
increment(){
        this.state.count = this.state.count +1
        console.log(this.state.count);
    }
```

Sauf que…..

Props VS state

```
increment(){
    this.setState({
        count: this.state.count +1
    })
    console.log(this.state.count);
}
```

Props VS state

Appelons 5 fois cette fonction :

```
 incrementFive(){
        this.increment()
        this.increment()
        this.increment()
        this.increment()
        this.increment()
}
<button onClick={()=>this.incrementFive()}>Increment five</button>
```

Props VS state

Appelons 5 fois cette fonction :

```
increment(){
    this.setState((prevState)=>({
        count:prevState.count +1
}))
```

Nous pouvons ajouter personnaliser et ajouter la props directement dans la fonction.

```
this.setState((prevState, props)=>({
        count:prevState.count + props.value
}))
```

Props VS state

Voici comment le component sera appelé :

```
<Counter value={10} />
```

Déstructuration des props

Les props peuvent être déstructuré afin de simplifié la lecture et son utilisation.

```
const Greet = ({name, age}) => {
    return (
        <div>
            <h1>Hello {name} vous avez {age}!</h1>
        </div>
    )
}
```

Déstructuration des props par fct

Les props peuvent être déstructuré afin de simplifié la lecture et son utilisation.

```
const Greet = ({name, age, children } ) => {
    return (
        <div>
            <h1>Hello {name} vous avez {age}!</h1>
            <p>{children}</p>
        </div>
    )
}
```

Déstructuration des props par fct

```
const Greet = (props) => {
    const {name, age, children } = props
    return (
        <div>
            <h1>Hello {name} vous avez {age}!</h1>
            <p>{children}</p>
        </div>
    )
}
```

Déstructuration des props par class

```
const Greet = (props) => {
    const {name, age, children } = this.props
    return (
        <div>
            <h1>Hello {name} vous avez {age}!</h1>
            <p>{children}</p>
        </div>
    )
}
```

Gestionnaire d’événement

Gestionnaire d’événement

Créons un nouveau component FunctionClick.js :

rfce

Gestionnaire d’événement

Créons un nouveau component FunctionClick.js :

```
function FunctionClick() {
    function clickHandler(){
        console.log("Button clicked !");
    }
  return (
    <div><button onClick={clickHandler}>Click</button></div>
  )
}
```

Gestionnaire d’événement

```
return (
   <div><button onClick={clickHandler}>Click</button></div>
 )
```

Attention **clickHandler n’as pas de parenthèse** lors de l’appel sinon ça serait un appel de fonction

Gestionnaire d’événement

```
return (
   <div><button onClick={clickHandler}>Click</button></div>
 )
```

Attention **clickHandler n’as pas de parenthèse** lors de l’appel sinon ça serait un appel de fonction

Gestionnaire d’événement

Voyons la même chose avec une class, créons un nouveau component : ClassClick.js

Rce

Gestionnaire d’événement

Voyons la même chose avec une class, créons un nouveau component : ClassClick.js

Rce

Gestionnaire d’événement

```
export class ClassClick extends Component {
   clickHandler(){
      console.log("Button clicked !");
   }
  render() {
   return (
      <div><button onClick={this.clickHandler}>Click Me</button></div>
   )
  }
}
```

Affichage conditionné

Affichage conditionné

Créons un nouveau composant : UserGreeting.js

```
 constructor(props) {
    super(props)
    this.state = {
      isLoggedIn:false
    }
 }
```

Affichage conditionné

If et else :

```
render() {
    if (this.state.isLoggedIn) {
      return <div>Welcome Fred !</div>
    } else {
      return <div>Welcome Guest</div>
    }
  }
```

Affichage conditionné

Operateur « ternaire »

(ternary-operator)

```
render() {
  return this.state.isLoggedIn ? (
    <div>Welcome Fred !</div>
  ) : (
    <div>Welcome Guest</div>
  )
}
```

Affichage conditionné

Operateur circuit court

```
render() {
  return this.state.isLoggedIn && <div>Welcome Fréd</div>
 }
```

Affichage conditionné

Avec une variable intermédiaire

```
render() {
   let message
   if (this.state.isLoggedIn) {
     message = <div>Welcome Fréd</div>
   } else {
     message = <div>Welcome Guest</div>
   }
   return <div>{message}</div>
 }
```

Affichage conditionné

Crée un composant AdminTest

Créer une state admin.

Si il est vrai afficher un bouton edit sinon afficher "Veuillez-vous connecter en tant qu'administrateur"

Faire cela avec une variable intermédiaire et un ternaire.

Liste de données

Liste de données

Créons un nouveau Component NameList.js

```
const names = ['John', 'Malcolm', 'Richard'];
return (
  <div>
    <h2>{names[0]}</h2>
    <h2>{names[1]}</h2>
    <h2>{names[2]}</h2>
  </div>
)
```

Liste de données (arrays)

Pour éviter les répétitions, vous avez le .map qui permets de parcourir les listes de données

```
const names = ['John', 'Malcolm', 'Richard'];
 return (
  <div>
    {
       names.map(name => <h2>{name}</h2>)
    }
  </div>
 )
```

Liste de données

On peut raccourcir tout cela en une seule et même ligne de code

```
function NameList() {
    const names = ['John', 'Malcolm', 'Richard'];
    const nameList = names.map(name => <h2>{name}</h2>);
  return (<div>{nameList}</div>)
}
export default NameList;
```

Liste de données

Prenons un exemple un peu plus concret. Comment faire quand nous avons un object à afficher :

Récupérer l’object Person auprès de votre formateur

```
const persons = [
   {
     id: 1,
     name: 'Bruce',
     age: 30,
     skill: 'React'
   }, [….]
]
```

Liste de données

Pour accéder à cette objet rien de bien complexe :

```
 const personList = persons.map(person => (
     <h2>
       I am the {person.name}. I am {person.age} years old. I know {person.skill}
     </h2>
))
 return (<div>{personList}</div>)
```

Liste de données

Pour un code plus propre nous allons mettre la valeur du return dans un autre component

Liste de données

Pour accéder à cet objet rien de bien complexe :

```
function Person({person}) {
  return (
    <div>
      <h2>
        I am {person.name}. I am {person.age} years old. I know {person.skill}
      </h2>
    </div>
  )
}
```

Liste de données

Il faut ensuite modifier l'appel du component

```
const personList = persons.map(person => <Person person={person} />)
```

Cela fonctionne mais nous obtenons une erreur dans la console du navigateur. Pour la solutionner il faut rajouter une key lors de l'appel du composant :

```
<Person key={person.id} person={person} />
```

Styles avec React

Faire du style avec React

1. Feuille de style css
2. Style en inline
3. CSS Modules
4. Styled Component

Faire du style avec React

Créer un composant Inline.js

```
const heading = {
  fontSize: '72px',
  color: 'blue'
}
function Inline() {
  return (
    <div>
      <h1 style={heading}>Inline</h1>
    </div>
  )
}
```

Faire du style avec React

Créons un component Stylesheet

```
function Stylesheet() {
return (
    <div>
<h1> Stylesheets </h1>
</div>
  )
}

export default Stylesheet
```

Faire du style avec React

Nous allons créer un nouveau fichier myStyle.css directement dans le dossier components

```
.primary {
  color: orange;
}
```

Faire du style avec React

```
function Stylesheet() {
return (
   <div>
<h1 className="primary">Stylesheet</h1>
</div>
  )
}
export default Stylesheet
```

Faire du style avec React

On peut également conditionner le style grâce au props

```
function Stylesheet(props) {
  let className = props.primary ? 'primary' : ''
  return (
    <div>
      <h1 className={className}>Stylesheet</h1>
    </div>
  )
}
```

Faire du style avec React

App.js

<Stylesheet primary={true} />

Faire du style avec React

Créer un fichier dans le dossier src/ appStyles.module.css

```
.success{
        color: green;
}
```

Faire du style avec React

Dans App.js

```
import './appStyles.css'

import styles from './appStyles.module.css’


 <h1 className='error'>Error</h1>

<h1 className={styles.success}>Success</h1>
```

Faire du style avec React

Exercice

Stylisons le component Person :

Chaque h2 devrons avoir une bordure noire de 3px et des bords arrondis (10px).

Si le nom du hero est Diana le texte sera de couleur rose et un fond de couleur : darkslategray

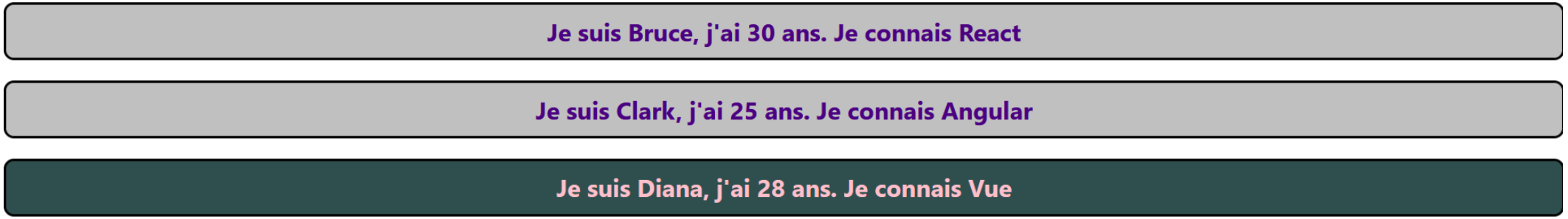
Dans les autres cas la couleur du texte sera indigo (#2e006c) et un fond de couleur : silver

Utilisez le fichier myStyle.css ou avec appStyles.module.css

Bonus :

Permettre à l'utilisateur de choisir la couleur du texte en props

Faire du style avec React



Les formulaires

Les formulaires

```
this.state = {
        email: ' '
}
```

```
this.changeEmailHandler = (event ) => {
        this.setState({email: event.target.value})
}
```

```
<input type='text'   value={this.state.email}     onChange={this.changeEmailHandler}    />
```

Les Formulaires

Crée un componant par class

```
class Form extends Component {
        constructor(props) {
                super(props)
                this.state = {
                        username: '',
                        comments: '',
                        topic: 'react'
                }
        }
}
export default Form
```

Les Formulaires

Récupérer le formulaire transmis par votre formateur et mettre dans le return :

```
render() {
        const { username, comments, topic } = this.state
        return (
            <form onSubmit={this.handleSubmit>
                [...]
            </form>
        )
    }
```

Les Formulaires

Pour chacune des valeurs mis en input il faut son « handle »

```
handleUsernameChange = event => {
        this.setState({
            username: event.target.value
        })
    }
```

Les Formulaires

Pour chacune des valeurs mis en input il faut son « handle »

```
handleCommentsChange = event => {
    this.setState({
        comments: event.target.value
    })
}
```

Les Formulaires

Pour chacune des valeurs mis en input il faut son « handle »

```
handleTopicChange = event => {
        this.setState({
                topic: event.target.value
        })
}
```

Les Formulaires

Pour chacune des valeurs mis en input il faut son « handle »

```
handleSubmit = event => {
    alert(`${this.state.username} ${this.state.comments} ${this.state.topic}`)
    event.preventDefault()
}
```

Attention au `` ce sont des backquotes

Les Formulaires

Exercice

Crée un formulaire qui permet de s’inscrire à une newsletter.

Un champ nom

Un champ prénom

Un champ email

Un bouton de validation

La page affichera un message : « Merci, [Nom] [prénom] d’avoir pris contact avec nous. Nous reviendrons vers vous à cet email : [email] »

Les Formulaires

Exercice

Crée un formulaire saisie controlé.

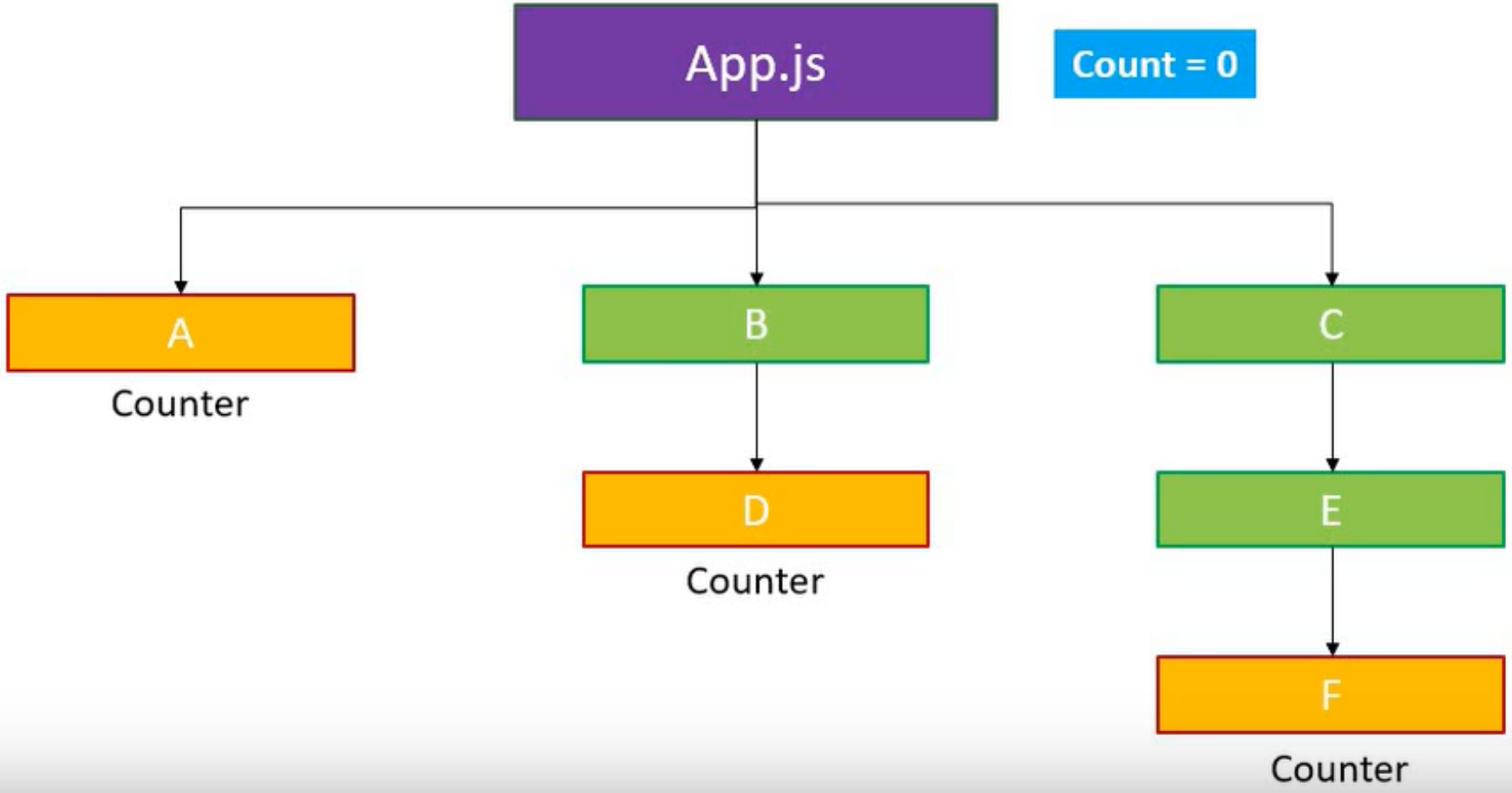
Un champ pseudo, si celui fait moins de 3 caractères l'input aura des bordures rouges.

Les bordures deviendront verte si le champ pseudo fait plus de 3 caractères

useReducer avec useContext

useReducer avec useContext

Nous allons voir les Reducer avec un counteur que nous allons passer de component à component (A, D,F). Le compteur va être initialisé dans App.js



useReducer avec useContext

Nous allons recréer une application pour qu'on est des bases propre.

Dans App.js, nous allons initialiser un compteur :

```
const initialState = 0

const reducer = (state, action) => {
        switch (action) {
                case 'increment':
                        return state + 1
                case 'decrement':
                        return state - 1
                case 'reset':
                        return initialState
                default:
                        return state
        }
}
```

useReducer avec useContext

Nous allons recréer une application pour qu'on est des bases propre.

Dans App.js, dans la fonction :

const [count, dispatch] = useReducer(reducer, initialState)

Pour appeler notre reduceur

useReducer avec useContext

Créons le componant A :

```
import React, {useContext} from 'react'
import { CountContext } from '../App';
function ComponentA() {
  const countContext = useContext(CountContext)
  return (
    <div>
      Component A {countContext.countState}
      <button onClick={() => countContext.countDispatch('increment')}>Increment</button>
     <button onClick={() => countContext.countDispatch('decrement')}>Decrement</button>
     <button onClick={() => countContext.countDispatch('reset')}>Reset</button>
    </div>
  )
}
export default ComponentA
```

useReducer avec useContext

Faire la même chose mais avec le composant D et F

useReducer avec useContext

```
import React, {useContext} from 'react'
import { CountContext } from '../App';
function ComponentD() {
  const countContext = useContext(CountContext)
  return (
    <div>
      Component D {countContext.countState}
      <button onClick={() => countContext.countDispatch('increment')}>Increment</button>
     <button onClick={() => countContext.countDispatch('decrement')}>Decrement</button>
     <button onClick={() => countContext.countDispatch('reset')}>Reset</button>
    </div>
  )
}
export default ComponentD
```

useReducer avec useContext

```
import React, {useContext} from 'react'
import { CountContext } from '../App';
function ComponentF() {
  const countContext = useContext(CountContext)
  return (
    <div>
      Component F {countContext.countState}
      <button onClick={() => countContext.countDispatch('increment')}>Increment</button>
     <button onClick={() => countContext.countDispatch('decrement')}>Decrement</button>
     <button onClick={() => countContext.countDispatch('reset')}>Reset</button>
    </div>
  )
}
export default ComponentF
```

useReducer avec useContext

Il faut ensuite donner un contexte dans l'App.js

```
<CountContext.Provider
value={{ countState: count, countDispatch: dispatch }}
>
```

useReducer avec useContext

Il faut ensuite donner un contexte dans l'App.js

```
<CountContext.Provider
value={{ countState: count, countDispatch: dispatch }}
>
export const CountContext = React.createContext()
```

Affichage data

AFFICHER UNE DATA DEPUIS UNE API (AXIOS)

Affichage data

Créons une nouvelle app.

Installons axios

Npm i axios

Créons un nouveau composant DataFetchingOne.js

rfce

DataFetchingOne.js

```
import React, {useState, useEffect} from 'react'
import axios from 'axios’;


function DataFetchingOne() {
  const [loading, setLoading] = useState(true)
  const [error, setError] = useState('')
  const [post, setPost] = useState({})
```

DataFetchingOne.js

```
useEffect(() => {

axios.get(`https://jsonplaceholder.typicode.com/posts/1`)
      .then(response => {
        setLoading(false)
        setPost(response.data)
        setError('')
      })
      .catch(error => {
        setLoading(false)
        setPost({})
        setError('Something went wrong!')
      })
  }, [])
```

DataFetchingOne.js

```
return (
  <div>
    {loading ? 'Loading' : post.title}
    {error ? error : null}
  </div>
 )
}
```

DataFetchingOne.js

Exercice

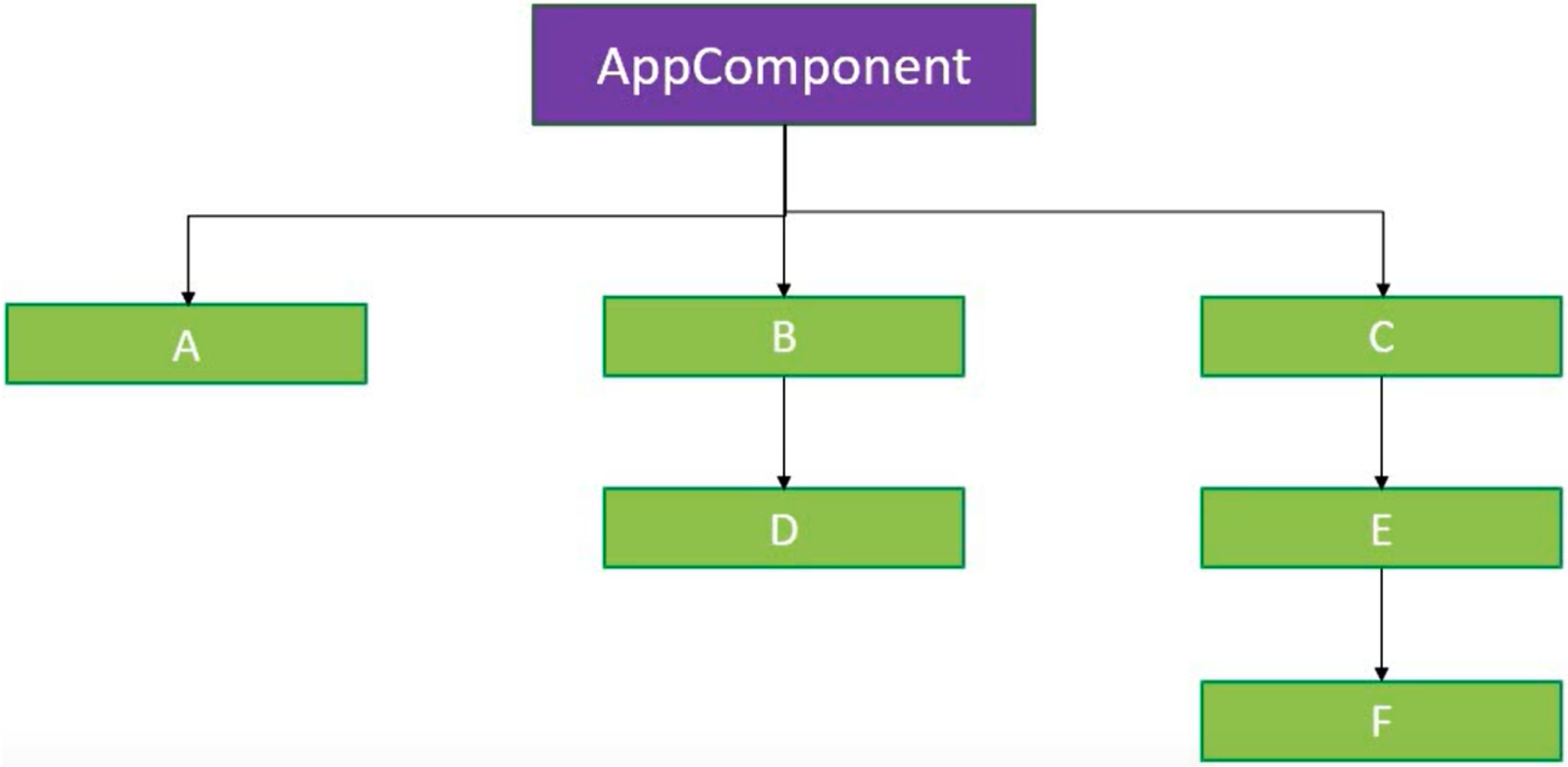
API FÏLM

Affichez le titre, le poster, l'année de sortie du film et combien il a rapporté au box office.

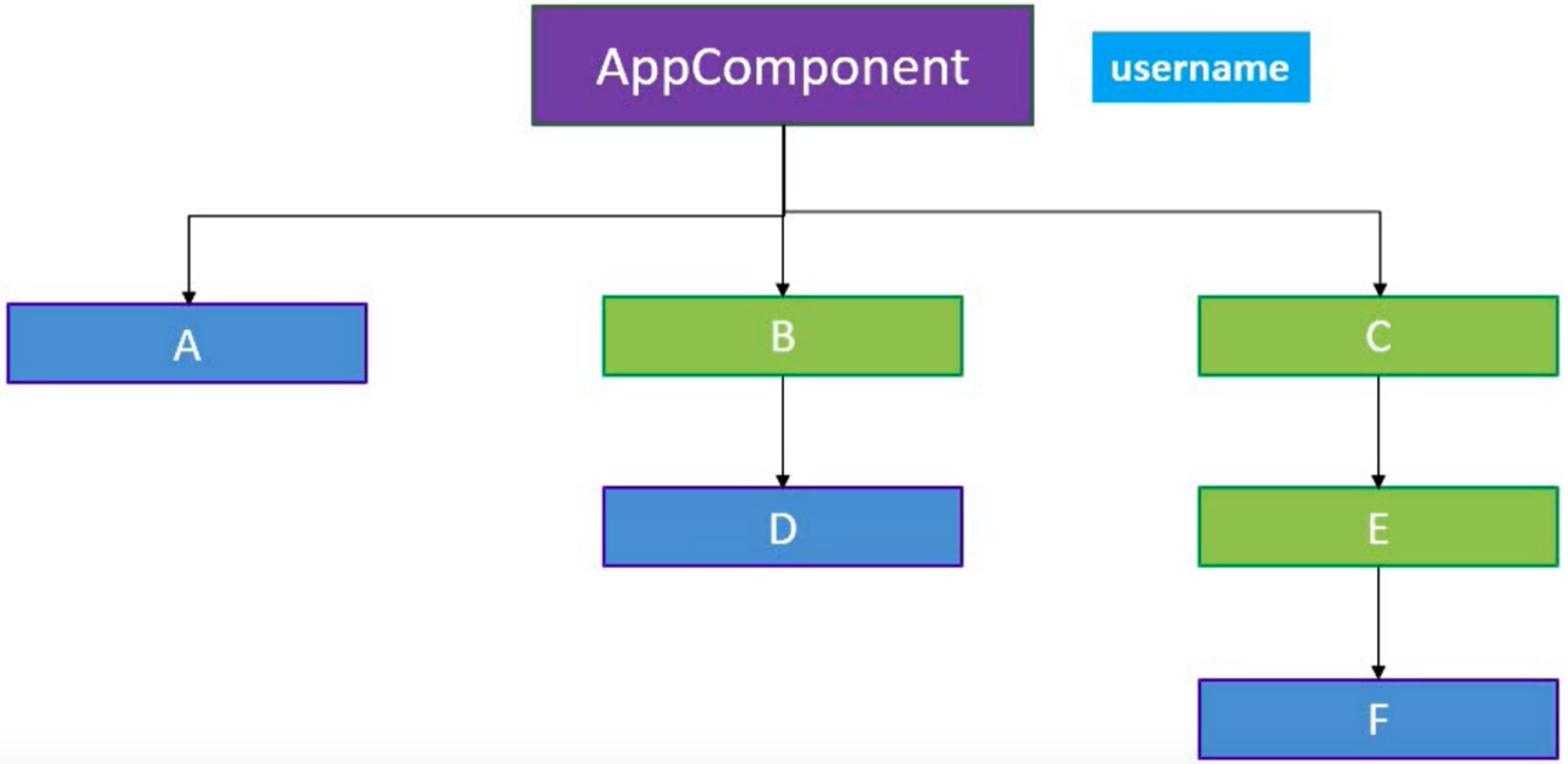
Bonus: on fait un input et l'utilisateur tape le nom du film

Contexte

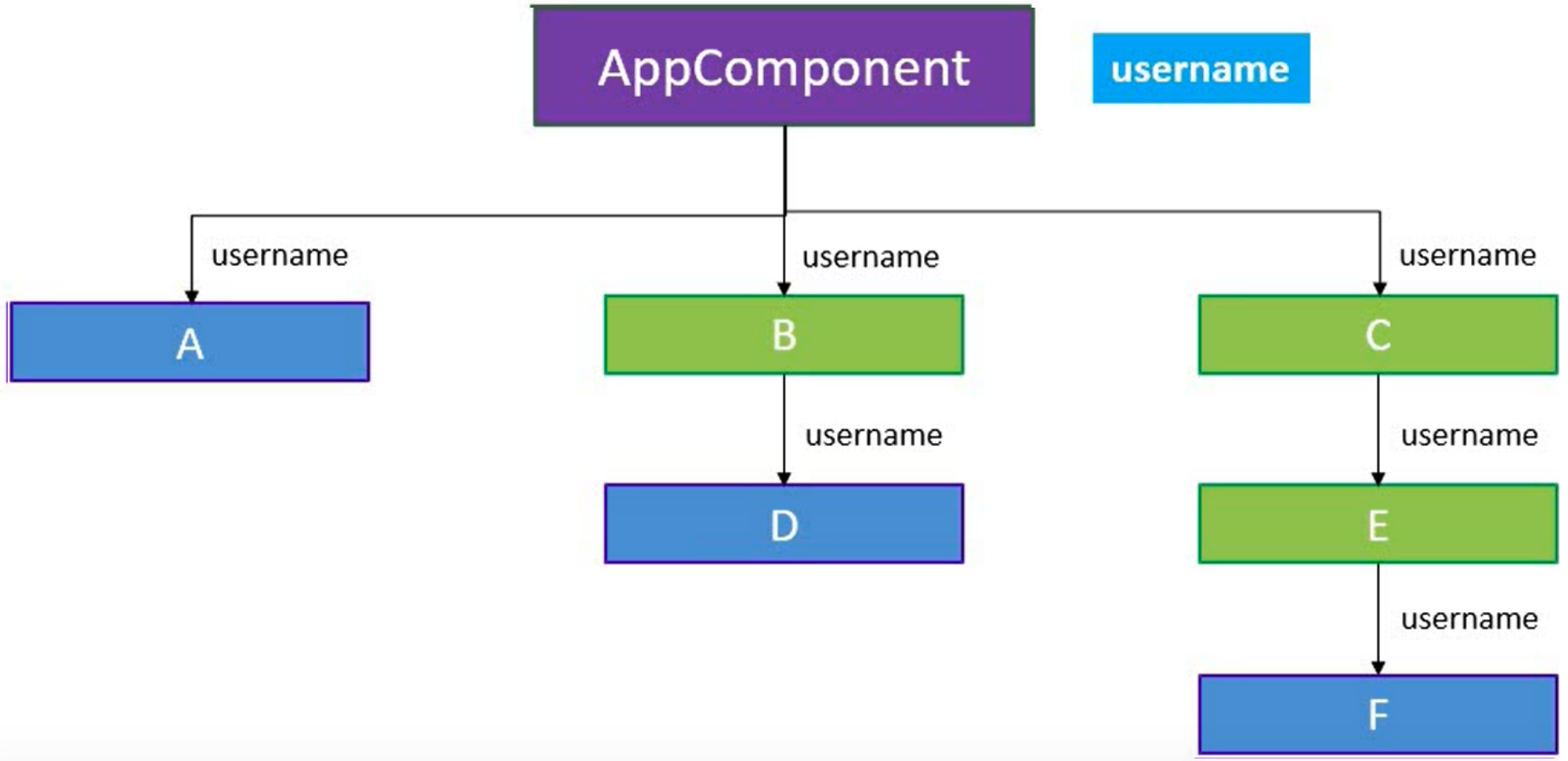
Contexte



Contexte



Contexte


AppComponent
username
username
username
username
A
B
C
username
username
D
E
username
F

Le Contexte

Contexte va permettre de faire passer une data sans passer par l’username de component en component

Le Contexte

Créons 3 composants. Component C, E, F qui s’appellent entre eux.

Le Contexte

Créons un component userContext.js

```
import React from 'react'


const UserContext = React.createContext('codevolution')


const UserProvider = UserContext.Provider
const UserConsumer = UserContext.Consumer


export { UserProvider, UserConsumer }
```

Le Contexte

1. Crée le contexte
2. Fournir une variable au contexte (Provide)
3. Consommer la variable du contexte (Consume)

Le Contexte

Dans app.js

```
import { UserProvider } from './components/userContext'

<UserProvider value="Fred">
        <ComponentC />
</UserProvider>
```

Le Contexte

Dans ComponentF.js

```
render() {
    return (
        <UserConsumer>
            {username => {
                return <div>Hello {username}</div>
            }}
        </UserConsumer>
    )
}
```

Les Fragments

Les Fragments

Lors d’un return

```
<React.Fragment>
    {loading ? 'Loading' : "title : " + post.title}
    {error ? 'error': null}
</React.Fragment>
```

Les Fragments

Lors d’un return

```
<React.Fragment>
    {loading ? 'Loading' : "title : " + post.title}
    {error ? 'error': null}
</React.Fragment>
```

DataFetchAll.js

```
const [posts, setPost] = useState({})
 axios.get(`http://localhost:5000/`)
            .then(response =>{
                setLoading(false)
                setPost(response.data)
                setError('')
})
```

DataFetchAll.js

```
const [posts, setPost] = useState({})
 {loading ? 'Loading' : posts.map((post, index) => {
        return (
          <div key={index}>
            <h2>firstname: {post.firstname}</h2>
            <h2>email: {post.email}</h2>
            <hr />
          </div>
        );
})}
```

DataFetchAll.js

Exercice

Récupérer et afficher la liste de toutes les todos :

https://jsonplaceholder.typicode.com/todos/

Afficher le titre, userId et l'etat (completed ou non)

Mettre un fond vert si l'etat completed est true sinon mettre le fond en rouge si l'etat est false

Mettre une ligne de séparation pour chaque élément.

DataFetchAll.js - reducer

Les reducers sont conseillé lorsqu’on veux afficher de la data depuis un array ou un obj

DataFetchAll.js - reducer

```
const initialState = {
        loading: true,
        error: '',
        post: {}
}
```

DataFetchAll.js - reducer

```
const reducer = (state, action) => {
switch (action.type) {
        case 'FETCH_SUCCESS’:
                return {
                        loading: false,
                        posts: action.payload,
                        error: ''
                }
```

DataFetchAll.js - reducer

```
        case 'FETCH_ERROR':
                return {
                        loading: false,
                        posts: {},
                        error: 'Something went wrong!'
                }
            default:
                return state
        }
}
```

DataFetchAll.js - reducer

```
function DataFetchingTwo() {
        const [state, dispatch] = useReducer(reducer, initialState)
useEffect(() => {
                axios
                        .get(`http://localhost:5000/`)
                        .then(response => {
                                dispatch({ type: 'FETCH_SUCCESS', payload: response.data })
                        })
                        .catch(error => {
                                dispatch({ type: 'FETCH_ERROR' })
                        })
        }, [])
```

DataFetchAll.js - reducer

```
function DataFetchingTwo() {
        const [state, dispatch] = useReducer(reducer, initialState)
useEffect(() => {
                axios
                        .get(`http://localhost:5000/`)
                        .then(response => {
                                dispatch({ type: 'FETCH_SUCCESS', payload: response.data })
                        })
                        .catch(error => {
                                dispatch({ type: 'FETCH_ERROR' })
                        })
        }, [])
```

DataFetchAll.js

```
{state.loading ? 'Loading' : state.posts.map((post, index) => {
      return (
        <div key={index}>
          <h2>firstname: {post.firstname}</h2>
          <h2>email: {post.email}</h2>
          <hr />
        </div>
      );
})}
```

DataFetchAll.js

Exercice

Récupérer et afficher la liste de toutes les todos :

https://mocki.io/v1/78c793c7-58cb-416f-b2ca-d5d737f7c17e

Afficher le titre, description et la catégorie dans un tableau

Utiliser un reducer.

DataFetchAll.js

Scenario	useState	useReducer
Type de state	Number, String, Boolean	Object ou Array
Nombre de state de transition	Un ou deux	Beaucoup
State de transition	Non	Oui
Local vs global	Local	Global