React.JS

Framework vs Library:

une **library** est une **collection fonctions/méthodes”** à employer comme un couteau suisse, un **framework** (littéralement “cadre de travail”) est une collection de classes/méthodes qui impose une manière de faire, un **protocole de travail**, basée sur des bonnes pratiques répétées.

SPA:

Single **P**age **A**pplication. Application mono page, permet de fluidifier l'experience utilisateur en **évitant le temps de chargement des pages**.

Les composants:

il y a deux facons décrire les composants:

sous forme de fonction:

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

sous forme de class:

```
class composant extends react.Component {
  render (){
    return <h1>Hello, {props.name}</h1>
  }
}
```

State vs Props:

Les **props** (propiétés) sont des **parametrès** de notre composant. Ils sont **immutables**. Ne peuvent pas etre modifiés. Ils sont un peu comme les arguments d'une fonction

Le **state**: **permet une mutation** de notre composant (il y aura un rerender quand on change le state, un rerender juste du composant et pas de toute la page, c'est la puissance de react)

https://www.youtube.com/watch?v=T5kMboavS1M

https://github.com/uberVU/react-guide/blob/master/props-vs-state.md

Props:

Composant :

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

class Composant extends Component{
  constructor(){
    super()
    this.state = {
        count : 0
    }
  }
  addOne (){
    this.setState({
      count : this.state.count + 1
    })
  }
  render() {
    return (
      <div>
        <h1>Welcome {this.props.name}</h1>
        <p>Mon compteur : {this.state.count}</p>
        <button onClick = {() => this.addone()}>Ajouter 1 au compteur</button>
      //<button onClick = {this.addone.bind(this)}>Ajouter 1 au compteur</button>
      </div>
    );
  }
}

export default Composant
```

App:

```
import React, {Component} from 'react'
import Composant from './Composant'
import './App.js'

class App extends Component{
  render () {
    return (
      <Composant name = 'Jackson'/>
    )
  }
}

export default App
```

State:

composant:

```
import React, {Component} from 'react';
class Composant extends Component{
  render() {
    return (
      <h1>Welcome {this.props.name}</h1>
    );
  }
}

export default Composant
```

App:

```
import React, {Component} from 'react'
import Composant from './Composant'
import './App.js'

class App extends Component{
  render () {
    return (
      <Composant name = 'Jackson'/>
    )
```

```
  }
}

export default App
```

Key:

Une **key** est un **attribut** qu'on doit inclure quand on crée une list d'éléments Elle donne à l'élément une **identité** "stable". Elle aide react à savoir quel élément a changé, bougé ou a été supprimé.

Package.json:

Le **package.json** est un peu le **manifeste** de notre projet. on y retrouve les **dépendances** nécessaires au projet, les **scripts** qui seront lancés à la commande 'npm start' ou 'nmp test',

et d'autres éléments de configuration type linter.

Store:

un **store** permet de stocker et gérer des données de manière partagée entre plusieurs composants de notre application.

Auparavant on utilisait Redux ou Flux ou Mobx pour gérer le store. Désormais il y **Context** et les **Hooks.**

Les **context** permettent de partager des données entre les composants sans passer par les props .

Pour utiliser les hooks il faut écrire ses composants sous forme de fonction.

Les hooks retournent des methodes qui permettent de modifier le store.

Hook:

Api fournit par react qui comprend une série de fonction qui permettent de gérer l'état

```
function Composant (props) {

  const state = useState(0)
  return <h1>Hello, {props.name}</h1>
}
```