# ReactJS - Create React App

# L'outil Create React App

Outil en ligne de commande pour créer des applications ReactJS avec:

- Serveur de développement
- Outils de build avec PostCSS
- Architecture optimisée pour une bonne modularité

En interne, utilise Webpack, Babel, ESLint etc...

https://create-react-app.dev/

# Créer une nouvelle app ReactJS

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

# Fonctionnement de CRA

Pour que le projet puisse bien recharger, ces 2 fichiers doivent exister avec exactement le même nom:

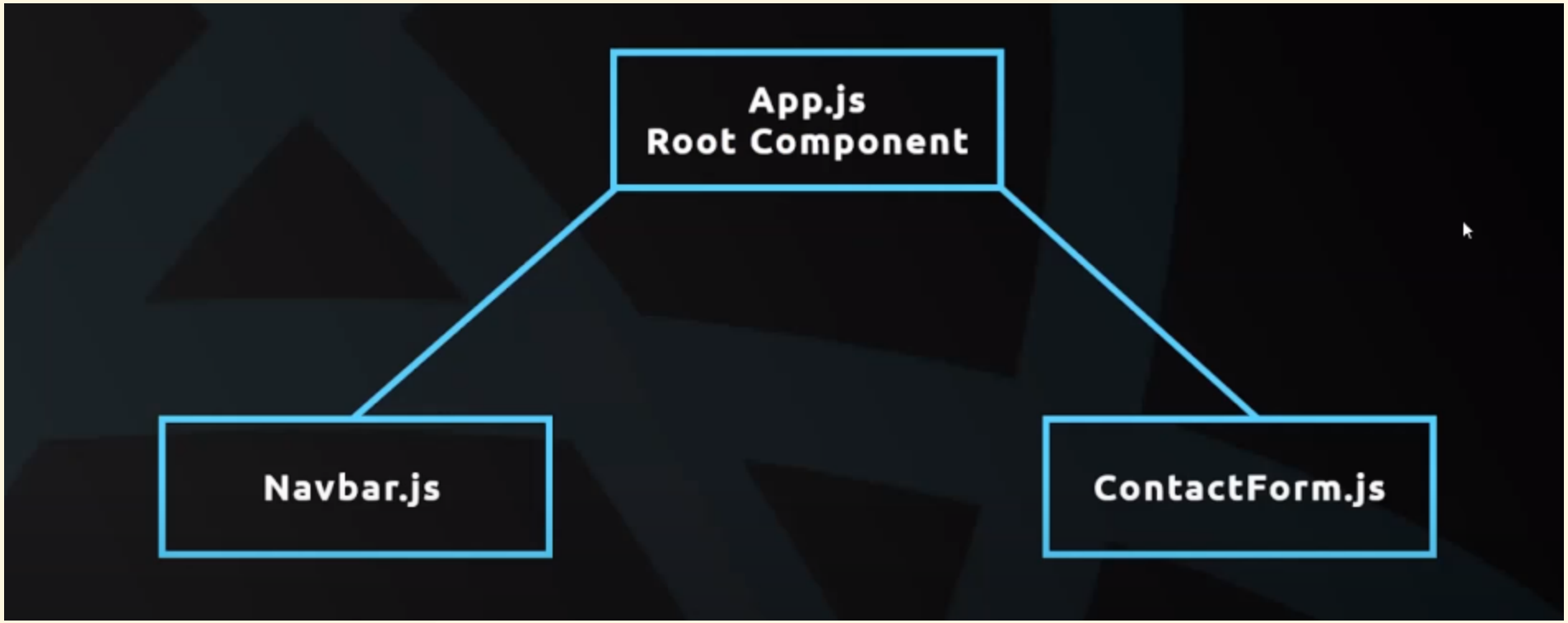
- `public/index.html` sera notre fichier `.html`
- `src/index.js` sera le point d'entrée de toute notre app `.js`

Tous les autres fichiers peuvent être supprimés
**Tous nos futurs fichiers CSS ou JS doivent être dans le dossier src, sinon seront pas vus par Webpack**

Il est conseillé de **créer des sous-dossiers dans** `src/` pour une meilleure organisation

# Composants d'imbrication

“ Une grande partie de la puissance de ReactJS réside dans sa capacité à permettre l’imbrication de composants. ”



# Créer un nouveau composant

```
// File Counter.js
import React, { Component } from 'react';

class Counter extends Component {
    state = {  }
    render() {
        return (
            <button>Cliquez-moi</button>
         );
    }
}

export default Counter;
```

# Import du nouveau composant

```
// File App.js
import React from 'react';
import Counter from './Counter';


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

export default App;
```

- Il est possible d'importer un composant plusieurs fois
- Les états ( `state` ) sont indépendants pour chaque instance du composant

```
state = {count: 0}

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

render() {
    return (
        <button onClick={this.increment}>Cliqué {this.state.count} fois</button>
     );
}
/* Each Counter has its own count
```

# Les props

Les `props` permettent de **transmettre des données** d'un **composant parent à un composant enfant**

Raccourci de `properties`, càd propriétés.

```
//App.js
<Player pseudo="Blondie75" age="75" />
```

```
//Player.js
<h2>{this.props.pseudo} is {this.props.age} years</h2>
```

- Les props sont en lecture seule

Utiliser le principe de décomposition ( `unpacking` ) pour une méthode raccourcie:

```
    const {pseudo, age} = this.props;
    return (
        <h2>{pseudo} is {age} years</h2>
    )
```

- On peut transmettre un objet complet comme `props`

```
<ListPlayers players={this.state.players} />
```

# Afficher une liste d'éléments

Grâce à la méthode `map`

```
const players = props.players;

const listPlayers = players.map((player) =>
  <li>{player}</li>
);

return (
    <ul>{listPlayers}</ul>
)
```

https://fr.reactjs.org/docs/lists-and-keys.html

# Les composants sans état

Les composants sans état ont différents noms:

- Stateless Components
- UI Components

Ont des `props`, mais pas de `state`
Ils servent principalement à l'interface utilisateur

Il est alors recommandé d'utiliser la déclaration en **programmation fonctionnelle**

# Déclaration d'un composant sans état

```
import React from 'react';

const Player = (props) => {
    return (
        <h2>{props.pseudo} is {props.age} years </h2>
     );
}

export default Player;
```

# Affichage conditionnelle

```
// user.js
 const isLoggedIn = props.isLoggedIn;
 if (isLoggedIn) {
     return <LoggedInUser />;
     }
 else {
      return <GuestUser />;
   }
 }


 // App.js
 <User isLoggedIn={false} />
```

# Tri conditionnel dans une liste d'éléments

```
const players = props.players;

const listPlayers = players
    .filter(player => player.age >= 18)
    .map((player) => <li>{player}</li>);

return (
    <ul>{listPlayers}</ul>
)
```

# Passer une fonction en tant que props

```
// Player.js

return(
    <form onSubmit={this.props.addPlayer(this.state)}>
    ...
)
```

```
// App.js
    addPlayer = (player) => {
        let players = [...this.state.players, player];
        this.setState({
            players: players
        })
    }}
```

```
// App.js
    <FormPlayer addPlayer={this.addPlayer} />
```

**L'objectif: Penser en ReactJS**

https://fr.reactjs.org/docs/thinking-in-react.html

- Définir dans quel composant doit se situer les états
- Toutes les méthodes qui modifient ces états doivent être dans le même composant

# Identifier où les états doivent vivre

**Exemple:**

- Si on a un composant pour ajouter dans une liste `players` un élément: `<AddPlayer/>`
- Si on a un composant pour l'affichage : `<ListPlayers/>`
- Alors la liste `players` et toutes les méthodes qui modifient cet état doivent se situer dans un composant parent de `<AddPlayer/>` et `<ListPlayers/>`
- Les méthodes seront transmises aux enfants grâce aux props

# Importer du CSS ou des images

```
// Button.js

import "./Button.css";
import Logo from "./logo.png";

return(
    <button className="button button--primary">
        <img src={Logo} alt="Logo" />
    </button>
)
```