formation.tech

React

React - Installation

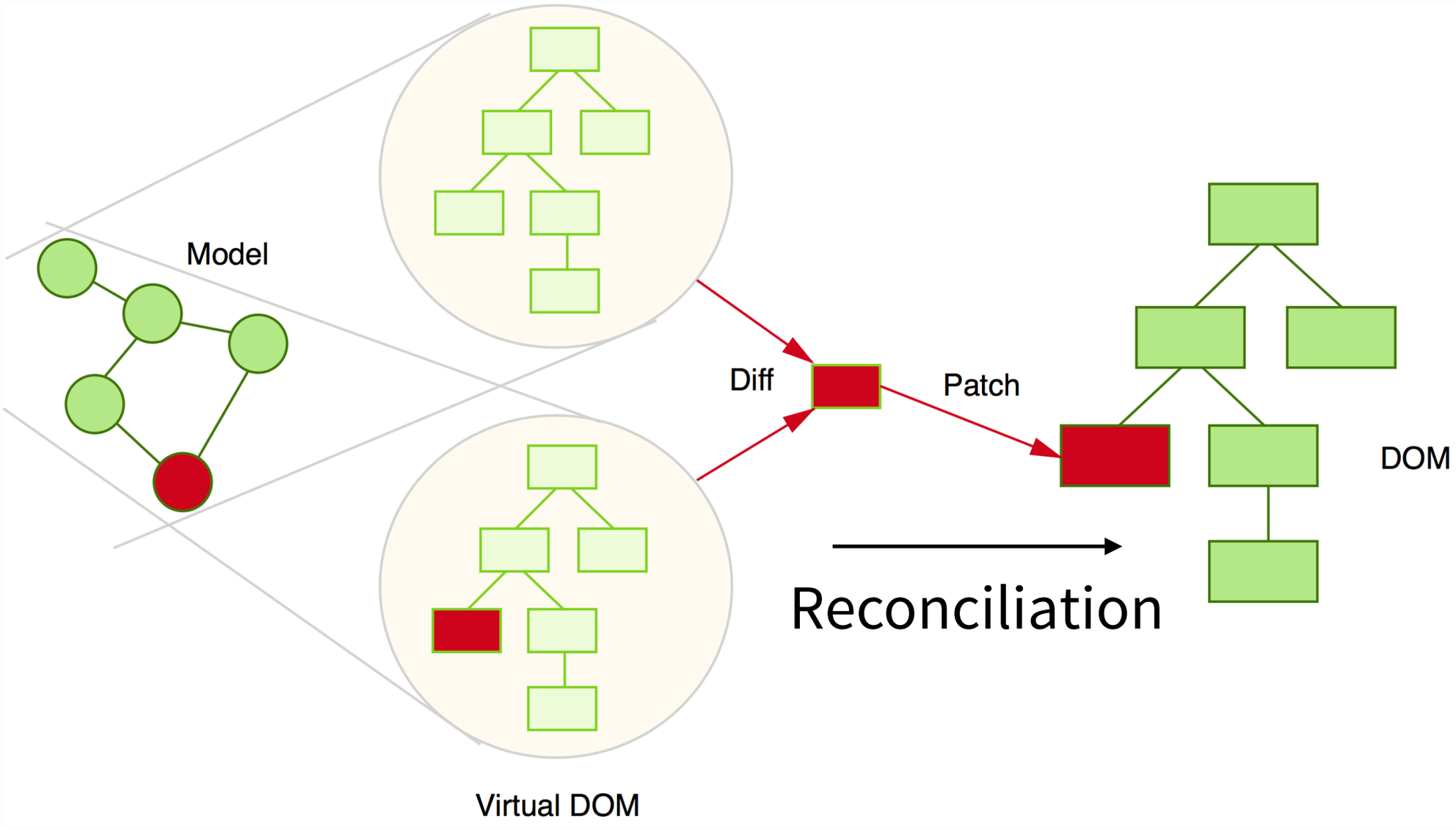
- Pour mettre en place rapidement un environnement React fonctionnel, on peut utiliser le package `create-react-app`
- Installation avec npm :
  ```
  npm install -g create-react-app
  ```
- Installation avec Yarn :
  ```
  yarn add create-react-app
  ```
- Pour créer le projet :
  ```
  create-react-app NOM_DU_DOSSIER
  ```
- Créer le projet directement
  ```
  npx create-react-app NOM_DU_DOSSIER
  npm init react-app NOM_DU_DOSSIER
  yarn create react-app NOM_DU_DOSSIER
  ```

React - Premier composant

- 2 dépendances :
  - react : permet la création de composants
  - react-dom : permet le rendu de ces composants dans le contexte du DOM
- Notre premier composant App est une simple fonction, qui retourne une syntaxe proche du HTML appelée JSX (l'import de React est obligatoire dans ce cas)
- Par convention les composants React commencent par une majuscule (si vous ne le faites pas, React voit du HTML)

```
import React from 'react';
import ReactDOM from 'react-dom';

const App = () => <h1 className="my-app">Hello</h1>;

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

React - Virtual DOM



React - JSX

- Documentation
  https://facebook.github.io/jsx/
- Language proche du HTML nécessitant une compilation (avec Babel par exemple et son plugin babel-plugin-transform-react-jsx)
- Exemple précédent sans JSX :

```
import React from 'react';
import ReactDOM from 'react-dom';

ReactDOM.render(
  React.createElement('h1', {className: 'my-app'}, 'Hello'),
  document.getElementById('root'),
);
```

React - JSX

- Conditions en JSX
  - if simple

```
import React from 'react';

export const Todo = (props) => {
  return (
    <div>
      <input value={props.value} />
      {props.isDeletable && <button>-</button>}
    </div>
  );
};
```

  - if … else

```
import React from 'react';

export const Todo = (props) => {
  return (
    <div>
      <input value={props.value} />
      {(props.isDeletable) ? <button>-</button> : <button disabled>-</button> }
    </div>
  );
};
```

React - JSX

- Listes en JSX

```
import React from 'react';

export const TodoList = (props) => {
  const listItems = props.todos.map((val, i) =>
    <div key={i}>{val}</div>
  );
  return <div>{listItems}</div>;
};
```

- L’attribut key est obligatoire
  Il permet à React de savoir si cet élément de la liste doit être ou non rafraîchi.
  Idéalement une clé id d’un Enregistrement ou Document de base de données.

- Bonne pratique sinon : générer un id avec uuid par exemple.

React - Stateful components

- Autant les composants les plus simple peuvent être de simple fonction comme vu précédemment, autant la plupart du temps il faudra créer des fonctions constructeurs JS (class en ES6).
- Ces composants auront la possibilité d'entrer en interaction avec d'autres fonctions et leur « state ».
- A minima, écrire une classe qui hérite de `React.Component` et qui implémentent une méthode `render` retournant un sous-arbre JSX.

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
  render() {
    return <h1 className="my-app">Hello</h1>
  }
}

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

React - Stateful components

- Sur plusieurs lignes :

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1 className="my-app">Hello</h1>
        <p>World</p>
      </div>
    );
  }
}

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

- Des parenthèses sont obligatoires si le JSX ne commence pas sur la même ligne que le return.
- Un composant doit avoir un seul élément racine (ici), depuis React 16 on peut retourner un tableau et depuis React 16.2 on peut retourner un Fragment (voir doc)

React - Propriétés

- Les propriétés ou props, permettent de passer des valeurs au moment du rendu du composant (syntaxe proche d'un attribut HTML)
- Pour accéder à une propriété depuis le composant on passer par sa propriétés props (ici en JSX {this.props.content})

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
  render() {
    return <h1 className="my-app">{this.props.content}</h1>
  }
}

ReactDOM.render(
  <App content="Hello props"/>,
  document.getElementById('root'),
);
```

React - Propriétés

- Définir la typologie et la validation des propriétés du composants
- Installer prop-types (voir aussi airbnb-prop-types)
  `npm install prop-types`

```
import PropTypes from 'prop-types';

class Contact extends React.Component {
  render() {
    return <p>
      Hello my name is {this.props.name},
      I'm {this.props.age}
      </p>;
  }
}

Contact.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number,
};
```

React - Propriétés

- Validateurs personnalisés :

```
Contact.propTypes = {
  name: PropTypes.string.isRequired,
  age(props, propName, component) {
    if (props[propName] && (props[propName] < 0 || props[propName] > 120)) {
      return new Error(`${propName} should be higher than 0 and lower than 120`)
    }
  },
};
```

- Autres validateurs possibles :
  https://github.com/facebook/prop-types

- Valeurs par défaut :

```
Contact.defaultProps = {
  name: 'John'
};
```

React - Refs

- Référencer des éléments du DOM avec refs

```
class ContactAdd extends React.Component {
  add(e) {
    e.preventDefault();
    console.log(this.refs.prenom.value);
    console.log(this.refs.nom.value);
  }
  render() {
    return <form onSubmit={this.add.bind(this)}>
      <div>
        Prénom : <input ref="prenom" />
      </div>
      <div>
        Nom : <input ref="nom" />
      </div>
      <button>+</button>
    </form>;
  }
}
```

React - State

- Props permet de communiquer avec le composant, state est son état interne, la méthode render est appelée à chaque modification
- On ne peut pas modifier le state directement, il faut utiliser la méthode setState

```
class CounterButton extends React.Component {
  constructor() {
    super();
    this.state = {
      count: 0,
    };
  }
  increment() {
    this.setState({
      count: this.state.count + 1,
    });
  }
  render() {
    return <button onClick={this.increment.bind(this)}>
        {this.state.count}
      </button>;
  }
}
```

React - State

- Il n’est pas nécessaire de modifier tout le state à chaque appel de setState
- Pour des questions de performance, les objets et tableaux du state seront de préférence immuables

```
export class Todo extends Component {
  constructor(props) {
    super(props);
    this.state = {
      saisie: '',
      liste: []
    };
  }

  inputHandler(e) {
    this.setState({
      saisie: e.target.value,
    })
  }

  formHandler(e) {
    this.setState((prevState) => ({
      liste: [...prevState.liste, this.state.saisie]
    }))
  }

  // ...
}
```

React - Imbrication de composants

- Lorsque qu'un state doit être accessible par plusieurs composant, il faut le définir sur le plus proche ancêtre commun, les composants imbriqués y accéder au travers de props

```
class App extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }
  increment() {
    this.setState({ count: this.state.count + 1 });
  }
  decrement() {
    this.setState({ count: this.state.count - 1 });
  }
  render() {
    return <div className="App">
      <h1>{this.state.count}</h1>
      <CounterButton update={this.increment.bind(this)}>+</CounterButton>
      <CounterButton update={this.decrement.bind(this)}>-</CounterButton>
    </div>;
  }
}

class CounterButton extends React.Component {
  render() {
    return <button onClick={this.props.update}>
      {this.props.children}
      </button>;
  }
}
```

React - Formulaires

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

export class ContactAdd extends Component {
  constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(e) {
    this.setState({
      [e.target.name]: e.target.value
    });
  }

  handleSubmit(e) {
    e.preventDefault();
    // fetch()...
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <div className="form-group">
          <label>Prénom</label>
          <input type="text" className="form-control" name="firstName" onChange={this.handleChange} />
        </div>
        <div className="form-group">
          <label>Name</label>
          <input type="text" className="form-control" name="lastName" onChange={this.handleChange} />
        </div>
        <button type="submit" className="btn btn-default">Add</button>
      </form>
    );
  }
}
```

React - Cycle de vie

- Chaque composant à un certain nombre de méthodes liées à son cycle de vies :
- Chargement
  - constructor()
  - ~~componentWillMount~~() (dépréciée)
  - render()
  - componentDidMount()
- Destruction
  - componentWillUnmount()
- Mise à jour
  - ~~componentWillReceiveProps~~() (dépréciée)
  - shouldComponentUpdate()
  - ~~componentWillUpdate~~() (dépréciée)
  - render()
  - componentDidUpdate()
- https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html

React - Cycle de vie

- constructor()
  - Appelée côté client et serveur
  - constructor sert à initialiser state et props

React - Cycle de vie

- componentDidMount()
  - Le rendu initial du composant a été effectué
  - Il est possible de manipuler le DOM
  - N’existe que côté client
  - Le bon endroit pour charger un plugin jQuery ou tout ce qui ne s’exécute qu’une seul fois et qui a besoin d’accéder au DOM
  - Démarrer des requêtes AJAX, des timers

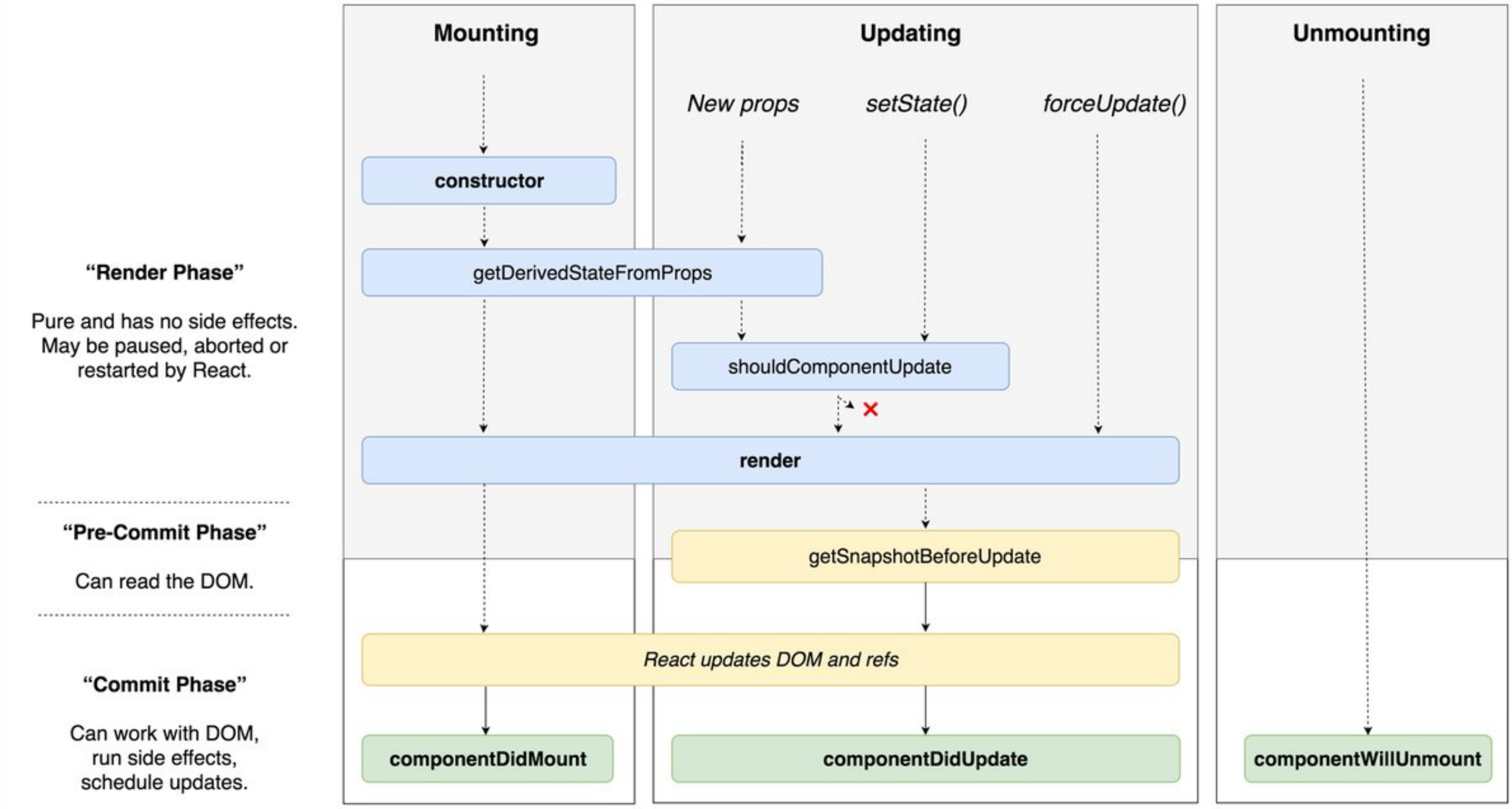
React - Cycle de vie

- shouldComponentUpdate()
  - Permet d'empêcher un render(), doit répondre true ou false
  - Utile pour optimiser une application, ne pas faire de rendu si les props ou le state on été modifié d’une manière qui ne nécessite pas un nouveau rendu (voir aussi PureComponent)

React - Cycle de vie

- componentDidUpdate()
  - Juste après un rendu autre que initial
  - On a accès au DOM
  - Le bon endroit pour un update d’un plugin jQuery (Chosen, Select2…)
- componentWillUnmount()
  - Le composant va être supprimer
  - Permet de supprimer des listeners, libérer la mémoire, appeler clearInterval/Timeout, sinon l'objet associé au composant ne sera jamais détruit (si ref interne dans un callback)

React - Cycle de vie



React - Higher Order Components

- Un Higher Order Component (HOC) est une fonction qui reçoit un composant en entrée et retourne un nouveau composant (une liste filtrée, remplie, etc…)
- Exemple, connect dans react-redux, qui injecte la fonction dispatch à un composant

- Voir Recompose (bibliothèque d'utilitaire HOC)
  https://github.com/acdlite/recompose/

```
export default connect()(TodoApp);
```

React - Higher Order Components

- Ajouter de nouvelles props via un HOC

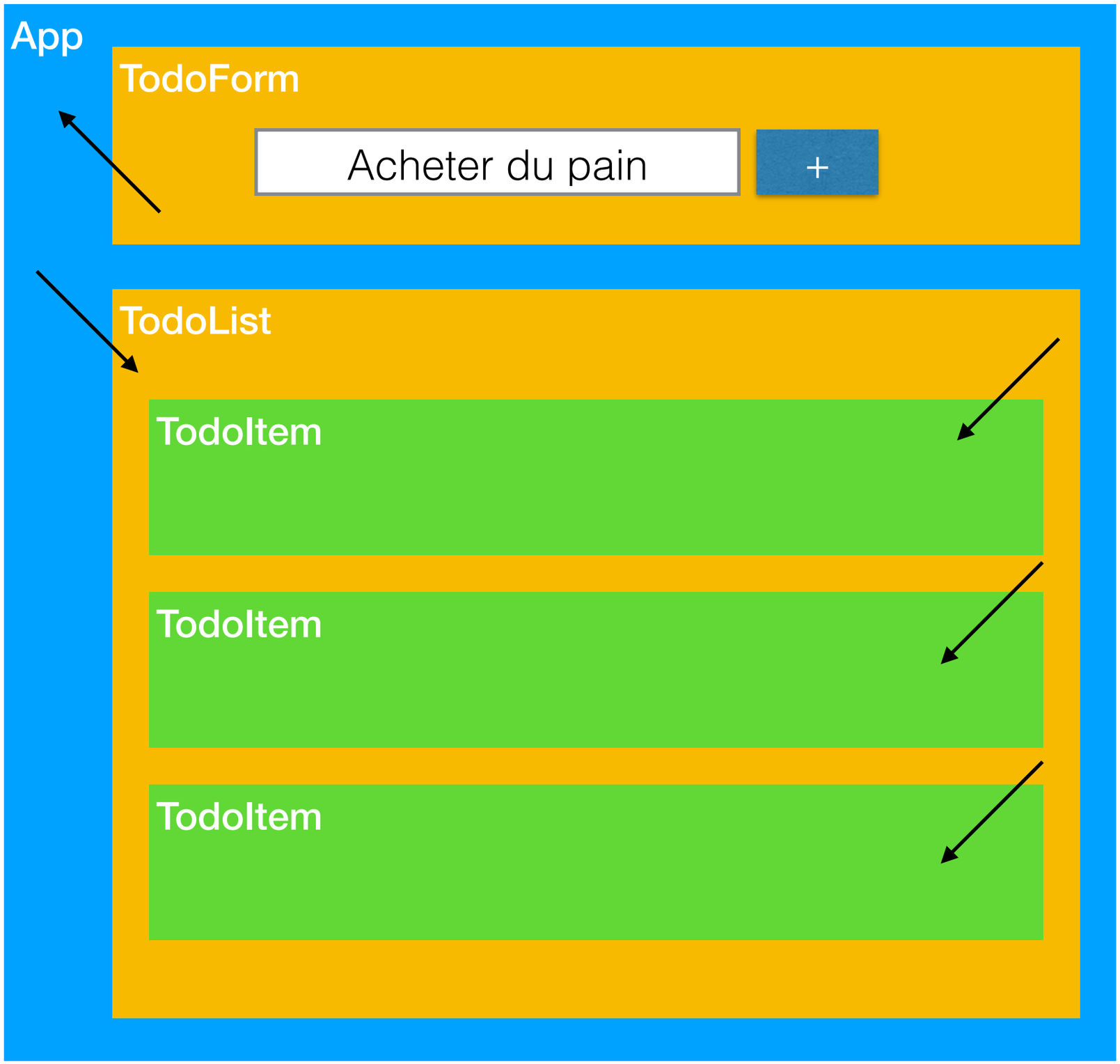
```
render() {
  // Filter out extra props that are specific to this HOC and shouldn't be
  // passed through
  const { extraProp, ...passThroughProps } = this.props;

  // Inject props into the wrapped component. These are usually state values or
  // instance methods.
  const injectedProp = someStateOrInstanceMethod;

  // Pass props to wrapped component
  return (
    <WrappedComponent
      injectedProp={injectedProp}
      {...passThroughProps}
    />
  );
}
```

React - Higher Order Components

- Renommer le composant retourné (bonne pratique)

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

function getDisplayName(WrappedComponent) {
  return WrappedComponent.displayName || WrappedComponent.name || 'Component';
}

export const logLifeCycle = (WrappedComponent) => {

  class LogLifeCycle extends Component {
   // ...
  }

  LogLifeCycle.displayName = `LogLifeCycle(${getDisplayName(WrappedComponent)})`;


  return LogLifeCycle;
};
```

React - Exercice



- Créer un projet todo-react avec create-react-app
- Créer 3 composants :
  - TodoForm
  - TodoList
  - TodoItem
- Dans le state de App créer une liste de todos (string[] ou object[])
- Passer ce tableau à TodoList, qui créera autant de TodoItem qu'il y a d'élément dans le tableau
- TodoItem reçoit un élément et l'affiche
- TodoForm Controlled Component, reçoit un callback de App et lui passe la valeur saisie au submit du form
- Le callback de App ajoute l'élément au tableau

formation.tech

Introduction

Introduction

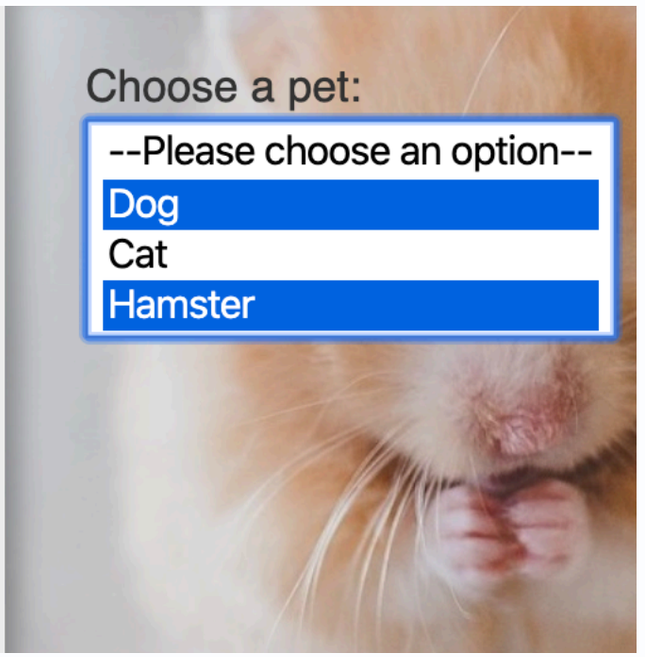
- React est une bibliothèque de création de composants capables de se rafraîchir à chaque changement d'état
- Créée par un employé Facebook en 2011
- Rendue Open-Source en 2013
- Licence MIT depuis novembre 2017
- Qui utilise React ?
  - Les produits de Facebook : Facebook, Instagram, WhatsApp...
  - AirBnb
  - Dropbox
  - Yahoo! Mail
  - formation.tech

Introduction - Qu'est-ce qu'un composant ?

- Un composant est un moyen simple et isolé de regrouper :
  - Du HTML pour la structure de données
  - Du CSS pour la mise en forme
  - Du JavaScript pour le comportent
- Exemple : la balise select

https://developer.mozilla.org/fr/docs/Web/HTML/Element/select

```
<label for="pet-select">Choose a pet:</label>

<select name="pets" id="pet-select" multiple>
    <option value="">--Please choose an option--</option>
    <option value="dog">Dog</option>
    <option value="cat">Cat</option>
    <option value="hamster">Hamster</option>
    <option value="parrot">Parrot</option>
    <option value="spider">Spider</option>
    <option value="goldfish">Goldfish</option>
</select>
```



Introduction - Qu'est-ce qu'un composant ?

- Le composant Select de react-select
  https://react-select.com/

```
import React from 'react';

import Select from 'react-select';
import { colourOptions } from '../data';

export default () => (
  <Select
    defaultValue={[colourOptions[2], colourOptions[3]]}
    isMulti
    name="colors"
    options={colourOptions}
    classNamePrefix="select"
  />
);
```

Purple × Red ×

Introduction - Ecosystème

- A la différence de Google qui maintient tout un écosystème de bibliothèques dans Angular (pour gérer les requêtes HTTP, les formulaires, les pages ou les animations), React ne propose qu'un nombre limité de bibliothèques
- react
  Bibliothèque permettant la création de composants et la communication
- react-dom
  Permet de faire le rendu d'un composant dans le navigateur via l'API DOM
- react-dom/server
  Permet de faire le rendu d'un composant dans Node.js
- react-native
  Permet de faire le rendu d'un composant dans une application native iOS ou Android
- create-react-app
  Programme en CLI qui permet la création d'un squelette d'application React

Introduction - Documentation

- React
  https://reactjs.org/ (existe également en Français : https://fr.reactjs.org/)
- Tutoriel Officiel
  https://reactjs.org/tutorial/tutorial.html
- Blog
  https://reactjs.org/blog/
- Create React App
  https://create-react-app.dev/

Introduction - Installation

- Pour installer React on peut soit ajouter les balises script directement https://reactjs.org/docs/add-react-to-a-website.html
- Soit utiliser le programme create-react-app qui va créer la structure d'une application permettant l'utilisation de technologies comme JSX, TypeScript ou SASS

Create React App

Create React App - Introduction

- Create React App est un programme CLI pour Node.js qui permet la création d'un nouveau squelette d'application React
- Pas besoin de l'installer globalement avec npm ou Yarn, il ne sert que pour créer le projet
- Création du projet
  - `npx create-react-app [chemin-projet] [options]`
  - `npm init react-app [chemin-projet] [options]`
  - `yarn create react-app [chemin-projet] [options]`

Create React App - Introduction

- Options :
  - --use-npm ou --use-pnp sinon l'install se fait via Yarn quand il est installé
  - --template pour utiliser un autre squelette
  - --scripts-version pour utiliser un fork ou une autre version de react-scripts
- Trouver des templates
  https://www.npmjs.com/search?q=cra-template-*
- Exemple
  ```
  npx create-react-app hello-react --use-npm --template
  @formation.tech/cra-template-basic
  ```

Create React App - Squelette cra-template

- Les deux templates officiel sont cra-template et cra-template-typescript
  https://github.com/facebook/create-react-app/tree/master/packages/cra-template
  https://github.com/facebook/create-react-app/tree/master/packages/cra-template-typescript

- Exemple
  npx create-react-app hello-react

```
hello-react
├── README.md
├── node_modules
├── package.json
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
├── src
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   ├── serviceWorker.js
│   └── setupTests.js
└── yarn.lock
```

Create React App - Squelette cra-template

- Le package.json

```
{
  "name": "hello-react",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.3.2",
    "@testing-library/user-event": "^7.1.2",
    "react": "^16.12.0",
    "react-dom": "^16.12.0",
    "react-scripts": "3.3.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```