
# Formation JavaScript React Redux

Romain Bohdanowicz
Twitter : @bioub - Github : https://github.com/bioub
http://formation.tech/


# Introduction

# Présentations

- ## Romain Bohdanowicz
  Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle
- ## Expérience
  Formateur/Développeur Freelance depuis 2006
  Plus de 8000 heures de formation animées
- ## Langages
  Expert : HTML / CSS / JavaScript / PHP / Java
  Notions : C / C++ / Objective-C / C# / Python / Bash / Batch
- ## Certifications
  PHP 5 / PHP 5.3 / PHP 5.5 / Zend Framework 1
- ## Particularités
  Premier site web à 12 ans (HTML/JS/PHP), Triathlète à mes heures perdues
- ## Et vous ?
  Langages ? Expérience ? Utilité de cette formation ?


# React

# React - Introduction

- React est une bibliothèque de création de composants capables d'être « rendus » (render en anglais) à chaque changement d'état
- React n'offre pas d'architecture comme MVC, on organise en général ses composants autour d'un concept nommé Flux

- Créée par un employé Facebook en 2011
- Rendue Open-Source en 2013
- Licence MIT depuis novembre 2017

# 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`

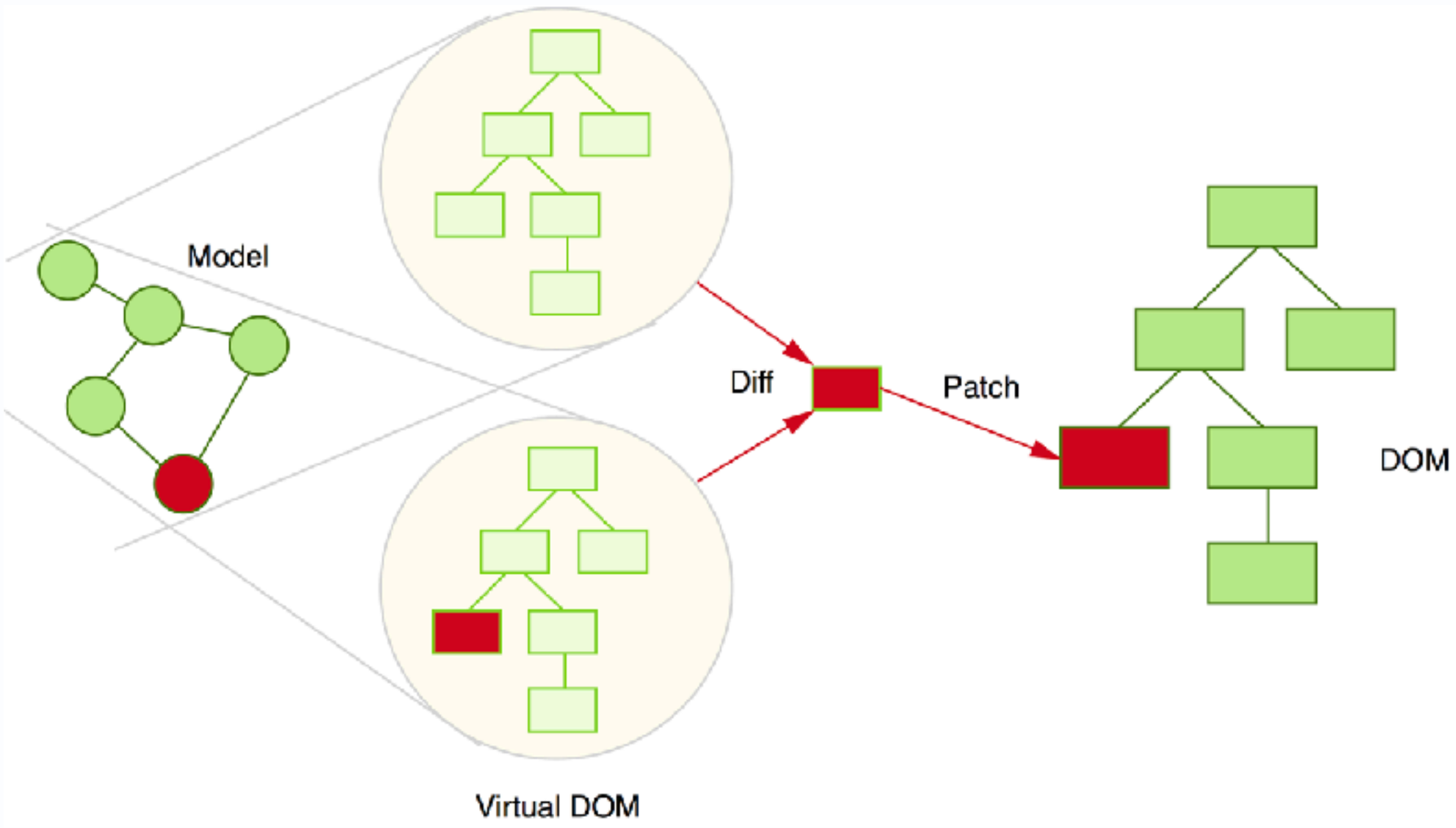
```
My-React-Project
├── README.md
├── node_modules
│   └── ...
├── package.json
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
├── src
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   └── registerServiceWorker.js
└── yarn.lock
```

# React - Premier composant

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

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

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

- 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

# 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.

# 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 )

# 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
  `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

```
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>
      <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>;
  }
}
```

- 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

# 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()
  - render()
  - componentDidMount()
- Destruction
  - componentWillUnmount()
- Mise à jour
  - componentWillReceiveProps()
  - shouldComponentUpdate()
  - componentWillUpdate()
  - render()
  - componentDidUpdate()

# React - Cycle de vie

- constructor() et componentWillMount()
  - Assez similaires, appelés côté client et serveur
  - constructor peut modifier le state avec this.state, componentWillMount avec this.setState()
  - componentWillMount() sert principalement lorsqu'un composant est créé via React.createClass() (déprécié, sera supprimé dans React 16)
  - Servent principalement à initialiser props et state

# React - Cycle de vie

- componentDidMount()
  - Le rendu 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

# React - Cycle de vie

- componentWillReceiveProps(nextProps)
  - Appelée à chaque fois que les props sont modifiées
  - Utile avec React Router pour savoir qu'un param d'URL a changé (et donc déclencher un nouveau fetch par exemple)
- 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

- componentWillUpdate()
  - Appeler juste avant un prochain render
  - Ne pas appeler setState ici pour éviter une boucle infinie
  - N’a pas accès au DOM
- componentDidUpdate()
  - Le composant est rendu
  - 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

# 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

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

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

# React - Higher Order Components

- Ajouter de nouvelles props via un HOC

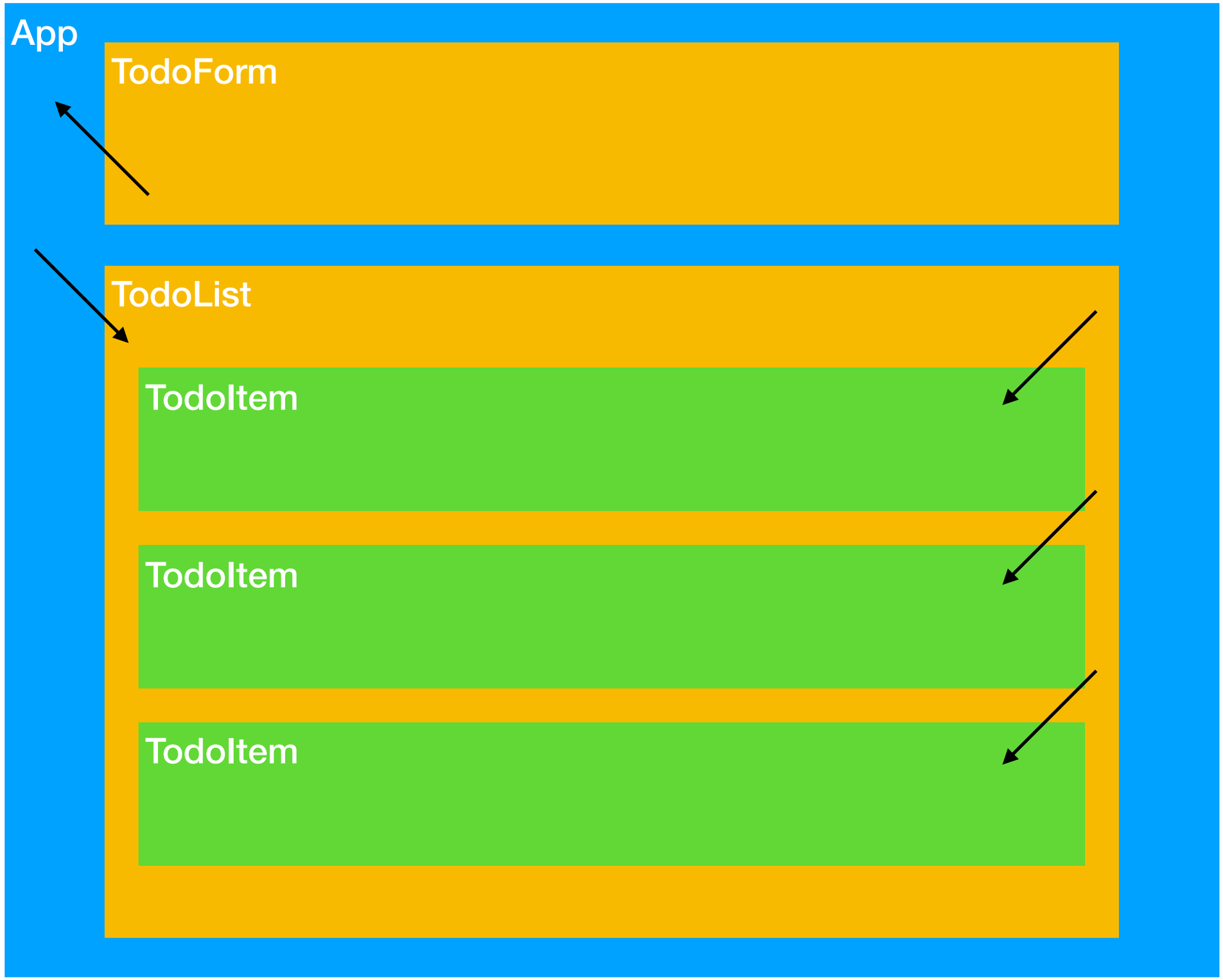
```
render() {
  // Filter out extra props that are specific to this HOC and shouldn't be
  // passed through
  const { extraProp, ...passThroughProps } = this.props;

  // 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;
};
```

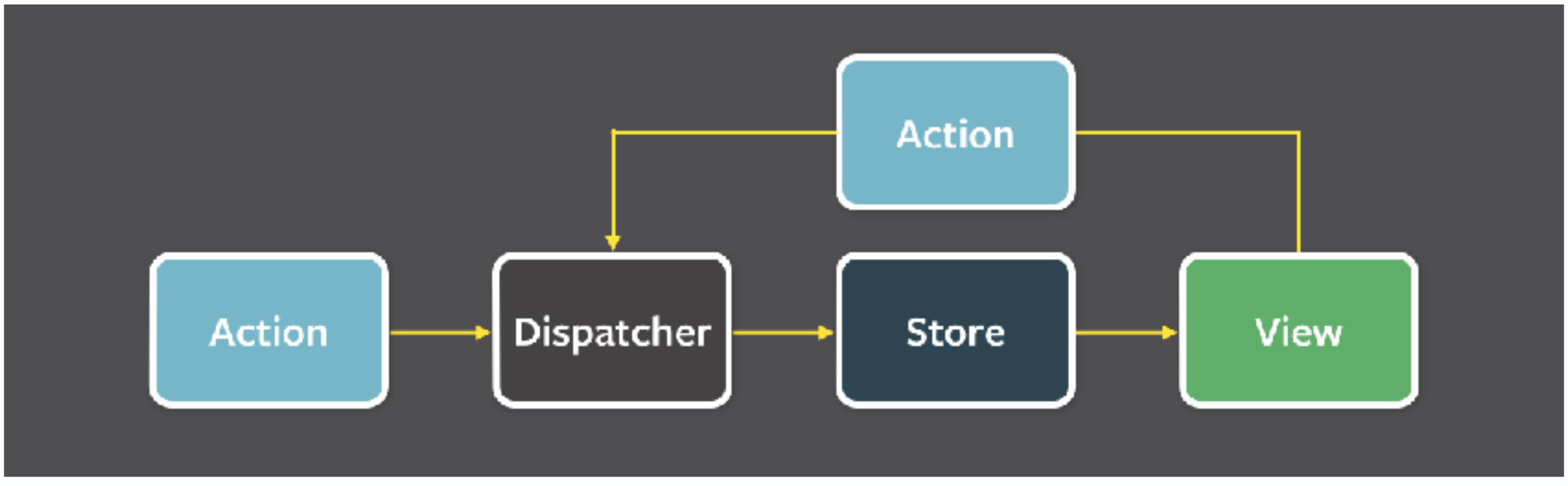
# React - Exercice




# Redux

# Redux - Introduction

- Redux est un conteneur d’état (state container)
  Une bibliothèque dont le rôle et de stocker l’état de l’application et ainsi de la réaffirmer dans l’état précédent lorsque l’historique est manipulé.
- Inspiré par Flux (Facebook)
  Redux est inspiré de Flux, une architecture proposée par Facebook pour les applications front-end. Différente bibliothèque propose de simplifier leur mise en place, dont Redux, même si quelques concepts sont différents.



# Redux - Quand utiliser Redux ?

- Extrait de la doc du Redux : https://redux.js.org/docs/faq/OrganizingState.html#organizing-state-only-redux-state
  - D'autres partie de l'application ont-elles besoin de ces données ?
  - Avez vous besoin de créer de nouvelles données dérivées de ces données ?
  - La même données est-elle utilisée pour piloter différents composants ?
  - Y-a-t'il une valeur que vous souhaiteriez restaurer dans un état précédent (time travel debugging) ?
  - Voulez-vous mettre cette donnée en cache ?
- Si oui à l'une de ces question : Redux

# Redux - Installation

- Redux
  - `npm install redux`
  - `yarn add redux`
- Intégration avec react
  - `npm install react-redux`
  - `yarn add redux`
- Intégration avec l’extension Redux DevTools
  - `npm install redux-devtools-extension`
  - `yarn add redux`

# Redux - Quelques principes

- Immutable State Tree
  - Contrairement à Flux, Redux maintient l'ensemble de l'état de l'application dans un arbre unique.
  - Cet arbre stocké sous la forme d'un plain object JavaScript ou bien tout autre structure (voir Immutable.js).
  - Il doit être immuable, une modification doit entrainement la création d'un nouvel objet en mémoire et non la modification de l'objet existant, ceci pour permettre des fonctionnalités plus avancées comme un undo/redo.
- Pas de modification directe du State Tree
  - Il ne faut pas modifier directement le State Tree, au lieu de ça on va « dispatcher » des actions pour indiquer les modifications à apporter à l'arbre.
  - Une action est un objet JS qui décrit le changement à apporter au State Tree.

# Redux - Quelques principes

- Actions
  - Une action doit avoir à minima une propriété nommée type.
    Exemple pour un compteur :

```
const incrementAction = {
  type: 'INCREMENT',
};

const decrementAction = {
  type: 'DECREMENT',
};
```

  - Chaque action doit décrire le minimum du changement à effectuer dans l'application Redux.

```
const addTodoAction = {
  id: 123,
  value: 'Apprendre à utiliser Redux',
  type: 'ADD_TODO',
};
```

  - Chaque changement intervenant dans l'application, clic utilisateur, nouvelle données reçues du serveur, texte saisi… devrait être décrit par une action la plus simple possible.

# Redux - Quelques principes

- Actions creators
  - Pour faciliter la création d'actions et les pouvoir les réutiliser plus facilement à différents endroit de l'application, on utilise des fonctions appelées actions creators

```
export const setVisibilityFilter = (filter) => ({
  type: 'SET_VISIBILITY_FILTER',
  filter
});
```

# Redux - Quelques principes

- Fonction pure
  - Retourne toujours la même valeur lorsque appelée avec les des  paramètres identiques.
  - Aucun effet parallèle comme l'écriture dans un fichier
  - Ne modifie par ses paramètres d'origines (objets, tableaux…)

```
// fonction pure
const addition = function(a, b) {
  return Number(a) + Number(b);
};

// fonctions impures
const getRandomIntInclusive = function(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
};

const validateUser = function(user) {
  localStorage.setItem('user', user);
  return user === 'Romain';
};

const userToUpperCase = function(user) {
  user.prenom = user.prenom.toUpperCase();
  return user;
};
```

# Redux - Quelques principes

- Reducers
  - Fonction pure
  - Reçoit l'état précédent et l'action dispatchée
  - Retourne l'état suivant
  - Peut conserver les références vers les objets non-concernés par l'action

```
var counterReducer = function(state, action) {
  if (state === undefined) {
    return 0;
  }

  switch (action.type) {
    case 'INCREMENT':
      return state + 1;
    case 'DECREMENT':
      return state - 1;
  }

  return state;
};
```

# Redux - Quelques principes

- Reducers en ES6
  - Privilégier des constantes plutôt que des chaînes de caractères pour les types d’actions
  - Utiliser une valeur par défaut pour l’état initial
  - Ne pas utiliser Symbol() si le state doit être sérialisé.

```
const Counter = {
  INCREMENT: Symbol(),
  DECREMENT: Symbol(),
};

const counterReducer = (state = 0, action) => {
  switch (action.type) {
    case Counter.INCREMENT:
      return state + 1;
    case Counter.DECREMENT:
      return state - 1;
  }

  return state;
};
```

# Redux - Mise en place

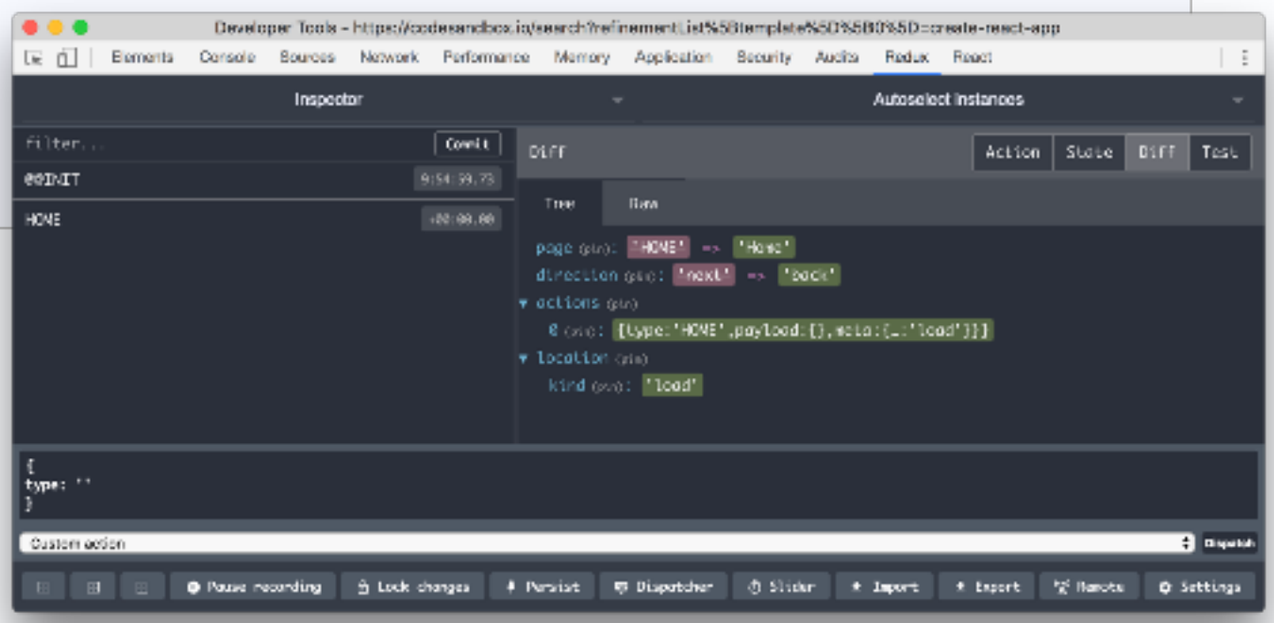
- Store
  - Objet dans lequel est stocké le state (store.getState())
  - Doit être créé à partir d'un reducer ou d'une combinaison de reducers
  - Peut recevoir un state initial, qui pourrait être persisté dans le localStorage par exemple
  - Peut également recevoir des plugins appelées middlewares

```
import { createStore } from 'redux';

const counterReducter = (state, action) => {
  // ...
};

const store = createStore(counterReducter);
```

# Redux - Mise en place

- Redux DevTools
  - Installer *redux-devtools-extension*
  - Passer *composeWithDevTools* en 2e paramètres de *createStore*

```
import { createStore, combineReducers } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import { counter } from './reducers/counter';

export const store = createStore(
  counter,
  composeWithDevTools()
);
```



# Redux - Mise en place

- React Redux
  - Pour que le store soit disponible dans l'application React, on utilise le composant Provider de react-redux (il doit être à la racine)

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './components/App';
import { store } from './store';
import { Provider } from 'react-redux';

ReactDOM.render(
  <Provider store={store}><App /></Provider>,
  document.getElementById('root'),
);
```

# Redux - Mise en place

- La fonction connect de react-redux permet d'associer de rendre disponible la méthode dispatch de redux et d'accéder à certaines propriétés du state

```
import React, { Component } from 'react';
import { connect } from 'react-redux'
import { counterIncrement } from '../actions/counter';

export class ButtonCounter extends Component {
  constructor() {
    super();
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.props.dispatch(counterIncrement());
  }
  render() {
    return <button onClick={this.handleClick}>{this.props.count}</button>;
  }
}

const mapStateToProps = (state) => ({
  count: state.count,
});

export const ButtonCounterContainer = connect(mapStateToProps)(ButtonCounter);
```

# Redux - Mise en place

- On peut également associer directement des méthodes avec *mapDispatchToProps*

```
import React from 'react';
import { connect } from 'react-redux'
import { counterIncrement } from '../actions/counter';

export const ButtonCounter = (props) => {
  return <button onClick={props.handleClick}>{props.count}</button>;
};

const mapStateToProps = (state) => ({
  count: state.count,
});

const mapDispatchToProps = (dispatch) => ({
  handleClick: () => dispatch(counterIncrement()),
});

export const ButtonCounterContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(ButtonCounter);
```