ReactJs 101

Roadmap

- Pourquoi React ?
- Les grands principes
- Let's play with my todo list !
- Aller plus loin

React

> A JavaScript library for building user interfaces

/!\ WARNING /!\

Beaucoup de nouveau vocabulaire et de spécificités:

JSX, props, state, redux, render...

Tout cela a du bon :)

Once upon a time

https://www.youtube.com/watch?v=XxVg_s8xAms

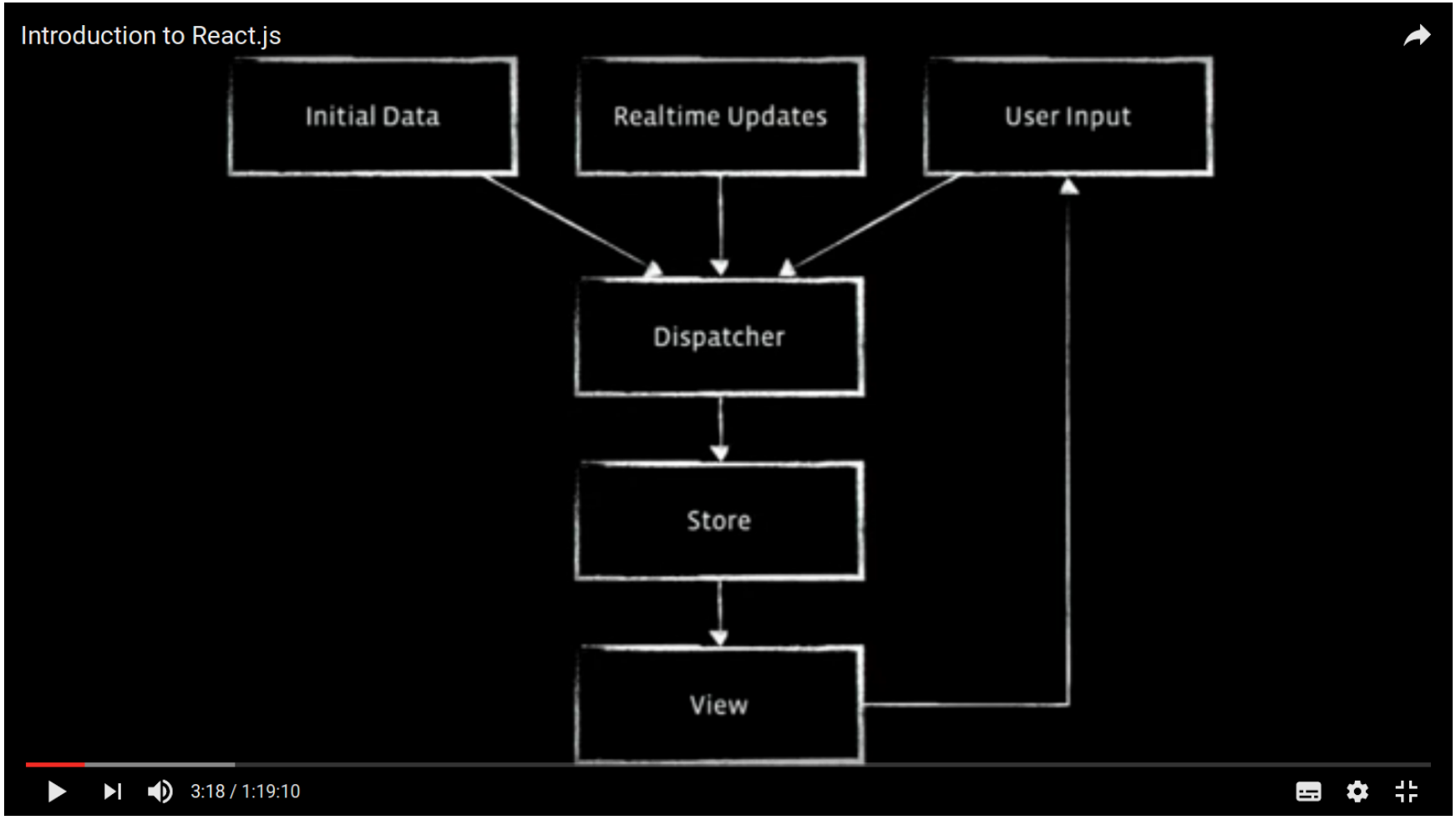
- 2013 : How should we structure JavaScript applications ?

Already, used internally for several years...

Trends

- MVC, MVVM, MVW
- They all have *Models*
- Models : observables, events, changes, mutations
- Mutation

One directional data-binding



View

> "when my data changes, just blow it all away and re-render it from scratch."

Declarative components

- Describe what your component looks like : how they look like.
- Reusable API. Hide the implementation. Like a basic html input.
- No Explicit data-binding. Just say which properties they share.
- render() : function(){...}
    - -> return a representation of a view

Update your view as you data changes.

Re call render and compare, find the differences, and update the minimum.

Interoperability

at Facebook : a lot of legacy stuff

```
<Comment Box/>
```

Instagram

Pas de version web. ALL REACT !

Then they went : Open Source

Les grandes lignes

Everything is a component

```
// MyComponent.js
import React from 'react';

class MyComponent extends React.Component {
  render() {
    return (
      <div>
        Je suis un composant
      </div>
    );
  }
}

export default MyComponent;
```

ES6

JSX

- not html
- JSX => XML-like syntax
- JSX is optional

Jsx is optional (but HIGLHY RECOMMENDED !)

```
class HelloMessage extends React.Component {
  render() {
    return <div>Hello Jane</div>;
  }
}
```

```
class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hello Jane"
    );
  }
}
```

Everything is a component

```
// MyComponent.js
import React from 'react';

class MyComponent extends React.Component {
  render() {
    return (
      <div>
        Je suis un composant
      </div>
    );
  }
}

export default MyComponent;
```

Everything is a component

```
// UnAutreComposant.js
import React from 'react';
import MyComponent from './MyComponent';

class UnAutreComposant extends React.Component {
  render() {
    return (
      <div>
        <h2>Je suis un autre composant.</h2>
        <MyComponent />
      </div>
    );
  }
}

export default UnAutreComposant;
```

Everything is *a* component

```
render() {
  return (
    <h1>Présentation</h1>
    <p>
        Simplon 2 touche à sa fin...
    </p>
  )
}
```

```
Syntax error: Adjacent JSX elements must be wrapped
in an enclosing tag
```

Curly braces

```
render() {
  const maVariable = 19*3/19;
  return (
    <div>
        <h2>Et le résultat est ... </h2>
        <p> { maVariable } </p>
    </div>
  )

}
```

Passing Data Through Props

```
class UnAutreComposant extends React.Component {
  render() {
    const maVariable = 19*3/19;
    return (
      <div>
        <h2>Je suis un autre composant.</h2>
        <MyComponent valeur={ maVariable } />
      </div>
    );
  }
}
```

Passing Data Through Props

```
class MyComponent extends React.Component {
  render() {
    return (
      <p>
        Ma valeur est { this.props.valeur }
      </p>
    );
  }
}
```

Passing Data Through Props

```
class MyComponent extends React.Component {

  handleClick = (event) => {
    const date = Date.now();
    // On envoie la valeur à la fonction donnée en *props
    this.props.onClick(date);
  }

  render() {
    return (
      <button onClick={ this.handleClick } >
        Cliquez moi !
      </button>
    );
  }
}
```

Passing Data Through Props

```
class UnAutreComposant extends React.Component {

  onClick = (dateDeClick) => {
        // ...
  }

  render() {
    return (
      <div>
        <h2>Je suis un autre composant.</h2>
        <MyComponent onClick={ this.onClick } />
      </div>
    );
  }
}
```

Typechecking With PropTypes

Typechecking With PropTypes

```
class Greeting extends React.Component {
  static propTypes = {
    name: React.PropTypes.string
  }

  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}
```

Default props

```
class Greeting extends React.Component {
  static propTypes = {
    name: React.PropTypes.string
  }

  static defaultProps = {
    name: 'Stranger'
  };

  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}
```

Lifting State Up

```
class Clock extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      date: new Date()
    };
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>
          It is { this.state.date.toISOString() }.
        </h2>
      </div>
    );
  }
}
```

```
handleClick = (event) => {
  this.setState({
    date: new Date();
  });
}

render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>
          It is { this.state.date.toISOString() }.
        </h2>
        <button>
          onClick={ this.handleClick }
        </button>
      </div>
    );
  }
```

Lifecycle Methods

Lifecycle Methods

- constructor
- componentDidMount
- shouldComponentUpdate
- componentWillUpdate
- componentWillUnmount
- ...
- render

Lists and Keys

Lists and Keys

```
render() {
  const numbers = [2, 382, 421, 24];
  const listItems = numbers.map((number) => {
   return <li>{ number }</li>
  });

  return (
    <ul>{ listItems }</ul>
  );
}
```

Lists and Keys

```
render() {
  const numbers = [2, 382, 421, 24];
  const listItems = numbers.map((number) => {
   return <li key={ number.toString() }>{ number }</li>
  });

  return (
    <ul>{ listItems }</ul>
  );
}
```

Better with unique id : `key={ item.id }`

Refs and the DOM

```
let textInput = null;

focus() {
  // Explicitly focus the text input using the raw DOM AP
  this.textInput.focus();
}

render() {
  // Use the `ref` callback to store a reference
  // to the text input DOM element in an
  // instance field (for example, this.textInput).
  return (
    <div>
      <input
        type="text"
        ref={(input) => { this.textInput = input; }} />
      <input
        type="button"
        value="Focus the text input"
        onClick={this.focus}
      />
    </div>
  );
}
```

Common errors

className

- `class` is a keyword in js

```
class MyComponent extends React.Component {
// ...
```

- `class` can not be used in JSX as a CSS class

```
render() {

  return (
    <h2 className="ui header">Mon titre</h2>
  );
}
```

others

- All elements wrapped under one

Recap

- ES6 & JSX
- Everything is a component
- Props
- Typechecking
- State & Lifecycle
- List & keys
- Common errors

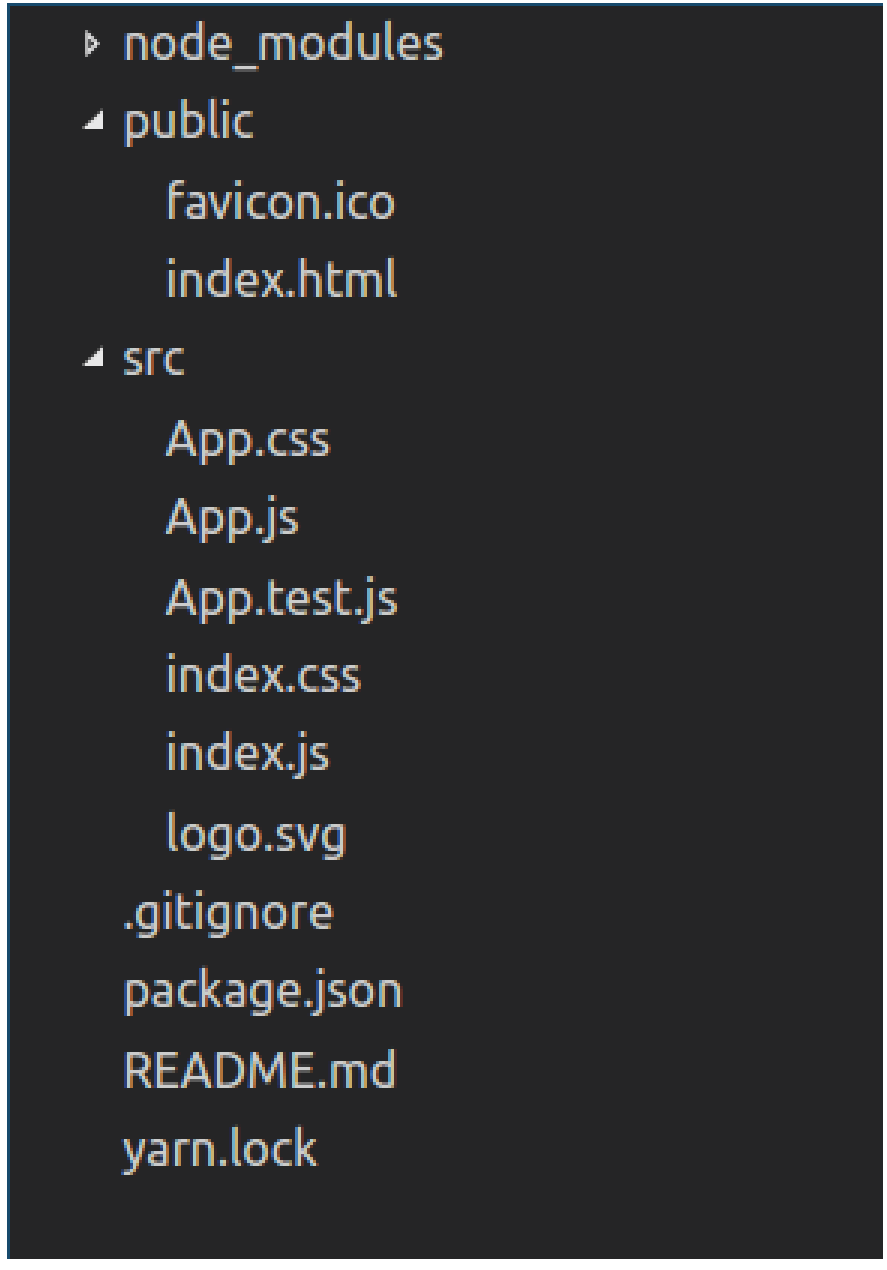
Piece of advice

- La *rigidité*, c'est pour votre bien
- La structure de React force un découpage en composant
- Quand on doit ajouter un nouveau composant, on le sent...

Show Time

```
$ create-react-app mon-app-react
```

```
// package.json
{
  "name": "mon-app-react",
  "version": "0.1.0",
  "private": true,
  "devDependencies": {
    "react-scripts": "0.9.4"
  },
  "dependencies": {
    "react": "^15.4.2",
    "react-dom": "^15.4.2"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
```


node_modules
public
favicon.ico
index.html
src
App.css
App.js
App.test.js
index.css
index.js
logo.svg
.gitignore
package.json
README.md
yarn.lock

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo"
               alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code>
          and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

```
<!-- public/index.html -->

<!-- ... -->
  <body>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser,
      you will see an empty page.

      You can add webfonts, meta tags, or analytics
      to this file.
      The build step will place the bundled scripts
      into the <body> tag.

      To begin the development, run `npm start`.
      To create a production bundle, use `npm run build`.
    -->
  </body>
</html>
```

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './index.css';

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

```
$ npm start
```

To Do List

- Voir la liste des choses à faire
- Ajouter un élément dans ma Todo List
- Marquer un élément comme "Fait"
- Filtrer l'affichage en fonction des états : Tous, A faire, faits

Think in React

```
<App>

  <ItemForm/>

  <Items>
    <Item />
    <Item />
  </Items>

  <Filters>
        <Filter/>
        <Filter/>
        <Filter/>
  </Filters>

</App>
```

=> Créer le composant Items qui va afficher une liste de choses à faire

Prérequis :

- constructor()
- state
- list & keys
- render()

```
// Items.js
// ...
  constructor(){
    super();
    this.state = {
      items: [
        {
          id:1,
          content: "Finir Simplon 2"
        },
        {
          id:2,
          content : "Mettre mon CV en ligne"
        }
      ]
    }
  }
```

```
// Items.js
// ...
  render() {
    const itemNodes = this.state.items.map( item => {
      return <li key={ item.id }>{ item.content }</li>;
    });

    return (
      <div>
        <ul>
          { itemNodes }
        </ul>

      </div>
    );
  }
```

=> Ajouter un input pour ajouter un item

Prérequis :

- Refs and the DOM

```
<form onSubmit={ this.handleSubmit }>
  <label htmlFor="item">Ajouter un item :</label>
  <div>
    <input
      ref={ input => this.itemInput = input }
      type="text"
      name="item"
      placeholder="Acheter du pain"/>
    <button>Ajouter</button>
  </div>
</form>
```

```
handleSubmit = (event) => {
    event.preventDefault();

    const newItemContent = this.itemInput.value;
    const newItem = {
      id: Date.now(),
      content: newItemContent
    }

    let myNewItems = Object.assign([], this.state.items);
    myNewItems.push(newItem);

    this.setState({
      items: myNewItems
    });

    this.itemInput.value = "";
 }
```

=> Ajouter un état aux items

=> Créer un nouveau composant ItemForm

Prérequis :

- props

```
// Items.js
return (
      <div>
        <ItemForm onSubmit={ this.onSubmit } />
        <ul>
          { itemNodes }
        </ul>

      </div>
    );
```

```
//Items.js
  onSubmit = (newItem) => {
    let myNewItems = Object.assign([], this.state.items);
    myNewItems.push(newItem);

    this.setState({
      items: myNewItems
    });
  }
```

```
// ItemForm.js
render() {
    return (
      <form onSubmit={ this.handleSubmit }>
        <label htmlFor="item">Ajouter un item :</label>
        <div>
          <input
            ref={ input => this.itemInput = input }
            type="text"
            name="item"
            placeholder="Acheter du pain"/>
          <button>Ajouter</button>
        </div>
      </form>
    );
  }
```

```
// ItemForm.js
  handleSubmit = (event) => {
    event.preventDefault();

    const newItemContent = this.itemInput.value;
    const newItem = {
      id: Date.now(),
      content: newItemContent
    }

    this.props.onSubmit(newItem);

    this.itemInput.value = "";
  }
```

```
static propTypes = {
    onSubmit : React.PropTypes.func.isRequired
  }
```

=> Ajouter une checkbox pour l'état

==> Ajouter une classe en fonction de cet état pour les différencier visuellement

```
// Items.js
this.state = {
      items: [
        {
          id:1,
          checked: false,
          content: "Finir Simplon 2"
        },
        {
          id:2,
          checked: true,
          content : "Mettre mon CV en ligne"
        }
      ]
}
```

```
// Items.js
const itemNodes = this.state.items.map( item => {
      return (
        <li key={ item.id }
          className={ (
            item.checked ?"item-done":"item-to-do"
          )}
        >
          <input type="checkbox"
            defaultChecked={ item.checked}
            onChange={ this.handleChangeItem }
            data-itemId={ item.id }
          />
          { item.content }
        </li>
      );
    });
```

```
// Items.js
handleChangeItem = (event) => {
  const isChecked = event.target.checked;
  let itemId = event.target.getAttribute("data-itemId");
  itemId = parseInt(itemId, 10);

  const updatedItems = this.state.items.map( item => {
    if(item.id === itemId){
      item.checked = isChecked;
    }
    return item;
  });

  this.setState({
    items: updatedItems
  })
}
```

=> Refactoriser avec Item.js

```
// Items.js
const itemNodes = this.state.items.map( item => {
      return (
        <Item key={ item.id }
          item={ item }
          onChangeItem={ this.onChangeItem }
        />
      );
    });
```

```
// Items.js
  onChangeItem = (itemId, isChecked) => {
    const updatedItems = this.state.items.map( item => {
      if(item.id === itemId){
         item.checked = isChecked;
      }
      return item;
    });

    this.setState({
      items: updatedItems
    })
  }
```

```
// Item.js
render() {
    const item = this.props.item;

    return (
      <li key={ item.id }
        className={ (
          item.checked ?"item-done":"item-to-do"
        )}
      >
        <input type="checkbox"
          defaultChecked={ item.checked}
          onChange={ this.handleChangeItem }
          data-itemId={ item.id }
        />
        { item.content }
      </li>
    );

  }
```

```
// Item.js
handleChangeItem = (event) => {
  const isChecked = event.target.checked;
  let itemId = event.target.getAttribute("data-itemId");
  itemId = parseInt(itemId, 10);

  this.props.onChangeItem(itemId, isChecked);
}
```

```
// Item.js
  static propTypes = {
    item: React.PropTypes.object.isRequired,
    onChangeItem : React.PropTypes.func.isRequired
  }
```

=> Ajouter des filtres

```
<form>
  <input type="radio"
    id="filter-all"
    data-filter="all"
    name="filter"
    onChange={ this.handleFilterChange }
    defaultChecked="true"
  />
  <label htmlFor="filter-all">Tous</label>

  <input type="radio"
    id="filter-to-do"
    data-filter="to-do"
    name="filter"
    onChange={ this.handleFilterChange }
  />
  <label htmlFor="filter-to-do">A faire</label>

  <input type="radio"
    id="filter-done"
    data-filter="done"
    name="filter"
    onChange={ this.handleFilterChange }
  />
  <label htmlFor="filter-done">dFaits</label>
</form>
```

```
handleFilterChange = (event) => {
    const filter = event.target
        .getAttribute('data-filter');

    this.setState({
      filter
    });
  }
```

```
// Items.js render()
const itemNodes = this.state.items
    .filter( item => {
      let keepItem = false;

      if(this.state.filter === "to-do"){
        keepItem = !item.checked;
      }else if(this.state.filter === "done"){
        keepItem = item.checked;
      }else{
        keepItem = true;
      }

      return keepItem;
    })
    .map(
```

Suite

- Refactoriser les filtres dans un composant
- Nettoyer son workspace
- Mettre des tests unitaires avec Jest
- Mettre firebase
- Mettre en place Redux

Any questions ?