REACT Précis et concis

Table des matières

VS Code extensions

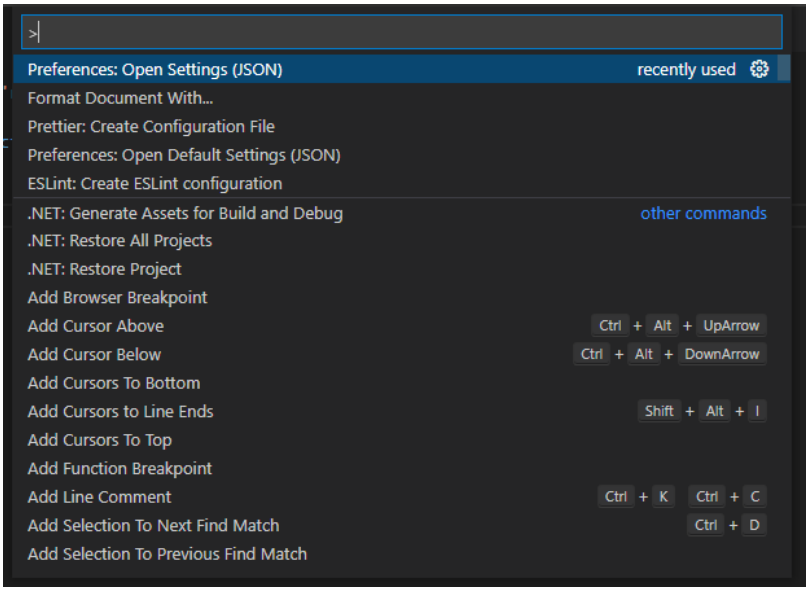
ES7 React/Redux/GraphQL,React-Native snippets

Emmet (inclus avec VS Code)

Pouvoir utiliser emmet dans le JSX

Menu « file » … « preferences » … « settings » … onglet « extensions » … « Emmet »

Ou plus rapide CTRL+MAJ+P puis « Preferences : Open Settings (JSON) »



Ajouter

```
"emmet.includeLanguages": {
  "javascript": "javascriptreact"
},
```

Utiles

Editeurs en ligne

Codesandbox permet de faire (et shared) des projets rapidement en ligne

https://codesandbox.io/

Créer rapidement une page sans installer les dépendances

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <script src="https://unpkg.com/@babel/standalone/babel.js"></script>
        <script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>
        <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
    </head>
    <body>
        <div id="root"></div>

        <script type="text/babel">
            const App = () => {
                return (
                    <div>
                        <h1>Sample</h1>
                    </div>
                );
            };

            ReactDOM.render(<App />, document.getElementById("root"));
        </script>
    </body>
</html>
```

Ajout de babel

On peut également mettre « text/jsx » mais pas de colorisation syntaxique par défaut avec VS Code

On peut aussi créer un fichier js à part (indiqué avec src)

Création de projet

Avec « create-react-app »

Github

Installation en global de create-react-app

```
npm i -g create-react-app
```

Création de projet

```
npx create-react-app my-app
```

Il est possible créer une application React avec **Typescript** (adding typescript)

```
npx create-react-app my-app --template typescript
```

À partir de zero

Création du dossier de l'application

Création de « package.json », depuis une invite de commande (ou terminal de VS Code)

```
npm init -f
```

Aiout des dépendances

```
npm i react react-dom react-scripts
```

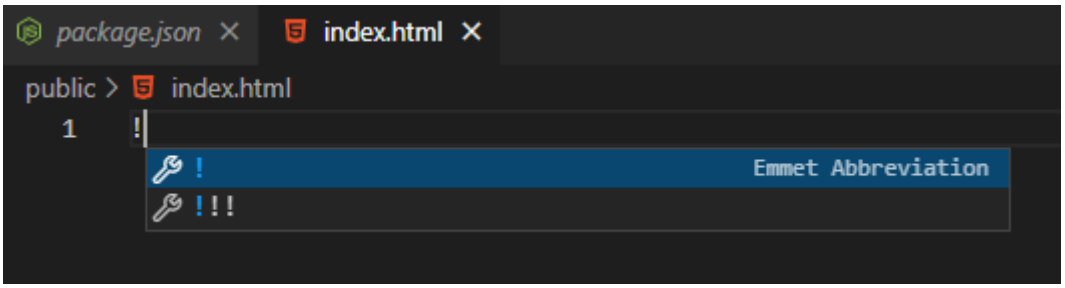
Création d'un dossier public avec une page « index.html » contenant une div avec id root

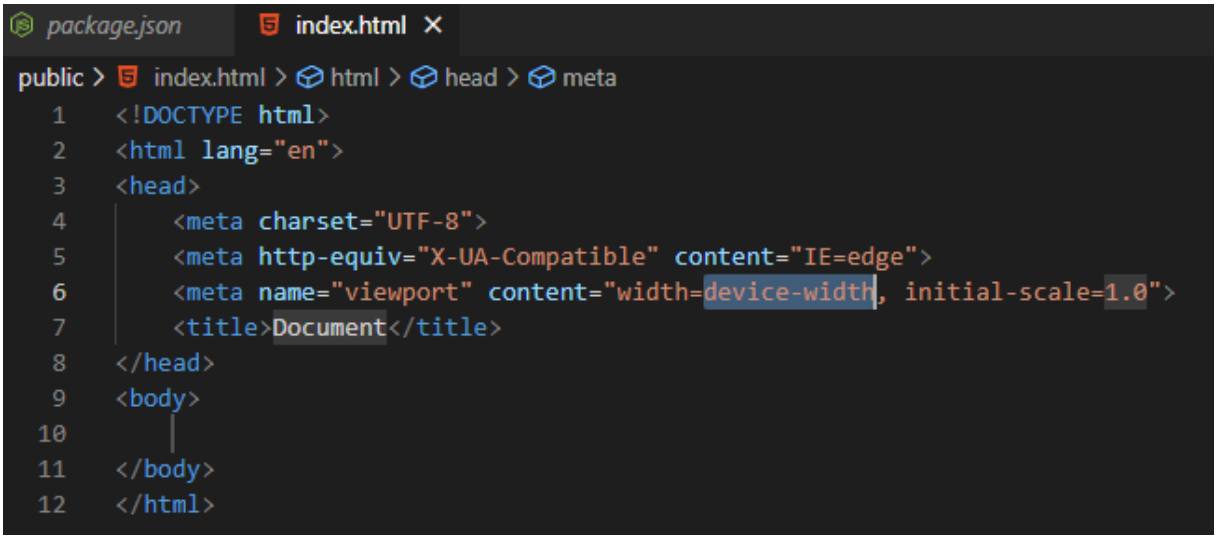
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sample</title>
</head>
<body>
    <div id="root"></div>
</body>
</html>
```

Astuces

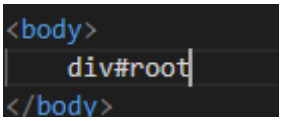
« snippet »

Page html





« emmet »



Création du dossier « src » avec « index.js » et « App.js »

```
import ReactDOM from "react-dom";
import App from "./App";

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

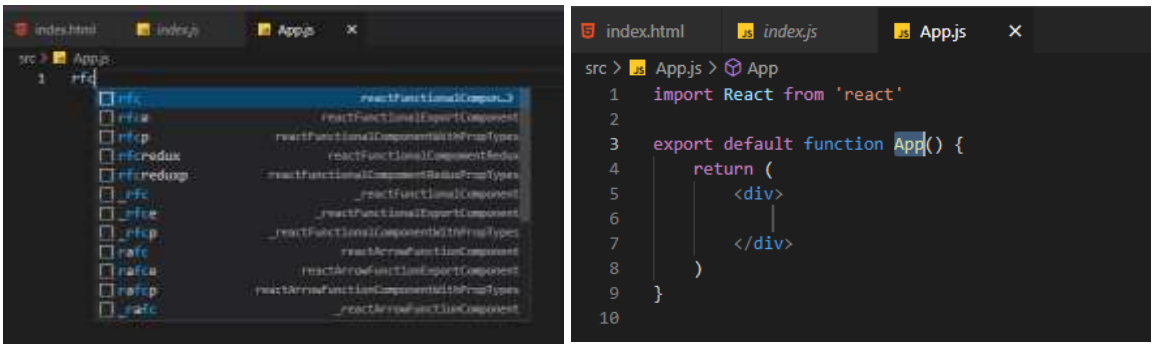
App.js

```
import React from 'react'

export default function App() {
  return (
    <div>

    </div>
  )
}
```

Astuces :

« snippet »

« rfc » pour créer un component react



Ajout de scripts à « package.json »

```
  "scripts": {
    "start": "react-scripts start"
  },
```

Note : au premier lancement il sera proposé d'ajouter la section « target browsers »

```
? We're unable to detect target browsers.

Would you like to add the defaults to your package.json? » (Y/n)
```

Exemple de « package.json » de départ

```
{
  "name": "react-sample",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "react-scripts start"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-scripts": "^4.0.3"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Starter/boilerplate

2 types de projets:

- **Site web** :

    - « src » et « build »
    - Pouvoir lancer le site en mode dev et build le site en mode production
    - Webpack ou react-scripts peuvent être suffisants.
    - React Router peut être ajouté.
    - Tests avec Jest
    - Docker ?
- **Library** :
    - « src » (packages ?)
    - « build » (ou dist) avec library au format umd (rollup pour un code propre ?), minifié
    - « example » peut être utile (es6, browser) en mode dev
    - et/ou « docs » pour le site de documentation de la library.
    - L'utilisation de typescript peut être recommandée + types

Recommandations de React

https://reactjs.org/docs/create-a-new-react-app.html#you-might-not-need-a-toolchain

Recommended Toolchains

The React team primarily recommends these solutions:

- If you're **learning React** or **creating a new single-page app,** use Create React App.
- If you're building a **server-rendered website with Node.js,** try Next.js.
- If you're building a **static content-oriented website,** try Gatsby.
- If you're building a **component library** or **integrating with an existing codebase**, try More Flexible Toolchains.

ES6 utile

Immutable avec le spread operator

number, string, boolean, null, undefined: sont **deja immutables**

objects, arrays, functions sont **mutables**

Array

```
const array = [1, 2, 3];
const newArray = [...array, 4];
console.log(...newArray); // 1 2 3 4
```

Note: éviter push,pop, reverse, ect.

Utiliser: map, reduce, filter, concat, spread qui retournent un nouvel array

Object

```
const obj = { name: "test", email: "test@mail.com" };
const newObj = { ...obj, age: 25 }; // { name: 'test', email: 'test@mail.com',
 age: 25 }
console.log(newObj);
```

Avec **Object.assign**

1er parameter: nouvel objet vide, 2nd parmètre, l'objet à coper, 3ème parameter: valeurs à ajouter.

```
const obj = { name: "test", email: "test@mail.com" };
const newObj = Object.assign({}, obj, { age: 25 });
```

Nested objects

```
const user = {
  name: "test",
  email: "test@email.com",
  address: {
    city: "test-city",
  },
};
const newUser = { ...user, address: { ...user.address } };
```

Librairies pour gérer les immutables

- Immer
- Seamless-immutable
- React-addons-update
- Immutable.js

Libraries de cloning (même si à éviter)

- Clone-deep
- Lodash-merge

Rest operator (avec function)

```
function fn(a, b, ...rest) {
  console.log(a, b, rest); // 1 2 [ 'a', 'b', 'c' ]
}

fn(1,2, "a","b","c");
```

Destructuring

Array

```
const array = [1, 2, 3];
const [v1, v2] = array;
console.log(v1, v2); // 1 2
```

Object

```
const obj = { name: "test", email: "test@mail.com", age: 25 };
const { email, age } = obj;
console.log(email, age); // test@mail.com 25
```

Gestion des états de l'application

- props (passage de valeurs à un component child)
- state (gestion de state dans le scope du component)
- ref (accès à l'élement du DOM)
- useReducer (gestion de state plus large)
- context API (shared context pour tous les child components)

Plus

- Local state: XState
- Global state (context API) : redux, MobX, Recoil
- « Server » state : fetch, Axios mais aussi react-query, swr, relay, apollo

JSX

Permet de faciliter l'écriture du code. Celui-ci est transpilé grace à un plugin (« @babel/plugin-transform-react-jsx » inclus dans « @babel/preset-react »)

On peut avoir un aperçu sur le site de Babel

https://babeljs.io/repl

Babel

https://babeljs.io/docs/en/

En général il faudra installer @babel/cli (pour les lignes de commandes) et @babel/core (pour les méthodes comme « transform ») pour une application avec node.js (voir setup)

Puis selon les besoins installer un preset et des plugins. Un preset installe automatiquement les différents plugins.

Pour utiliser les dernières features de Javascript (exemple : spread operator)	@babel/preset-env
Support de react et JSX	@babel/preset-react
Support de typescript, inclus le plugin @babel/plugin-transform-typescript permettant de transformer du typescript en javascript	@babel/preset-typescript

Note : on peut cumuler les presets

On crée un fichier de configuration « .babelrc » ou « babel.config.json » dans lequel on ajoute les presets et les plugins https://babeljs.io/docs/en/configuration . Exemple

```
{
  "presets": ["@babel/preset-env"],
  "plugins": ["@babel/plugin-transform-react-jsx"]
}
```

Pour transpiler le code soit en ligne de commande soit avec un script dans package.json (-d indique le dossier de destination)

```
"build": "babel src -d dist"
```

On pourrait aussi le faire avec node.js

```
const fs = require("fs");
const { transform } = require("@babel/core");

const code = fs.readFileSync("./src/index.js");
const transpiled = transform(code).code;

fs.writeFile("./dist/build.js", transpiled, (err) => {
  if (err) console.log("err", err);
  else console.log("ok");
});
```

Puis « node build/babel.js » par ex pour un fichier nommé babel.js qu'on mettrait dans un dossier « build »

On peut aussi utiliser un bundler comme webpack ou rollup.

Hooks

Désormais React est base sur les hooks. Plus besoin de manipuler « this ».

Attention les class based components ne peuvent pas utiliser les hooks

useState

Permet de faire du « two-way databinding»

Le component est re-render à chaque changement de valeur du state

```
import { useState } from "react";

export default function App() {
  const [message, setMessage] = useState("");

  const inputHandler = (ev) => {
    setMessage(ev.target.value);
  };

  return (
    <div>
      <input
        id="message"
        name="message"
        value={message}
        onChange={inputHandler}
      />
      <p>{message}</p>
    </div>
  );
}
```

1ère valeur : le state

2nde valeur : méthode permettant de modifier le state, ce qui provoque le re-render du component

On passe l'initial value à useState

message 1

message 1

useRef (uncontrolled input)

Permet d'accéder directement à l'élément dans le DOM

Quand utiliser ref ?

- Quand on a besoin d'accéder au DOM
- Quand l'état ne change pas / n'est pas affiché

Exemple : on donne le focus à l'input au click sur le bouton

```
import { useRef } from "react";

const InputWithFocusButton = () => {
  const inputRef = useRef();

  const onButtonClick = () => {
    inputRef.current.focus();
  };

  return (
    <>
      <input ref={inputRef}  />
      <button onClick={onButtonClick}>Donner le focus au champ</button>
    </>
  );
};
```

Accès à l'élément avec « current »

Note : on peut créer un **tableau de refs**

```
const inputs = useRef([])
```

On accède aux values par l'index

```
inputs.current[0].value
```

Création d'une méthode utilisée par les inputs pour l'ajout des refs

```
  const addInputs = (el) => {
    if (el && !inputs.current.includes(el)) {
      inputs.current.push(el);
    }
  };
```

Exemple

```
 <input type="email" ref={addInputs} htmlFor="mail" />
```

On peut utiliser useRef pour persister des données

Une ref ne déclenche pas le re-render mais peut être utilisé pour persister des informations entre plusieurs re-render (sans lié la ref à un élément du DOM). Dans l'exemple on crée un hook pour forcer le re-render (forceUpdate).

```
import React, { useState, useRef } from "react";
import { unstable_batchedUpdates } from "react-dom";

const useForceUpdate = () => {
  const [count, setCount] = useState(0);
  const increment = () => setCount(prevCount => prevCount + 1)
  return [increment, count];
}

export default function Sample1() {
  const myRef = useRef({ count : 1 });
  const [forceUpdate] = useForceUpdate();
```

```
  console.log("RENDER", myRef.current.count);

  const handleClick = () => {
    myRef.current.count += 1;
    forceUpdate();
  };

  return (
    <div>
      <button onClick={handleClick}>Click</button>
    </div>
  );
}
```

formwardRef + useImperativeHandle

Permet d'accéder à un component child. Exemple focus depuis le parent

```
import React, { useRef, useImperativeHandle } from "react";

const Input = React.forwardRef((props, ref) => {
  const inputRef = useRef();

  const activate = () => {
    inputRef.current.focus();
  };

  useImperativeHandle(ref, () => {
    return {
      focus: activate
    };
  });

  return (<input ref={inputRef} />);
});

export default function App() {
  const inputRef = useRef();
  const handleClick = () => {
    inputRef.current.focus();
  };
  return (
    <div>
      <Input ref={inputRef} />
      <button onClick={handleClick}>Focus</button>
    </div>
  );
}
```

Depuis le parent au click on focus l'input du component child

useEffect

Appelé à chaque re-render

```
import { useState, useEffect } from "react";

useEffect(() => {
```

```
  console.log("UseEffect called");
});
```

Appelé une seule fois au chargement

```
  useEffect(() => {
    console.log("UseEffect called");
  },[]);
```

Appelé quand un state change

```
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log("UseEffect called after count changed");
  },[count]);
```

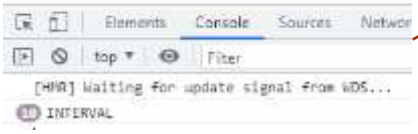
On peut avoir plusieurs useEffect

```
  useEffect(() => {
    console.log("UseEffect called");
  },[]);

  useEffect(() => {
    console.log("UseEffect called after count changed");
  },[count]);
```

Attention au cache

« useEffect » utilise la valeur initiale des states (cache). Exemple

Count: 1



setInterval s'exécute mais le count ne change pas

```
const Sample = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setInterval(() => {
      console.log("INTERVAL");
      //setCount(count + 1); // ne marche pas
      setCount((count) => count + 1);
    }, 1000);
  }, []);
```

Utiliser une function

```
  return <p>Count: {count}</p>;
};
```

Clear

```
  useEffect(() => {
    const intervalId = setInterval(() => {
      setCount((count) => count + 1);
    }, 1000);

    return () => {
      clearInterval(intervalId);
    };
  }, []);
```

On peut aussi l'utiliser pour se désabonner d'un event (« resize » par exemple)

```
  useEffect(() => {
    window.addEventListener("resize", onResize);

    function onResize() {
      console.log("Window resized");
    }

    return () => {
      window.removeEventListener("resize", onResize);
    };
  }, []);
```

Equivalences

Avant	Avec hooks
componentDidMount() { … }	useEffect(()=>{ … })
componentDidUpdate() { … }	useEffect(()=>{ … })
componentWillUnmount() { … }	useEffect(()=>{ … return ()=>{…}})

memo

Eviter qu'un child component soit re-render si les props passées n'ont pas changé.

```
// PROBLEME: mis à jour à chaque fois quand on modifie un état dans le component parent même si les pro
ps passées ne changent pas
// const Content = (props) => {
//   console.log("Content updated");
//   return <div>{props.count}</div>;
// };

// SOLUTION: avec react memo le component child n'est pas re render si les props ne changent pas
const Content = React.memo((props) => {
  console.log("Content updated");
  return <div>{props.count}</div>;
});

export default function App() {
  const [count, setCount] = useState(0);
  const onClick = () => {
    console.log("click");
```

```
    setCount(count + 1);
  };

  return (
    <div>
      <Content count={5} />
      <button onClick={onClick}>Click!</button>
    </div>
  );
}
```

useCallback

Eviter le re-render d'un child component quand on passe une function qui ne change pas en props

```
const Content = React.memo((props) => {
  console.log("Content updated");
  return;
  <div>{props.count}</div>;
});

export default function App() {
  const [count, setCount] = useState(0);
  const onClick = () => {
    console.log("click");
    setCount(count + 1);
  };

  // PROBLEME 2: le component child est mis à jour en passant une function en props même si elle ne cha
nge pas
  //   const log = () => {
  //     console.log("Une function qui ne change pas");
  //   };

  // SOLUTION 2
  const log = useCallback(() => {
    console.log("Une function qui ne change pas");
  }, []);

  return (
    <div>
      <Content count={5} log={log} />
      <button onClick={onClick}>Click!</button>
    </div>
  );
}
```

useReducer (de react)

Reducer :

(previousState, action) => newState

```
import React, { useReducer } from "react";

const initialState = { name: "", email: "" };

const INPUT_CHANGE = "INPUT_CHANGE";

// reducers
const formReducer = (state, action) => {
  switch (action.type) {
    case INPUT_CHANGE:
      return { ...state, [action.inputId]: action.value };
    default:
```

```
      return state;
  }
};

export default function App() {
  const [formState, dispatch] = useReducer(formReducer, initialState);

  const inputHandler = (ev) => {
    dispatch({
      type: INPUT_CHANGE,
      inputId: ev.target.name,
      value: ev.target.value,
    });
  };

  const submitHandler = (ev) => {
    ev.preventDefault();
    console.log(formState);
  };

  return (
    <div>
      <form onSubmit={submitHandler}>
        <div className="form-control">
          <label htmlFor="name">Name</label>
          <input
            id="name"
            name="name"
            value={formState.name}
            onChange={inputHandler}
          />
        </div>
        <div className="form-control">
          <label htmlFor="email">E-mail</label>
          <input
            id="email"
            name="email"
            type="email"
            value={formState.email}
            onChange={inputHandler}
          />
        </div>
        <input type="submit" />
      </form>
    </div>
  );
}
```

Custom hooks

Pas mal de hooks utiles sur https://usehooks-typescript.com/

useFetch

Il faut parfois pouvoir interrompre soit la requête, soit ne pas modifier les états, pour éviter qu'un component unmounted voit ses states changés (à un changement de page alors qu'une requête http est en cours par exemple, ce qu'on peut simuler facilement avec un « setTimeout » au moment de modifier « data » et « isLoading »)

« use-fetch.js » dans un dossier « src/hooks » par exemple

version avec unmount

```
const useFetch = url => {
  const isMounted = useRef(false);
  const [data, setData] = useState();
  const [error, setError] = useState();
  const [isLoading, setLoading] = useState(true);

  useEffect(() => {
    isMounted.current = true;
```

```
    setLoading(true);
    fetch(url)
      .then(response => {
        if (!response.ok) throw new Error(response.statusText);
        return response.json();
      })
      .then(result => {
        setTimeout(() => {
          if (isMounted.current) {
            setData(result);
            setLoading(false);
          }
        }, 4000);
      })
      .catch(err => {
        if (isMounted.current) {
          setError(err);
          setLoading(false);
        }
      });

    return () => {
      isMounted.current = false;
    };
  }, [url]);
  return { data, error, isLoading };
};
```

Version avec AbortController

```
const useFetch = (url, options = {}) => {
  const [data, setData] = useState();
  const [error, setError] = useState();
  const [isLoading, setLoading] = useState(true);

  useEffect(() => {
    const abortController = new AbortController();
    setLoading(true);
    fetch(url, {
      ...options,
      signal: abortController.signal
    })
      .then(response => {
        if (!response.ok) throw new Error(response.statusText);
        return response.json();
      })
      .then(data => {
        setData(data);
        setLoading(false);
      })
      .catch(error => {
        setError(error);
        setLoading(false);
      });

    return () => {
      abortController.abort();
    };
  }, [url]);
  return { data, error, isLoading };
};
```

… en version **async**

```
const useFetch = (url, options = {}) => {
  const [isLoading, setIsLoading] = useState(true);
  const [data, setData] = useState();
  const [error, setError] = useState();

  useEffect(async () => {
    try {
```

```
      setIsLoading(true);

      const abortController = new AbortController();
      const response = await fetch(url, {
        ...options,
        signal: abortController.signal
      });
      if (!response.ok) throw new Error(response.statusText);

      const data = await response.json();

      setData(data);
      setIsLoading(false);
    } catch (error) {
      setError(error);
      setIsLoading(false);
    }

    return () => {
      abortController.abort();
    };
  }, [url]);

  return { data, error, isLoading };
};
```

Ou une version avancée permettant de gérer plusieurs requêtes

```
const useFetch = (url, options = {}) => {
  const [data, setData] = useState();
  const [error, setError] = useState();
  const [isLoading, setLoading] = useState(true);
  const activeHttpRequests = useRef([]);

  useEffect(() => {
    const abortController = new AbortController();
    activeHttpRequests.current.push(abortController);
    setLoading(true);
    fetch(url, {
      ...options,
      signal: abortController.signal
    })
      .then(response => {
        if (!response.ok) throw new Error(response.statusText);
        return response.json();
      })
      .then(result => {
        activeHttpRequests.current = activeHttpRequests.current.filter(
          ctrl => ctrl !== abortController
        );

        setTimeout(() => {
          // for sample
          setData(result);
          setLoading(false);
        }, 4000);
      })
      .catch(err => {
        setError(err);
        setLoading(false);
      });

    return () => {
      activeHttpRequests.current.forEach(ctrl => ctrl.abort());
    };
  }, [url]);
  return { data, error, isLoading };
};
export default useFetch;
```

Utilisation (exemple avec « infinite loading »)

```jsx
import React, { useState } from "react";
import useFetch from "../hooks/use-fetch";

export default function InfiniteLoadingHook() {
  const [limit, setLimit] = useState(2);
  const { data: posts, error } = useFetch(
    `https://jsonplaceholder.typicode.com/posts?_start=0&_limit=${limit}`
  );

  if (!posts) return <p>Loading ...</p>;

  if (error) return <p>Failed to load</p>;

  return (
    <>
      <h1>Infinite Loading - useFetch</h1>
      <section>
        {posts.map((post, index) => (
          <article key={index}>
            <h2>{post.title}</h2>
            <p>{post.body}</p>
          </article>
        ))}
        <div class="text-center">
          <button onClick={() => setLimit(limit + 2)}>Load more</button>
        </div>
      </section>
    </>
  );
}
```

Alternatives : swr et react query.

useHttp

peut-être plus utile que useFetch, permet d'appeler explicitement la méthode send request

```jsx
import { useState, useCallback, useRef, useEffect } from "react";

export const useHttpClient = () => {
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState();

  const activeHttpRequests = useRef([]);

  const sendRequest = useCallback(async (url, options = {}) => {
    setLoading(true);
    const abortController = new AbortController();
    activeHttpRequests.current.push(abortController);

    try {
      const response = await fetch(url, {
        ...options,
        signal: abortController.signal,
      });

      const responseData = await response.json();
```

```
      activeHttpRequests.current = activeHttpRequests.current.filter(
        (ctrl) => ctrl !== abortController
      );

      if (!response.ok) {
        throw new Error(responseData.message);
      }

      setLoading(false);
      return responseData;
    } catch (err) {
      setError(err.message);
      setLoading(false);
      throw err;
    }
  }, []);

  const clearError = () => {
    setError(null);
  };

  useEffect(() => {
    return () => {
      activeHttpRequests.current.forEach((abortCtrl) => abortCtrl.abort());
    };
  }, []);

  return { loading, error, sendRequest, clearError };
};
```

useLocalStorage

```
import { useEffect, useState } from "react";

const getItemFromLocalStorage = (key, defaultValue) => {
  const item = window.localStorage.getItem(key);
  if (item) return JSON.parse(item);
  return defaultValue;
};

const useLocalStorage = (key, defaulValue) => {
  const [value, setValue] = useState(getItemFromLocalStorage(key, defaulValue));

  useEffect(() => {
    window.localStorage.setItem(key, JSON.stringify(value));
  }, [value]);

  return [value, setValue];
};
export default useLocalStorage;
```

utilisation

```
const [messagesA, setMessagesA] = useState(JSON.parse(localStorage.getItem("messagesA")) || []);
 // …

setMessagesA([...messagesA, m];);
```

Services et librairies tiers

Plusieurs possibilités:

1er possibilité : on peut créer un dossier « src/services » avec des fichiers « productService.js » par exemple qui utilisent fetch (axios, etc.)

```
const baseUrl = process.env.REACT_APP_API_BASE_URL;

export async function getProducts(category) {
  const response = await fetch(baseUrl + "products?category=" + category);
  if (response.ok) return response.json();
  throw response;
}

export async function getProductById(id) {
  const response = await fetch(baseUrl + "products/" + id);
  if (response.ok) return response.json();
  throw response;
}
```

… que l'on importe et utilise dans les components (dans useEffect ou au click d'un button par exemple)

2ème possibilité : on crée (dans un dossier « src/hooks ») / installe (avec npm) des custom hooks

Exemple « useFetch » (section précédente), useSWR, etc.

… que l'importe et utilise dans les components (dans useEffect ou au click d'un button par exemple)

3ème possibilité : on met directement le code (fetch par exemple) dans le component (useEffect)

SWR

https://swr.vercel.app/

Installation

```
npm i swr
```

Exemple

```
import React from "react";
import useSWR from "swr";

const fetcher = url => fetch(url).then(res => res.json());

export default function WithSWR() {
  const { data: posts, error } = useSWR(`https://jsonplaceholder.typicode.com/posts`, fetcher);

  if (!posts) return <p>Loading ...</p>;

  if (error) return <p>Failed to load</p>;

  return (
```

```
    <>
      <h1>SWR</h1>
      <section>
        {posts.map((post, index) => (
          <article key={index}>
            <h2>{post.title}</h2>
            <p>{post.body}</p>
          </article>
        ))}
      </section>
    </>
  );
}
```

« infinite loading »

```
import React, { useState } from "react";
import useSWR from "swr";

const fetcher = url => fetch(url).then(res => res.json());

export default function InfiniteLoadingSWR() {
  const [limit, setLimit] = useState(2);
  const { data: posts, error } = useSWR(
    `https://jsonplaceholder.typicode.com/posts?_start=0&_limit=${limit}`,
    fetcher
  );

  if (!posts) return <p>Loading ...</p>;

  if (error) return <p>Failed to load</p>;

  return (
    <>
      <h1>Infinite Loading - SWR</h1>
      <section>
        {posts.map((post, index) => (
          <article key={index}>
            <h2>{post.title}</h2>
            <p>{post.body}</p>
          </article>
        ))}
        <div class="text-center">
          <button onClick={() => setLimit(limit + 2)}>Load more</button>
        </div>
      </section>
    </>
  );
}
```

React Query

https://react-query.tanstack.com/

npm i react-query

```
import React from "react";
import { QueryClient, QueryClientProvider, useQuery } from "react-query";

const queryClient = new QueryClient();

function ReactQuerySample() {
    const {
      data: posts,
      error,
      isLoading,
      isFetching
    } = useQuery(
      "posts",
      () =>
        fetch(`https://jsonplaceholder.typicode.com/posts`).then(res =>
          res.json()
        ) // fetcher
    );
```

```
    if (isLoading) return <p>Loading ...</p>;

    if (error) return <p>Failed to load</p>;

    return (
      <>
        <h1>React Query</h1>
        <section>
          {posts.map((post, index) => (
            <article key={index}>
              <h2>{post.title}</h2>
              <p>{post.body}</p>
            </article>
          ))}
        </section>
      </>
    );
  }

const WithReactQuery = () => {
  return (
    <QueryClientProvider client={queryClient}>
      <ReactQuerySample />
    </QueryClientProvider>
  );
};

export default WithReactQuery;
```

« infinite loading »

```
import React, { useState } from "react";
import { QueryClient, QueryClientProvider, useQuery } from "react-query";

const queryClient = new QueryClient();

function ReactQuerySample() {
  const [limit, setLimit] = useState(2);
  const {
    data: posts,
    error,
    isLoading
  } = useQuery(
    ["posts", limit],
    () =>
      fetch(
        `https://jsonplaceholder.typicode.com/posts?_start=0&_limit=${limit}`
      ).then(res => res.json()) // fetcher
  );

  if (isLoading) return <p>Loading ...</p>;

  if (error) return <p>Failed to load</p>;

  return (
    <>
      <h1>React Query</h1>
      <section>
        {posts.map((post, index) => (
          <article key={index}>
            <h2>{post.title}</h2>
            <p>{post.body}</p>
          </article>
        ))}
        <div className="text-center">
          <button onClick={() => setLimit(limit + 2)}>Load more</button>
        </div>
      </section>
    </>
  );
}

const WithReactQuery = () => {
  return (
```

```
    <QueryClientProvider client={queryClient}>
      <ReactQuerySample />
    </QueryClientProvider>
  );
};

export default WithReactQuery;
```

Class based Component

C'est l'ancienne manière de créer ses components. On ne peut pas utiliser de hooks (useState, useEffect, etc.) donc c'est plutôt à éviter

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

export default class ComponentA extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
    // this.increment = this.increment.bind(this); // or ...
  }

  componentDidMount() {
    console.log("use effect");
  }

  increment(ev) {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment.bind(this)}>Increment</button>
      </div>
    );
  }
}
```

Eliminer le constructor

Il est possible d'éliminer le constructor

```
export default class ComponentA extends Component {
  state = {
    count: 0
  };

  // ...
}
```

Eliminer bind de this avec les arrow functions

```
export default class ComponentA extends Component {
  state = {
    count: 0
  };
```

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

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

Utiliser les hooks avec les class

Solution 1 : créer une function wrapper qui passe les props

```
import React, { Component, useState } from "react";

export default function ComponentAWrapper() {
  const [count, setCount] = useState(0);
  return <ComponentA count={count} setCount={setCount} />;
}

class ComponentA extends Component {
  increment = (ev) => {
    this.props.setCount(this.props.count + 1);
  };

  render() {
    return (
      <div>
        <p>Count: {this.props.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

Solution 2 : function as child

```
import React, { Component, useState } from "react";

function ComponentAWrapper({ children }) {
  const [count, setCount] = useState(0);
  return children(count, setCount);
}

export default class ComponentA extends Component {
  render() {
    return (
      <ComponentAWrapper>
        {(count, setCount) => {
          return (
            <div>
              <p>Count: {count}</p>
              <button onClick={() => setCount(count + 1)}>Increment</button>
            </div>
          );
        }}
      </ComponentAWrapper>
    );
  }
}
```

Cycle de vie du component

- componentWillMount: avant le render initial
- componentDidMount : après le render initial
- componentWillReceiveProps : quand le component reçoit de nouveaux props
- shouldComponentUpdate : avant le render après changement de state, etc. (on peut retourner false pour annuler le rendering)
- componentWillUpdate
- componentDidUpdate
- componentWillUnmount : avant de supprimer le component du DOM

Stateless vs statefull components

Stateless component

Pas de states, se contente d'afficher les informations

```
import React from "react";

const ComponentA = props => {
  return (
    <header>
      <h1>{props.title}</h1>
    </header>
  );
};
export default ComponentA;
```

Statefull component

```
import React, { useEffect, useState } from "react";

const ComponentA = (props) => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const id = setTimeout(() => {
      setCount((count) => count + 1);
    }, 1500);

    return () => {
      clearTimeout(id);
    };
  }, []);

  return (
    <div>
      <p>Count: {count}</p>
    </div>
```

```
  );
};
export default ComponentA;
```

Fragments

Permet de retourner plusieurs éléments sans les mettre dans un conteneur. Cela évite les imbrications (avec div par exemple) inutiles

2 possibilités

Avec « <></> »

```
export default function MainNavigation() {
  const context = useContext(AuthContext);

  return (
    <>
      <NavLink to="/login">Login</NavLink>
      <NavLink to="/signup">Signup</NavLink>
    </>
  );
}
```

Ou avec « React.Fragment »

```
export default function MainNavigation() {
  const context = useContext(AuthContext);

  return (
    <React.Fragment>
      <NavLink to="/login">Login</NavLink>
      <NavLink to="/signup">Signup</NavLink>
    </React.Fragment>
  );
}
```

Context API

Permet de créer un shared context pour l'App par exemple

Dans un dossier « **context** » création d'un contexte « auth-context.js » par exemple

```
import React from "react";

const AuthContext = React.createContext({
    isLoggedIn:false
});

export default AuthContext;
```

Initial value

Dans « **App.js** ». **Import du contexte** et **wrapping** des components child et défintion de la **value**

```
import React from "react";

import AuthContext from "./context/auth-context";
import MainNavigation from "./MainNavigation";

export default function App() {
  return (
    <AuthContext.Provider value={{ isLoggedIn: true }}>
      <MainNavigation />
    </AuthContext.Provider>
  );
}
```

Dans les **child** components, on utilise le contexte

```
import React, { useContext } from "react";

import AuthContext from "./context/auth-context";

export default function MainNavigation() {
  const context = useContext(AuthContext);

  return (
    <>{context.isLoggedIn ? <p>You are logged in!</p> : <p>Please, login</p>}<
/>
  );
}
```

Autre façon de procéder

Création de context, provider et hook dans un fichier « auth-context.js »

```
import React, { useContext } from "react";

const AuthContext = React.createContext();

let initialValue = {
  isLoggedIn: false,
  // other functions ...
};

export function AuthProvider(props) {
  return (
    <AuthContext.Provider value={initialValue}>
      {props.children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error(
      "useAuth must be used within a AuthProvider. Wrap a parent component in <AuthProvider> to fix thi
s error."
```

Hook évitant de devoir importer le contexte et useContext dans les child components du provider

```
    );
  }
  return context;
}
```

Wrap du provider

```
import React from 'react';
import { AuthProvider } from './auth-context';
import MainNavigation from './MainNavigation';

export default function App() {
  return (
    <div>
      <AuthProvider>
        <MainNavigation />
      </AuthProvider>
    </div>
  )
}
```

Utilisation du hook dans le child component (au lieu de useContext)

```
import React from "react";
import { useAuth } from "./auth-context";

export default function MainNavigation() {
  const { isLoggedIn } = useAuth();

  return (
    <>{isLoggedIn ? <p>You are logged in!</p> : <p>Please, login</p>}</>
  );
}
```

On peut utiliser le contexte ou le destructuring

Styles

Inline styles

Directement dans propriété **style**

Double accolade

```
<button style={{ color:"white", background:"blue", padding: "0.5rem 1rem", borderRadius: "5px" }}>Inlin
e styles</button>
```

Ou en dehors pour plus de clarté

```
import React from "react";

const styles = {
  color: "white",
  background: "blue",
  padding: "0.5rem 1rem",
  borderRadius: "5px",
};
```

Les noms des propriétés css peuvent sont « javascript-isées »

```
export default function App() {
  return (<button style={styles}>Inline styles</button>);
}
```

Utilisé pour un rendu conditionnel

```
<label style={{ color: isValid ? 'black':'red' }}>Message</label>
```

```
<div className={`from-control ${!isValid ? "invalid" : ""}`}>…</div>
```

Création d'un theme context et d'un theme provider

```
import React from 'react';

export const ThemeContext = React.createContext({ theme: {} });

export function ThemeProvider(props) {
  return (
    <ThemeContext.Provider value={{ theme: props.theme || {} }}>
      {props.children}
    </ThemeContext.Provider>
  )
}
```

Dans App

```
import React from "react";
import { ThemeProvider } from "./context/theme-context";

import Sample from "./Sample";

const theme = {
    header: {
      color: '#ff598a',
    },
    input: {
      color: '#fff',
      background: '#070222',
      textAlign: 'center',
    },
    inputFocus: {
      outline: '2px solid #5e9fff',
    },
  }

export default function App() {
  return (
    <ThemeProvider theme={theme}>
      <Sample />
    </ThemeProvider>
  );
}
```

Définition du theme à utiliser pour l'application

Utilisation dans un component

```
import React, { useEffect, useState, useContext } from "react";

import { ThemeContext } from "./context/theme-context";
```

Utilisation du theme context pour retrouver les valeurs du theme

```jsx
export default function Sample() {
  const { theme } = useContext(ThemeContext);
  const { width } = useWindowDimensions();
  return (
    <div style={styles.container({ width })}>
      <header style={styles.header({ theme })}>
        <h2 style={styles.headerH2()}>Get the newsletter</h2>
      </header>
    </div>
  );
}

const styles = {
  container: ({ width }) => ({
    position: "relative",
    maxWidth: width >= 800 ? "700px" : "100%",
    fontSize: width >= 800 ? "2.25em" : "1.25em",
    padding: "1em 1em 2em 1em",
    background: "#2b283d",
  }),
  header: ({ theme }) => ({
    position: "relative",
    color: theme.header.color || "white",
    zIndex: "1",
    textTransform: "uppercase",
    fontSize: "0.85em",
    textShadow: "0 3px 2px #000",
  }),
  headerH2: () => ({
    margin: "0 0 0.5em 0",
  }),
};

const useWindowDimensions = () => {
  const [windowDimensions, setWindowDimensions] = useState({
    width: window.innerWidth,
  });

  useEffect(() => {
    function handleResize() {
      setWindowDimensions({ width: window.innerWidth });
    }

    window.addEventListener("resize", handleResize);
    return () => window.removeEventListener("resize", handleResize);
  }, []);

  return windowDimensions;
};
```

Utilisation de la taille de la window pour adapter les styles

On peut définir des styles pour ce component qui utilisent les valeurs du theme

Code généré

```
<html lang="en">
  ▼<head>
     <meta charset="UTF-8">
     <meta http-equiv="X-UA-Compatible" content="IE=edge">
     <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <title>Document</title>
   </head>
  ▼<body>
  ▼<div id="root"> == $0
    ▼<div style="position: relative; max-width: 700px; font-size: 2.25em; padding: 1em 1em 2em; background: rgb(4
    3, 40, 61);">
      ▼<header style="position: relative; color: rgb(255, 89, 138); z-index: 1; text-transform: uppercase; font-s
      ize: 0.85em; text-shadow: rgb(0, 0, 0) 0px 3px 2px;">
         <h2 style="margin: 0px 0px 0.5em;">Get the newsletter</h2>
       </header>
      </div>
    </div>
    <script src="/static/js/bundle.js"></script>
    <script src="/static/js/vendors~main.chunk.js"></script>
    <script src="/static/js/main.chunk.js"></script>
   </body>
  </html>
```

Les styles sont ajoutés inline

« CSS-in-js »

Styled-components

https://styled-components.com/

Installation

```
npm i styled-components
```

```
import React from "react";
import styled from "styled-components";

const Button = styled.button`
    color: white;
    background: blue;
    padding: 0.5rem 1rem;
    border-radius: 5px;
`;

export default function App() {
  return (<Button>Styled components</Button>);
}
```

Nom de l'élément html auquel sera appliqué le style

CSS : ne pas entourer les valeurs par des guillemets

Utiliser « comme » un component React

Supporte la syntaxe scss

```
const Button = styled.button`
  color: white;
  background: blue;
  padding: 0.5rem 1rem;
  border-radius: 5px;

  &:hover,
  &:active{
   background: orange;
  }
`;
```

props

On peut passer des props comme à un component

```
const Button = styled.button`
  color: ${(props) => props.color || "white"};
```

```
`;
```

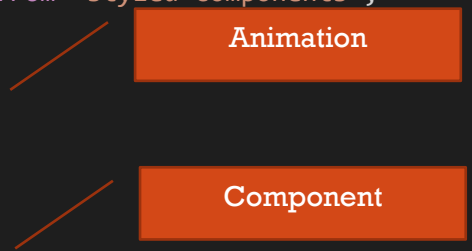
Exemple de passage de props

```
<Button color="red">Styled components</Button>
```

Animations

Import de « keyframes »

```
import styled, { keyframes } from "styled-components";
const rotate = keyframes`
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
`;

const Rotate = styled.div`
  display: inline-block;
  animation: ${rotate} 2s linear infinite;
  padding: 2rem 1rem;
  font-size: 1.2rem;
`;
```



Utilisation

```
<Rotate>&lt; 🖉 &gt;</Rotate>
```

ThemeProvider de styled components

App

```
import React from "react";

import { ThemeProvider } from 'styled-components';

import Sample from "./Sample";

const theme = {
    header: {
      color: '#ff598a',
    },
    input: {
      color: '#fff',
      background: '#070222',
      textAlign: 'center',
    },
    inputFocus: {
      outline: '2px solid #5e9fff',
    },
  }

export default function App() {
  return (
    <ThemeProvider theme={theme}>
      <Sample />
    </ThemeProvider>
  );
}
```

Utilisation

```
import React from 'react';
import styled, { keyframes } from 'styled-components';
```

```jsx
const Header = styled.header`
  position: relative;
  color: ${(props) => props.theme.header.color};
  text-transform: uppercase;
  font-size: 0.85em;

  h2 {
    margin: 0 0 0.5em 0;
  }
`;

export default function Sample() {
  return (
    <div>
      <Header>
        <h2>Title</h2>
      </Header>
    </div>
  );
}
```

Code généré

```
<html lang="en">
▼<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  ▼<style data-styled="active" data-styled-version="5.3.0">
      .bHFcsw{position:relative;z-index:1;text-transform:uppercase;font-size:0.85em;text-shadow:0 3px 2px #000;}
      .bHFcsw h2{margin:0 0 0.5em 0;}
      .bnytDS{position:relative;z-index:1;text-transform:uppercase;font-size:0.85em;}
      .bnytDS h2{margin:0 0 0.5em 0;}
      .TclYx{position:relative;text-transform:uppercase;font-size:0.85em;}
      .TclYx h2{margin:0 0 0.5em 0;}
      .lmZKwj{position:relative;color:#ff598a;text-transform:uppercase;font-size:0.85em;}
      .lmZKwj h2{margin:0 0 0.5em 0;}
    </style>
    <script charset="utf-8" src="/main.125e53a...hot-update.js"></script>
    <script charset="utf-8" src="/main.5cfc0f8...hot-update.js"></script>
    <script charset="utf-8" src="/main.3920a99...hot-update.js"></script>
  </head>
▼<body>
  ▼<div id="root">
    ▼<div>
...   ▼<header class="sc-hKFxyN lmZKwj"> == $0
        <h2>Title</h2>
      </header>
    </div>
  </div>
  <script src="/static/js/bundle.js"></script>
  <script src="/static/js/vendors~main.chunk.js"></script>
  <script src="/static/js/main.chunk.js"></script>
</body>
</html>
```

Des classes css sont ajoutées dans le head de la page

Feuille de styles

On crée une feuille de styles avec le même nom que le component

ComponentA.css

```
.title{
    color: red;
}
```

ComponentA.js

```
import React from 'react';

import "./ComponentA.css";

export default function ComponentA() {
    return (
        <div>
            <h2 className="title">Component A</h2>
        </div>
    )
}
```

Les styles importés sont ajoutés dans le header de la page. L'avantage c'est qu'il n'y a plus de styles « inline »

Import de feuille de styles de packages tiers

Exemple avec boostrap 4

```
npm i bootstrap@4.6.0
```

Dans index.js

```
import "../node_modules/bootstrap/dist/css/bootstrap.min.css";
```

CSS Modules

Un fichier « module css » est scoped au component ou il est importé. Ce qui fait par exemple que l'on peut avoir les mêmes noms de classes dans des modules différents.

Exemple :

On a deux components avec chacun un module css. Chacun a une class css « title » sauf que pour le premier la couleur du texte est définie sur rouge et pour le second sur bleu.

Component A

Component B

ComponentA

```
import React from 'react';
```

```jsx
import styles from "./ComponentA.module.css";

export default function ComponentA() {
    return (
        <div>
            <h2 className={styles.title}>Component A</h2>
        </div>
    )
}
```

ComponentA.module.css

```css
.title{
    color: red;
}
```

ComponentB

```jsx
import React from 'react';

import styles from "./ComponentB.module.css";

export default function ComponentB() {
    return (
        <div>
            <h2 className={styles.title}>Component B</h2>
        </div>
    )
}
```

ComponentB.module.css

```css
.title{
    color: blue;
}
```

Code généré

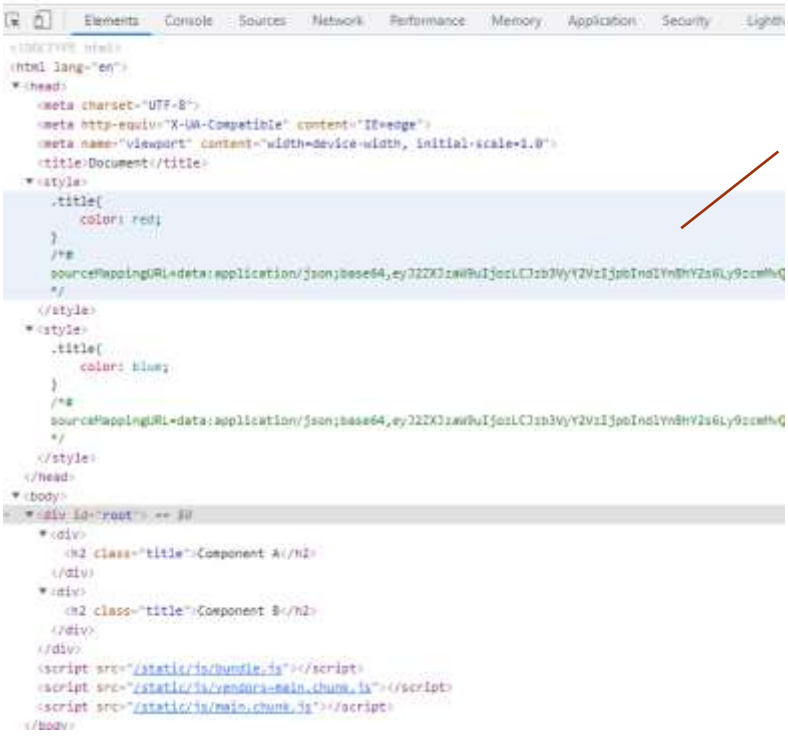
Format : [filename]_[classname]_[hash]

```
.ComponentA_title__2Iq79{
    color: red;
}
/*#
```

Comparaison avec une feuille de styles. Cela ne marche pas

Component A

Component B



Styles ajoutés dans le header, le dernier avec le même nom est appliqué

Libraires

- styled-components
- emotion
- aphrodite
- glamor
- styled-jsx
- radium
- astroturf

Rendering list & conditional content

List avec la function « map »

```
import React, { useState } from "react";

export default function ComponentA() {
  const [messages, setMessages] = useState(["message 1", "message 2"]);

  return (
    <div>
      <ul>
        {messages.map((message, index) =><li key={index}>{message}</li>)}
      </ul>
    </div>
  );
}
```

Indiquer une key pour chaque élément pour aider le re-rendering.

Attention à bien return le content si utilisation d'accolades, sinon cela n'affichera rien

Conditionnel

Le label n'est affiché que s'il y a une erreur

```
{!isValid && <label>My errror</label>}
```

Ternaire

```
{isValid ? <label>Pas d'erreur</label>: <label>My errror</label>}
```

Proptypes & default props

https://reactjs.org/docs/typechecking-with-proptypes.html

Même si l'on voit moins on peut encore utiliser proptypes. Il faut installer le package (https://www.npmjs.com/package/prop-types)

```
npm i prop-types
```

Si le title n'est pas passé en props, on a une erreur dans la console du browser

```
import React from "react";
import PropTypes from "prop-types";
```

```
const ComponentA = (props) => {
  return (
    <header>
      <h1>{props.title}</h1>
    </header>
  );
};

ComponentA.propTypes = {
  title: PropTypes.string.isRequired
};

export default ComponentA;
```

Note : on pourrait définir une default value

```
ComponentA.defaultProps = {
    title: "Default title"
};
```

Redux avec React

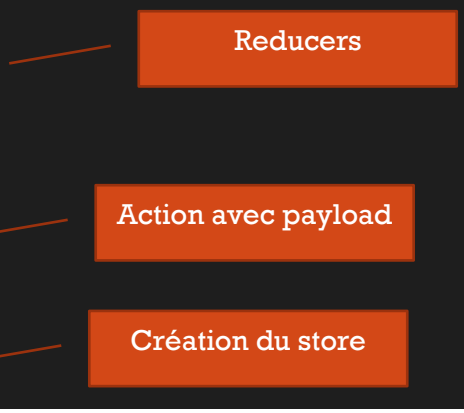
Installation

```
npm i react-redux
```

Reducers, store

Création d'un fichier « index.js » dans un dossier « store »

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

const initialState = {
  count: 0
};

const counterReducer = (state = initialState, action) => {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
      case "INCREASE":
        return { count: state.count + action.amount };
    default:
      return state;
  }
};

const store = createStore(counterReducer);
export default store;
```



Wrapping du provider de React Redux avec le store

```
import ReactDOM from "react-dom";
import { Provider } from "react-redux";
import store from "./store";

import App from "./App";

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

Utilisation

```
import React from "react";
import { useSelector, useDispatch } from "react-redux";

export default function Sample() {
  const dispatch = useDispatch();
  const count = useSelector((state) => state.count);

  const increment = () => {
    dispatch({ type: "INCREMENT" });
  };

  const decrement = () => {
    dispatch({ type: "DECREMENT" });
  };

  const increase = () => {
    dispatch({ type: "INCREASE", amount: 10 });
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={increase}>+10</button>
    </div>
  );
}
```

Sélection du state

Dispatch avec type d'action

Dispatch avec payload

Avec plusieurs reducers

```
import { createStore, combineReducers } from "redux";

const store = createStore(combineReducers({ counterReducer, cartReducer }));
```

Sélection

```
  const { count } = useSelector((state) => ({
    ...state.counterReducer
  }));
```

De plusieurs

```
  const { cart, count } = useSelector((state) => ({
    ...state.cartReducer,
    ...state.counterReducer,
  }));
```

Asynchrone

Avec « redux-thunk »

```
npm i redux-thunk
```

Ajout du middleware

```
import { createStore, combineReducers, applyMiddleware } from "redux";
import thunk from "redux-thunk";
```

```
const store = createStore(combineReducers({ counterReducer, cartReducer }), applyMiddleware(thunk));
```

On crée une function « async » qui retourne une function recevant en paramètre dispatch . Cette function dispatch est utilisée en « callback » une fois l'action finie

```
function increaseAsync() {
  return (dispatch) => {
    setTimeout(() => {
      dispatch({ type: "INCREASE", amount: 100 });
    }, 1000);
  };
}
```

Utilisation du dispatch du component

```
const dispatch = useDispatch();
```

```
<button onClick={() => dispatch(increaseAsync())}>+100 delayed</button>
```

Custom middleware

On peut ajouter des middlewares qui seront appelés à chaque dispatch

```
const customMiddleware = store => next => action => {
  console.log("custom middleware");
  return next(action);

};

const store = createStore(
  combineReducers({ counterReducer, cartReducer }),
  applyMiddleware(customMiddleware, thunk)
);
export default store;
```

Recoil

Alternative à redux

https://recoiljs.org/fr/docs/introduction/getting-started/

```
npm i recoil
```

« atom » state simple

Création des states dans un dossier "atoms" par exemple "todos.js"

```
import { atom } from "recoil";

export const todosState = atom({
  key: "todosState",
  default: []
});
```

Utilisation

```
const [todos, setTodos] = useRecoilState(todosState);
```

Seulement la valeur

```
const todos = useRecoilValue(todosState);
```

Seulement la méthode « update »

```
const setTodos = useSetRecoilState(todosState);
setTodos(oldTodos => [newTodo, ...oldTodos]);
```

« selector » state utilisant d'autres states

```
// ...

export const todosFilterState = atom({
  key: "todosFilterState",
  default: "Show all" // Show dones, Show remaining
});

export const filteredTodosSate = selector({
  key: "filteredTodosSate",
  get: ({ get }) => {
    const filter = get(todosFilterState);
    const todos = get(todosState);

    switch (filter) {
      case "Show dones":
        return todos.filter(x => x.completed === true);
      case "Show remaining":
        return todos.filter(x => x.completed === false);
      default:
        return todos;
    }
  }
});
```

Utilisation

```
const todos = useRecoilValue(filteredTodosSate);
```

Forms

2 types de forms :

- uncontrolled (avec ref)
- controlled (avec state, etc.)

Une form uncontrolled peut être utile avec des forms ayant auto complétion (email et password par exemple) ou qui n'affichent pas/n'ont pas besoin de state (input type file par exemple)

Redux n'est pas nécessaire pour gérer les states de la form, car ceux-ci sont plutôt liés à un formulaire plutôt qu'à l'application.

Libraires :

- Formik
- React hook form

Exemple sans react

```
<body>
  <h1>From Scratch</h1>
  <form id="login-form">
    <input id="email" name="email" type="email" />
    <label id="email-error"></label>
    <input name="password" type="password" />
    <input type="submit" />
  </form>

  <script>
    const form = document.getElementById("login-form");
    const inputField = document.getElementById("email");
    const label = document.getElementById("email-error");

    function handleInput(event) {
      console.log("input", event.target.name, event.target.value);
    }

    function handleBlur(event) {
      console.log("blur", event.target.name, event.target.value);
    }

    function validateEmailField(email) {
      if (!email) {
        return "Email is required";
      }
      return "";
    }

    function handleSubmit(event) {
      event.preventDefault();
      console.log("submit", event);

      // validate
      const error = validateEmailField(inputField.value);
      // affichage errors
      if (error) {
        label.innerText = error;
        label.style = "display:inline";
      } else {
        label.innerText = "";
        label.style = "display:none";
```

```
      }
    }

    inputField.oninput = handleInput; // on typing
    inputField.onblur = handleBlur; // when leaves input
    form.onsubmit = handleSubmit;

    // set initial values (events dont fired)
    inputField.value = "test@email.com";
    label.style = "display:none";
  </script>
</body>
```

Exemple de form « login » uncontrolled

```
import React, { useRef, useState } from "react";

import "./LoginForm.css";

// validation on submit, after submitted

export default function LoginForm() {
  const emailRef = useRef();
  const passwordRef = useRef();
  const [errors, setErrors] = useState({});
  const [isSubmitted, setSubmitted] = useState(false);

  const changeHandler = (ev) => {
    if (isSubmitted) {
      const errors = getErrors();
      setErrors(errors);
    }
  };

  function getErrors() {
    const result = {};
    if (!emailRef.current.value) result.email = "Email is required";
    if (!passwordRef.current.value) result.password = "Password is required";
    return result;
  }

  const submitHandler = (ev) => {
    ev.preventDefault();
    const errors = getErrors();
    const isValid = Object.keys(errors).length === 0;
    if (isValid) {
      // send
    }
    setErrors(errors);
    setSubmitted(true);
  };

  return (
    <>
      <div className="container login-container">
        <div className="row justify-content-md-center">
          <div className="col-md-6 login-form">
            <h3>Login</h3>
            <form onSubmit={submitHandler} noValidate>
              <div className="form-group">
                <input
                  ref={emailRef}
                  id="email"
                  name="email"
                  type="email"
                  className="form-control"
                  placeholder="Your Email *"
                  onChange={changeHandler}
                />
                <p role="alert">{errors.email}</p>
              </div>
              <div className="form-group">
                <input
                  ref={passwordRef}
                  id="password"
```

```
                  name="password"
                  type="password"
                  className="form-control"
                  placeholder="Your Password *"
                  onChange={changeHandler}
                />
                <p role="alert">{errors.password}</p>
              </div>
              <div className="form-group">
                <input type="submit" className="btnSubmit" value="Login" />
              </div>
            </form>
            {/* {JSON.stringify(errors)} */}
          </div>
        </div>
      </div>
    </>
  );
}
```

Exemple form controlled

```
import React, { useState } from "react";

const initialValues = {
  email: "",
  password: "",
};

function getErrors(values) {
  const result = {};
  if (!values.email) result.email = "Email is required";
  if (!values.password) result.password = "Password is required";
  return result;
}

export default function FromScratchForm() {
  const [values, setValues] = useState(initialValues);
  const [errors, setErrors] = useState({});
  const [isSubmitted, setSubmitted] = useState(false);

  const handleChange = (event) => {
    console.log("change", event.target.name, event.target.value); // fired parfois sous chrome au
premier click si le champ est auto complété
    setValues({ ...values, [event.target.name]: event.target.value });

    if (isSubmitted) {
      const errors = getErrors(values);
      setErrors(errors);
    }
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    const errors = getErrors(values);
    const isValid = Object.keys(errors).length === 0;
    if (isValid) {
      // send
    }
    setErrors(errors);
    setSubmitted(true);
  };
  return (
    <>
      <h1>From Scratch</h1>
      <form onSubmit={handleSubmit}>
        <input
          name="email"
          type="email"
          value={values.email}
          onChange={handleChange}
        />
        {errors.email && <label>{errors.email}</label>}
```

```
        <input
          name="password"
          type="password"
          value={values.password}
          onChange={handleChange}
        />
        {errors.password && <label>{errors.password}</label>}
        <input type="submit" />
      </form>
    </>
  );
}
```

TabIndex et focus

« tabIndex » permet de donner l'autre de focus quand on presse la touche TAB des éléments.

On utiliser une « ref » affectée à l'élément auquel on désire donner le focus après le chargement.

```
import React, { useEffect, useRef } from "react";

export default function TabIndexSample() {
  const inputRef = useRef();

  useEffect(() => {
    inputRef.current.focus();
  }, []);

  return (
    <div>
      <form>
        <input tabIndex="1" ref={inputRef} />
        <input tabIndex="4" />
        <input tabIndex="2" />
        <input />
        <input tabIndex="5" />
        <input tabIndex="3" />
        <input type="submit" />
      </form>
    </div>
  );
}
```

Création d'un component Field

Au plus simple

```
const Field = (props)=>{
    return (
        <input {...props}/>
    )
}
```

Permet de faire un input (par défaut), mais aussi textarea, select (avec en children les « options »)

```
const Field = ({ tag, children, ...props }) => {
  return React.createElement(tag, { ...props }, children);
};
Field.defaultProps = {
  tag: "input",
  type: "text"
};

export default function AutoFocusSample() {
  const handleChange = () => {};
  return (
    <form>
      <Field tag="select" name="color">
        <option value="red">Red</option>
```

```
        <option value="green">Green</option>
        <option value="blue">Blue</option>
      </Field>
      <Field tag="textarea" value="test" onChange={handleChange} />
      <Field type="email" value="test@email.com" autoFocus onChange={handleChange} />
    </form>
  );
}
```

Form avec reducer (react)

```
import React, { useContext, useReducer } from "react";

function formReducer(state, action) {
  switch (action.type) {
    case "SET_VALUES":
      return { ...state, values: action.payload };
    case "SET_ERRORS":
      return { ...state, errors: action.payload };
    case "SET_STATUS":
      return { ...state, isSubmitted: action.payload };
    default:
      return state;
  }
}

export function Form1() {
  const initialFormState = {
    values: { email: "", password: "" },
    errors: {},
    isSubmitted: false,
  };
  const [{ values, errors, isSubmitted }, dispatch] = useReducer(
    formReducer,
    initialFormState
  );

  console.log("RENDER", values, errors, isSubmitted);

  const validate = (values) => {
    const result = {};
    if (!values.email) result.email = "Email is required";
    if (!values.password) result.password = "Password is required";
    return result;
  };

  const handleChange = (event) => {
    const newValues = { ...values, [event.target.name]: event.target.value };
    console.log("DISPATCH set values on change", newValues);
    dispatch({
      type: "SET_VALUES",
      payload: newValues,
    });
    if (isSubmitted) {
      const errors = validate(newValues);
      console.log("DISPATCH validation on change", newValues, errors);
      dispatch({
        type: "SET_ERRORS",
        payload: errors,
      });
    }
  };
  const handleSubmit = (event) => {
    event.preventDefault();

    const errors = validate(values);
    dispatch({
      type: "SET_ERRORS",
      payload: errors,
    });

    dispatch({
      type: "SET_STATUS",
```

On doit obtenir et passer les new values pour bien valider les values changées

```
      payload: true,
    });
  };

  return (
    <>
      <h2>Form 1</h2>
      <form onSubmit={handleSubmit} noValidate>
        <input
          name="email"
          type="email"
          value={values.email}
          onChange={handleChange}
        />
        {errors.email && <label>{errors.email}</label>}
        <input
          name="password"
          type="password"
          value={values.password}
          onChange={handleChange}
        />
        {errors.password && <label>{errors.password}</label>}
        <input type="submit" />
      </form>
      {JSON.stringify(values)}
    </>
  );
}
```

Form avec components gérant les changements et la validation « automatiquement »

On utilise un contexte pour les components. On a :

- Un premier component « FormLib » qui crée le context provider et passe aux children différents informations (values, errors, isSubmitted, handleChange, handleSubmit). Ce qui permet d'utiliser les éléments html et bind les éléments values, etc.
- On peut également utiliser les components Form, Field pour que les values, onChange, onSubmit soient automatiquement bindés

```
import React, { useContext, useReducer } from "react";
import PropTypes from "prop-types";

// UTILS

const isFunction = (obj) => typeof obj === "function";

// CONTEXT

const FormContext = React.createContext({});

function FormContextProvider({ children, value }) {
  return <FormContext.Provider value={value}>{children}</FormContext.Provider>;
}

const useFormContext = () => {
  const context = useContext(FormContext);
  if (!context) throw new Error("Context is null");
  return context;
};

// REDUCER

function formReducer(state, action) {
  switch (action.type) {
```

```
    case "SET_VALUES":
      return { ...state, values: action.payload };
    case "SET_ERRORS":
      return { ...state, errors: action.payload };
    case "SET_STATUS":
      return { ...state, isSubmitted: action.payload };
    default:
      return state;
  }
}

// WRAPPER

const FormLib = ({ initialValues, validate, children }) => {
  const [{ values, errors, isSubmitted }, dispatch] = useReducer(formReducer, {
    values: initialValues,
    errors: {},
    isSubmitted: false,
  });
  const handleChange = (event) => {
    const newValues = { ...values, [event.target.name]: event.target.value };
    dispatch({
      type: "SET_VALUES",
      payload: newValues,
    });
    if (isSubmitted) {
      if (isFunction(validate)) {
        const errors = validate(newValues);
        dispatch({
          type: "SET_ERRORS",
          payload: errors,
        });
      }
    }
  };
  const handleSubmit = (event) => {
    event.preventDefault();

    if (isFunction(validate)) {
      const errors = validate(values);
      dispatch({
        type: "SET_ERRORS",
        payload: errors,
      });
    }

    dispatch({
      type: "SET_STATUS",
      payload: true,
    });
  };
  return (
    <FormContextProvider
      value={{ values, errors, isSubmitted, handleSubmit, handleChange }}
    >
      {children({ values, errors, isSubmitted, handleChange, handleSubmit })}
    </FormContextProvider>
  );
};
FormLib.propTypes = {
  initialValues: PropTypes.object.isRequired
};

const Form = ({ children, ...props }) => {
  const { handleSubmit } = useFormContext();
  return (
    <form onSubmit={handleSubmit} {...props}>
      {children}
    </form>
  );
};

// COMPONENTS THAT USE CONTEXT
```

```
const Field = ({ tag, children, name, ...props }) => {
  const { handleChange, values }  = useFormContext();
  return React.createElement(
    tag,
    { onChange: handleChange, value: values[name], name, ...props },
    children
  );
};
Field.defaultProps = {
  tag: "input",
  type: "text",
};

// SAMPLE

export function Form1() {
  const validate = (values) => {
    const result = {};
    if (!values.email) result.email = "Email is required";
    if (!values.password) result.password = "Password is required";
    return result;
  };

  return (
    <FormLib validate={validate} initialValues={{ email: "", password: "" }}>
      {({ values, errors, isSubmitted, handleChange, handleSubmit }) => (
        <>
          <h2>Form Lib 1</h2>
          <Form noValidate>
            <Field name="email" type="email" autoFocus />
            {errors.email && <label>{errors.email}</label>}
            <Field name="password" type="password" />
            {errors.password && <label>{errors.password}</label>}
            <input type="submit" />
          </Form>
          <p>Values: {JSON.stringify(values)}</p>
          <p>IsSubmitted: {isSubmitted}</p>
        </>
      )}
    </FormLib>
  );
}
```

Formik

Documentation

Installation

```
npm i formik
```

Component Formik

Membres utiles du component Formik :

- initialValues : permet de définir les values initiales du forumlaire
- initialErrors
- initialStatus
- initialTouched
- onSubmit : submit handler
- validationSchema
- validate : function de validation retournant un object (ou promise) avec erros

Membres de la function passée en children du component « Formik »

« States »

- values
- errors
- touched
- initialValues
- initialStatus
- initialTouched

Functions « handle »:

- handleBlur
- handleChange
- handleSubmit
- handleReset

Functions « actions » :

- resetForm
- setErrors
- setTouched
- setValues
- submitForm

Variables liées à la validation

- isSubmitting
- isValid
- isValidating
- dirty
- validateOnChange (true)
- validateOnBlur (true)

Méthodes de validation

- validateField
- validateForm

On peut ainsi définir son propre formulaire, il suffit de connecter les différentes informations passées

```
            <form onSubmit={handleSubmit} noValidate>
              <div className="form-group">
                <input
                  id="email"
                  name="email"
                  type="email"
                  value={values.email}
                  onChange={handleChange}
                />
                <p>{errors.email}</p>
              </div>
              <div className="form-group">
                <input
                  id="password"
                  name="password"
                  type="password"
                  value={values.password}
                  onChange={handleChange}
                />
                <p>{errors.password}</p>
              </div>
              <div className="form-group">
                <input type="submit" />
              </div>
            </form>
          </>
        )}
      </Formik>
    </div>
  );
}
```

Components Form, Field, ErrorMessage pour ne pas tout bind soi-même

Components de Formik

- Formik
- Form
- Field
- ErrorMessage

Form, Field, ErrorMessage évitent d'avoir à tout bind soi-même

Equivalent du code précédent avec les components de Formik

```
import React from "react";
import { ErrorMessage, Field, Form, Formik } from "formik";

export default function FormikSample() {
  const validate = (values) => {
    const result = {};
    if (!values.email) result.email = "Email is required";
    if (!values.password) result.password = "Password is required";
    return result;
  };
  const handleSubmit = (values) => {
    console.log(values);
  };
  return (
    <div>
      <h1>Formik</h1>
      <Formik
        initialValues={{ email: "", password: "" }}
        onSubmit={handleSubmit}
        validate={validate}
      >
        {({ values, errors }) => (
          <>
            <Form>
              <div className="form-group">
```

```
              <Field name="email" type="email" />
              <ErrorMessage name="email" component="p" />
            </div>
            <div className="form-group">
              <Field name="password" type="password" />
              <ErrorMessage name="password" component="p" />
            </div>
            <input type="submit" />
          </Form>
        </>
      )}
    </Formik>
  </div>
);
}
```

Variantes pour ErrorMessage

```
<ErrorMessage name="email">
    {error => <label>{error}</label>}
</ErrorMessage>
```

Avec component

On peut imaginer passer d'autres props comme la couleur du texte

```
const ErrorLabel = ({children}) => (<label>{children}</label>);
```

Utilisation

```
<ErrorMessage name="email">
    {error => <ErrorLabel>{error}</ErrorLabel>}
</ErrorMessage>
```

Donner le focus à un élément Field avec « autoFocus »

```
<Field name="email" type="email" autoFocus/>
```

Form level validation et field level validation

Les deux peuvent coexister

Form validation

```
  const validate = (values) => {
    const result = {};
    if (!values.email) result.email = "Email is required";
    if (!values.password) result.password = "Password is required";
    return result;
  };
```

Bind validate sur le component Formik

```
 <Formik
       initialValues={{ email: "", password: "" }}
       onSubmit={handleSubmit}
       validate={validate}
     >
```

Field validation

```
  const validateEmail = (email) => {
    if (!/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i.test(email))
      return "Invalid email address";
    return "";
  };
```

Bind validate sur le component Field

```
 <Field name="email" type="email" validate={validateEmail} />
```

Validation avec Yup

Github

Installation

```
npm i yup
```

Création d'un schéma de validation (plus besoin de la méthode validate)

```
import React from "react";
import { ErrorMessage, Field, Form, Formik } from "formik";
import * as Yup from "yup";

const signinSchema = Yup.object().shape({
  email: Yup.string()
    .email("Invalid email address")
    .required("Email is required"),
  password: Yup.string().required("Password is required"),
});

export default function FormikSample() {
  const handleSubmit = (values) => {
    console.log(values);
  };
  return (
    <div>
      <h1>Formik</h1>
      <Formik
        initialValues={{ email: "", password: "" }}
        onSubmit={handleSubmit}
        validationSchema={signinSchema}
      >
        {() => (
          <>
            <Form>
              <div className="form-group">
                <Field name="email" type="email" />
                <ErrorMessage name="email" component="p" />
              </div>
              <div className="form-group">
                <Field name="password" type="password" />
                <ErrorMessage name="password" component="p" />
              </div>
              <input type="submit" />
            </Form>
          </>
        )}
      </Formik>
    </div>
  );
}
```

Note : on peut définir aussi un validation schema au niveau des fields

React hook form

Installation

```
npm i react-hook-form
```

```
import React from "react";
import { useForm } from "react-hook-form";

export default function ReactHookSample() {
  const {
    register,
    handleSubmit,
    formState: { errors }
  } = useForm();

  const onSubmit = (data) => console.log(data);
  return (
    <>
      <h1>React Hook Form Sample</h1>
      <form onSubmit={handleSubmit(onSubmit)} noValidate>
        <div className="form-group">
          <input
            type="email"
            {...register("email", {
              required: { value: true, message: "Email is required" },
              pattern: {
                value: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i,
                message: "Invalid email address",
              },
            })}
          />
          {errors.email && <p>{errors.email.message}</p>}
        </div>
        <div className="form-group">
          <input
            type="password"
            {...register("password", {
              required: { value: true, message: "Password is required" },
            })}
          />
          {errors.password && <p>{errors.password.message}</p>}
        </div>
        <div className="form-group">
          <input type="submit" />
        </div>
      </form>
    </>
  );
}
```

Sélection du mode de validation

```
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm({ mode: "onSubmit", reValidateMode: "onBlur" });
```

Validation avec Yup

La libraire supporte Yup. Cela n'est pas mal étant donné qu'il y a assez peu de validateurs différents de base.

Installer

```
npm install @hookform/resolvers yup
```

On crée un schema avec Yup et on le passe en options de « useForm » avec yupResolver. On enlève les règles de validation des champs

```
import React from "react";
import { yupResolver } from "@hookform/resolvers/yup";
import * as Yup from "yup";
import { useForm } from "react-hook-form";

const signinSchema = Yup.object().shape({
  email: Yup.string()
    .email("Invalid email address")
    .required("Email is required"),
  password: Yup.string().required("Password is required"),
});

export default function ReactHookSample() {
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm({ mode: "onSubmit", resolver: yupResolver(signinSchema) });

  const onSubmit = (data) => {
    console.log(data);
  };

  return (
    <>
      <h1>React Hook Form Sample</h1>
      <form onSubmit={handleSubmit(onSubmit)} noValidate>
        <div className="form-group">
          <input name="email" type="email" {...register("email")} />
          {errors.email && <p>{errors.email.message}</p>}
        </div>
        <div className="form-group">
          <input name="password" type="password" {...register("password")} />
          {errors.password && <p>{errors.password.message}</p>}
        </div>
        <div className="form-group">
          <input type="submit" />
        </div>
      </form>
    </>
  );
}
```

Disabled le button submit pendant la submission

```
function wait(time) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve();
    }, time);
  });
}
```

```
  const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting, isValid },
  } = useForm({ mode: "onSubmit", resolver: yupResolver(signinSchema) });
```

```
  const onSubmit = async (data) => {
    await wait(4000);
  };
```

```
<input disabled={isSubmitting} type="submit" />
```

React Router

Documentation

Installation

```
npm i react-router-dom
```

Les liens et routes doivent être des children du router

Une solution est de wrap l'App dans « index.js » avec BrowserRouter.

```
import ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";
import App from "./App";

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

Exemple

```
import React from "react";
import {
  BrowserRouter,
  Link,
  NavLink,
  Route,
  Switch,
  useLocation,
  useParams,
} from "react-router-dom";

import classes from "./index.css";

const HomePage = () => <h1>Home</h1>;

const DetailPage = () => {
  const { id } = useParams();
  const location = useLocation();

  console.log(location);
```

```
  return <h1>Id reçu: {id}</h1>;
};

export default function App() {
  return (
    <BrowserRouter>
      <ul>
        <li>
          <NavLink to="/" activeClassName={classes.active} exact>
            Home
          </NavLink>
        </li>
        <li>
          <NavLink
            to={{
              pathname: "/products/10",
              search: "?sort=name",
              hash: "#the-hash",
            }}
            activeClassName={classes.active}
            exact
          >
            Detail
          </NavLink>
        </li>
      </ul>

      <Switch>
        <Route path="/" exact>
          <HomePage />
        </Route>
        <Route path="/products/:id" exact>
          <DetailPage />
        </Route>
        <Route path="*">
          <h1>Not found</h1>
        </Route>
      </Switch>
    </BrowserRouter>
  );
}
```

Liens sont children de BrowserRouter

Routes dans un Switch

Link et NavLink

Utilisation de NavLink permet d'avoir « activeClassName »

Link

```
<Link to="/products/10">Detail</Link>
```

Lien dynamique

```
<Link to={`/products/${props.id}`}>Detail</Link>
```

NavLink

On ajoute « exact » si besoin, pour éviter par exemple qu'un NavLink vers « / » soit sélectionné aussi

```
<NavLink to="/products/10" activeClassName={classes.active} exact>
   Detail
</NavLink>
```

Définition d'un style « active » dans la feuille de style

```
.active{
    background: orange;
}
```

Et import

```
import classes from './index.css';
```

Récupération de params avec useParams

Par exemple on a une route

```
<Route path="/products/:id" exact>
     <DetailPage />
</Route>
```

On récupère l'id avec le hook useParams

```
import { useParams } from "react-router-dom";
```

```
const DetailPage = () => {
  const { id } = useParams();
  return <h1>Id reçu: {id}</h1>;
};
```

Lien avec hash, search, etc.

```
<NavLink
            to={{
              pathname: "/products/10",
              search: "?sort=name",
              hash: "#the-hash",
            }}
            activeClassName={classes.active}
            exact
          >
            Detail
          </NavLink>
```

Récupération des informations avec le hook useLocation

```
const DetailPage = () => {
  const { id } = useParams();
  const location = useLocation();

  console.log(location);
```

```
  return <h1>Id reçu: {id}</h1>;
};
```

```
▼ {pathname: "/products/10", search: "?sort=name", hash: "#the-hash", key: "rglscd"}
    hash: "#the-hash"
    key: "rglscd"
    pathname: "/products/10"
    search: "?sort=name"
  ▶ [[Prototype]]: Object
```

Utiliser l'history pour la navigation dans le code

```
const DetailPage = () => {
  const history = useHistory();

  const handleClick = () => {
    history.goBack();
  };

  return (<button onClick={handleClick}>Back</button>);
};
```

Pour naviguer

```
history.push("/");
```

Redirection

Exemple au lieu d'afficher une page not found on fait une redirection vers la page d'accueil

```
<Route path="*">
          <Redirect to="/"/>
</Route>
```

REST API

Plusieurs possibilités :

- XMlHttpRequest
- fetch
- Axios

Avec fetch

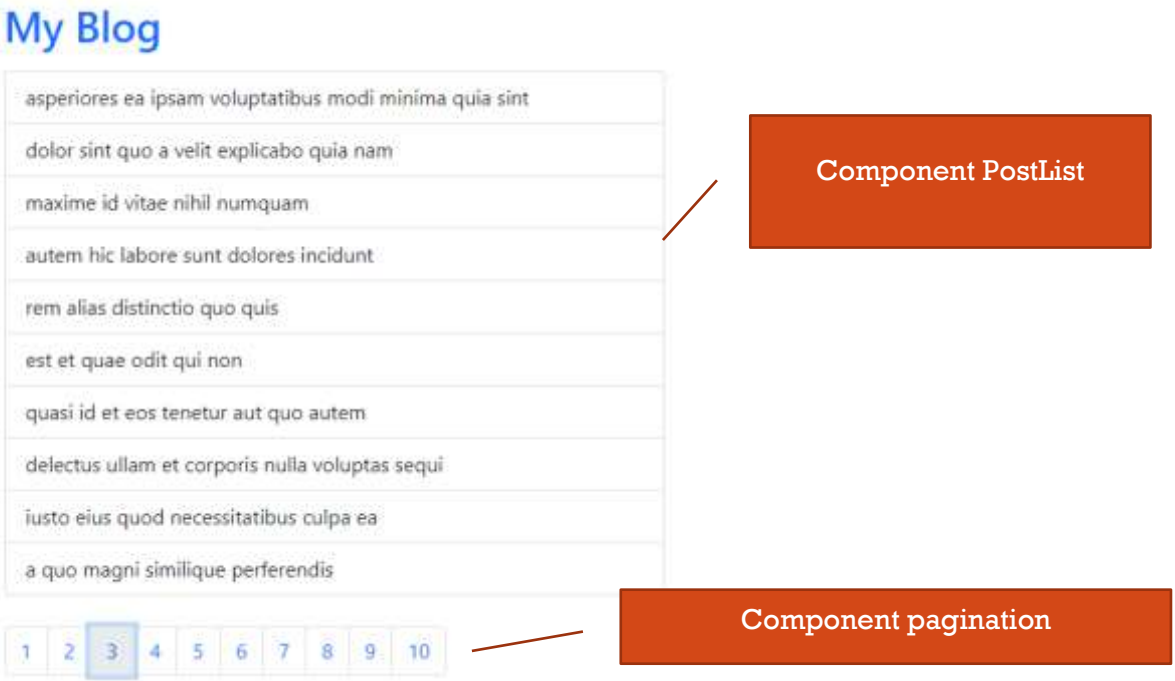
On l'utilisera en général au chargement donc dans useEffect

```
  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then((response) => response.json())
      .then((responseData) => console.log(responseData))
      .catch((err) => console.log("Error", err));
  }, []);
```

Avec async await

```
useEffect(async () => {
  try {
    const response = await fetch(
      "https://jsonplaceholder.typicode.com/posts"
    );
    const responseData = await response.json();
    console.log(responseData);
  } catch (error) {
    console.log("Error", error);
  }
}, []);
```

Pagination



Page « Posts.js »

```
import React, { useEffect, useState } from "react";
import Pagination from "../components/Pagination";
import PostList from "../components/PostList";

export default function Posts() {
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState();
  const [posts, setPosts] = useState([]);

  const [postsPerPage] = useState(10); // 10 posts per page
  const [currentPage, setCurrentPage] = useState(1); // page 1...2...3...

  useEffect(() => {
    const fetchPosts = async () => {
      try {
        setLoading(true);

        const response = await fetch(
          "https://jsonplaceholder.typicode.com/posts"
        );

        const responseData = await response.json();
        setPosts(responseData);
      } catch (err) {
```

```
        setError(err.message);
        console.log(err);
      } finally {
        setLoading(false);
      }
    };

    fetchPosts();
  }, []);

  // change current page
  const paginate = pageNumber => setCurrentPage(pageNumber);

  // Get current posts
  const indexOfLastPost = currentPage * postsPerPage; // 10 (1 * 10) ... 20 (2 * 10) ... 30
  const indexOfFirstPost = indexOfLastPost - postsPerPage; // 0 (10 - 10) ... 10 (20 - 10) ... 20
  const currentPosts = posts.slice(indexOfFirstPost, indexOfLastPost);

  return (
    <div className="container mt-5">
      <h1 className="text-primary mb-3">My Blog</h1>
      <PostList posts={currentPosts} loading={loading} />
      <Pagination
        postsPerPage={postsPerPage}
        totalPosts={posts.length}
        paginate={paginate}
      />
    </div>
  );
}
```

Compoent « PostList.js »

Affiche la liste de posts de la page courante

```
import React from "react";

export default function PostList({ loading, posts }) {
  if (loading) return <p>Loading...</p>;

  return (
    <ul className="list-group mb-4">
      {posts.map(post => (
        <li key={post.id} className="list-group-item">
          {post.title}
        </li>
      ))}
    </ul>
  );
}
```

Component « Pagination.js »

Affiche des liens avec les numéros des pages. Le click sur un lien permet de change la page courante

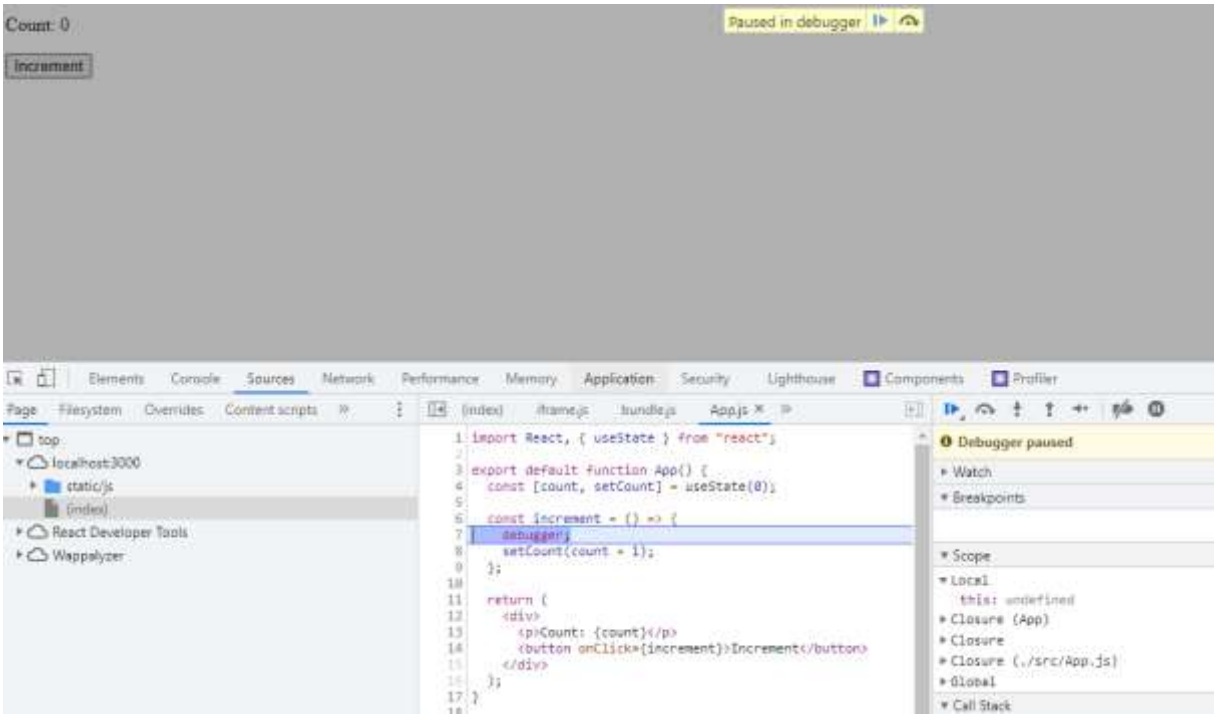
```
import React from "react";

export default function Pagination({ postsPerPage, totalPosts, paginate }) {
  const pageNumbers = [];
```

```
  for (let i = 1; i <= Math.ceil(totalPosts / postsPerPage); i++) {
    pageNumbers.push(i);
  }

  return (
    <nav>
      <ul className="pagination">
        {pageNumbers.map(number => (
          <li key={number} className="page-item">
            <a onClick={() => paginate(number)} href="!#" className="page-
link">
              {number}
            </a>
          </li>
        ))}
      </ul>
    </nav>
  );
}
```

Debugger

Il est possible de forcer l'arrêt du debugger du browser avec le mot clé « debugger » dans le code react

Activer les breakpoints dans le panneau « sources » de Chrome



```jsx
import React, { useState } from "react";

export default function App() {
  const [count, setCount] = useState(0);

  const increment = () => {
    debugger;
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

SEO

On peut utiliser Next.js ou Gatsby. Sinon il reste possible d'ajouter des balises dans le header

```jsx
document.title = "New Title";

let meta = document.createElement('meta');
meta.name = "description";
meta.content = "My content";
document.getElementsByTagName('head')[0].appendChild(meta);
```

On peut aussi utiliser une librairie comme react-helmet

```
npm i react-helmet
```

```jsx
import "./styles.css";
import { Helmet } from "react-helmet";

export default function App() {
  return (
    <div className="App">
      <Helmet>
```

```
        <title>Title with Helmet</title>
        <meta name="description" content="My content" />
      </Helmet>

      <h1>SEO with Helmet</h1>
    </div>
  );
}
```

Typescript avec React

Souvent on aura à se demander quel type donner :

cheatSheet : https://github.com/typescript-cheatsheets/react ou le site (mieux structuré)

Les infos bulles pourront aussi donner une information sur les types attendus

Tests unitaires

Tests avec Jest et @testing-library

On teste plutôt le html qui serait généré. Pareil pour la sélection des éléments

Installation et configuration

- Jest
- @testing-library :
    - jest-dom https://github.com/testing-library/jest-dom (permet de faire expect … toBe…)
    - react permet par exemple de faire un fake « render » et de sélectionner les éléments avec « screen » https://testing-library.com/docs/react-testing-library/cheatsheet
    - user-event permet de simuler des events (exemple click d'un button) https://testing-library.com/docs/ecosystem-user-event/

```
npm i jest @testing-library/react @testing-library/jest-dom @testing-library/user-event -D
```

A cela on pourrait ajouter un polyfill de fetch pour les tests afin d'exécuter réellement les fetch des components plutôt que de créer un mock

```
npm i isomorphic-fetch -D
```

Création d'un fichier de configuration de jest « **jest.config.js** »

```
module.exports = {
  roots: ["<rootDir>/src"],
  transform: {
    "^.+\\.(js|jsx)$": "babel-jest"
  },
  setupFilesAfterEnv: [
    "@testing-library/jest-dom/extend-expect",
    "<rootDir>/setupTests.js"
  ],
  testMatch: ["**/__tests__/**/*.[jt]s?(x)", "**/?(*.)+(spec|test).[jt]s?(x)"],
  moduleFileExtensions: ["ts", "tsx", "js", "jsx", "json", "node"],
  testEnvironment: "jsdom"
```

```
};
```

Note : « babel-jest » est normalement installé en tant que dépendance.

On peut créer des fichiers de tests dans un dossier « src/__tests__ » ou à côté des components testés en terminant leur nom par « test.js » ou « spec.js »

Création d'un fichier « **setupTests.js** » à la racine du projet ou autre. Cela permettra d'éviter de répéter des imports dans chaque fichier de test.

```
import "@testing-library/jest-dom";
import 'isomorphic-fetch';
```

Scripts : Création d'un script dans package.json

```
"test": "jest"
```

On peut également définir explicitement le fichier de configuration s'il n'est pas à la racine du projet avec « **jest -c config/jest.config.js** »

On pourrait également ajouter le **coverage** avec « **jest –coverage** »

```
"test:coverage": "npm test -- --coverage"
```

Watch requiert au moins d'avoir fait un « git init » pour éviter l'erreur « --watch is not supported without git/hg »

```
"test:watch": "npm test -- --onlyChanged --watch"
```

Queries

2 versions : retournant un node (getBy… par ex) ou une liste de nodes (getAllBy… par ex)

- « getBy… », « getAllBy.. » retournent les nodes correspondants ou throw une error
- « queryBy », « queryAllBy » retournent les nodes correspondants ou null
- « findBy… », « findAllBy… » retournent une promise (permettant de faire async / await)

Type of Query	0 Matches	1 Match	>1 Matches	Retry (Async/Await)
Single Element				
`getBy...`	Throw error	Return element	Throw error	No
`queryBy...`	Return `null`	Return element	Throw error	No
`findBy...`	Throw error	Return element	Throw error	Yes
Multiple Elements				
`getAllBy...`	Throw error	Return array	Return array	No
`queryAllBy...`	Return `[]`	Return array	Return array	No
`findAllBy...`	Throw error	Return array	Return array	Yes

Trouver des éléments

Avec « container » qui permet d'utiliser querySelector

```
import React from "react";
import { render, screen } from "@testing-library/react";
```

```
const { container } = render(<Sample />);
const button = container.querySelector("button");
const h2 = container.querySelector("h2");
```

Permet de verifier le contenu d'un élément par exemple

```
const { container } = render(<Sample />);
const h2 = container.querySelector("h2");

expect(container).toBeDefined();
expect(h2.outerHTML).toEqual("<h2>Mon titre</h2>");
expect(h2.innerHTML).toEqual("Mon titre");
```

Vérifier qu'un texte est bien présent

Exemple un component qui afficherait le message passé

```
test("should displays message", async () => {
  const r = render(<Sample message="Mon message" />);

  const element = r.getByText("Mon message");
  // or
  // const element = screen.getByText("Mon message"); // avec screen
```

```
  expect(element).toBeInTheDocument();
});
```

Quelques rôles pour « getByRole »

- link
- article
- button
- textbox (input)
- img
- combobox (select)
- table
- list (ul)
- listitem (li)

```
render(<Sample />);
const link = screen.getByRole("link");
expect(link.href).toEqual("http://localhost/test");
```

Astuce : pour obtenir le role d'un élément https://testing-library.com/docs/dom-testing-library/api-accessibility#getroles

Sélection par « data-testid ». On ajoute l'attribute « data-testid » sur le component, ce qui permet de le sélectionner plus facilement avec « getByTestId »

```
import React from "react";
import { render, screen } from "@testing-library/react";

const MyButton = ({ handleClick }) => {
  return (
    <button data-testid="my-button" onClick={handleClick}>
      Button
    </button>
  );
};

describe("MyButton component", () => {
  it("should call once on click", () => {
    const fn = jest.fn();
    const { getByTestId } = render(<MyButton handleClick={fn} />);

    const buttonElement = getByTestId("my-button");
    // etc.
  });
});
```

Expect de Jest pour tester un résultat

https://jestjs.io/docs/expect

- expect(value)
- expect.extend(matchers)
- expect.anything()
- expect.any(constructor)
- expect.arrayContaining(array)
- expect.assertions(number)
- expect.hasAssertions()
- expect.not.arrayContaining(array)
- expect.not.objectContaining(object)
- expect.not.stringContaining(string)
- expect.not.stringMatching(string | regexp)
- expect.objectContaining(object)
- expect.stringContaining(string)
- expect.stringMatching(string | regexp)
- expect.addSnapshotSerializer(serializer)
- .not
- .resolves
- .rejects
- .toBe(value)
- .toHaveBeenCalled()
- .toHaveBeenCalledTimes(number)
- .toHaveBeenCalledWith(arg1, arg2, ...)
- .toHaveBeenLastCalledWith(arg1, arg2, ...)
- .toHaveBeenNthCalledWith(nthCall, arg1, arg2, ...)
- .toHaveReturned()
- .toHaveReturnedTimes(number)
- .toHaveReturnedWith(value)
- .toHaveLastReturnedWith(value)
- .toHaveNthReturnedWith(nthCall, value)
- .toHaveLength(number)
- .toHaveProperty(keyPath, value?)
- .toBeCloseTo(number, numDigits?)
- .toBeDefined()
- .toBeFalsy()
- .toBeGreaterThan(number | bigint)
- .toBeGreaterThanOrEqual(number | bigint)
- .toBeLessThan(number | bigint)
- .toBeLessThanOrEqual(number | bigint)
- .toBeInstanceOf(Class)
- .toBeNull()
- .toBeTruthy()
- .toBeUndefined()
- .toBeNaN()
- .toContain(item)
- .toContainEqual(item)
- .toEqual(value)
- .toMatch(regexp | string)
- .toMatchObject(object)

- .toMatchSnapshot(propertyMatchers?, hint?)
- .toMatchInlineSnapshot(propertyMatchers?, inlineSnapshot)
- .toStrictEqual(value)
- .toThrow(error?)
- .toThrowErrorMatchingSnapshot(hint?)
- .toThrowErrorMatchingInlineSnapshot(inlineSnapshot)

Globals (beforeEach, beforeAll, etc.)

https://jestjs.io/docs/api

- afterAll(fn, timeout)
- afterEach(fn, timeout)
- beforeAll(fn, timeout)
- beforeEach(fn, timeout)
- describe(name, fn)
- describe.each(table)(name, fn, timeout)
- describe.only(name, fn)
- describe.only.each(table)(name, fn)
- describe.skip(name, fn)
- describe.skip.each(table)(name, fn)
- test(name, fn, timeout)
- test.concurrent(name, fn, timeout)
- test.concurrent.each(table)(name, fn, timeout)
- test.concurrent.only.each(table)(name, fn)
- test.concurrent.skip.each(table)(name, fn)
- test.each(table)(name, fn, timeout)
- test.only(name, fn, timeout)
- test.only.each(table)(name, fn)
- test.skip(name, fn)
- test.skip.each(table)(name, fn)
- test.todo(name)

Test présence DOM

```
import React, { useState } from "react";

export default function Welcome() {
  return (
    <div>
      <h1>Hello World!</h1>
    </div>
  );
}
```

```
import React from "react";
import Welcome from "./Welcome";
import { render, screen } from "@testing-library/react";

describe("Welcome component", () => {
  test("should renders hello world text", () => {
    render(<Welcome />); // render de @testing-library/react
```

```
    const element = screen.getByText("Hello World!");
    expect(element).toBeInTheDocument();
  });

});
```

Test event

```
import React, { useState } from "react";

export default function Welcome() {
  const [count, setCount] = useState(0);
  const handleClick = () => {
    setCount(count + 1);
  };
  return (
    <div>
      <h1>Hello World!</h1>
      <p>Count: {count}</p>
      <button onClick={handleClick}>Increment</button>
    </div>
  );
}
```

```
import React from "react";
import Welcome from "./Welcome";
import { render, screen } from "@testing-library/react";
import userEvent from '@testing-library/user-event';

describe("Welcome component", () => {

  test("renders count 1 when button is clicked once", () => {
    render(<Welcome />);

    const buttonElement = screen.getByRole("button");
    userEvent.click(buttonElement); // trigger click with @testing-
library/user-event'

    const element = screen.getByText("Count: 1", { exact: false });
    expect(element).toBeInTheDocument();
  });
});
```

Test async (async / await) + fetch (polyfill isomorphic-fetch)

```
import React from "react";
import { useEffect, useState } from "react";

const Async = () => {
  const [posts, setPosts] = useState([]);
```

```
  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then(response => response.json())
      .then(data => {
        setPosts(data);
      });
  }, []);

  return (
    <div>
      <ul>
        {posts.map(post => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
};

export default Async;
```

```
import React from "react";
import { render, screen } from "@testing-library/react";
import Async from "./Async";

describe("Async component", () => {
  test("renders posts if request succeeds", async () => {
    render(<Async />);

    const listItemElements = await screen.findAllByRole("listitem");
    expect(listItemElements).not.toHaveLength(0);
  });
});
```

Tester directement le component avec React

Exemple on vérifie le type de l'élément child crée et le message affiché. C'est basique, mais on voit le principe. Il pourrait par exemple y avoir une condition et que l'affichage diffère selon.

```
import React from "react";

const ErrorMessage = ({ error }) => {
  return <>{error && <p className="invalid-feedback">{error}</p>}</>;
};

describe("ErrorMessage component", () => {
  test("should renders error", () => {
    const component = ErrorMessage({ error: "My error message" });
    expect(component.props.children.type).toBe("p"); // <p>
    expect(component.props.children.props.children).toBe("My error message");
    expect(component.props.children.props.className).toBe("invalid-feedback");
  });
});
```

Tests avec "react-dom/test-utils"

On teste plutôt le html qui serait généré

https://fr.reactjs.org/docs/test-utils.html

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

```jsx
import { act } from "react-dom/test-utils";

const ErrorMessage = ({ error }) => {
  return <>{error && <p className="invalid-feedback">{error}</p>}</>;
};

describe("ErrorMessage component", () => {
  let container;

  beforeEach(() => {
    container = document.createElement("div");
    document.body.appendChild(container);
    act(() => {
      ReactDOM.render(<ErrorMessage error="My error message" />, container);
    });
  });

  afterEach(() => {
    document.body.removeChild(container);
    container = null;
  });

  test("should renders error", () => {
    const p = container.querySelector("p");
    expect(container.outerHTML).toBe(
      '<div><p class="invalid-feedback">My error message</p></div>'
    );
    expect(p.innerHTML).toBe("My error message");
    expect(p.className).toBe("invalid-feedback");
    // or
    // expect(p.getAttribute("class")).toBe("invalid-feedback");
  });
});
```

Préparation

DispatchEvent

```jsx
import React, { useState } from "react";
import ReactDOM from "react-dom";
import { act } from "react-dom/test-utils";

const Counter = () => {
  const [count, setCount] = useState(0);
  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={handleClick}>Increment</button>
    </div>
  );
};

describe("Counter component", () => {
  let container, p, button;

  beforeEach(() => {
    container = document.createElement("div");
    document.body.appendChild(container);
    act(() => {
```

```
      ReactDOM.render(<Counter />, container);
    });
    button = container.querySelector("button");
    p = container.querySelector("p");
  });

  afterEach(() => {
    document.body.removeChild(container);
    container = null;
  });
  it("should increment", () => {
    act(() => {
      button.dispatchEvent(new MouseEvent("click", { bubbles: true }));
    });
    expect(p.textContent).toBe("Count: 1");
  });
});
```

Tests avec "react-test-render"

Philosophie différente, on teste les objets (type == tag html), props qui seraient définis avec « createElement » et React.

https://fr.reactjs.org/docs/test-renderer.html

Installation

```
npm i react-test-renderer -D
```

Sélection des éléments avec findByType (le « type » est le nom du tag html généré), findByProps (filtre sur une valeur de props)

Assertions : attention on teste les objets qu'on aurait avec « createElement » et non le html qui « serait généré »

```
import React from "react";
import TestRenderer from "react-test-renderer";

const ErrorMessage = ({ error }) => {
  return <>{error && <p className="invalid-feedback">{error}</p>}</>;
};

describe("ErrorMessage component", () => {
  test("should renders error", () => {
    const tr = TestRenderer.create(<ErrorMessage error="My error message" />);
    const targetInstance = tr.root;

    expect(targetInstance.findByType("p").props.children).toBe(
      "My error message"
    );
    expect(
      targetInstance.findByProps({ className: "invalid-feedback" }).children
    ).toEqual(["My error message"]);
  });
});
```

Act + onClick

```
import React, { useState } from "react";
import TestRenderer from "react-test-renderer";

const Counter = () => {
  const [count, setCount] = useState(0);
  const handleClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={handleClick}>Increment</button>
    </div>
  );
};

describe("Counter component", () => {
  it("should increment on click", () => {
    const tr = TestRenderer.create(<Counter />);
    const targetInstance = tr.root;

    const button = targetInstance.findByType("button");

    TestRenderer.act(() => button.props.onClick());
    expect(targetInstance.findByType("p").props.children).toEqual(["Count: ",
1]);
  });
});
```

Mocks avec Jest

https://jestjs.io/fr/docs/mock-functions

```
import React from "react";
import { render, screen } from "@testing-library/react";
import userEvent from "@testing-library/user-event";

const MyButton = ({ handleClick }) => {
  return (
    <button onClick={handleClick}>
      Button
    </button>
  );
};

describe("MyButton component", () => {
```

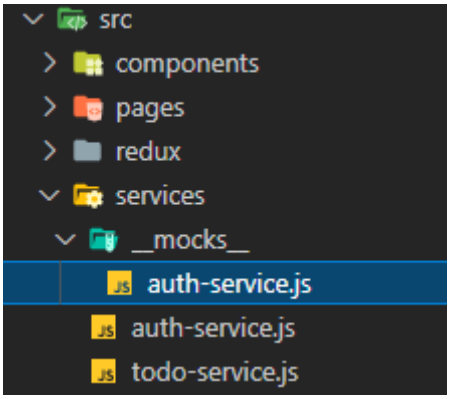
```
  it("should call once on click", () => {
    const fn = jest.fn();
    render(<MyButton handleClick={fn} />);

    const buttonElement = screen.getByRole("button");
    userEvent.click(buttonElement); // trigger click with @testing-
library/user-event'

    expect(fn.mock.calls.length).toBe(1);
  });
});
```

Valeur retournée

```
const myMock = jest.fn();

console.log(myMock()); // > undefined

myMock.mockReturnValueOnce(10).mockReturnValueOnce("x").mockReturnValue(true);

console.log(myMock(), myMock(), myMock(), myMock()); // > 10, 'x', true, true
```

Mocks fichiers

Exemple on crée un service "auth-service" (dans dossiers "__mocks___") qui remplacera le service de base. Il faut que le mocks soient dans le même repertoire que la cible



Dans les tests on inclue

```
jest.mock("../services/auth-service");
```

exemple de function du mock qui retourne une promise

```
export const signInWithEmailAndPassword = async (email, password) => {
  return new Promise((resolve, reject) => {
    resolve({ user: email });
  });
};
```

```
import * as actions from "./auth-actions";
import * as types from "./auth-action-types";

jest.mock("../services/auth-service");

describe("Auth Actions tests", () => {
  it("Fake signin", async () => {
    const result = [];
    const dispatch = jest.fn(t => result.push(t));
    await actions.signInWithEmailAndPassword("test@mail.com", "test")(dispatch
);

    expect(result.length).toEqual(2);
    expect(result[0]).toEqual({ type: types.SIGNIN_START });
    expect(result[1]).toEqual({
      type: types.SIGNIN_SUCCESS,
      payload: "test@mail.com"
    });
  });
});
```

MockStore

Installer

```
npm I redux-mock-store
```

```
import * as actions from "./auth-actions";
import * as types from "./auth-action-types";
import thunk from "redux-thunk";
import configureMockStore from "redux-mock-store";

const middleware = [thunk];
const mockStore = configureMockStore(middleware);

jest.mock("../services/auth-service");

describe("Auth Actions tests", () => {
  it("Fake signin", async () => {
    const store = mockStore({ user: null });

    return store
      .dispatch(actions.signInWithEmailAndPassword("test@mail.com", "test"))
      .then(() => {
        const result = store.getActions();
        expect(result.length).toEqual(2);
```

```
        expect(result[0]).toEqual({ type: types.SIGNIN_START });
        expect(result[1]).toEqual({
          type: types.SIGNIN_SUCCESS,
          payload: "test@mail.com"
        });
      });
  });
});
```

Tests E2E avec Cypress

https://docs.cypress.io/api/table-of-contents

Installation et configuration

npm i cypress -D

Ajout des scripts à package.json

```
"cypress:open": "cypress open",
"cypress:run": "cypress run"
```

- "cypress open" permet de lancer les tests dans le browser (chrome par défaut)
- "cypress run" permet d'afficher seulement les résultats des tests dans la console sans ouvrir le browser

Il est possible d'indiquer un match pour sélectionner les tests

```
"cypress:run": "cypress run -s cypress/integration/**/*.spec.js"
```

Note : Au premier lancement des tests, le dossier cypress, le fichier de configuration « cypress.json » sont ajoutés

Créer un dossier « **cypress/integration** » et y ajouter les fichiers de tests.

Fichier de configuration « **cypress.json** »
https://docs.cypress.io/guides/references/configuration#Global

```
{
  "baseUrl": "http://localhost:3000",
  "video":false
}
```

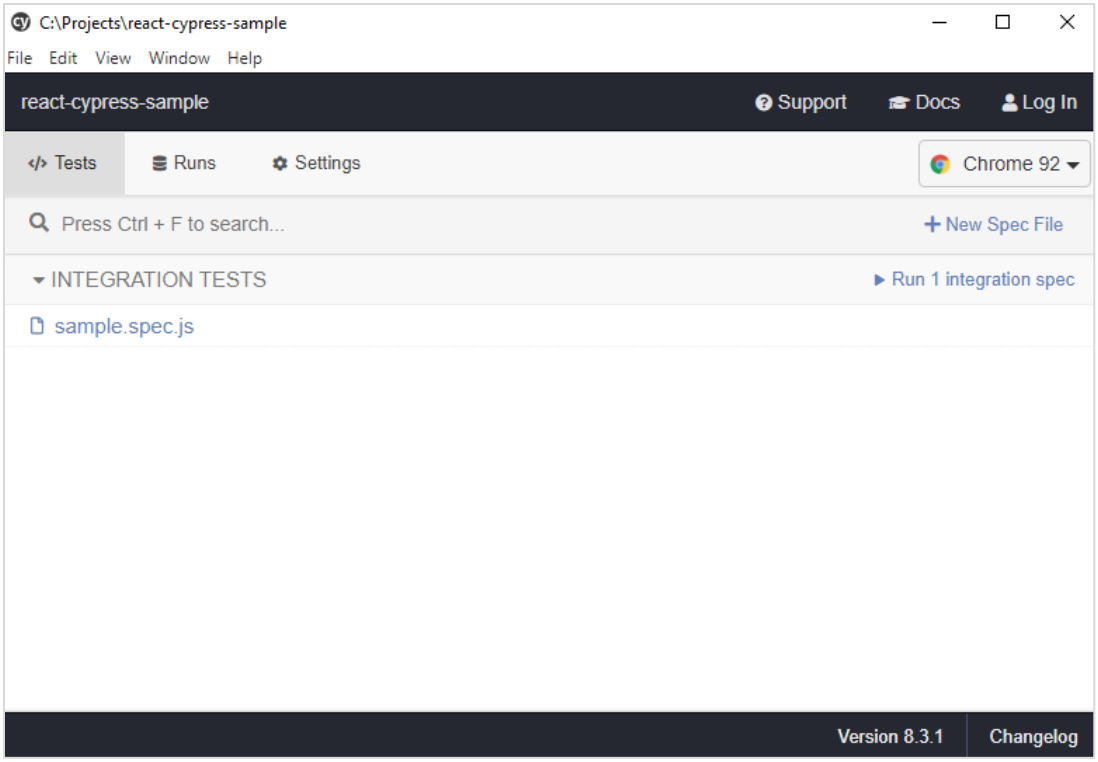
Permet par exemple de définir l'url de base permettant de faire directement **cy.visit("/")**

Avoir l'intellisense dans les fichiers de tests

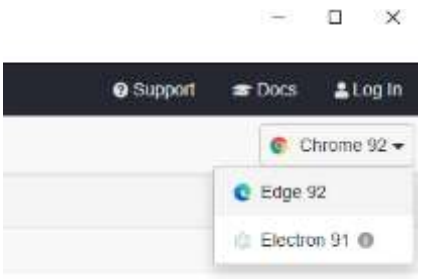
```
/// <reference types="Cypress" />

describe("First Test", () => {
// etc.
```

Lancer les tests

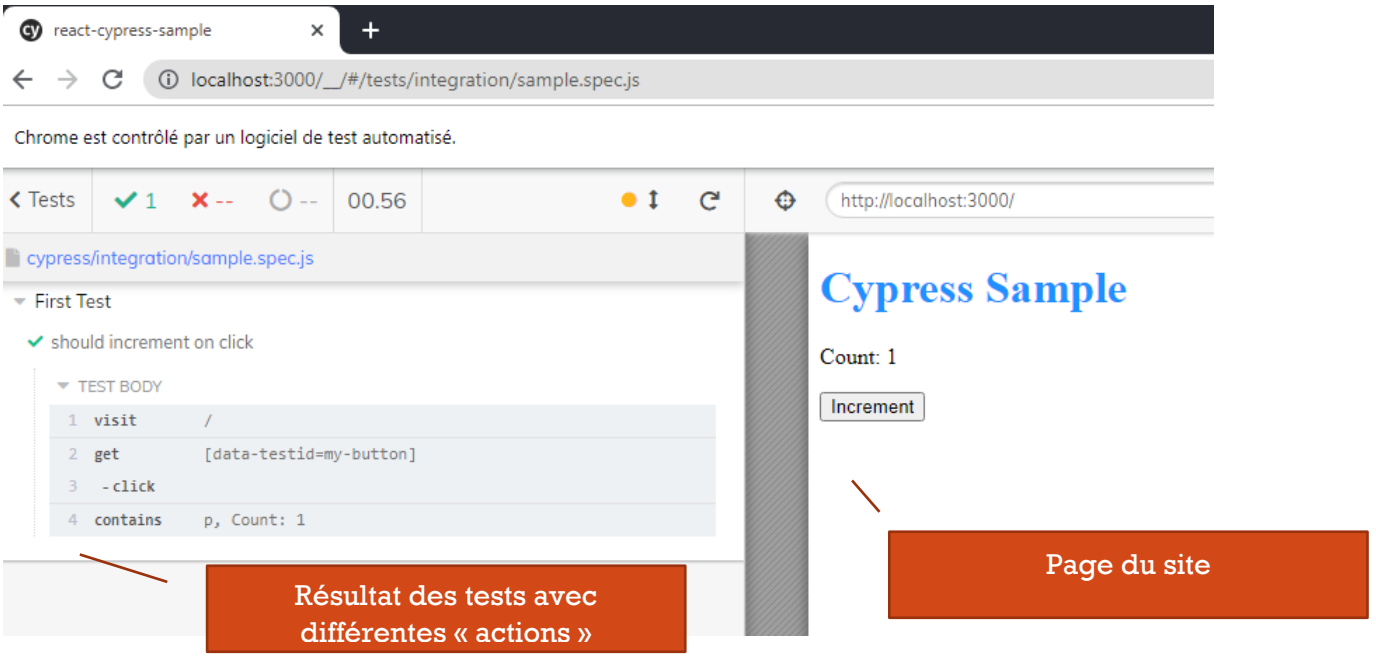
Avec la commande « cypress open » , une tool permettant de sélectionner les tests s'ouvre.



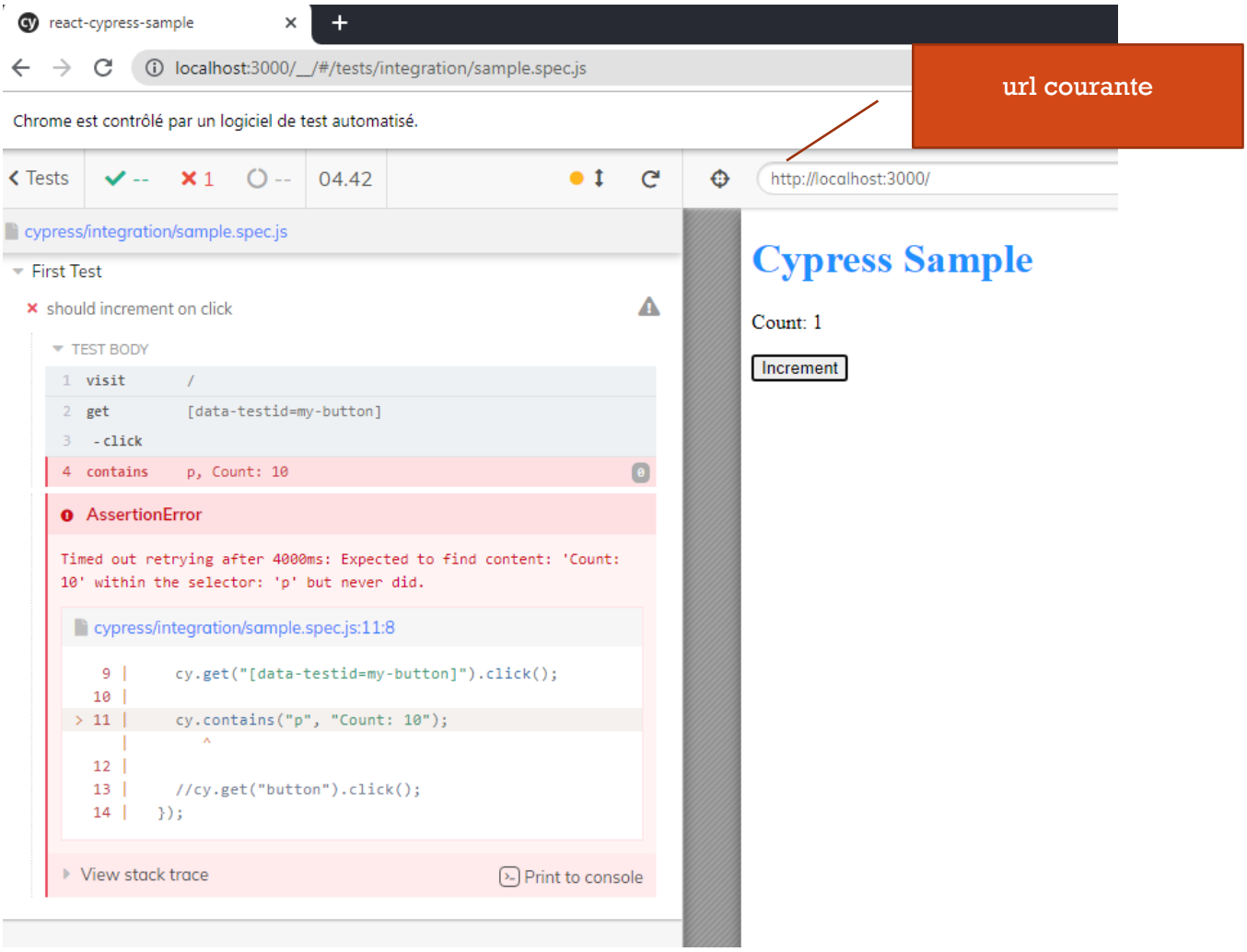
Browser support : On peut sélectionner le brwser à ouvrir



Lorsque l'on clique sur un test, le browser s'ouvre.



Exemple en cas d'erreur



Création de tests

Ajouter les tests dans le dossier « cypress/integration ». Ils peuvent être nommés comme on veut (sauf si match est défini) et ne pas forcément finir « .test.js » ou « .spec.js »

- Commands
- Assertions (basé sur chai)
- Events
- Plugins

Commandes

Custom commands

Il est possible de définir des custom commands

https://docs.cypress.io/api/cypress-api/custom-commands#Syntax

Visiter une page

```
cy.visit("http://localhost:3000/");
// ou si baseUrl définie dans cypress.json
cy.visit("/");
```

Visiter une autre page du site

```
cy.visit("/page-b");
```

Il est possible également de visiter une page externe au site.

Sélectionner des éléments dans la page

Sélection simple comme avec querySelector

Exemples

```
    cy.get("p")
    cy.get("#my-id")
    cy.get(".my-class")
```

Attribut « data-testid » (ou autre nom d'attribut)

Comme pour Jest et @testing-library on peut définir un attribut « data-testid » sur les components pour les sélectionner

Dans le component

```
<button data-testid="my-button" onClick={handleClick}>Increment</button>
```

Dans le test

```
cy.get("[data-testid=my-button]")
```

Il existe d'ailleurs une libraire par @testing-library https://testing-library.com/docs/cypress-testing-library/intro/

Déclencher un event

On sélectionne l'élément puis on trigger l'event

```
cy.get("button").click();
```

Avec une form

```
cy.get("form").submit();
```

Changer le contenu d'un élément

Exemple input

```
const username = "testuser";
cy.get("input").type(username);
```

Contains

Permet de vérifier la présence d'un contenu. Exemple on vérifie que le paragraphe contient le texte « Count : 1 »

```
cy.contains("p", "Count: 1");
```

Note contains peut également permettre de chainer. Exemple un link pour lequel on vérifie le contenu puis on chaine par « click() »

Location

Permet de récupérer les informations d'url

```
cy.location('pathname').should('equal', '/login')
```

Assertions

Avec should

Liste des assertions utiles:
https://docs.cypress.io/guides/references/assertions#Common-Assertions

Have text (exact) sinon utilizer "include-text"

```
cy.get("[data-testid=message]").should("have.text", "Message");
```

Not contain

```
cy.get("[data-testid=message]").should("not.contain", "!!");
```

exists et not exist

```
cy.get("[data-testid=message]").should("exist");
cy.get("[data-testid=message]").should("not.exist");
```

Equal

```
cy.location('pathname').should('equal', '/')
```

has value

```
 cy.get("input").should("have.value", "testuser");
```

visible

```
cy.get("[data-testid=message]").should('be.visible');
```

have class et not have class (css)

```
cy.get("p").should("have.class", "valid");
```

```
cy.get("form").find("input").should("not.have.class", "disabled");
```

Faire un screenshot

```
 cy.screenshot("screenshot-page-b");
```

Aliases

On définit un alias avec la function « as » et on utilise l'alias en précédent son nom de @

```
// peut être défini dans before each
cy.get("button").as("btn");

cy.get("@btn").click();
```

Server

https://docs.cypress.io/api/commands/server#Options

Plugins

https://docs.cypress.io/plugins/directory

Next.js

React est considéré comme une librairie pour le frontend.

Next est considéré comme un framework (« for production »).

- Routing : pas besoin de react router, basé sur les fichiers/dossiers du dossier « pages »
- Pre-rendering des pages permettant de charger les données en avance plutôt qu'une fois la page chargée avec useEffect
- « Fullstack » on peut définir des api utilisant node
- Génération de pages « statiques », très bon pour le SEO, on n'a plus une div root vide avec un bundle sans header mais des pages avec du contenu et un header.

Exemples

https://github.com/vercel/next.js/tree/master/examples

Création de projet

Depuis une invite de commande

```
npx create-next-app <project_name>
```

Pour lancer en mode dev (hot reloading)

```
npm run dev
```

La page est disponible sur http://localhost:3000

Pour build, générer les pages et tester le code généré

```
npm run build
npm start
```

Structure projet :

- Dossier « **pages** » contient les pages intervenant dans la navigation basée sur les files
- Dossier « **styles** » pour les fichiers css et modules css des pages
- Dossier « **public** » contenant les assets utilisés par le site (pas d'index.html)

Exemples d'application

https://github.com/vercel/next.js/tree/master/examples

Création d'un projet à partir d'un exemple, exemple avec tailwind

```
npx create-next-app -e with-tailwindcss next-tailwind-sample
```

File based routing

File « index.js »

« pages/index.js » => http://localhost:3000/

Dans un sous dossier « pages/blog/index.js » => http://localhost:3000/blog

Il suffit de retourner un component par default nommé comme on veut. Exemple :

```
import React from "react";
import Image from "next/image";

export default function Home({ array }) {
  return (
    <>
      <h1>Home</h1>
      <Image src="/logo.svg" alt="Logo" width={72} height={16}/>
    </>
  );
}
```

Route/file « named »

« pages/contact.js » => http://localhost:3000/contact

Dans un sous dossier « pages/authors/list.js » => http://localhost:3000/authors/list

« Dynamic » route/path

On entoure de crochets le nom de la page. Exemples : [productId].js [slug].js

Peuvent recevoir n'importe quel path

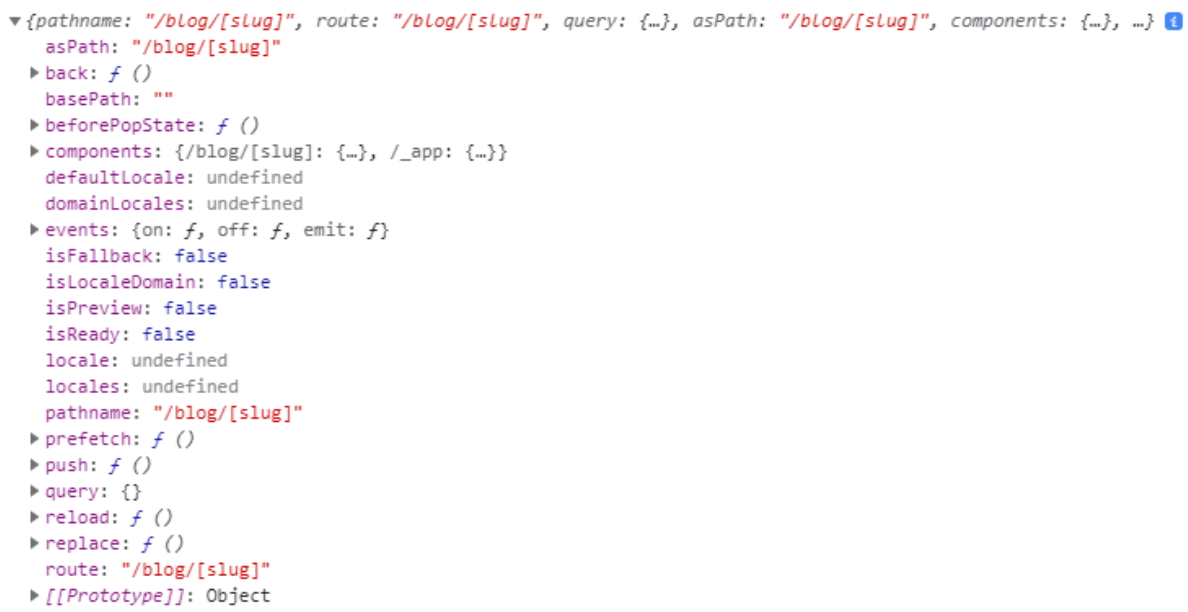
Exemple « pages/blog/[slug].js » => http://localhost:3000/blog/abc12345

Pour récupérer les informations passées : « useRouter »

```
import React from "react";
import { useRouter } from "next/router";

export default function PostDetailPage() {
  const router = useRouter();
```

```
  console.log(router);

  return (
    <div>
      <h1>Page article</h1>
      <p>{router.query.slug}</p>
      <button onClick={()=> router.back()}>Back</button>
      <button onClick={()=> router.push("/")}>Home</button>
    </div>
  );
}
```

On peut récupérer diverses informations de routes (pathname, query)



Catching all

Page avec crochet + 3 points […slug].js

Intercepte tous les paths. Exemple « pages/blog/[…slug].js] »

http://localhost:3000/blog/abc/123

http://localhost:3000/blog/abc/123/456

Custom page 404

Il suffit de créer une page 404.js à la racine de « pages »

```
import React from "react";

export default function NotFound() {
  return <h1>Oups... not found</h1>;
}
```

Navigation par code

Avec les méthodes push, replace, back de « useRouter »

```
import { useRouter } from "next/router";
```

Push

```
router.push("/blog/post-xyz-54321")
```

Ou avec le path et query définis

```
router.push({
            pathname: "/blog/[slug]",
            query: { slug: "post-xyz-54321" },
          })
```

Replace

Fait la même chose que push sauf le state current est remplacé dans l'history

Back

```
router.back()
```

Link

```
import Link from "next/link";
```

```
<Link href="/blog/create">New Post</Link>
```

Supporte également l'écriture en object (avec pathname, query)

Pre rendering

2 formes : génération de pages static ou server side rendering

```
λ  (Server)  server-side renders at runtime (uses getInitialProps or getServerSideProps)
o  (Static)  automatically rendered as static HTML (uses no initial props)
●  (SSG)     automatically generated as static HTML + JSON (uses getStaticProps)
   (ISR)     incremental static regeneration (uses revalidate in getStaticProps)
```

Le code de getStaticProps, getStaticPaths et getServerSideProps n'apparait pas dans le code côté client.

getStaticProps

Permet de précharger les données pendant le pre-rendering à l'inverse de useEffect qui doit attendre le chargement de la page.

```
import React from "react";

export default function Posts({ posts }) {
  return (
    <>
      <h1>Posts</h1>
      {posts.map((post, index) => (
        <div key={index}>
          {post.title}
        </div>
```

```
      ))}
    </>
  );
}

export function getStaticProps() {
  const posts = [
    { id: 1, title: "Post 1" },
    { id: 2, title: "Post 2" },
  ];
  return {
    props: {
      posts,
    },
  };
```

On pourrait aussi récupérer des informations dynamiques

```
export async function getStaticProps() {
  const response = await fetch("https://jsonplaceholder.typicode.com/posts");
  const posts = await response.json();
  return {
    props: {
      posts,
    },
  };
}
```

Si on regarde la page générée après « npm run build » on peut constater qu'elle contient les informations

ISR avec revalidate

À utiliser seulement pour des données qui ont besoin d'être mis à jour régulièrement/souvent. Exemple toutes les 10 secondes

```
export async function getStaticProps() {
  const posts = [
```

```
    { id: 1, title: "Post 1" },
    { id: 2, title: "Post 2" },
  ];
  return {
    props: {
      posts,
    },
    revalidate : 10
  };
}
```

On peut ajouter un

```
console.log("RENDER");
```

pour voir dans la console de VS Code que la function est régulièrement exécutée

redirect et not found dans getStaticProps

```
export async function getStaticProps() {
  const response = await fetch("https://jsonplaceholder.typicode.com/posts");
  const posts = await response.json();

  if (!posts) {
    return {
      redirect: {
        destination: "/",
      },
    };
  }

  return {
    props: {
      posts,
    },
  };
}
```

notFound redirige vers 404

```
  if (!posts) {
    return { notFound: true };
  }
```

getStaticPaths pour les routes dynamiques

Pour les routes telles que [slug].js

Permet de définir les chemins et pages « static » générées pendant le build

```
import React from "react";

export default function PostDetailPage({ selectedPost }) {
  return (
    <div>
      <h1>{selectedPost.title}</h1>
    </div>
  );
}

export async function getStaticPaths() {
  return {
    paths: [{ params: { slug: "post-abc" } }, { params: { slug: "post-xyz" } }],
    fallback: false
  };
}

export async function getStaticProps(context) {
  const slug = context.params.slug;

  // const post = await getPost(slug); // dynamic
  // or for sample
  const post = { slug, title: "Post " + slug };

  return {
    props: {
      selectedPost: post,
    },
  };
}
```

getStaticProps est requis avec getStaticPaths

Le fallback (requis) indique comment gérer les paths qui ne sont définis. A false, ils ne sont pas gérés dynamqiuement.

Avec le build (« npm run build »), on constate que des pages sont générées pour chaque path (404)

```
Page                                Size     First Load JS
┌ ○ /                               3.57 kB        71.8 kB
├   └ css/6b9cae372f078ab4a172.css  25.7 kB
├   /_app                          0 B            68.3 kB
├ ○ /404                           283 B          68.5 kB
├ λ /api/hello                     0 B            68.3 kB
├ ● /blog (345 ms)                 342 B          68.6 kB
├ ○ /blog/[...slug]                302 B          68.6 kB
├ ● /blog/[slug] (ISR: 30 Seconds) 333 B          68.6 kB
├   ├ /blog/post-abc
├   └ /blog/post-xyz
└ ○ /contact                       283 B          68.5 kB
```

Les pages contiennent déjà le contenu

Avec fallback true

```
export default function PostDetailPage({ selectedPost }) {
  if (!selectedPost) return <p>Loading ...</p>;

  return (
    <div>
      <h1>{selectedPost.title}</h1>
    </div>
  );
}

export async function getStaticPaths() {
  return {
    paths: [{ params: { slug: "post-abc" } }, { params: { slug: "post-
xyz" } }],
    fallback: true,
  };
}

export async function getStaticProps(context) {
  const slug = context.params.slug;

  // const post = await getPost(slug); // dynamic
  // or for sample
  const post = { slug, title: "Post " + slug };

  return {
    props: {
      selectedPost: post,
    },
    revalidate: 30,
  };
```

Obligé d'ajouter un loading screen afin d'éviter une erreur de build

```
}
```

getServerSideProps

Permet d'avoir accès à req et res de node

```
export async function getServerSideProps(context) {
  const { req, res, params } = context;

  console.log(req, res);

  const posts = [
    { id: 1, title: "Post 1" },
    { id: 2, title: "Post 2" },
  ];
  return {
    props: {
      posts,
    },
  };
}
```

Création de fichiers utils

Exemple création d'un dossier « lib » à la racine du projet avec un fichier « post-util.js » contenant des functions utilisables par getStaticProps, getServicerSideProps, getStaticPaths

```
export async function getPosts() {
  const response = await fetch("https://jsonplaceholder.typicode.com/posts");
  const posts = await response.json();
  return posts;
}

// autres functions
```

Utilisation

```
import React from "react";
import { getPosts } from "../../lib/post-util";

// …

export async function getServerSideProps(context) {
  const posts = await getPosts();
  return {
```

```
    props: {
      posts,
    },
  };
}
```

API

File based également. On a un dossier « pages/api » ce qui donne donc des routes http://localhost:3000/api/... Supporte également les sous dossiers

Exemple si on a un fichier nommé « hello.js » « pages/api/hello.js » => http://localhost:3000/api/hello

https://nextjs.org/docs/api-routes/introduction

```
export default function handler(req, res) {
  res.status(200).json({ name: "John Doe" });
}
```

Route dynamique [message].js => http://localhost:3000/api/world

```
export default function handler(req, res) {
  const { message } = req.query;
  res.status(200).json({ message: `Hello ${message}!` });
}
```

Astuces

Mongoose

Models

Création des models dans un dossier « models » à la racine du projet

```
const mongoose = require("mongoose");

const Schema = mongoose.Schema;
const postSchema = new Schema({
  title: { type: String, required: true },
  content: { type: String, required: true },
  created_date: { type: Date, default: Date.now },
});
// evite l'erreur Cannot overwrite `Post` model once compiled.
export default mongoose.models.Post ||  mongoose.model("Post", postSchema);
```

Connexion

Création d'un helper « db-connect.js » dans un dossier « helpers » à la racine du projet

```
const mongoose = require("mongoose");

export default async function dbConnect() {
```

```
  let client;
  try {
    const url = "mongodb://jerome:secret@127.0.0.1:27017/blog?authSource=admin
";
    client = await mongoose.connect(url, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
  } catch (error) {
    console.log(error);
  }
}
```

Post (create.js d'un dossier « api/blog » par exemple)

```
import dbConnect from "../../../lib/db-connect";
import Post from "../../../models/post";

export default async function handler(req, res) {
  if (req.method === "POST") {
    await dbConnect();

    const { title, content } = req.body;
    if (!title || !content) {
      res.status(422).json({ message: "Invalid inputs." });
    }

    try {
      const newPost = new Post({ title, content });
      newPost.save();
      res.status(201).json({ message: "Post added", post: newPost });
    } catch (error) {
      res.status(500).json({ message: "Inserting post failed!" });
    }
  }
}
```

Get (index.js d'un dossier « api/blog » par exemple)

```
import dbConnect from "../../../lib/db-connect";
import Post from "../../../models/post";

export default async function handler(req, res) {
  await dbConnect();
```

```
  const posts = await Post.find({});
  res.status(200).json({ posts });
}
```

Helpers

On peut créer des helpers dans un dossier à la racine du projet nommé « helpers » par exemple. Exemple un helpers pour la connexion à une base mongoose

Head

On peut le définir dans les pages et dans _app.js

Ils sont mergés si besoin. Exemple

_app.js

```
import Head from "next/head";
import Layout from "../components/Layout/Layout";
import "../styles/globals.css";

function App({ Component, pageProps }) {
  return (
    <Layout>
      <Head>
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
      </Head>
      <Component {...pageProps} />
    </Layout>
  );
}

export default App;
```

blog/index.js

```
import Head from "next/head";
import React from "react";
import { getPosts } from "../../lib/post-util";

export default function Posts({ posts }) {
  return (
    <>
      <Head>
        <meta name="description" content="Post list" />
        <title>Posts</title>
      </Head>
      ;<h1>Posts</h1>
      {posts.map((post, index) => (
        <div key={index}>{post.title}</div>
      ))}
    </>
  );
}
```

On pourrait avoir un titre et une description plus dynamique pour les pages de détails

```
function PostDetailPage(props) {
  return (
    <>
      <Head>
        <title>{props.post.title}</title>
        <meta name='description' content={props.post.excerpt} />
      </Head>
      <h2>{props.post.title}</h2>
    </>
  );
}
```

_document.js structure des pages

On peut créer une page « _document.js » permettant de décrire la structure des pages

```
import Document, { Html, Head, Main, NextScript } from 'next/document';

class MyDocument extends Document {
  render() {
    return (
      <Html lang='en'>
        <Head />
        <body>
          <Main />
          <NextScript />
          <div id="notifications"></div>
        </body>
      </Html>
    );
  }
}

export default MyDocument;
```

Création d'un Layout

Création de Layout.js dans un dossier « components » à la racine du projet

```
import React from "react";
import Navbar from "../Navbar/Navbar";

export default function Layout({ children }) {
  return (
    <>
      <Navbar />
      {children}
    </>
  );
}
```

_app.js

Wrapping de component (qui représente les pages rendues)

```
import Head from "next/head";
import Layout from "../components/Layout/Layout";
import "../styles/globals.css";

function App({ Component, pageProps }) {
  return (
    <Layout>
      <Head>
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
      </Head>
      <Component {...pageProps} />
    </Layout>
  );
}

export default App;
```

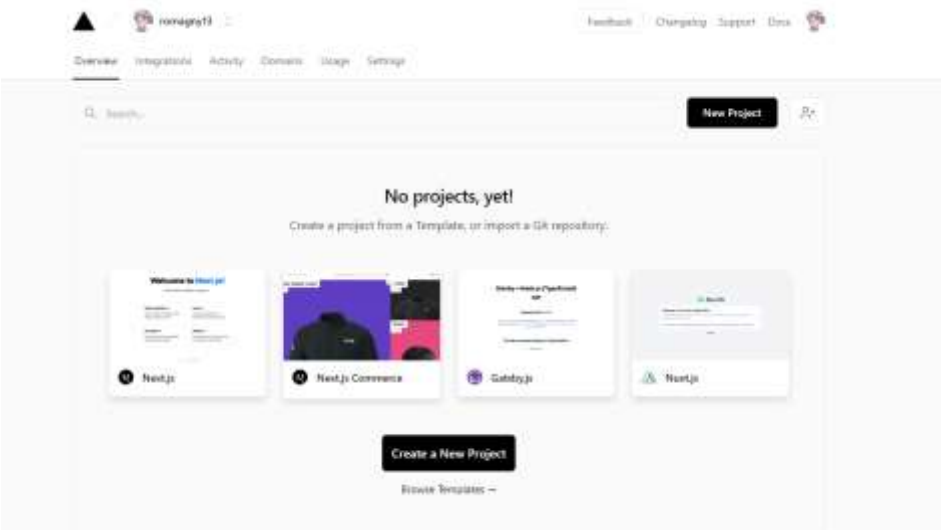
Déploiement avec Vercel

https://vercel.com/

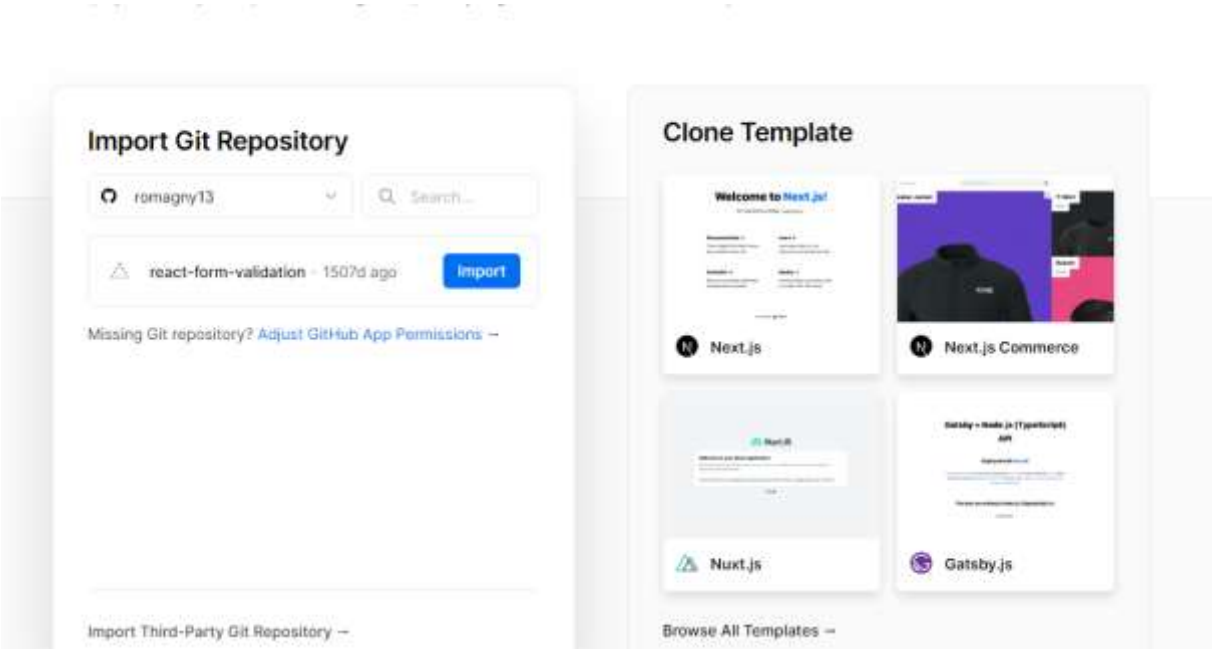
Créer un compte en se connectant avec github par exemple.

Créer un repository sur github, build et push le projet.

Depuis le dashboard « create new project



On peut importer depuis un repository github



… deploy

Gatsby

Permet de créer des sites statiques

Installation

On peut installer Gatsby CLI globalement

```
npm i -g gatsby-cli
```

Création de projet

A partir de starter

On peut créer un projet à partir d'un starter https://www.gatsbyjs.com/starters/

Plusieurs starters intéressants :

- Le classique « hello world » https://github.com/gatsbyjs/gatsby-starter-hello-world
- « gatsby-starter-default » https://github.com/gatsbyjs/gatsby-starter-default

2 possibilités : avec npx et avec gatsby installé en global. A savoir que le projet généré est strictement le même.

Exemple avec le starter « hello world »

Avec npx

```
npx gatsby new gatsby-hello-world https://github.com/gatsbyjs/gatsby-starter-hello-world
```

Avec Gatsby installé globalement

```
gatsby new gatsby-hello-world https://github.com/gatsbyjs/gatsby-starter-hello-world
```

Lancement

```
npm run develop
```

Et/ou si gatsby installé globalement

```
gatsby develop
```

On peut visiter la page http://localhost:8000/ et pour graphql (graphiql)http://localhost:8000/___graphql

Navigation basé sur les fichiers du dossier « pages »

File name	Route
index.js	/
about.js	/about

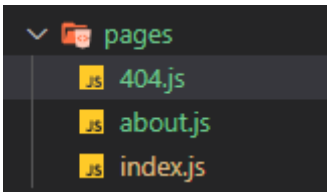
Note : attention si on nomme un fichier avec une capitale (exemple « About.js ») il faudra naviguer vers « /About » sinon cela ne le trouvera pas (404)

Ce qui est possible : faire des components sans nom. Exemple « about.js »

```
import React from "react"

export default function () {
  return (
    <div>
      <h1>About page</h1>
    </div>
  )
}
```

Création d'une page « 404.js »

```
import React from "react"

export default function () {
  return (
    <div>
      <h1>Page not found</h1>
    </div>
  )
}
```



Créer des liens vers les pages

Avec la balise anchor

```
<a href="/">Home</a> | <a href="/About">About</a>
```

Ou avec le component « Link » de Gatsby

```
import { Link } from "gatsby"
```

```
<Link to="/">Home</Link>| <Link to="/about">About</Link>
```

Styles

https://www.gatsbyjs.com/docs/how-to/styling/

- CSS, SASS, LESS
- CSS-inJS
- CSS modules
- Etc.

CSS modules convention de nom

componentName.module.css

Exemple : « about.module.css »

CSS	JS
.header	styles.header
.content-message	styles.contentMessage

Exemple

```
.header {
  color: deepskyblue;
}

.content-message {
  color: #777;
}
```

```
import React from "react"
```

```
import * as styles from "./about.module.css"

export default function () {
  return (
    <div>
      <h1 className={styles.header}>About page</h1>
      <p className={styles.contentMessage}>
        Lorem Ipsum is simply dummy text of the printing and typesetting
        industry. Lorem Ipsum has been the industry standard dummy text ever
        since the 1500s, when an unknown printer
      </p>
    </div>
  )
}
```

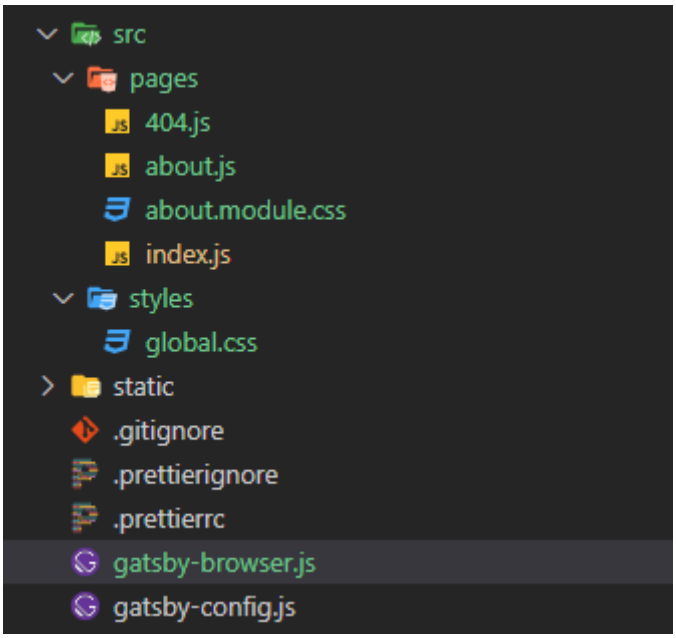
Création de style global

Création d'un dossier « src/styles » et ajout de « global.css ». Exemple

```
body {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 17px;
  line-height: 22px;
  margin: 0;
}
```

Création d'un fichier « gatsby-browser.js » à la racine du projet et import de la feuille de style

```
import "./src/styles/global.css"
```



SASS

Installation

```
npm i sass gatsby-plugin-sass
```

Ajout du plugin dans « gatsby-config.js »

```
module.exports = {
  plugins: [`gatsby-plugin-sass`],
}
```

Création d'un fichier .scss (ou.sass) et utilisation

```
import React from "react"
import * as styles from "./404.module.scss";

export default function () {
  return (
    <div>
      <h1 className={styles.header}>Page not found</h1>
    </div>
  )
}
```

Déploiement

- Gatsby Cloud https://www.gatsbyjs.com/products/cloud/hosting/
- Surge https://surge.sh/
- Netlify https://www.netlify.com/
- Github pages
- Azure
- Aws CloudFront

- Heroku
- Vercel
- Firebase

Fichier « .env » avec variables d'environnement

Création d'un **fichier** « **.env** » à la racine du projet de **frontend**.

Les **variables** avec react doivent **commencer** par « **REACT_APP_** »

Exemple avec firebase

```
REACT_APP_API_KEY="AIzaSyBtGhB2h5l3r2wLxeYLfAAuTmEkUxU9O5Y"
REACT_APP_AUTH_DOMAIN="reactsample-10246.firebaseapp.com"
REACT_DATABASE_URL="https://reactsample-10246-default-rtdb.europe-
west1.firebasedatabase.app"
REACT_APP_AUTH_PROJECT_ID="reactsample-10246"
REACT_APP_STORAGE_BUCKET="reactsample-10246.appspot.com"
REACT_APP_MESSAGING_SENDER_ID="372471858877"
REACT_APP_APP_ID="1:372461859877:web:0bbf79326df17ef29c7a54"
```

Utilisation

```
import firebase from 'firebase/app';

firebase.initializeApp({
  apiKey: process.env.REACT_APP_API_KEY,
  authDomain: process.env.REACT_APP_AUTH_DOMAIN,
  projectId: process.env.REACT_APP_AUTH_PROJECT_ID,
  storageBucket: process.env.REACT_APP_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_MESSAGING_SENDER_ID,
  appId: process.env.REACT_APP_APP_ID,
});
```

Documentation de sa libraire de composants avec Storybook

https://storybook.js.org/docs/react/get-started/introduction

Storybook permet de faire rapidement la documentation des composants d'une application. Elle ne supporte pas que react, mais aussi vue, angular, … voir même du pur javascript.

Il faut installer Storybook dans un projet ayant au minimum les dépendances dans le « package.json »

```
npx sb init
```

- Un dossier « .storybook » est ajouté à la racine du projet contenant la configuration

- Un dossier « stories » est ajouté au dossier « src » avec quelques exemples de componants.
- Des scripts sont ajoutés à package.json

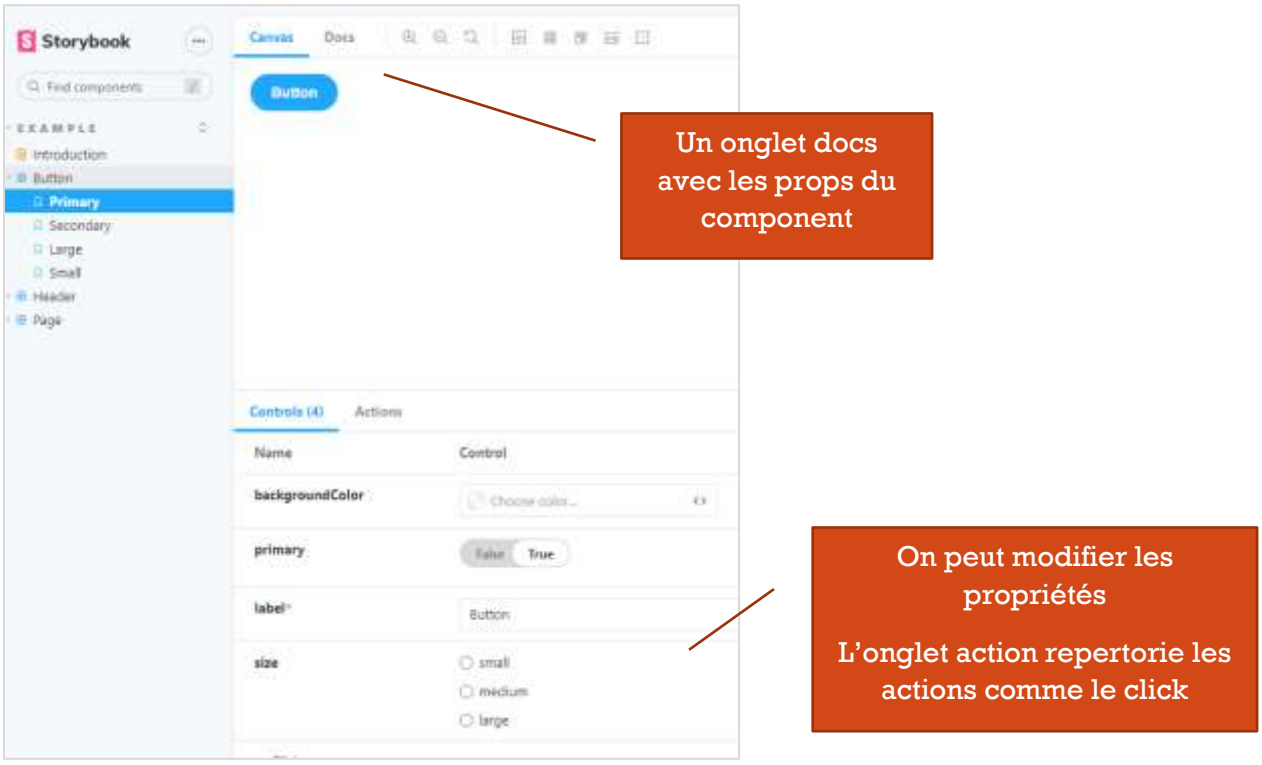
```
"storybook": "start-storybook -p 6006",
"build-storybook": "build-storybook"
```

- Il y a une page « welcome » (Introduction.stories.mdx du dossier « src/stories ») et des pages pour les composants « *.stories.tsx »

Pour lancer le storybook (supporte le hot reloading)

```
npm run storybook
```

La page s'ouvre sur : http://localhost:6006/



Ajout de component stories

Par dfaut Storybook cherche les fichiers *.stories dans le dossier « src ». On peut l'ajouter dans le dossier stories ou à côté du component qu'il documente.

.storybook > main.js > <unknown>

```
module.exports = {
  "stories": [
    "../src/**/*.stories.mdx",
    "../src/**/*.stories.@(js|jsx|ts|tsx)"
  ],
  "addons": [
    "@storybook/addon-links",
    "@storybook/addon-essentials"
  ]
}
```

Il détecte les propTypes (installer la dépendance), les defaultProps, les commentaires sur les interfaces de props, sur les components

Configuration

Preview.js : on peut par exemple importer une feuille de style commune ou définir l'onglet par défaut

```
export const parameters = {
  actions: { argTypesRegex: "^on[A-Z].*" },
  viewMode:"docs",
  controls: {
    matchers: {
      color: /(background|color)$/i,
      date: /Date$/,
    },
  },
}
```