Formation React

6H - Yoann Eichelberger

Objectifs

- Comprendre et maitriser les principes fondamentaux de React.
- Savoir créer une application React fonctionnelle.

Prérequis

- NodeJS
- Skill JavaScript + Typescript
- VSCode
- Accès à https://pokeapi.co/

Contenu

Développement d'un **Pokemon Guesser** pas à pas.

React (aka React.JS)

C'est quoi ?

- une **bibliothèque javascript**.
- créé par la multinationnal **Meta**.
- née en 2013.
- maintenu par Meta et par une communauté de développeurs.

Comment ça marche ?

JSX / TSX Kezako ?

Code JS

```
React.createElement(
  "div",
  undefined,
  React.createElement("h1", undefined, "Yoann Eichelberger"),
  React.createElement(
    "p",
    undefined,
    "Je suis ",
    React.createElement(
      "span",
      { style: { backgroundColor: "red" } },
      "développeur"
    ),
    "."
  )
);
```

Code JSX

```
<div>
  <h1>Yoann Eichelberger</h1>
  <p>
    Je suis <span style={{ backgroundColor: "red" }}>développeur</span>.
  </p>
</div>
```

Composant & éléments

Définition d'un composant

```tsx
function Fullname() {
  return <h1>Yoann Eichelberger</h1>;
}
```

```tsx
interface FullnameProps {
  firstname: string;
  lastname: string;
}

function Fullname(props: FullnameProps) {
  return (
    <h1>
      {props.firstname} {props.lastname}
    </h1>
  );
}
```

```tsx
interface MyJobIsProps {
  jobName: string;
  color?: string;
}

function MyJobIs(props: MyJobIsProps) {
  return (
    <p>
      Je suis{" "}
      <span style={{ backgroundColor: props.color }}>{props.jobName}</span>.
    </p>
  );
}

MyJobIs.defaultProps = {
  color: "red",
};
```

```jsx
function MyProfile() {
  return (
    <div>
      <h1>Yoann Eichelberger</h1>
      <p>
        Je suis <span style={{ backgroundColor: "red" }}>développeur</span>.
      </p>
    </div>
  );
}
```

```
function MyProfile() {
  return (
    <div>
      <Fullname firstname="Yoann" lastname="Eichelberger" />
      <MyJobIs jobName="développeur" />
    </div>
  );
}
```

Atelier

Initialisation du projet

Créer un projet React via Vite. (https://vitejs.dev/guide/)

```
npm create vite@latest
```

Structure du projet

- `public/` tout ce qui va être en dehors du "bundler" mais accessible.
- `src/` code source. (ts, tsx, css, ...)
- `.eslintrc.cjs` config. du linter. (eslint)
- `.gitignore`
- `index.html` index de l'application.
- `package-lock.json` fige les versions des dépendances.
- `package.json` config. centrale du projet.
- `README.md`
- `tsconfig.json` config. TypeScript utilisée pour le build.
- `tsconfig.node.json` config. TypeScript utilisée par Vite pour le run.

Démarrer le projet

1. Installer les dépendances `npm install`
2. Lancer le projet `npm run dev`

1. Préparer le jeu

Nettoyer les sources.

2. Contruire la structure HTML

A faire

1. Importer les données
    - pokemon-1.json, pokemon-2.json, pokemon-3.json : données sur les 3 pokemons.
    - pokemon-types.json : tous les types des pokemons.

2. Construire le squelette du jeu :
    - Placer un titre (ex: `My Pokemon Guesser`)
    - Afficher un des 3 pokemons au hasard (nom + sprite).
        - `pokemon.name` : nom du pokemon
        - `pokemon.sprites.front_default` : url de l'image par défaut.
    - Construire un formulaire à partir de la liste des types de pokemons.
    - Dans le formulaire, ajouter un bouton `Submit` pour valider notre choix des types et un bouton `Skip` pour passer au pokemon suivant sans validation.

3. Ajouter une gestion des interactions

A faire ensemble

- Cliquer sur un type dans le formulaire coche sa case.
- Cliquer sur `Submit` compare les types du pokemon à l'écran avec nos choix.
  - Afficher en log `true` si les types sont les bons, sinon `false`.
  - Remettre à zéro le formulaire.
  - Tenter de changer le pokemon afficher (ça ne devrait pas fonctionner !)
- Cliquer sur `Skip` pour passer au pokemon suivant.
  - Remettre à zéro le formulaire.
  - Tenter de changer le pokemon afficher (ça ne devrait pas fonctionner !)

4. Mettre à jour le jeu

```
function Counter() {
  const [count, setCount] = useState<number>(0);
  return (
    <>
      <div>{count}</div>
      <button onClick={() => setCount(count + 1)}>Add</button>
    </>
  );
}
```

- A chaque changement met à jour la vue.
- Prends un argument pour s'initialiser. (valeur ou fonction callback)
- Renvoie un tableau qui contient la valeur de l'état `count` et une fonction de mise à jour `setCount`.

A faire

1. Faire en sorte que le pokemon s'actualise à l'écran.

5. Cycle de vie

```
function AutoCounter() {
  const [count, setCount] = useState<number>(0);

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

  const decrement = () => {
    setCount((prevCount) => prevCount - 1);
  };

  // Appelé quand le Composant est monté.
  useEffect(() => {
    const id = setInterval(increment, 500);
    // Appelé quand le Composant est démonté.
    return () => clearInterval(id);
  }, []);

  return (
    <>
      <div>{count}</div>
      <button onClick={decrement}>Substract</button>
    </>
  );
}
```

A faire

1. Ajouter la dépendance `pokenode-ts` au projet.
2. Instancier le `PokemonClient` dans l'application.
3. Utiliser le `useEffect` pour récupérer un pokemon aléatoirement par id (nombre entre 1 et 150) en appelant la fonction `

TODO !