Note tuto react :

Installation :

Copié package.json et tsconfig.json

Et lancer npm i

-Dans tsconfig.json dans le include on spécie notre src ou se trouve nos code :

-danst index.tsx on définit labas ou on inject le contenue de notre App.tsx dans le index.html

State

Représente les données internes d’un composant et peut être modifié par rapport au interaction de l’utilisateur.

Exemple : const [name, setName]= useState('React');

const [pokemon, setPokemon] = useState<Pokemon|null>(null); //signifie crée un état qui peut contenir un Pokémon ou une valeur null par défaut c’est null

Cycle de vie:

Hook effet permet de les définir 1-3

Exemple :

 useEffect(() => {

    POKEMONS.forEach(pokemon => { //boucle for

      if (match.params.id === pokemon.id.toString()) {

        setPokemon(pokemon);

      }

    })

  }, [match.params.id]);// ça change à chaque fois match.params.id change

Boucle :

Boucle for n’éxiste pas dans notre jsx :

On peut utiliser : key est obligatoire on peut utiliser comme valeur de key index

<ul>

    {pokemons.map((pokemon)=>(

      <li key={pokemon.name}>{pokemon.name}</li>

    ))}

  </ul>

-recupérer simplement les donnés qu’on a besoins :

{pokemons.map(({id,name,picture,created})=>(

          <div className='col s6 m4' key={id}>

            <div className='card horizontal'>

              <div className='card-image'>

                <img src={picture} alt={name} />

              </div>

              <div className='card-stacked'>

                <div className='card-content'>

                  <p>{name}</p>

                  <p> <small>{created.toString}</small></p>

                </div>

              </div>

            </div>

          </div>

      ))}

IF aussi n’existe pas :

{age>18(vous etes majeur) :(vous etes mineur)} ;

Les props :

Il permet de découper notre application en composants enfin qu’ils communiquent entre eux

-Pour un Props nous avons besoins d’un composant parent et fils

Appeler un composants  dans le App.tsx:

 return (

<PokemonList />

 )

Helpers extensions .ts

HOOk personnalisé :

Dont son nom commence par use

Routes :

import { BrowserRouter as Router, Route,Switch,Link } from 'react-router-dom';

// Router permet de metre en place le systéme de navigation

//Switch permet de mettre d’afficher une seul route à la fois

//Route permet de décrire les routes de notre app

Exemple d’utilisation :

 return (

  <Router>

    <div>

    {/\* la barre de navigation commun à toutees les pages sera dispo dans toutes les pages parceque il ne se trouve pas dans le Switch \*/}

    <nav>

      <div className='nav-wrapper teal'>

        <Link to="/" className='brand-logo center'>Pokédex</Link>

      </div>

    </nav>

    {/\* Le systéme de gestion es routes de notre application\*/}

    <Switch>

    </Switch>

    <Route exact path='/' component={PokemonList} />

    <Route exact path='/pokemons' component={PokemonList} />

    <Route path='/pokemons/:id' component={PokemonsDetail} />

    </div>

  </Router>

 )

useHistory

Lorsqu’on clique sur un composants card on veut faire un redirection pour cela on peut utiliser useHistory qui est une hook

Importation : import { useHistory } from 'react-router-dom';

Exemple d’utilisation

const history=useHistory();

const goToPokemon=(id:number)=>{

    history.push(`/pokemons/${id}`) //permet de faire une redirection

   }

return (

<div className="col s6 m4" onMouseEnter={showBorder} onMouseLeave={hideBorder} onClick={()=>goToPokemon(pokemon.id)}>

Route not found 404:

<Route component={PageNotFound} /> //on le met en derniére

Recupération donné dans la route :

FunctionComponent<RouteComponentProps<Params>> = ({ match }) => {

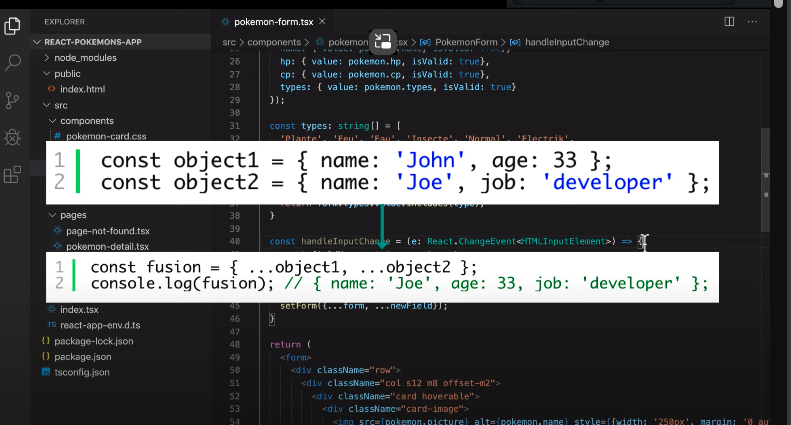
Const id =match.params.id

}

Formulaire :

Regarder dans le commit formulaire avant sauvegarde edition

Remarque : On peut fusionner deux objects avec react



Validation checkbox entre 1 et 3 :

const hasType=(type:string):boolean=>{

    return form.types.value.includes(type);

  }

  const isTypesValid=(type:string):boolean=>{

    if (form.types.value.length===1 && hasType(type)){

      return false;

    }

    if(form.types.value.length >=3 && !hasType(type)){

         return false;

    }

    return true;

  }

-html

 <input id={type} type="checkbox" className="filled-in" value={type} checked={hasType(type)}   onChange={e => selectType(type, e)} disabled={!isTypesValid(type)}></input>

Expression régulier voir dans le commit validation checkbox entre 1 et 3 choix dans pokemon-form dans component

Exemple :

    if(!/^[0-9]{1,3}$/.test(form.hp.value)) {

      const errorMsg: string = 'Les points de vie du pokémon sont compris entre 0 et 999.';

      const newField: Field = {value: form.hp.value, error: errorMsg, isValid: false};

      newForm = { ...newForm, ...{ hp: newField } };

    } else {

      const newField: Field = { value: form.hp.value, error: '', isValid: true };

      newForm = { ...newForm, ...{ hp: newField } };

    }

Message d’erreur formulaire:

 {form.name.error && <div className='card-panel red accent-1'>

                    {form.name.error}

                     </div>

                     }

Attention: le form c’est un **useState** qui est modifié dans la fonction validateForm qui ajoute un champs erreur en cas d’erreur

Utilisation d’une api rest json server :

npm install -g json-server :

-on crée dans model db.json

-on allume le server avec json-server --watch src/models/db.json --port=3001

Requête http avec fetchApi

useEffect(() => {

    fetch('http://localhost:3001/pokemons')

    .then(response=>response.json())

    .then((pokemons)=>{

      setPokemons(pokemons)

    });

  }, []);

NB : on a pas besoin d’installer une librairie fetch c’est une méthode js native

SERVICES :

Nb :l’ extension d’une service est .ts ,d’un helpers et d’un model