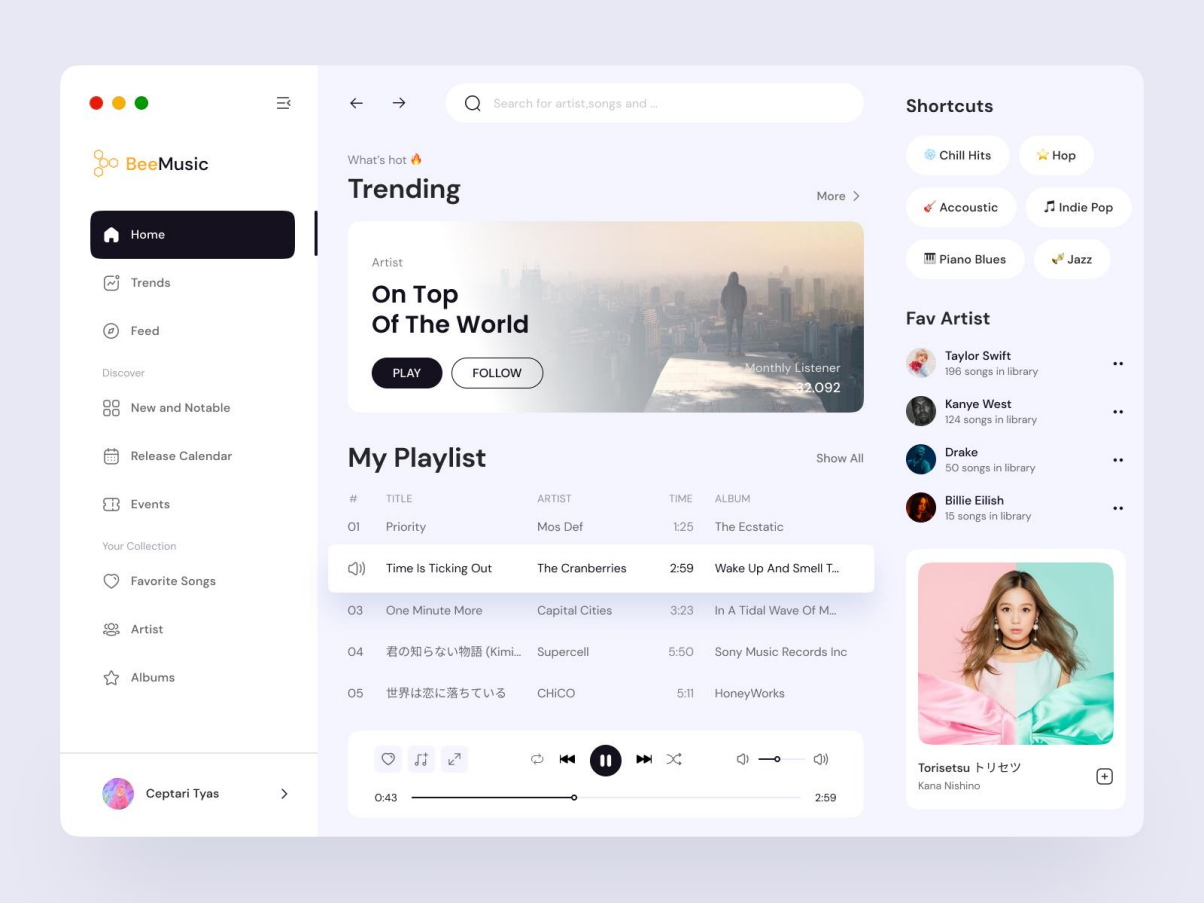
DOCUMENTATION PERSONNEL

Ce mini projet à été réalisé avec NextJs, un framework ReactJS, Tailwindcss et MaterialUi.

Le but était de développer une application de musique, nous nous sommes inspirés d'une maquette Dribbble (Dribbble étant un site où toutes sortent de maquette y sont répertorié).



Lien de la maquette : https://dribbble.com/shots/16618273/attachments/11648035?mode=media

Ayant utilisé ReactJs je n'ai pas eu de mal a me familiariser avec NextJs, nous retrouvons presque les mêmes logique et mêmes fonctions (useState,useEffect…).

Cependant l'architecture est différente, ReactJs permet une énorme flexibilité sur la manière dont nous mettons en place nos Routes.

Ex

```
<ProtectedRoute exact path="/" component={Home} />
<ProtectedRoute exact path="/lives" component={Home} />
<ProtectedRoute exact path="/stats" component={Home} />
<ProtectedRoute exact path="/forecasts" component={Home} />
<ProtectedRoute exact path="/analitics" component={Home} />
<ProtectedRoute exact path="/podcasts" component={Home} />
<ProtectedRoute exact path="/podcasts/:cate" component={Home} />
<ProtectedRoute exact path="/blog/post/:id/:titre" component={Home} />
<ProtectedRoute exact path="/me/articles" component={Home} />
<ProtectedRoute exact path="/me/articles/:status" component={Home} />
```

Figure 1 Fichier Router React Js

Contrairement a React, Next utilise les fichiers qui sont dans le dossier /pages, et les créer en route. Personnellement je n'aime pas trop cette méthode, en développant notre application nous nous sommes rendus-compte qu'il n'était pas possible d'avoir une page avec un composant drawer qui englobe la vue main et une autre page sans le composant drawer, nous avons donc utilisé des conditions.

Ex

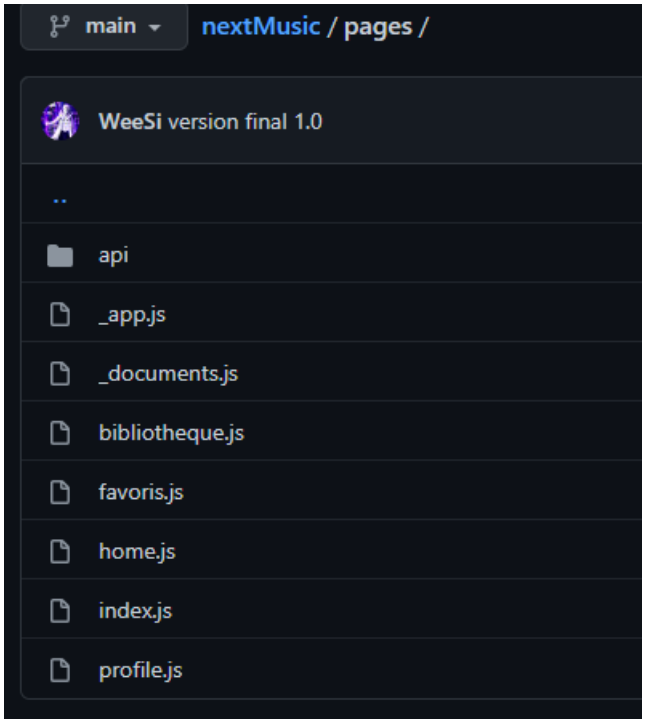


Figure 2 NextJs Pages Router

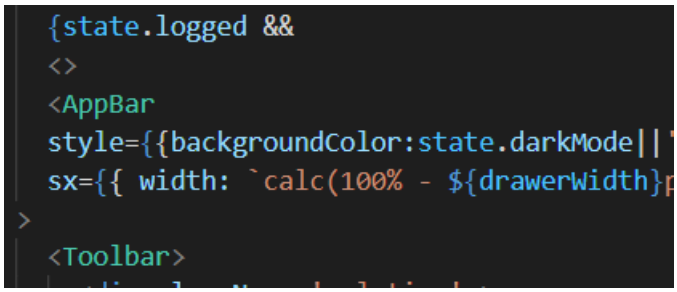


Figure 3 NextJs Condition sur l'affichage du drawer en fonction de l'etat logged

Next Js est aussi un framework permettant de faire du back-end et du SSR (server-side-rendering), nous ne l'avons pas utilisé pour ce projet mais pour certaine personne ne voulant pas utiliser plusieurs technologie cette fonctionnalité peut être bénéfique car il n'y a qu'un seul langage a utiliser.

Je trouve aussi que le serveur met du temps à charger (peut-être que c'est ma machine), typiquement quand nous voulons accéder a une page qui n'a pas été ouverte, NextJs met au moins 5second à afficher cette page, puis là garde dans le cache et les prochaines fois l'ouverture se fait plus rapidement.

Pour finir, j'ai appris à utiliser le React Context (Permet de stocker des données et des fonctions globalement dans la mémoire du navigateur, donc possibilité d'utiliser ces données et ces fonctions

de partout dans notre code), c'est exactement le même fonctionnement que Redux, mais moins performant car il est assez récent.

```
import { useReducer, createContext } from "react";
import { global } from "./reducers/global.reducers";

// initial state
const initialState = {
  user: {},
  logged:false,
  darkMode: false,
  inPlay: {},
  volume: 0.9,
  index: 0,
  isPlaying: false,
  favoris:[]
};

// create context
const Context = createContext({});

// combine reducer function
const combineReducers = (...reducers) => (state, action) => {
  for (let i = 0; i < reducers.length; i++) state = reducers[i](state, action);
  return state;
};

// context provider
const Provider = ({ children }) => {
  const [state, dispatch] = useReducer(combineReducers(global), initialState);
  const value = { state, dispatch };

  return <Context.Provider value={value}>{children}</Context.Provider>;
};

export { Context, Provider };
```

Figure 4 Fichier création du provider, state, et du Context

```
export function global(state, action) {
    switch (action.type) {
      case "LOGGED_IN_USER":
        return { ...state, user: action.payload, logged:true };
      case "CHANGE_DARK":
        return { ...state, darkMode: action.payload };

      case "CHANGE_VOLUME":
        return { ...state, volume: action.payload};

      case "CHANGE_SONG":
        return { ...state, inPlay: action.payload};

        case "ADD_TO_FAV":
          return { ...state, favoris: [...state.favoris, action.payload]};

        case "REMOVE_FROM_FAV":
          return { ...state, favoris: state.favoris.filter((el) => el.id != action.payload)};
      default:
        return state;
    }
  }
```

Figure 6 Fichier de mise à jour des state selon le type de l'action

```
return (
 <CacheProvider value={emotionCache}>
    <CssBaseline />
    <Provider>
      <MiniDrawer>
          <Component {...pageProps} />
      </MiniDrawer>
    </Provider>
 </CacheProvider>
```

Figure 5 Notre app qui wrapper par le Provider