Documentation Next-Music

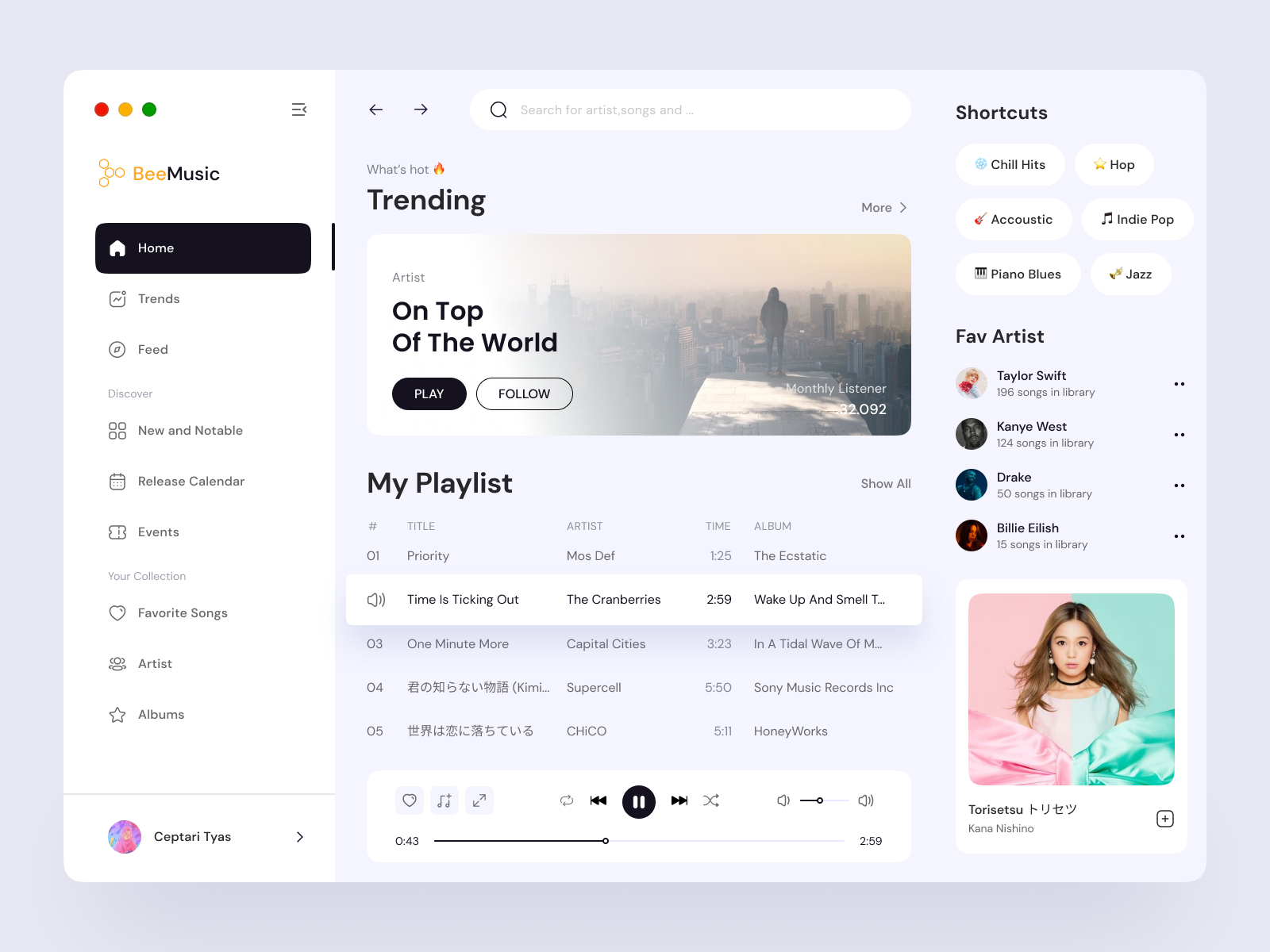
Ce projet réalisé avec NEXTJS, un Framework REACTJS.

Lien : <https://next-music-isitech.herokuapp.com/>

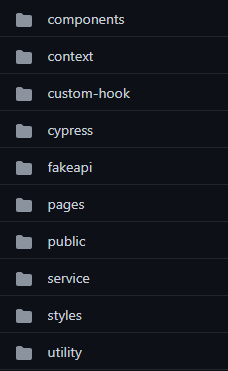
Utilisation de Tailwindcss et MaterialUI deux bibliotheques de composant différente.

Le projet consistait à la création d’une application web musical, a la façon de Spotify. Cette application permet l’écoute de musique et l’ajout de titre en favoris.

Pour ce faire nous avons utilisé une maquette Dribbble (Dribbble étant un site où toutes sortent de maquette y sont répertorié).

<https://dribbble.com/shots/16618273/attachments/11648035?mode=media>

Pour n’avoir aucune complication pour notre environnement de travail, nous avons créer plusieurs dossiers.



* Le dossier components correspond aux composants réutilisables que nous créons
* Le dossier Context, contient plusieurs fichiers consistant à stocker des données de l’application pendant la session de l’utilisateur.
* Le dossier Cypress pour nos tests unitaires
* Le dossier Fakeapi pour stocker des données en dur
* Le dossier pages pour créer nos pages sur l’application. Un fichier sera converti directement en page avec NextJs
* Le dossier public qui sert à stocker des images, des sons, des assets...
* Styles pour les fichiers CSS de l’app
* Et utility pour des fonctions réutilisable

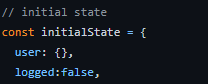
Notre application est composée de 4 pages principal

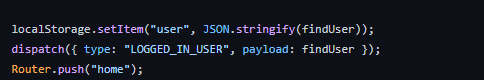
* Page de connexion
* Page d’accueil
* La bibliothèque de musique
* La page de favoris

Si l’utilisateur n’est pas connecté il naviguera vers la page de login sinon sur la page d’accueil.

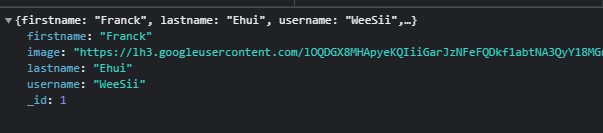
**Comment ça marche ?**

Lorsqu’un utilisateur veut se connecter nous enregistrons ces données non sensibles dans le localstorage et dans le Context avec le Hook useReducer de ReactJs. Une fois ces données enregistrées dans le Context, nous pouvons les réutiliser de partout, dans n’importe quel composant.





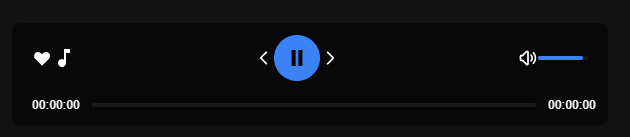




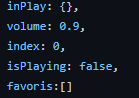
Grâce à tout ça nous savons si l’utilisateur est connecté ou pas.

**Comment fonctionne l’écoute d’une musique ?**

L’écoute d’un son fonctionne de la même manière que la connexion de l’utilisateur. Seulement le composant est un peu plus complexe.



Nous avons ce composant Player que nous affichons tout le temps peu importe la page. Comme le login il utilise aussi le Context.



Nous avons des états initiaux, puis quand on clique sur le bouton Play nous mettons a jours l’état avec la fonction dispatch venant de Context.

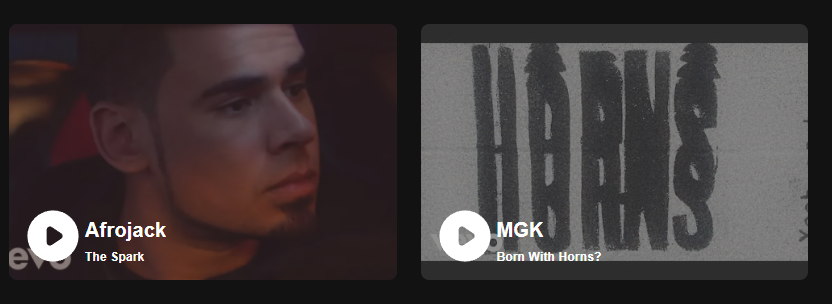


S'il n’y a pas de son il prend le premier de la liste.

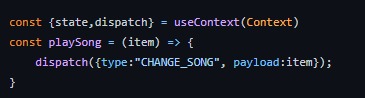


Grace au Context nous pouvons démarrer de la musique depuis n’importe quelle page. C’est le cas des deux pages suivantes (Bibliotheque et Favoris).

Pour la page Bibliotheque nous avons juste repris les sons stocker dans notre projets.



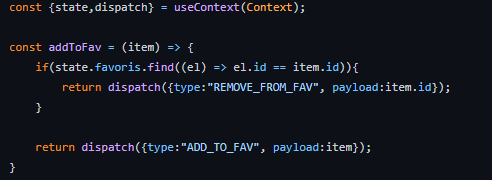
En cliquant sur une carte il se passe le même procédé que lorsque l’on clique sur le bouton Play.



Un évènement est envoyé avec l’objet en tant que payload et le state est mis à jour avec les nouvelles données reçu et donc un nouveau son commence.

**Mais comment fait-on pour ajouter des sons sur la page Favoris ?**

Encore une fois, tout simplement grâce au Context en appelant un autre type d’évènement.

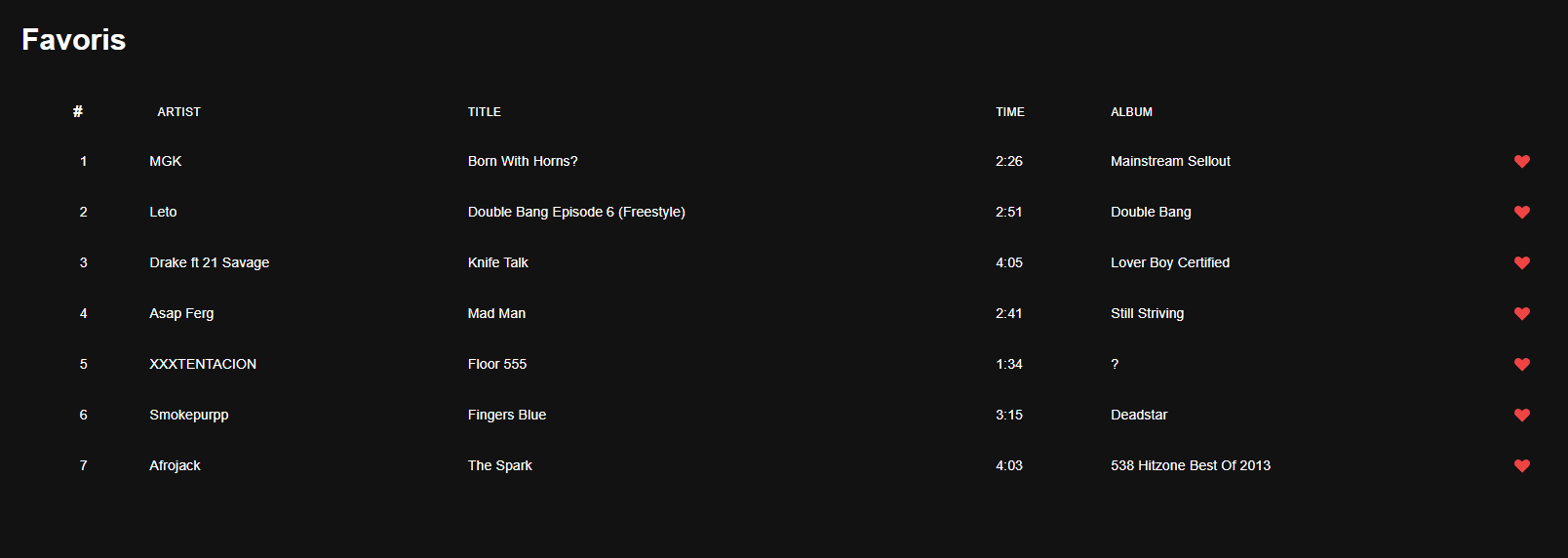


Tout d’abord il faut s’assurer que le son n’existe pas dans notre page favori, si c’est le cas on l’ajoute sinon on le supprime. Donc nous avons deux évènements gérant cette fonctionnalité “REMOVE\_FROM\_FAV” qui va juste enlever l’objet que l’on veut enlever.



Et l’évènement “ADD\_TO\_FAV” qui va juste pousser l’objet dans notre état “favoris” en dernière position.





**Recherche de musique**

Pour la recherche nous avons utilisé l’API Spotify, grâce à la fonction fetch de javascript nous somme capable de faire des appels http, puis récupérer les informations retourner en JSON et les afficher. Nous utilisons une fonctionnalité nommée debounce pour éviter de faire des appels API à chaque fois que l’utilisateur tape au clavier. On utilise aussi le Hook useCallback de react, qui va nous renvoyer une version mémoïsée (mis en cache) de la fonction appelée.

