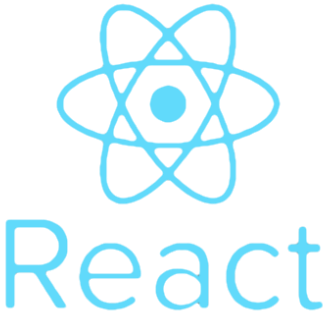
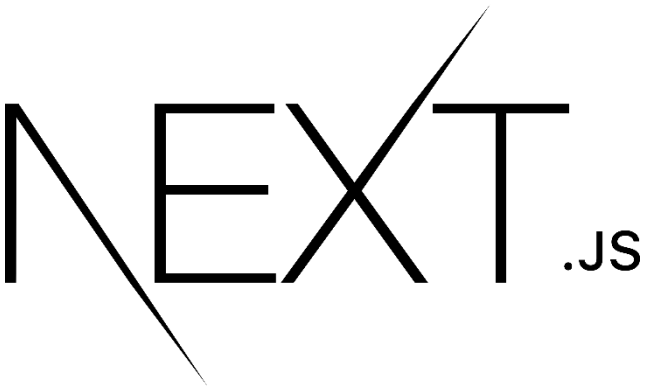




RAPPORT ARCHITECTURE LOGICIEL

FRONT-END WEB

TCHANTEU Thierry

FEREKH Khaled

BEN AMMI Achrafe

BUKARSA Zakaria

Equipe Frontend web

Nous avons décidé d'implémenter notre projet en utilisant le Framework React, donc du JavaScript (ou TypeScript). Ce choix a été pour nous une évidence car sa documentation et l'apport de la communauté est très grande. React est très utilisé pour la création de tel projet. Nous avons créé notre projet React sur l'environnement NodeJS. Il est important pour la reprise du projet d'installer NodeJS (npm version utilisée : 8.3.0). De plus, à l'aide de NodeJS, il est possible d'avoir la même configuration que notre projet (les outils utilisés, les libraires…) (grâce au package.json). Il est possible de lire sur le readme, les étapes nécessaires pour pouvoir reprendre le projet.

Lien du GitHub : https://github.com/Web-Architecture-Project/Frontend-Web

Avancement général :

Nous avons seulement travaillé sur la partie entraide des étudiants. L'utilisateur peut créer, commenter un post, aimer, ne pas aimer, voir les trending threads, filtrer par catégorie, etc.

Nous avons connecté pour l'instant notre api pour pouvoir tester notre blog, par la suite, il faudrait connecter le site web aux API créées par les autres groupes. L'API que nous avons utilisée est FastAPI, il s'agit d'un Framework écrit en Python.

Pour l'API "entraide étudiant" nous avons déjà créé les différentes fonctions qui nous permettent d'effectuer toutes les requêtes demandées. Il ne nous reste plus qu'à les implémenter.

La structure de base est là, il suffira juste de créer les composants restants et de les intégrer dans les pages adéquats. Vous pouvez suivre notre technique de mise en place des composants qui est décrite dans les pages suivantes de ce rapport.

Hand over :

Comme énoncé précédemment, on utilise du React et plus particulièrement du NextJS v12 qui est un Framework React avec la particularité d'être prêt au déploiement (« gratuit ») grâce à son créateur Vercel « Next.js gives you the best developer experience with all the features you need for production: hybrid static & server rendering, TypeScript support, smart bundling, route pre-fetching, and more. No config needed ». (https://nextjs.org/) React repose sur le système de composant, chaque composant a un état qui peut changer et chaque composant peut contenir d'autres composants. Vous pouvez voir ça comme un système d'arbre de composants.

Les composants principaux sont repris ici, beaucoup d'autres petits composants qui sont importé de la librairie antd (comme Button, input, select, layout, …) sont utilisés partout dans le projet. Et c'est ça qui fait la puissance de React, beaucoup de librairie sont déjà présentes avec des composants prêts à l'emploi, vous n'aurez qu'à faire npm install nom_librairie et puis import des composants dans votre projet et les personnaliser comme vous voulez.

```
import { Button, Input, Carousel } from "antd";
import { useState, useEffect } from "react"
import { PlusOutlined, UnorderedListOutlined } from "@ant-design/icons"
```

Un composant **doit** retourner quelque chose et **doit** être appelé comme cela :

```
<Button type="primary" htmlType="submit" >Confirm</Button>
```

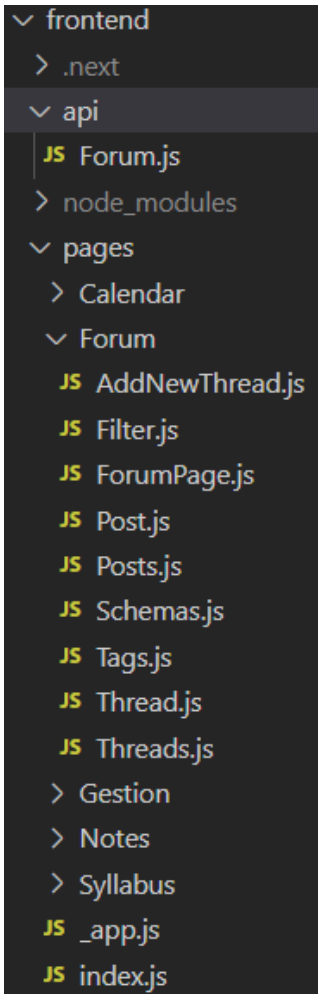
Cela vous rappel surement les tags de HTML, c'est le même concept. Et avec React vous pouvez combiner de l'HTML avec du JS :

```
<div style={{ marginLeft: "48px", marginBottom: "10px" }}>
  {post?.content}
</div>
{showAnswers ? <Posts firstPostId={post?.id} /> : ""}
```

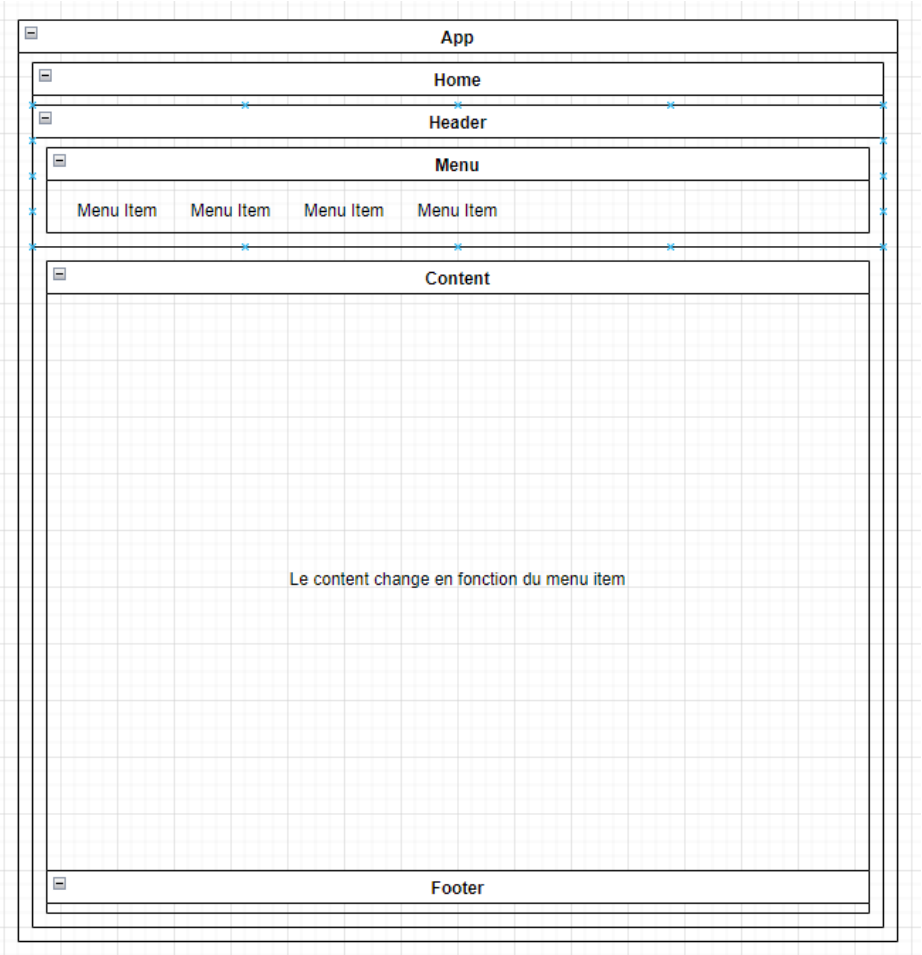
Vous pouvez vous-même créer votre propre composant et le personnaliser puis l'importer dans votre page mais il faudra faire attention aux règles à suivre pour un composant. Ces règles vont aider à diminuer les bugs mais vu que c'est du JavaScript, qui est un langage dynamique, des bugs et des erreurs vous allez en avoir surement. Cependant passer à du TypeScript peut être une bonne solution pour éviter au maximum les erreurs de typage .

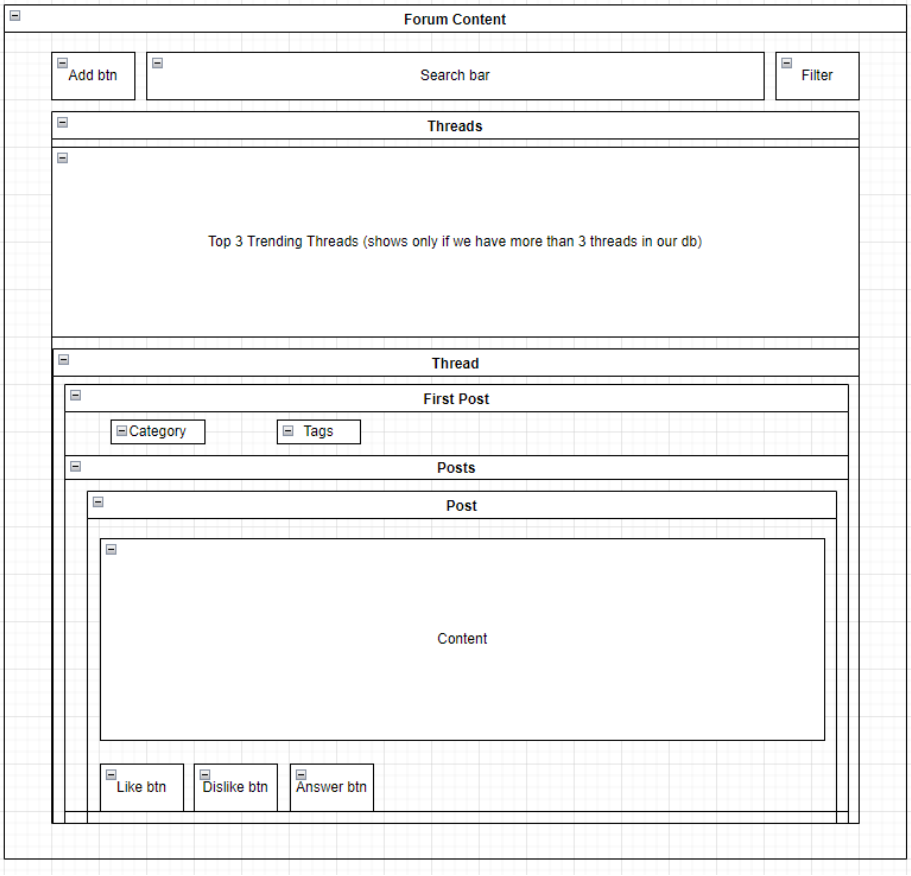
Pour plus d'informations vous pouvez consulter le site de React qui contient toutes les informations nécessaires : https://fr.reactjs.org/docs/getting-started.html

La structure actuelle de notre projet est la suivante :



Plus d'infos ici : https://github.com/Web-Architecture-Project/Frontend-Web


App
Home
Header
Menu
Menu Item
Menu Item
Menu Item
Menu Item
Content
Le content change en fonction du menu item
Footer

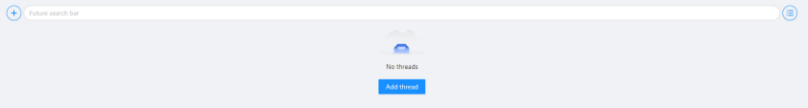

Forum Content
Add btn
Search bar
Filter
Threads
Top 3 Trending Threads (shows only if we have more than 3 threads in our db)
Thread
First Post
Category
Tags
Posts
Post
Content
Like btn
Dislike btn
Answer btn

Comment ça marche ?

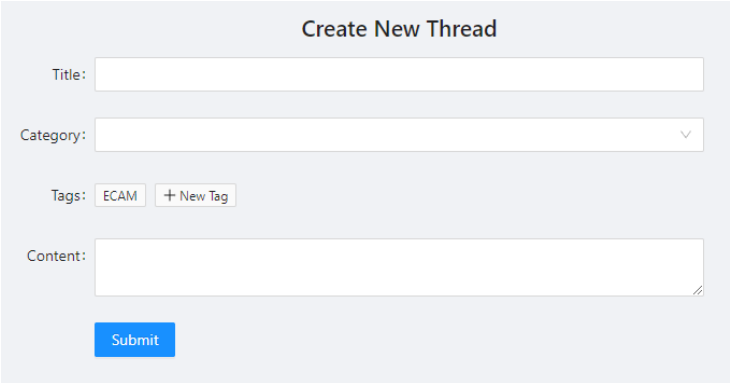
A l'arriver sur la page forum, le composant <Threads /> est appelé qui lui contient une fonction nommée GetAllThreads (importé de./api/forum.js) fait une requête à l'api et nous renvois une liste de tous les threads.

```
export async function GetAllThreads() {
  try {
    const result = await axios.get(`http://localhost:8000/threads`)
    return result.data
  } catch (error) {
    console.error(error);
  }
}
```

Si aucun thread le composant Empty de antd nous propose cela :



On pourra ensuite cliquer sur Add thread qui lui va faire appel à notre composant <AddNewThread/> qui return cela :



Après avoir compléter les cases en haut, on clique sur submit et ça nous crée un thread grâce à la fonction CreateThread(thread) qui envoi un post à l'API ou thread est fait à l'aide d'un schéma de thread pour éviter les bugs lors du post vers l'api :

```
const ThreadSchema = (props) => {
  return {
    title: props.title,
    first_post: props.first_post,
    tags: props.tags,
    category: props.category,
    answered: props.answered ? props.answered : false,
  };
};
```

```
export async function CreateThread(thread) {
  try {
    const result = await axios.post(`http://localhost:8000/threads`, thread)
    return result.data
  } catch (error) {
    console.error(error);
  }
}
```

GetAllThread() est rappélé et maintenant qu'on a des threads dans la liste, si la longueur de la liste est > 3, on affichera le top 3 des threads. On affiche ensuite les thread grâce à notre composant <Thread react et threadId /> avec les paramètres (react et threadId).

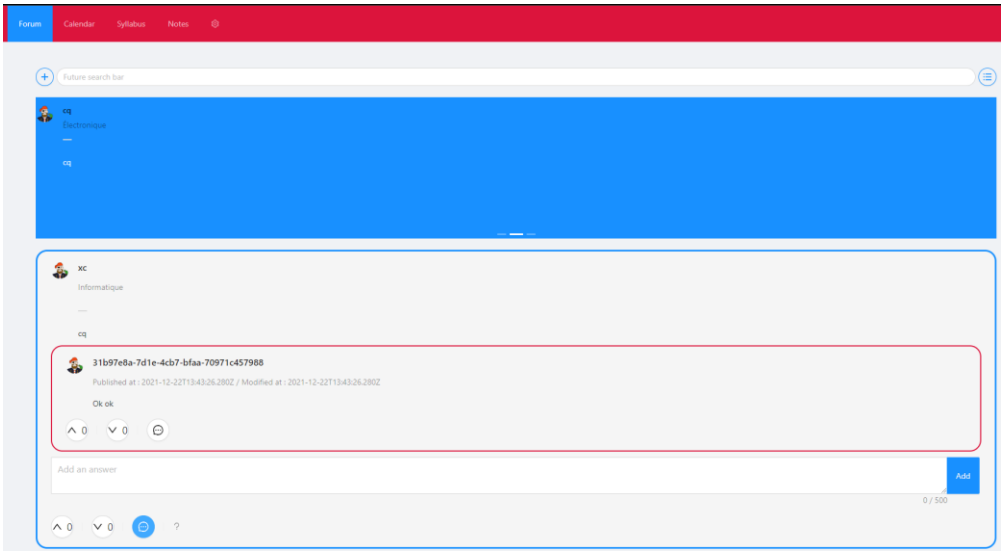
```
return (threads.length > 0 ?
    <>
        {threads.length > 3 ? <TrendingThreads /> : ""}
        <List
            itemLayout="vertical"
            size="large"
            pagination={{
                onChange: page => {
                    console.log(page);
                },
                pageSize: 3,
            }}
            dataSource={threads}

            renderItem={item => (
                <Thread react threadId={item.id} />
            )}
        /> </> :
    <Empty
```

Le composant <Thread/> quant a lui fait appel à la fonction GetThread() pour obtenir les informations de ce thread particulier (grâce au threadId)

```
export async function GetThread(threadId) {
  try {
    const result = await axios.get(`http://localhost:8000/thread/${threadId}`)
    return result.data
  } catch (error) {
    console.error(error);
  }
}
```

Thread lui fait appel au composant <Posts firstPostId /> qui lui affiche le first post du thread et les enfants du post grâce au composant <Post postId/> (GetPost(postId) nous renvoie les infos du post) pour finir avec un résultat comme celui-ci :



Cette structure est récursive un post peut avoir des posts qui peuvent avoir des posts etc.
Note : Beaucoup de détails ont été épargné, mais la logique reste la même partout.

Difficulté :

Pour la majorité du groupe, nous avons mis un temps important à se former sur le Framework et le langage javascript qui n'est pas facile à prendre en main. Sans réel connaissance, on ne savait pas la gestion du temps pour aboutir à l'ajout d'un service. De plus, nous n'avons pas réellement utilisé de méthode de travail ce qui a rendu compliquer la communication et l'avancement du travail. En effet, les trois premières semaines, nous avions chacun assigné une personne à une implémentation à un service, ce qui était beaucoup de travail pour une personne. C'est pourquoi, par la suite, nous nous sommes principalement concentrés à la partie *entraide des étudiants* car leur api était très bien documentée.

Un autre problème était de travailler avec les autres groupes. En travaillant dans le front end , nous ne savions pas à quoi nous attendre en termes d'implémentation. Les besoins des utilisateurs étaient flous pour nous. En effet, les autres groupes étaient en réflexion, de penser à la structure de leur api et ne pouvaient pas vraiment nous aiguiller au commencement du projet.

Amélioration :

Premièrement, on vous propose de passer à du TypeScript pour limiter les erreurs de typage. Ensuite, il est possible de travailler en classe ou en fonction. Dans notre cas, nous avons préféré travailler avec des fonctions. Cependant, vous pouvez tout à fait travailler à l'aide de classe si vous êtes plus à l'aise avec, on peut même combiner les 2 techniques.

Utilisé un container comme Redux, Appolo / GraphQL sera nécessaire pour stocker des variables/fonctions globales mais aussi pour vérifier les états (connecté, …) et autres choses.

Nous pensons également par la suite au groupe de directement s'intéresser à une fonctionnalité, un service et de communiquer avec le groupe *frontend Android* pour implémenter des services qui n'ont pas été réalisé.

Pour finir, nous recommandons vivement de travailler de façon agile car nous avancions totalement dans le noir, les taches que nous nous fixions étaient trop grandes ou pas respectées. Il est très difficile d'avancer et de travailler sur un tel projet sans réel méthode d'approche. Nous encourageons aussi la communication entre étudiant du même groupe lorsqu'une tache est mal comprise.