

NOLLE Damien
2^{ème} Année BTS SIO (Option : SLAM)
Etablissement Saint-Adjutor (Vernon)
Promotion : 2021 – 2022

RAPPORT DE STAGE



Entreprise d'accueil : Altameos Multimedia

Période de stage : Du 9 Janvier au 17 Février 2023

Tuteur de stage : Monsieur SCHNEKENBURGER

Enseignant référent et de visite : Monsieur GAULIER

Remerciements :

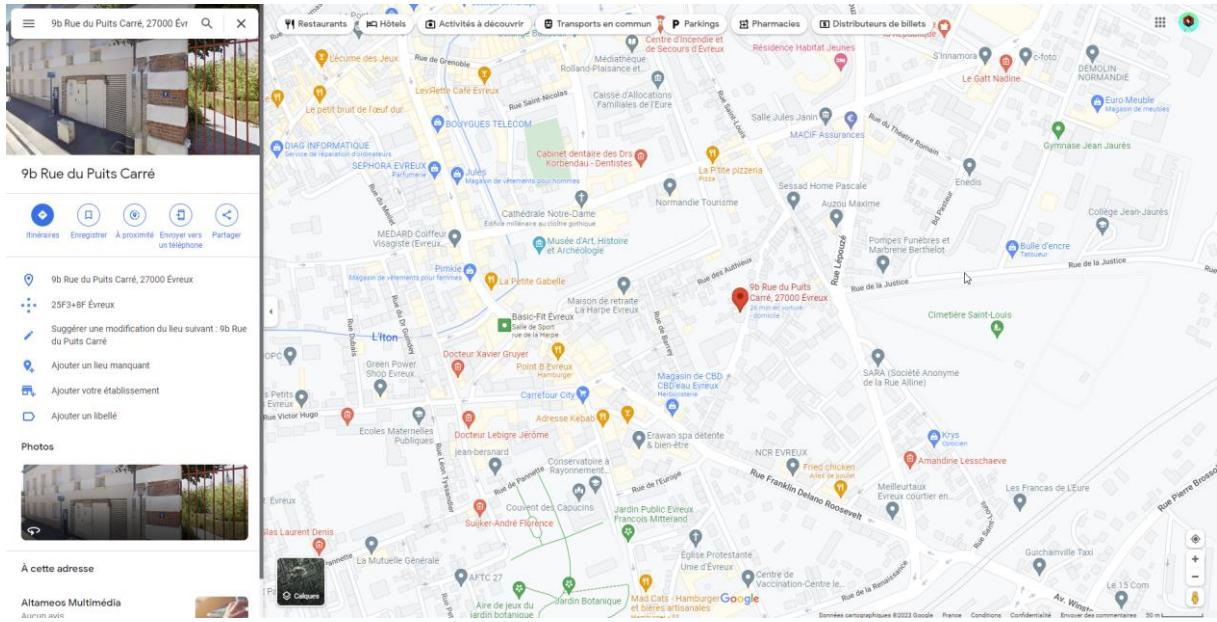
Je tiens à remercier Monsieur SCHNEKENBURGER d'Altameos Multimedia de m'avoir accueilli et fait confiance pour le passage du site d'e-commerce sous Angular ainsi que d'avoir été mon tuteur de stage et d'avoir eu le temps et la patience de m'accompagner tout le long du stage.

Je tiens aussi naturellement à remercier toute l'équipe pédagogique du BTS SIO de Saint-Adjutor à Vernon de m'avoir accueilli pour cette deuxième année et de m'avoir apporté les compétences et les connaissances nécessaires à la réalisation de ce stage.

Présentation de l'entreprise :

Altameos Multimedia est une entreprise de développement Web proposant ses services aux entreprises souhaitant concevoir leur site Web afin d'améliorer leurs visibilités et leurs notoriétés sur internet.

Fondé par Monsieur SCHNEKENBURGER en 2003, l'entreprise se situe à Evreux (27000) au 9b Rue du Puits Carré.

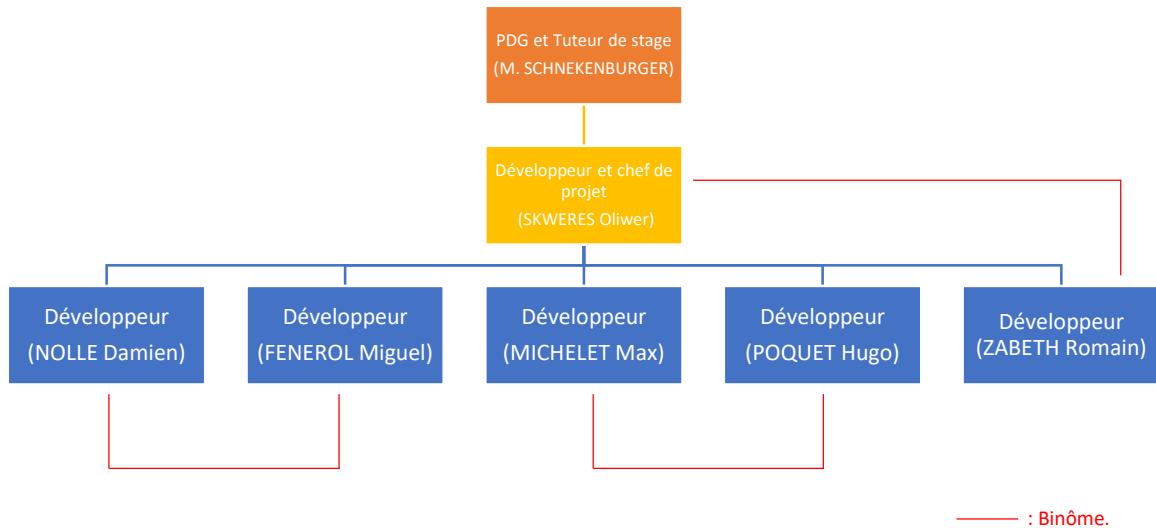


L'entreprise est composée uniquement de Monsieur SCHNEKENBURGER car il travaille en tant qu'indépendant libéral.



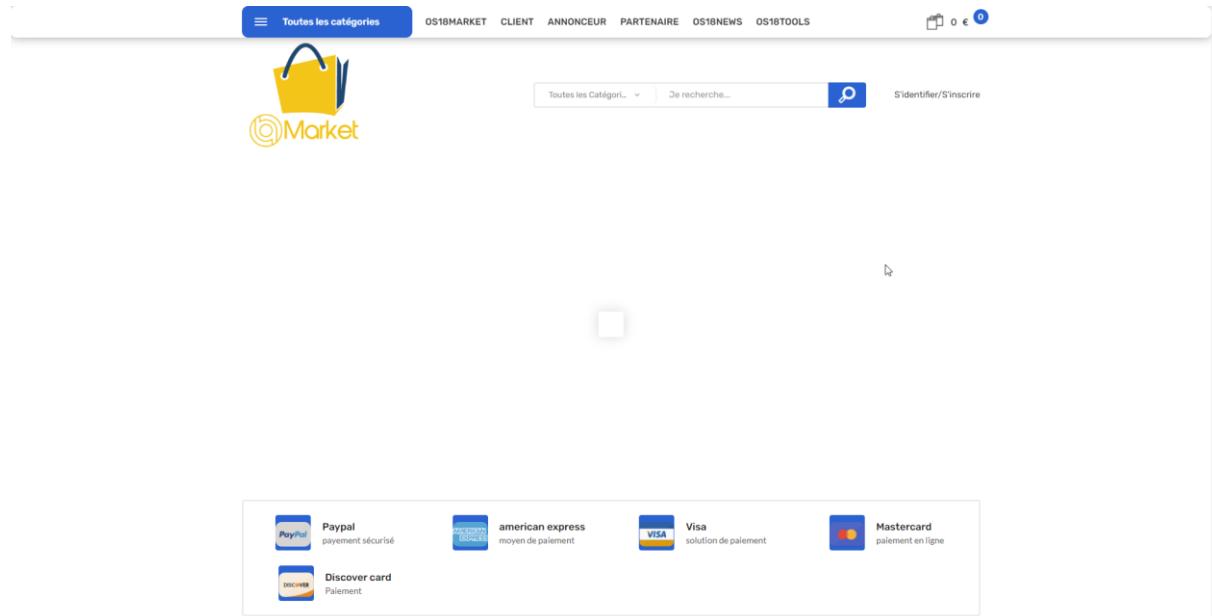
Nous étions une équipe de 6 étudiants venant du même BTS SIO (Saint-Adjutor à Vernon) sur le même projet : Hugo POQUET, Max MICHELET, Miguel FENEROL, Romain ZABETH, Oliwer SKWERES et moi. Nous étions tous en tant que développeurs hormis Oliwer SKWERES qui était en plus chef de projet.

Organigramme de Altameos Multimedia et de notre équipe :



Présentation du service d'accueil et des moyens informatiques :

Le stage s'est principalement réalisé en télétravail, les moyens mis en place sont donc nos propres moyens à disposition dont nos propres postes informatiques. Nous avions un site déjà existant puisque l'objectif était de le reporter en Angular. Il était prévu qu'une réunion se fasse le matin pour parler de l'organisation de la journée avec notre tuteur de stage et une le soir pour faire un compte rendu de la journée.



Nous avions aussi accès à deux documents (un Dashboard et une fiche UX) permettant de connaître les besoins du client pour le site d'e-commerce.

dashboard-eCommerce.odt

| | C | D | E | F | G |
|----|--|--|--------------------|-------------|-------------|
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |
| 7 | FRONT | UX | Spécificité | UX.1 | UX.2 |
| 8 | | | | | |
| 9 | Mise en place structure Angular de base | Gestion des administrateurs | | UX.1 | |
| 10 | | Gestion des utilisateurs | | UX.2 | |
| 11 | | Gestion des annonceurs | | UX.3 | |
| 12 | | Gestion des partenaires | | UX.4 | |
| 13 | Sécurisation et accès sécurisé | | | | |
| 14 | | Gestion des catégories | | | |
| 15 | | Gestion des articles | | | |
| 16 | | Gestion des produits | | | |
| 17 | | Gestion des offres pour annonceurs / packs | | | |
| 18 | | Gestion des commandes | | | |
| 19 | | Gestion des informations de la boutique | | | |
| 20 | | Gestion des partenaires publicitaires | | | |
| 21 | | Gestion du carrousel de la page d'accueil | | | |
| 22 | | Gestion de pages annexes (mentions / crédits / CGV...) | | | |
| 23 | | Gestion des erreurs de commande | | | |
| 24 | | Gestion de la commande | | | |
| 25 | | Gestion des erreurs de commande | | | |
| 26 | | Gestion des erreurs commandes | | | |
| 27 | | Gestion Satisfaction | | | |
| 28 | | Gestion Actualités | | | |
| 29 | | Gestion Faits en ligne | | | |
| 30 | | | | | |
| 31 | Entité | | | | |
| 32 | | | | | |
| 33 | Recherche produits | | | | |
| 34 | Inscription Client | | | | |
| 35 | Gestion Client | | | | |
| 36 | Gestion Annonceur | | | | |
| 37 | Identification Annonceur | | | | |
| 38 | Gestion Annonces | | | | |
| 39 | Gestion Annonces - Produits | | | | |
| 40 | Gestion Annonces - Produits | | | | |
| 41 | Gestion Annonces - Packets choisis | | | | |
| 42 | | | | | |
| 43 | Identification Partenaires | | | | |
| 44 | Gestion | | | | |
| 45 | Gestion Processus commande | | | | |
| 46 | Gestion Page Catalogue | | | | |
| 47 | Gestion Page Accueil | | | | |
| 48 | Gestion Page Accueil | | | | |
| 49 | Gestion Actualités et détails | | | | |
| 50 | Gestion Témoignages et détails | | | | |
| 51 | Gestion Contact | | | | |
| 52 | Gestion Produits les plus vendus | | | | |
| 53 | Gestion Produits moins vendus | | | | |
| 54 | Gestion Produits les plus vus | | | | |
| 55 | Gestion Produits moins vus | | | | |
| 56 | Gestion Erreurs Commande / Livraison / reclam. | | | | |
| 57 | Initialisation au service Test | | | | |

Feuille 1 sur 3 + Sheet1 Sheet2 Sheet3 Par défaut Français (France) Moyenne : Somme : 0 0 0 0

UX-eCommerce.odt

| | A | B | C | D | E | F | G | H | I | J |
|----|--|---------------|------------------|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | |
| 2 | Gestion Partenaires | | | | | | | | | |
| 3 | Gestion des fiches partenaires | | | | | | | | | |
| 4 | + Gestion des fiches partenaires | | | | | | | | | |
| 5 | | | | | | | | | | |
| 6 | | | | | | | | | | |
| 7 | | | | | | | | | | |
| 8 | A faire | Ajout | | | | | | | | |
| 9 | | Modification | | | | | | | | |
| 10 | | Suppression | | | | | | | | |
| 11 | | | | | | | | | | |
| 12 | Inscription avec code de confirmation | | | | | | | | | |
| 13 | 1 email par inscription | | | | | | | | | |
| 14 | Complexité mot de passe . Majuscule Minuscule Chiffre et caractères spéciaux @ ! ? - _ | | | | | | | | | |
| 15 | Table Part | | | | | | | | | |
| 16 | idPartenaire | NomPartenaire | | | | | | | | |
| 17 | | NonPartenaire | | | | | | | | |
| 18 | | Prénom | | | | | | | | |
| 19 | | 20 | ModifDePasse | | | | | | | |
| 20 | | 21 | IlUu | | | | | | | |
| 21 | | 22 | Adrl | | | | | | | |
| 22 | | 23 | Email | | | | | | | |
| 23 | | 24 | Tel (facultatif) | | | | | | | |
| 24 | | 25 | token | | | | | | | |
| 25 | | | | | | | | | | |
| 26 | | | | | | | | | | |
| 27 | | | | | | | | | | |
| 28 | | | | | | | | | | |
| 29 | | | | | | | | | | |
| 30 | | | | | | | | | | |
| 31 | | | | | | | | | | |
| 32 | | | | | | | | | | |
| 33 | | | | | | | | | | |
| 34 | | | | | | | | | | |
| 35 | | | | | | | | | | |
| 36 | | | | | | | | | | |
| 37 | | | | | | | | | | |
| 38 | | | | | | | | | | |
| 39 | | | | | | | | | | |
| 40 | | | | | | | | | | |
| 41 | | | | | | | | | | |
| 42 | | | | | | | | | | |
| 43 | | | | | | | | | | |
| 44 | | | | | | | | | | |
| 45 | | | | | | | | | | |
| 46 | | | | | | | | | | |
| 47 | | | | | | | | | | |
| 48 | | | | | | | | | | |
| 49 | | | | | | | | | | |
| 50 | | | | | | | | | | |

Feuille 4 sur 4 + UX 1 UX 2 UX 3 UX 4 Par défaut Français (France) Moyenne : Somme : 0 0 0 0

UX-eCommerce.odt

En termes de logiciels, nous avions accès à l'IDE Visual Studio Code permettant le développement Web, de manière simple et efficace, développé par Microsoft. Nous avions aussi accès à WAMP Server qui est un serveur Web fonctionnant sous Windows et qui permet d'héberger des sites en local. Il est composé de tous les outils nécessaires pour la création et l'hébergement de sites Web (PHPMyAdmin, Apache, ...). Nous utilisions GitKraken, Github, et Git pour pouvoir faire le versioning et avoir accès à un dépôt afin de pouvoir tous travailler sur le même le projet et en même temps.



Logo de WAMP Server.



Logo de Visual Studio Code.



Logo de GitKraken.



Logo de Github.



Logo de Git.

Pour pouvoir entrer en contact avec le tuteur de stage afin d'obtenir des conseils, de l'aide ou pour le tenir au courant de l'avancement du projet ainsi que pour les réunions du matin et du soir, et pour rester en contact avec toute l'équipe et travailler ensemble, le logiciel Teams par Microsoft a été utilisé afin d'établir nos communications



Logo de Microsoft Teams.

En terme de technologies Web, on utilise le framework Angular pour la partie application Web, Angular Material et Bootstrap pour le design, MySQL pour le Système de Gestion de Base de Données (SGBD) et les langages utilisés pour le développement web : PHP, HTML, SCSS, JavaScript, TypeScript et SQL.



Logo d'Angular.



Logo de Angular Material.



Logo de Bootstrap



Logo de MySQL.



Logo de TypeScript.



Logo SCSS (SASS).

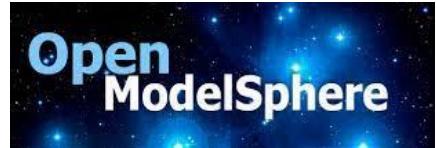


Logo de PHP, HTML, CSS, JavaScript et SQL.

Nous avons eu aussi recours au logiciel MySQL Workbench pour pouvoir créer une base de données rapidement et au logiciel Open ModelSphere pour pouvoir réaliser des MCD.



Logo de MySQL Workbench.



Logo de Open ModelSphere.

Nous avons aussi eu accès à Trello (<https://trello.com/>) qui est un site en ligne permettant de réaliser une organisation de projet visuel en utilisant des cartes et des colonnes. Cela nous a permis de réaliser la répartition des tâches et pour le chef de projet ainsi que notre tuteur de stage de suivre l'avancement du projet.



Logo de Trello.

Pour pouvoir réaliser des paiements sur le site, nous avons intégré et utilisé le module JavaScript de paiement nommé Stripe.



Logo de Stripe.

Pour pouvoir intégrer un éditeur de texte au projet web, le module CKEditor a été utilisé. C'est un module JavaScript permettant de réaliser un éditeur de texte enrichi de type WYSIWYG (« What You See Is What You Get ») et permettant d'obtenir un résultat en HTML.



Logo de CKEditor.

Présentation du sujet :

Notre tuteur de stage, M. SCHNEKENBURGER, nous a demandé de réaliser un site e-commerce/marketplace permettant à des annonceurs de rejoindre la plateforme et de payer une offre afin d'afficher leurs annonces pour vendre leurs produits (5 maximums par offre). A des partenaires, de pouvoir être affichés et donc faire leurs publicités. Et aux utilisateurs inscrits, de pouvoir commander les produits des annonceurs. Un système de modération a dû aussi être mis en place afin de gérer (valider ou refuser ainsi que supprimer) les annonces, il devait donc avoir des admins logistiques, des admins boutiques et des superadmins. Tout le système de commande et de paiement devait être pensé et conçu. Il exisait une première version de ce projet, notre objectif était de passer ce projet sous Angular.

Nous étions une équipe de 6 étudiants venant du même BTS, nous devions former des binômes et répartir les tâches entre nous, ainsi que décider qui serait le dev-leader (chef de projet).

Une fois les binômes formés, les tâches réparties et le dev-leader choisi, le projet fut découpé en plusieurs parties : Formation sur Angular, Angular Material et sur le CRUD. Réalisation de la structure de base de l'application Angular (en prenant en compte le Back et le Front) ainsi que de la base de données. Chaque binôme partit de son côté pour réaliser ses tâches et enfin, nous avons pu tester le projet sur un serveur en conditions réelles.

Cahier des charges venant du fichier « dashboard-eCommerce.odt » :

| FRONT | BACK | Spécificité | UX liées | nb jours |
|--|---|-----------------------------|----------|------------|
| | Gestion des administrateurs | | | |
| Mise en place structure Angular de base | Gestion des utilisateurs | UX 1 | | 1 |
| | Gestion des annonces | UX 2 | | 2 |
| | Gestion des partenaires | UX 3 | | 1 |
| | | UX 4 | | 1 |
| Sécurisation et accès sécurisé | | | | 4 |
| | Gestion des catégories | | | 0,5 |
| | Gestion des gammes | | | 0,5 |
| | Gestion des produits | | | 1 |
| | Gestion des Offres pour annonceurs / packs | | | 1 |
| | Gestion des annonces | | | 1 |
| | Gestion des informations de la boutique | | | 0,5 |
| | Gestion des bannières publicitaires | | | 1 |
| | Gestion du carrousel de la page d'accueil | | | 1 |
| | Gestion de pages connexes (mentions / crédits / CGV...) | | | 1 |
| | Gestion du panier de commande | | | 1 |
| | Gestion de la commande | | | 1 |
| | Gestion des états de commande | | | 1 |
| | Gestion des erreurs commandes | | | 1 |
| | Gestion logistique commande | | | 1 |
| | Gestion Satisfaction | | | 1 |
| | Gestion Actualités | | | 1 |
| | Gestion Tutoriels en ligne | | | 1 |
| Entête | | | | 0,5 |
| Footer | | | | 0,5 |
| Recherche produits | | | | 1 |
| Inscription Client | | | | 0,5 |
| Identification Client | | | | 0,5 |
| Gestion Client | | | | 0,5 |
| Inscription Annonceur | | | | 0,5 |
| Identification Annonceur | | | | 0,5 |
| Gestion Annonceurs | | | | 0,5 |
| Gestion Annonceurs – Produits | | | | 0,5 |
| Gestion Annonceurs – Packs choisis | | | | 1 |
| Inscription Partenaire | | | | 0,5 |
| Identification Partenaire | | | | 0,5 |
| Gestion Panier | | | | 1 |
| Gestion Processus commande | | | | 1 |
| Gestion Page Catalogue | | | | 0,5 |
| Gestion Page Produits | | | | 0,5 |
| Gestion Page Accueil | | | | 1 |
| Gestion Pages du site | | | | 1 |
| Gestion Actualités et détail | | | | 1 |
| Gestion Tutoriaux et détail | | | | 1 |
| Gestion Newsletter | | | | 0,5 |
| Gestion Contact | | | | 0,5 |
| Gestion Produits les plus vendus | | | | 0,5 |
| Gestion Produits notation | | | | 0,5 |
| Gestion Produits les plus vus | | | | 0,5 |
| Gestion Wishlist | | | | 1 |
| Gestion Erreurs Commande / Livraison / reclam. | | | | 1 |
| Installation sur serveur Test | | | | 1 |
| | | Total Jours projets | | 36,5 |
| | | Temps de travail par binome | | 12,1666667 |

SOMMAIRE :

I) **Base de données :**

1. Étude du besoin et réalisation du MCD.
2. Réalisation de la base de données.

II) **Application Web :**

1. Formation sur les technologies qui seront utilisées.
 - a. Angular.
 - b. CRUD.
 - c. Angular Material.
2. Démarrage et création du projet.
 - a. Création de la structure de l'application Angular et installation des librairies de bases pour ce projet.
 - b. Répartitions des taches.
3. Administration.
 - a. La gestion des administrateurs.
 - b. La gestion des annonceurs.
 - c. La gestion des produits.
 - d. La gestion des annonces.
4. Annonceurs.
 - a. Inscription.
 - b. Identification.
5. Tutoriels et actualités
 - a. BACK.
 - b. Partie affichage.
 - c. Partie gestionnaire.
6. Gestion des informations de la boutique.

III) **Conclusion**

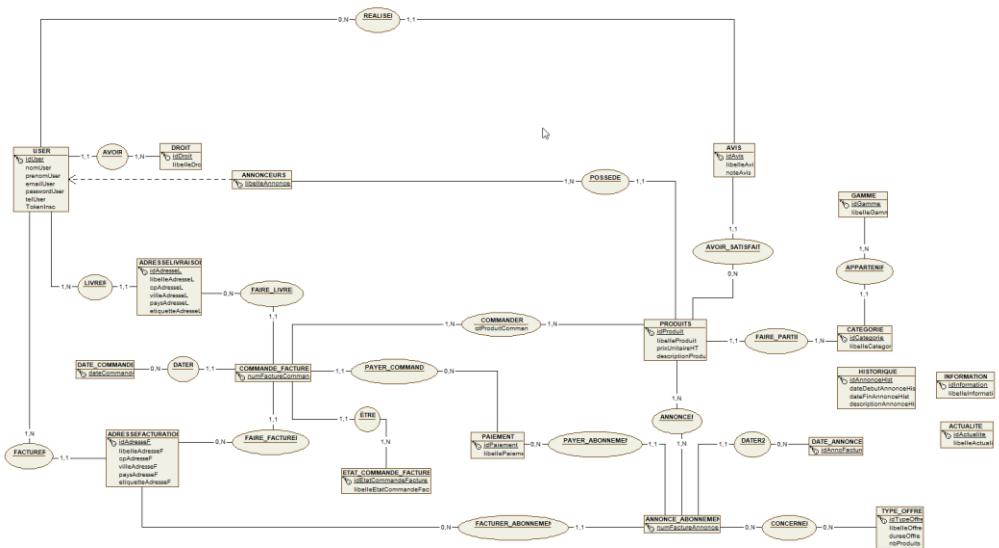
IV) **Annexes**

I) Base de données :

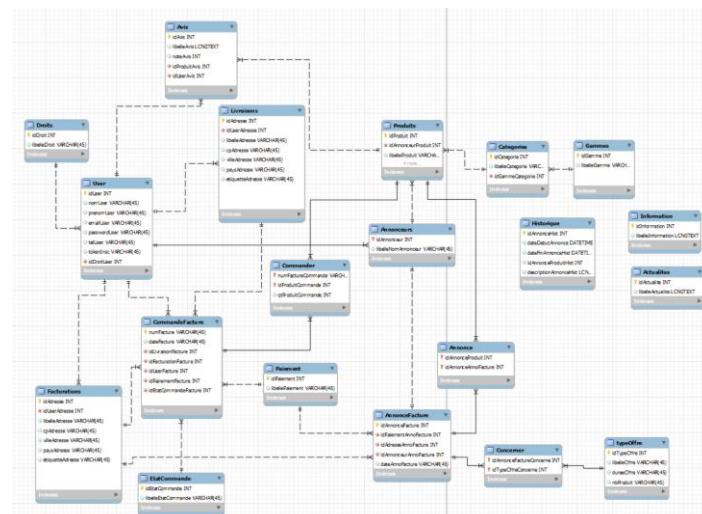
1. Étude du besoin et réalisation du MCD.

Toute l'équipe s'est réuni sur Teams afin d'étudier le besoin et les documents à notre disposition pour réaliser le MCD de la base de données nécessaire à l'application. Le document « *UX-eCommerce.ods* » nous donnait déjà certaines tables à mettre dans la base de données notamment pour la gestion des admins, des clients, des annonceurs et des partenaires. Il nous fallait donc réfléchir sur la façon de gérer en base de données les produits, les commandes et leurs états, les offres d'abonnements pour les annonceurs, les moyens de paiement et les moyens de facturations pour les commande ainsi que pour les annonceurs, les avis sur les produits, un historique des annonces postées, les actualités et les informations du site.

Une fois les idées échangées et mises en communs, nous avons utilisé le logiciel « *Open ModelSphere* » afin de réaliser le MCD de la base de données.



Après cela, nous avons présenté le MCD à notre tuteur de stage pour qu'il le valide et qu'on puisse réaliser la base de données physique. Nous l'avons réalisé sur le logiciel « *MySQL Workbench* » qui permet de réaliser une base de données de manière schématique et d'exporter un script SQL pour pouvoir importer sa structure dans le Système de Gestion de Base de Données (SGBD) MySQL.



I) Base de données :

2. Réalisation de la base de données.

Une fois le script exporté du logiciel « *MySQL Workbench* », j'ai accédé à « *MySQL* » du serveur « *WAMP* » via l'interface graphique « *PHPMyAdmin* » pour créer la base de données que l'on a nommée dans un premier temps « *mydb* » qui est le nom par défaut de la BDD utilisée par le script mais que l'on renommera plus tard.

The screenshot shows the PHPMyAdmin interface for a MySQL server on localhost. The left sidebar lists existing databases: Nouvelle base de données, bdd, damien-nolle_cms, information_schema, jv_bdd, mysql, performance_schema, phpmyadmin, pix_theatre, scrum_jv, sys, and test_etudiants. The main panel is titled 'Bases de données' and contains a sub-panel 'Création d'une base de données'. In this sub-panel, the database name is set to 'mydb' and the character set is 'utf8mb4_0900_ai_ci'. A 'Créer' button is visible. Below this, there's a 'Supprimer' button and a table listing existing databases with their character sets and options to verify privileges.

Puis j'ai importé le script précédemment exporté via l'onglet « *Importer* » de PHPMyAdmin afin de créer la structure de la BDD précédemment créée.

The screenshot shows the 'Importer' tab in the PHPMyAdmin interface. The left sidebar shows the same list of databases as before. The main area is titled 'Importation partielle :'. It shows a file selection dialog where 'scriptfinal.sql' has been chosen. The file is described as being in UTF-8 format. There are checkboxes for 'Permettre l'interruption de l'importation pour respecter la limite de temps définie dans PHP' and 'Ignorer ce nombre de requêtes (pour SQL), à partir du début'. Below these are sections for 'Autres options' (checkbox for foreign key verification) and 'Format' (set to 'SQL'). Under 'Options spécifiques au format :', there are settings for SQL compatibility mode ('NONE') and a checkbox for 'Ne pas utiliser AUTO_INCREMENT pour la valeur zéro'. At the bottom is an 'Importer' button.

Résultat de la BDD importée via le script :

The screenshot shows the phpMyAdmin interface for the 'mydb' database. The left sidebar lists various databases, and the main area displays the structure of the 'mydb' database, which contains 15 tables. A table at the bottom summarizes the total number of rows and file size.

| Table | Action | Lignes | Type | Interclassement | Taille | Perte |
|-----------------|--|--------|--------|--------------------|-----------|-------|
| annonce | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 48,0 kio | - |
| annoncefacture | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 80,0 kio | - |
| annoncateurs | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 32,0 kio | - |
| categories | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 32,0 kio | - |
| commandefacture | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 80,0 kio | - |
| commander | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 48,0 kio | - |
| droits | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 16,0 kio | - |
| facturations | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 32,0 kio | - |
| gammes | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 16,0 kio | - |
| historique | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 16,0 kio | - |
| livraisons | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 32,0 kio | - |
| paiement | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 16,0 kio | - |
| produits | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 48,0 kio | - |
| typeoffre | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 16,0 kio | - |
| user | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_0900_ai_ci | 32,0 kio | - |
| 15 tables | Somme | 0 | MyISAM | utf8mb4_0900_ai_ci | 544,0 kio | 0 |

Ceci est une première version de la base de données, il est possible qu'elle subisse des changements tout au long du projet. Dans mon cas, je prendrais uniquement en compte les changements que j'ai apporté à la base de données.

II) Application Web :

1. Formation sur les technologies qui seront utilisées.

a. Angular.

Avant la réalisation de la base de données, nous avons dû nous former sur les technologies que nous avons utilisées lors de ce stage. L'une des plus importantes est Angular qui est un Framework Open Source permettant, comme React, de créer des sites Web mono-pages (site qui ne se réactualise jamais et auquel toutes les interfaces s'affiche sur une seul page) qui sont les favoris des robots de tous les navigateurs car étant plus rapide à charger et à s'afficher, améliorant ainsi notre référencement. Ce Framework est basé sur TypeScript qui lui-même est basé sur JavaScript auquel on peut préciser le type des variables déclarées. L'avantage de ce langage est, qu'une fois compilé, le code est sécurisé de manière à rendre impossible la lecture de celui-ci.

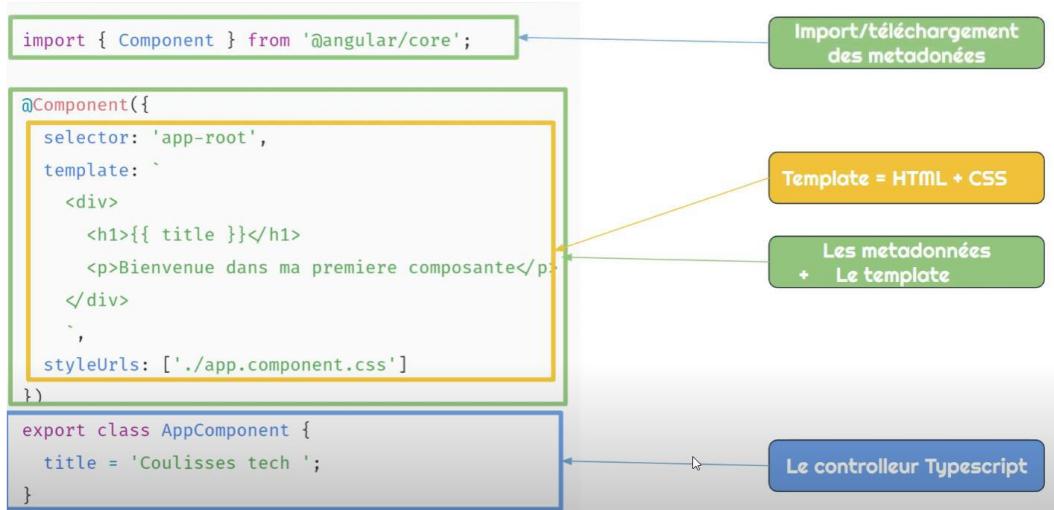
```
// Javascript
let maVariable;
maVariable = 'Angular';

// Typescript
let maVariable: string;
maVariable = 'Angular';
```

La logique d'Angular est exactement la même que celle de React : tout est un composant, chaque composant représentera une fonctionnalité de la page (Barre de navigation, barre de recherche, etc...). Les pages représentent aussi un gros composant qui regroupe un ensemble de sous-composants.

Un composant est découpé en deux parties : la partie Métadonnées, qui va correspondre aux informations du composant soit le nom de celui-ci et les liens entre son template HTML et son style. Par ailleurs, en Angular on utilise en style le SCSS qui va permettre de réaliser du style dynamique de manière plus simple et plus approfondie que le CSS en temps normal mais tout en conservant ce dernier. De plus, un style appliquée à un composant ne sera utilisé que par lui-même. Et la deuxième partie est la partie logique et fonctionnelle d'un composant.

La structure de base d'un composant :



En Angular, on travaille énormément avec des classes, la partie logique et fonctionnelle est d'ailleurs une classe exporter.

Il existe deux types de composants : le composant classique et le service. Un service est un composant qui ne dispose pas d'interface et donc pas de template HTML. Il est composé principalement d'une classe et a pour objectif de contenir la partie logique d'une fonctionnalité qui n'est pas affichable. On peut, par exemple, l'utiliser pour manipuler des données reçues par des requêtes HTTP ou encore réaliser une classe qui permettra de créer un ou plusieurs objets de celle-ci qui seront utilisables et contiendront des données tout au long de l'utilisation de l'application. Un service est mis à disposition de toute l'application en utilisant le système d'injection de dépendances d'Angular.

Exemple de service :

```
src > app > TS cart.service.ts > CartService
1 import { Product } from './products';
2 import { Injectable } from '@angular/core';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class CartService {
8   items: Product[] = [];
9
10  constructor() {
11
12  }
13
14  addToCart(product: Product) {
15    this.items.push(product);
16  }
17
18  getItems() {
19    return this.items;
20  }
21
22  clearCart() {
23    this.items = [];
24    return this.items;
25  }
26 }
```

Pour pouvoir réaliser des requêtes HTTP dans une application Angular et manipuler les données reçues, on utilise le module/la bibliothèque « *http* ».

Les composants, entre eux, peuvent être parent de l'un ou l'enfant de l'autre (composant et sous-composant). Ces composant peuvent s'échanger des données entre eux. Lorsqu'un composant parent envoi des données à un composant enfant, on va utiliser les « *inputs* » (on retrouve ici, les Props en React). Dans le cas contraire, on va utiliser des « *outputs* ».

On peut aussi venir créer des fichiers « *.ts* » qui ressemblent à des services mais qui contiendront uniquement des déclarations de variable ou de classe, ou qui contiendront tout simplement des variables déclarées et contenant des données.

Exemple de fichier « .ts » contenant des déclarations de variable et/ou de la donnée :

```
export interface Product {
  id: number;
  name: string;
  price: number;
  description: string;
}

export const products = [
  {
    id: 1,
    name: 'Phone XL',
    price: 799,
    description: 'A large phone with one of the best screens'
  },
  {
    id: 2,
    name: 'Phone Mini',
    price: 699,
    description: 'A great phone with one of the best cameras'
  },
  {
    id: 3,
    name: 'Phone Standard',
    price: 299,
    description: ''
  }
];
```

Pour appeler un autre composant sur un template HTML, on utilise le nom de la classe exportée comme une balise HTML et on saisit, en attributs, les paramètres d'entrées (les « *inputs* »).

Exemple : <app-nom-composant [input1] = "valeur1"></app-nom-composant>

Une application Angular dispose d'une structure bien spécifique dont voici les fichiers et répertoires importants et surtout les plus utilisés de cette structure :

- « *app* » : Ce dossier contiendra tous les composants/modules de l'application.
- « *assets* » : Ce dossier contiendra toutes les ressources (images, fonts, librairies, etc...) de l'application.
- « *index.html* » : Fichier HTML par défaut qui est le point de départ de l'application.
- « *main.ts* » : Il va chercher le « *AppModule* » qui lui-même ira chercher le composant, c'est le lien entre l'application en elle-même et les composants.
- « *app.module.ts* » : Fichier dans lequel on déclare nos éléments dont les composants dans le tableau « *declarations* » en utilisant le nom de la variable exportée par ce composant et dans lequel on crée et déclare le routeur.
- Les fichiers « *.spec.ts* » : Permettent de faire des tests unitaires.

Comme pour React, il est possible de réaliser un routeur pour naviguer entre les pages de l'application Angular. Le module permettant le routing (« *RouterModule* ») dispose de différentes fonctionnalités pour interagir avec une route. Le « *routeurLink* » permet dans un template HTML de réaliser un lien de navigation (fonctionnant comme le « *NavLink* » de React). La classe « *ActivatedRoute* » va permettre de vérifier si l'URL est valide et de récupérer cette dernière dans une variable. En proposant une route

avec « `:nomVar` » dans son URL, on peut venir y déclarer et y attribuer une variable appelée « paramètre » et qui sera récupérable via l'`« ActivatedRoute »`.

Exemple de routeur :

```
RouterModule.forRoot([
  { path: 'liste_produits', component: ProductListComponent },
  { path: 'products/:productId', component: ProductDetailsComponent },
  { path: 'cart', component: CartComponent },
  { path: 'shipping', component: ShippingComponent },
  { path: 'students', component: StudentsListComponent },
  { path: 'add-students', component: AddStudentsComponent },
  { path: 'edit/:id', component: EditStudentsComponent }
])
```

Dans les templates HTML, on peut venir appliquer des structures directives permettant ainsi de rendre dynamique un code HTML. Parmi les plus utilisées, on retrouve : « `*ngIf` » (Logique de la condition if) et « `*ngFor` » (Logique de la boucle for/foreach) que l'on va pouvoir attribuer à une balise HTML. Par exemple : `<h2 *ngIf="hotels && hotels.length > 0" >{{ title }}</h2>`. On retrouve aussi le « `{{ }}` » qui, comme Symfony, permet d'afficher une variable envoyée.

Une autre notion des templates HTML est le « `binding` » qui permet d'attribuer des variables/fonctions JS venant du composant à des attributs de balises HTML ou à des événements JS afin de réaliser des éléments HTML dynamiques. Le « `[]` » permet d'attribuer une variable à un attribut d'une balise HTML et le « `()` » permet d'attribuer à un évènement JS une fonction ou un événement. Attention, il faut que la valeur attribuée soit du même type que ce qui est déclaré dans le composant !

Exemple de binding :

```
<a [title]="{{ product.name + ' details' }}">
  {{ product.name }}
</a>
```

Une autre notion de ces templates HTML est les « `Pipes` » qui consiste à transformer une donnée/une variable (son affichage et, d'une certaine manière, son type) directement depuis le template.

Un composant à un cycle de vie avec lequel on peut interagir, notamment avec la classe « `OnInit` » que l'on va implémenter à la classe d'un composant et qui va permettre d'attendre que celui-ci soit monté (chargé ou complètement exécuté) avant d'exécuter du code de ce composant par la méthode « `ngOnInit()` ».

L'avantage d'Angular est que l'on va pouvoir créer des formulaires de manière simple et rapide en utilisant le « `formBuilder` ». Dans un premier temps, on vient créer notre formulaire en déclarant une variable qui contiendra les différents champs avec leurs options si besoin dans la classe du composant. Puis dans le templates, on vient l'afficher et on ajoute un bouton qui pourra faire appel à une méthode de la classe du composant pour récupérer les données.

Exemple de déclaration de formulaire :

```
17  constructor (private url:ActivatedRoute, private router: Router, private formBuilder: FormBuilder, private studentService:StudentsService)
18  {
19    this.EditForm = this.formBuilder.group({
20      id: '',
21      nom: '',
22      prenom: ''
23    });
24 }
```

Exemple d'affichage de formulaire dans un template HTML :

```
src > app > edit-students > edit-students.component.html > form
1  <p>edit-students works!</p>
2  <h1>Modifier un étudiant</h1>
3  <form [formGroup]="EditForm" (ngSubmit)="onEdit()">
4
5    <div>
6      <label for="id">
7        ID :
8        </label>
9        <input id="id" type="text" formControlName="id">
10   </div>
11
12   <div>
13     <label for="nom">
14       Nom :
15       </label>
16       <input id="nom" type="text" formControlName="nom">
17   </div>
18
19   <div>
20     <label for="prenom">
21       Prénom :
22       </label>
23       <input id="prenom" type="text" formControlName="prenom">
24   </div>
25
26   <button class="button" type="submit">Ajouter</button>
27 </form>
```

Un « *EventEmitter()* » dans Angular est une variable qui va permettre de changer de valeur uniquement si un événement JS se produit (« *.emit()* »).

Enfin, un « *Observable* » est une variable qui est vouée à changer dans le temps et qui va attendre les données grâce à la méthode « *.subscribe()* ».

II) Application Web :

1. Formation sur les technologies qui seront utilisées.
 - b. CRUD.

Jusque-là, nous nous sommes formés sur les bases d'Angular qui correspond à la partie Front. Pour la partie Back, soit la manipulation de la base de données, nous devons nous former sur la façon de réaliser un CRUD.

Le CRUD est une API réalisée en PHP qui fera le lien entre la base de données et l'application Angular.

La logique du CRUD est que le chemin menant à un fichier PHP correspond à une route. Ce fichier PHP permet de réaliser une action souhaitée sur la base de données. L'idée est qu'un fichier PHP correspond à une action sur une table. Par exemple : un fichier permettra de mettre à jour la table (UPDATE), un autre permettra de supprimer un enregistrement de la table (DELETE), un autre d'insérer un enregistrement dans la table (INSERT) et un autre permettra de récupérer une ou plusieurs données de cette table (SELECT).

Tout va se jouer sur la hiérarchie des dossiers de l'API et sur la façon dont les routes vont être formées par les chemins parcourus. Il faut donc être capable de ranger les répertoires et les fichiers PHP de manière à réaliser des chemins et donc des routes cohérentes.

Un fichier de connexion à la base de données voulue doit être créé à la racine de l'API de manière à ce que les fichiers PHP puissent interagir avec celui-ci.

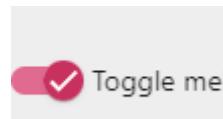
Vous trouverez aux annexes 1 à 5, un exemple de CRUD réalisé lors de notre formation. Le but de ce CRUD était de passer en revue les actions possibles et de réaliser un petit gestionnaire d'étudiants.

II) Application Web :

1. Formation sur les technologies qui seront utilisés.
 - c. Angular Material.

Material Angular est une bibliothèque de composants prédéfinis permettant ainsi de réaliser une interface utilisateur (UI) rapidement et facilement. Cette bibliothèque fonctionne dans la même veine que Bootstrap. L'objectif étant d'importer le composant souhaité, de la même manière que l'on importera un composant que l'on aurait créé, et de le modifier pour qu'il réponde à notre besoin. La documentation d'Angular Material est assez complète pour décrire toutes les possibilités (lien : <https://material.angular.io/>).

Imaginons par exemple que nous souhaitions importer la bascule suivante :



On vient l'importer dans le fichier « *app-modules.ts* » :

```
19 | import { MatSlideToggleModule } from '@angular/material/slide-toggle';
```

Puis on la rajoute dans les « *imports* » de ce même fichier :

```
imports: [
  BrowserModule,
  HttpClientModule,
  AppRoutingModule,
  FormsModule,
  ReactiveFormsModule,
  RouterModule.forRoot([
    { path: 'liste_produits', component: ProductListComponent },
    { path: 'products/:productId', component: ProductDetailsComponent },
    { path: 'cart', component: CartComponent },
    { path: 'shipping', component: ShippingComponent },
    { path: 'students', component: StudentsListComponent },
    { path: 'add-students', component: AddStudentsComponent },
    { path: 'edit/:id', component: EditStudentsComponent }
  ]),
  BrowserAnimationsModule,
  MatAutocompleteModule,
  MatSlideToggleModule,
  MatButtonToggleModule
],
```

Enfin, on vient appeler dans un template HTML via ses balises, de la même manière qu'on appellera un composant créé et importé :

```
<mat-slide-toggle>Toggle me!</mat-slide-toggle>
```

Il suffit juste après de modifier son style via le « `.scss` » du composant et d'interagir avec lui via le contrôleur de celui-ci, comme on le ferait avec un « `input` » HTML normal en Angular.

II) Application Web :

2. Démarrage et création du projet.

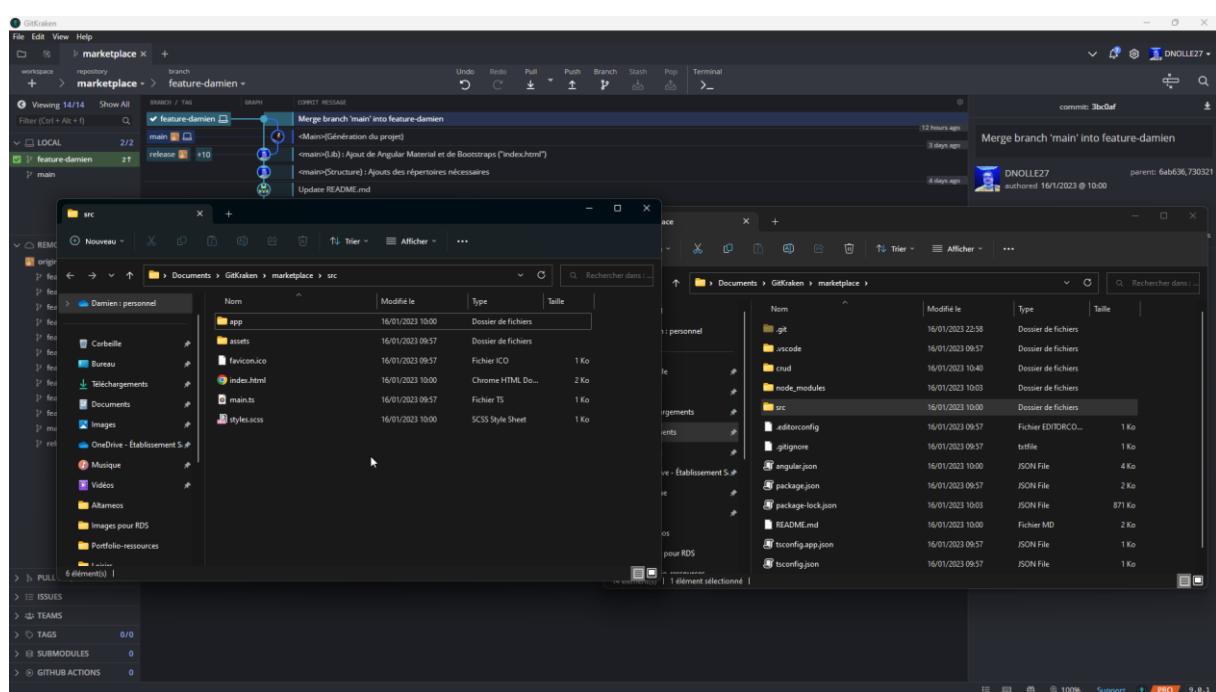
- a. Création de la structure de l'application Angular et installation des librairies importantes pour ce projet.

Dans un premier temps un dépôt Git (sur Github) a été créé via GitKraken et les branches respectant les règles de GitFlow ont été créées (« *Main* » correspondant à « *Master* » qui permettra d'y déposer le projet fini et d'y faire le versionning, « *feature_[Nom du membre de l'équipe]* » qui va permettre à chaque membre de l'équipe d'y développer ses fonctionnalité et donc de réaliser ses tâches, « *realease* » qui permettra de mettre une version finie mais à tester avant de la mettre dans « *Main* », « *develop* » qui correspond à la version à utiliser pour développer et « *hotfix* » qui va permettre de réaliser des corrections de bug sur la version présente dans « *Main* »).

Nous avons ensuite, dans la branche « *develop* », créé le projet Angular via la commande « *ng new marketplace* » puis nous avons installé tous les prérequis dont Angular Material avec la commande « *ng add @angular/material* », Stripe avec la commande « *npm install ngx-stripe @stripe/stripe-js* » et en important le module « *NgxStripeModule* » dans le fichier « *app.module.ts* ». Enfin, nous avons lié Bootstrap au fichier « *index.html* » pour pouvoir utiliser celui-ci dans tout le projet Angular.

```
link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-GhLQtQBRJ+ZJnVGyZq4Jd4E6Tt7iZ5q3CBvWfK9ccvF+F7DwBjPz07z7tC0Hmdt" crossorigin="anonymous">
```

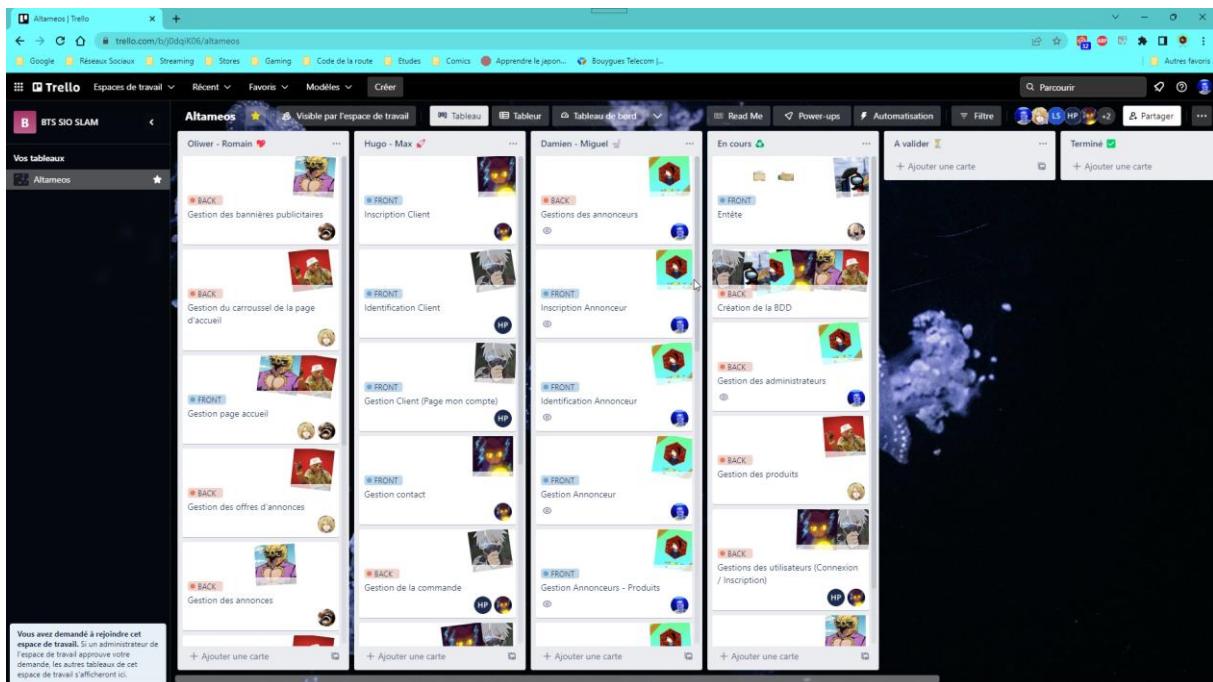
Enfin, nous avons utilisé la structure de base d'un projet Angular (dans « *src* ») afin d'y créer nos interfaces et donc toute notre partie Front, puis nous avons créé un répertoire « *crud* » dans la racine du projet pour pouvoir y placer notre CRUD et donc toute notre partie Back.



II) Application Web :

2. Démarrage et création du projet.
 - b. Répartitions des tâches.

Nous nous sommes réunis toute l'équipe et le tuteur de stage sur Teams pour pouvoir discuter sur la répartition des tâches de chaque binôme. Lors de cette discussion, un Trello a été créé pour représenter les binômes et leurs tâches attribuées. L'objectif de ce Trello est, pour nous, de marquer notre avancement dans le projet et d'être toujours à l'affût de potentielle modification d'organisation. Le chef de projet et notre tuteur de stage, pouvaient suivre l'avancement du projet et prendre des décisions en conséquence. Nous nous sommes ensuite concertés avec nos binômes respectifs afin de répartir les tâches et nous les avons consignées sur le Trello.



Voici la liste de mes tâches personnelles :

- Gestion des administrateurs.
- Gestion des annonceurs.
- Inscription Annonceur.
- Identification Annonceur.
- Gestion Annonceurs - Produits (Admin).
- Gestion Annonceurs - Annonces (Admin).
- Gestion actualités.
- Gestion tutoriel.
- Gestion actualités et détails.
- Gestion tutoriaux et détails.

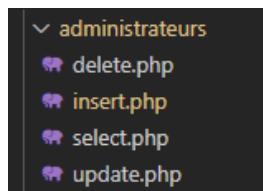
II) Application Web :

3. Administration.

a. La gestion des administrateurs.

L'objectif de cette page est de permettre à un super administrateur de pouvoir gérer les administrateurs du site. Il pourra ajouter, supprimer ou modifier un administrateur, en sachant qu'il en existe trois types (« superadmin », « admin_boutique » et « admin_logistique »). Le super administrateur connecté ne pourra pas supprimer son compte mais il pourra le modifier.

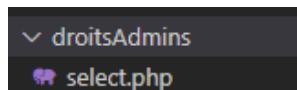
Dans un premier temps, j'ai réalisé un CRUD qui va me permettre de récupérer des informations uniquement des comptes des administrateurs pour l'affichage ainsi que la suppression, la modification et l'ajout d'un compte administrateur. Il est réalisé de la même manière que celui réalisé en exemple (Annexe 1 à 5) lors de la formation (cf. II) 1. b.) mais en le réadaptant pour répondre au besoin de gestion des administrateurs présent dans la table « user ».



La requête SQL suivante me permet de récupérer uniquement les utilisateurs de type administrateur.

```
SELECT iduser, nomUser, prenomUser, emailUser, passwordUser, telUser, tokenInsc, libelleDroit
FROM user
INNER JOIN Droits ON idDroit = IdDroitUser
WHERE libelleDroit IN ('superadmin','admin_boutique','admin_logistique')
```

Puis j'ai réalisé un fichier « select.php » afin de récupérer uniquement les droits des administrateurs présents dans la table « droits » .



La requête SQL suivante me permet d'uniquement récupérer les droits des administrateurs. En travaillant uniquement avec le libellé et non l'id, cela permet en cas de changement de ces derniers d'avoir toujours le code fonctionnel pour les bons droits.

```
SELECT idDroit, libelleDroit FROM droits WHERE libelleDroit IN ('superadmin','admin_boutique','admin_logistique');
```

Enfin, j'ai réalisé les modèles représentant les administrateurs et les droits des administrateurs dans des fichiers « .ts » qui vont nous permettre de récupérer et de manipuler les données.

```
src > app > model > TS Administrateurs > Administrateurs.ts
1  export class Administrateurs {
2    idUser? : number;
3    nomUser? : string;
4    prenomUser? : string;
5    emailUser? : string;
6    passwordUser? : string;
7    tellUser? : string;
8    tokenInsc? : string;
9    libelleDroit? : string;
10
11   constructor (idUser:number, nomUser:string, prenomUser:string, emailUser:string, passwordUser:string, tellUser:string, tokenInsc:string, libelleDroit:string)
12   {
13     this.idUser = idUser;
14     this.nomUser = nomUser;
15     this.prenomUser = prenomUser;
16     this.emailUser = emailUser;
17     this.passwordUser = passwordUser;
18     this.tellUser = tellUser;
19     this.tokenInsc = tokenInsc;
20     this.libelleDroit = libelleDroit;
21   }
22 }
```

Le fichier « Administrateurs.ts ».

```
src > app > model > TS Droits.ts > Droits.ts
1  export class Droits {
2    idDroit? : number;
3    libelleDroit? : string;
4
5
6    constructor (unIdDroit:number, unLibelleDroit:string)
7    {
8      this.idDroit = unIdDroit;
9      this.libelleDroit = unLibelleDroit;
10    }
11 }
```

Le fichier « Droits.ts ».

Puis j'ai réalisé des fonctions contenant les requêtes HTTP permettant de faire appel au CRUD pour réaliser les actions possibles dans un service nommé « *pannel-administration.service* » et créé avec la commande « *ng generate service* » (Ce service réunira toutes les requêtes HTTP nécessaires à ma partie de l'administration).

```
baseUrl: string = 'http://localhost/marketplace/crud/';

getLesAdministrateurs()
{
    return this.http.get<Administrateurs[]>(this.baseUrl+'administrateurs/select.php');
}

getUnAdministrateur(id:any)
{
    return this.http.get<Administrateurs[]>(this.baseUrl+'administrateurs/select.php?id='+id);
}

modifAdministrateur(admin:any)
{
    return this.http.put(this.baseUrl+'administrateurs/update.php', admin);
}

supprAdministrateur(id:any)
{
    return this.http.delete(this.baseUrl+'administrateurs/delete.php?id=' + id);
}

ajoutAdministrateur(admin:any)
{
    return this.http.post(this.baseUrl+'administrateurs/insert.php', admin);
}

getDroitsAdmins()
{
    return this.http.get<Droits[]>(this.baseUrl+'droitsAdmins/select.php');
}
```

Ensute, j'ai créé un composant « *list-administrateurs* » avec la commande « *ng generate component* » qui va permettre de gérer l'affichage des administrateurs ainsi que la suppression, la modification et l'ajout d'un administrateur de tout type.

Dans un premier temps, j'ai réalisé la partie logique du composant, soit dans le fichier « *list-utilisateurs.component.ts* ». Pour l'affichage, on vient récupérer toutes les données nécessaires en important le service précédent contenant toutes mes requêtes HTTP.

```
src > app > list-utilisateurs > TS list-utilisateurs.component.ts > ...
1  import { Component, OnInit, OnChanges, Input, ViewChild, ChangeDetectorRef } from '@angular/core';
2  import { PannelAdministrationService } from '../services/pannel-administration.service';
3  import { FormBuilder, FormGroup } from '@angular/forms';
4
```

Puis j'ai déclaré une variable « *pannelAdmin* » en privé dans le constructeur du composant afin de faire appel au service précédemment importé et aux fonctions pour exécuter les requêtes HTTP et récupérer leurs résultats.

```
constructor (private changeDetector : ChangeDetectorRef, private formBuilser: FormBuilder, private pannelAdmin : PannelAdministrationService)
{
```

Enfin, dans le « ngOnInit », je viens récupérer toutes les données nécessaires à la réalisation de l'interface de mon composant.

```
ngOnInit(): void {
  this.pannelAdmin.getLesAdministrateurs().subscribe(
    (result:any)=>{
      this.lesAdministrateurs = result.data;
    }
  );

  this.pannelAdmin.getDroitsAdmins().subscribe(
    (result:any)=>{
      this.lesDroitsAdmins = result.data;
    }
  );

  this.pannelAdmin.getUsers().subscribe(
    (result:any)=>{
      this.lesUtilisateurs = result.data;
    }
  );

  this.user = JSON.parse(localStorage.getItem('authUser')!);
}
```

Pour l'affichage des informations des comptes administrateur, je suis parti sur un affichage en Grid avec des FlexBox afin de réaliser des cartes qui représenteront chacune un compte de manière à avoir un affichage le plus responsive possible.

Lors de la réalisation de l'affichage des administrateurs, de nouvelles notions pour les structures directives ont été vues :

Deux nouvelles balises HTML, venant d'Angular, permettent de réaliser des structures directives plus proprement.

- **ng-container** : Permet d'exécuter des structures directives sans impacter le reste du code HTML. Par exemple, lors du « **ngFor* », permet d'éviter d'avoir des balises qui se répètent sans rien dedans.
- **ng-template** : Permet de réaliser du code HTML qui s'affiche uniquement lorsqu'on l'appelle par son nom. Cumulé avec un « *ng-container* » et un « **ngIf* », permet de réaliser un « *if ... else* ».

Cela m'a permis, lors de la réalisation du template HTML, de réaliser un « *if ... else* » afin de modifier le contenu d'une carte de manière à afficher l'interface de modification ou de suppression, lorsqu'on appuie sur le bouton modifier ou supprimer, ou bien l'affichage normal d'une carte.

J'ai donc réalisé un premier template afin d'avoir l'affichage de base d'une carte, affichant ainsi les informations de chaque compte et deux boutons (« Modifier » et « Supprimer »). Le bouton « Supprimer » ne s'affichera pas si le compte actuellement connecté correspond à l'un des comptes affichés.

```
<ng-template #normalList>
  <div class="flex-card-container">
    <div class="flex-card-containeur-row premiere-ligne">
      <div class="flex-card-element"><span class="labelForm">ID :</span> {{ unAdministrateur.idUser }}</div>
    </div>
    <div class="flex-card-containeur-row">
      <div class="flex-card-element"><span class="labelForm">Nom :</span> {{ unAdministrateur.nomUser }}</div>
    </div>
    <div class="flex-card-containeur-row">
      <div class="flex-card-element"><span class="labelForm">Prénom :</span> {{ unAdministrateur.prenomUser }}</div>
    </div>
    <div class="flex-card-containeur-row">
      <div class="flex-card-element"><span class="labelForm">E-Mail :</span> {{ unAdministrateur.emailUser }}</div>
    </div>
    <div class="flex-card-containeur-row">
      <div class="flex-card-element"><span class="labelForm">Mot de passe :</span> *****</div>
    </div>
    <div class="flex-card-containeur-row">
      <div class="flex-card-element"><span class="labelForm">Téléphone :</span> {{ unAdministrateur.telUser }}</div>
    </div>
    <div class="flex-card-containeur-row">
      <div class="flex-card-element"><span class="labelForm">Droit :</span> {{ unAdministrateur.libelleDroit }}</div>
    </div>
    <div class="flex-card-containeur-row">
      <div class="flex-card-element"><button class="button-blue" (click)="adminModif(unAdministrateur.idUser)">Modifier</button></div>
      <div class="flex-card-element" *ngIf="!user.email != unAdministrateur.emailUser"><button class="button-red" (click)="adminSuppr(unAdministrateur.idUser)">Supprimer</button></div>
    </div>
  </div>
</ng-template>
</div>
```

Des formulaires ont été déclarés et réalisés avec le « *formBuilder* » afin de permettre l'ajout et la modification d'un compte administrateur.

```
this.modifFormAdmin = this.formBuilder.group({
  idUser: '',
  nomUser: '',
  prenomUser: '',
  emailUser: '',
  passwordUser: '',
  telUser: '',
  idDroitUser: ''
});

this.ajoutFormAdmin = this.formBuilder.group({
  idUser: '',
  nomUser: '',
  prenomUser: '',
  emailUser: '',
  passwordUser: '',
  telUser: '',
  idDroitUser: ''
});
```

Lorsqu'on appuiera sur le bouton « Modifier », « Supprimer » ou « Ajouter un nouvel administrateur », une procédure s'exécutera permettant de mettre l'id du compte dans une variable (pour la modification et la suppression) ou mettre un booléen à « *true* » (pour l'ajout) afin d'accéder à l'interface de l'action voulue dans une même carte. Elle permet aussi de nettoyer les formulaires et les champs de ces derniers afin de ne pas conserver de possibles anciennes saisies. Les variables ne correspondant pas à l'action voulue sont vidées (soit mis à « *null* », soit à « *false* ») afin de toujours afficher la dernière interface de l'action voulue et non toutes les interfaces cumulées.

```
createAdmin()
{
    this.idModifAdmin = null;
    this.idSupprAdmin = null;
    this.ajoutAdmin = true;

    clearAdminForm(this.ajoutFormAdmin);

    this.changeDetector.detectChanges();

    this.ajoutFormAdmin.value.idUser = '';
    this.ajoutFormAdmin.value.nomUser = '';
    this.ajoutFormAdmin.value.prenomUser = '';
    this.ajoutFormAdmin.value.emailUser = '';
    this.ajoutFormAdmin.value.passwordUser = '';
    this.ajoutFormAdmin.value.telUser = '';
    this.ajoutFormAdmin.value.idDroitUser = '';

    this.nomInput.nativeElement.value = '';
    this.prenomInput.nativeElement.value = '';
    this.emailInput.nativeElement.value = '';
    thismdpInput.nativeElement.value = '';
    this.telInput.nativeElement.value = '';
    this.droitInput.nativeElement.value = '';
}
```

Procédure permettant d'accéder à l'interface d'ajout d'un nouvel administrateur.

On peut voir sur le bouton d'ajout d'un administrateur, l'utilisation du « *changeDetector* », c'est une variable déclarée en privé dans le constructeur et étant un objet de la classe « *ChangeDetectorRef* » qui permet de détecter tout changement dans l'interface HTML. Cela va permettre de re-exécuter le code se situant en dessous du détecteur. On peut y voir aussi l'utilisation de variable « *ViewChild* » (à savoir, les inputs). Ce sont des variables qui vont permettre de récupérer des éléments HTML de la même manière que le « *document.getElementById()* » par exemple en JavaScript. L'Objectif ici est de permettre, au moment du passage à l'interface d'ajout, de vider le formulaire et les inputs sans impacter la valeur par défaut d'une comboBox (« <select> »).

```
@ViewChild('nomUser', { static: false }) nomInput : any;
@ViewChild('prenomUser', { static: false }) prenomInput : any;
@ViewChild('emailUser', { static: false }) emailInput : any;
@ViewChild('mdpUser', { static: false }) mdpInput : any;
@ViewChild('telUser', { static: false }) telInput : any;
@ViewChild('droitUser', { static: false }) droitInput : any;
```

*Déclaration des variables « *ViewChild* ».*

```
adminModif(id:any)
{
    this.nrSelect = null;
    this.unAdministrateurVerifExiste = null;
    this.idModifAdmin = id;
    this.idSupprAdmin = null;
    this.ajoutAdmin = false;
    clearAdminForm(this.modifFormAdmin);
    this.pannelAdmin.getUnAdministrateur(id).subscribe(
        (result:any)=>{
            this.nrSelect = result.data.idDroitUser;
            this.unAdministrateurVerifExiste = result.data;
        }
    );
}
```

Procédure permettant d'accéder à l'interface de modification d'un administrateur.

```

adminSuppr(id:any)
{
    this.idSupprAdmin = id;
    this.idModifAdmin = null;
    this.ajoutAdmin = false;
}

```

Procédure permettant d'accéder à l'interface de suppression d'un administrateur.

```

function clearAdminForm(form:any) : void
{
    form.reset();

    form.value.idUser = '';
    form.value.nomUser = '';
    form.value.prenomUser = '';
    form.value.emailUser = '';
    form.value.passwordUser = '';
    form.value.telUser = '';
    form.value.idDroitUser = '';
}

function verifAndClearNULLForm(form:any) : void
{
    if (form.value.idUser == null)
    {
        form.value.idUser = '';
    }

    if (form.value.nomUser == null)
    {
        form.value.nomUser = '';
    }

    if (form.value.prenomUser == null)
    {
        form.value.prenomUser = '';
    }

    if (form.value.emailUser == null)
    {
        form.value.emailUser = '';
    }

    if (form.value.passwordUser == null)
    {
        form.value.passwordUser = '';
    }

    if (form.value.telUser == null)
    {
        form.value.telUser = '';
    }

    if (form.value.idDroitUser == null)
    {
        form.value.idDroitUser = '';
    }
}

```

Les deux procédures permettant de vider les formulaires d'ajout et de modification. La première vide le formulaire en utilisant la méthode « .reset() » qui met tous les champs à « null », puis on vient les mettre à vide (« '' »). Cette première procédure est principalement utilisée pour vider les formulaires afin que les champs ne soient plus remplis. La deuxième procédure permet de mettre à vide les champs qui sont rester « null » afin d'éviter les problèmes lors de la vérification après validation du formulaire.

Ensuite, j'ai réalisé chaque interface qui seront accessibles en appuyant sur les boutons adéquats. L'interface de modification présentera un formulaire pour saisir des données afin de les modifier en base de données. Il n'est pas obligatoire de saisir tous les champs, en revanche, il faut qu'il y ait au moins un champ saisi pour réaliser une modification. Il y a ensuite deux bouton, un « Valider » qui va me permettre de « submit » le formulaire et de vérifier si les données saisies sont correctes notamment grâce à des Regex et un bouton « Annuler » qui va me permettre, comme son nom l'indique, d'annuler l'action en cours et de faire disparaître l'interface de modification. Si les données sont correctes, alors la modification se fait, sinon un message d'erreur expliquant l'origine apparait via les « alert() ». L'interface d'ajout se présente de la même manière que celui de la modification sauf qu'il faut obligatoirement saisir tous les champs du formulaire. Le mot de passe, que ça soit pour la modification ou l'ajout d'un administrateur, est crypté. L'interface de suppression permet à l'utilisateur de confirmer s'il veut réellement supprimer tel administrateur. S'il le veut, il pourra cliquer sur le bouton « Oui » qui validera la suppression, sinon, il pourra cliquer sur le bouton « Non » qui fait la même action que le bouton « Annuler ». Quel que soit l'action voulue par l'utilisateur, lors de la validation, l'interface est remise à son état initial et la liste des administrateurs est réactualisée de manière à directement afficher toutes modifications réalisées (ajout, modification ou suppression).

```

src > app > list-administrateurs > list-administrateurs.component.html > div.grid-container > div.grid-element
1  <h3>Liste des administrateurs </h3>
2
3  <div class="grid-container">
4    <div class="grid-element" *ngFor="let unAdministrateur of lesAdministrateurs">
5      <ng-container *ngIf="!idModifAdmin == unAdministrateur.idUser; else supprAdmin">
6        <div class="flex-card-container">
7          <div class="flex-card-container-row premiere-ligne-form">
8            <div class="flex-card-element labelFormCenter"><span class="titreId">ID :</span> {{ unAdministrateur.idUser }}</div>
9          </div>
10         <div class="flex-card-container-row">
11           <div class="flex-card-element labelFormRight">Nom :</div>
12           <div class="flex-card-element" [formGroup]="modifFormAdmin"><input id="nomUser" type="text" formControlName="nomUser" placeholder="{{ unAdministrateur.nomUser }}" /></div>
13         </div>
14         <div class="flex-card-container-row">
15           <div class="flex-card-element labelFormRight">Prénom :</div>
16           <div class="flex-card-element" [formGroup]="modifFormAdmin"><input id="prenomUser" type="text" formControlName="prenomUser" placeholder="{{ unAdministrateur.prenomUser }}" /></div>
17         </div>
18         <div class="flex-card-container-row">
19           <div class="flex-card-element labelFormRight">E-Mail :</div>
20           <div class="flex-card-element" [formGroup]="modifFormAdmin"><input id="emailUser" type="text" formControlName="emailUser" placeholder="{{ unAdministrateur.emailUser }}" /></div>
21         </div>
22         <div class="flex-card-container-row">
23           <div class="flex-card-element labelFormRight">Mot de passe :</div>
24           <div class="flex-card-element" [formGroup]="modifFormAdmin"><input id="passwordUser" type="password" formControlName="passwordUser" placeholder="*****" /></div>
25         </div>
26         <div class="flex-card-container-row">
27           <div class="flex-card-element labelFormRight">téléphone :</div>
28           <div class="flex-card-element" [formGroup]="modifFormAdmin"><input id="telUser" type="text" formControlName="telUser" placeholder="{{ unAdministrateur.telUser }}" /></div>
29         </div>
30         <div class="flex-card-container-row">
31           <div class="flex-card-element labelFormRight">Droit :</div>
32           <div class="flex-card-element" [formGroup]="modifFormAdmin">
33             <select id="idDroitUser" formControlName="idDroitUser">
34               <ng-container *ngFor="let unDroit of lesDroitsAdmins">
35                 <option value="{{ unDroit.idDroit }}" [selected]="nrSelect==unDroit.idDroit">{{ unDroit.libelleDroit }}</option>
36               </ng-container>
37             </select>
38           </div>
39         </div>
40         <div class="flex-card-container-row">
41           <div class="flex-card-element"><form [formGroup]="modifFormAdmin" (ngSubmit)="validModifAdmin()"><button class="button-blue" type="submit">Valider</button></form></div>
42           <div class="flex-card-element"><button class="button-red" (click)="annuler()>Annuler</button></div>
43         </div>
44       </div>
45     </ng-container>

```

Interface de modification d'un administrateur.

Lors de la réalisation de l'interface précédente, une comboBox a été réalisée de manière à pouvoir sélectionner le droit. Une valeur sélectionnée par défaut peut être attribuée à celle-ci en utilisant soit l'attribut « *selected* » uniquement, soit en attribuant à ce dernier une valeur « *true* » ou « *false* ». Ici, la variable « *nrSelect* » contient l'id du droit à sélectionner par défaut. Si l'id correspond à celui présent pour le droit de l'administrateur que l'on souhaite modifier, alors le « *selected* » passera à « *true* » afin de sélectionner la valeur par défaut de la comboBox, sinon il passe à « *false* ».

```

<ng-template #supprAdmin>
  <ng-container *ngIf="idSupprAdmin == unAdministrateur.idUser; else normalList">
    <div class="flex-card-container-row premiere-ligne-suppr">
      <div class="flex-card-element">Voulez-vous supprimer cet administrateur ?</div>
    </div>
    <div class="flex-card-container-row">
      <div class="flex-card-element"><button class="button-blue" (click)="validSupprAdmin(unAdministrateur.idUser)">Oui</button></div>
      <div class="flex-card-element"><button class="button-red" (click)="annuler()">Non</button></div>
    </div>
  </ng-container>
</ng-template>

```

Interface de suppression d'un administrateur.

```

regEmail = new RegExp(/^[a-zA-Z-]+@[a-zA-Z-]+\.[a-zA-Z]{2,6}$/);
regMDP = new RegExp(/^(?=.*[A-Za-z])(?=.*[0-9])(?=.*[$!%*#?&])[A-Za-z\d@#$!%*#?&]{8,})$/);
regTel = new RegExp(/^(0|+33)(6|7)[0-9]{8}$/);
regNomPrenom = new RegExp(/^\w{1,}\w{1,}$/);

```

Regex permettant de vérifier les données. Pour tester une regex, on utilise la méthode « .test([string à tester]) » pour tester si la regex « match » avec une chaîne de caractères.

```

  annuler()
  {
    this.idModifAdmin = null;
    this.idSupprAdmin = null;
    this.ajoutAdmin = false;
  }

```

Procédure permettant le fonctionnement du bouton « Annuler » ou « Non ».

```

<div class="grid-element">
  <ng-container *ngIf="ajoutAdmin === true; else normalAjout">
    <div class="flex-card-container">
      <div class="flex-card-container-row premiere-ligne-form">
        <div class="flex-card-element labelFormRight">Nom :</div>
        <div class="flex-card-element" [formGroup]="ajoutFormAdmin"><input id="nomUser" #nomUser type="text" formControlName="nomUser" placeholder="Nom" /></div>
      </div>
      <div class="flex-card-container-row">
        <div class="flex-card-element labelFormRight">Prénom :</div>
        <div class="flex-card-element" [formGroup]="ajoutFormAdmin"><input id="prenomUser" #prenomUser type="text" formControlName="prenomUser" placeholder="Prénom" /></div>
      </div>
      <div class="flex-card-container-row">
        <div class="flex-card-element labelFormRight">E-Mail :</div>
        <div class="flex-card-element" [formGroup]="ajoutFormAdmin"><input id="emailUser" #emailUser type="text" formControlName="emailUser" placeholder="E-Mail" /></div>
      </div>
      <div class="flex-card-container-row">
        <div class="flex-card-element labelFormRight">Mot de passe :</div>
        <div class="flex-card-element" [formGroup]="ajoutFormAdmin"><input id="passwordUser" #mdpUser type="password" formControlName="passwordUser" placeholder="*****" /></div>
      </div>
      <div class="flex-card-container-row">
        <div class="flex-card-element labelFormRight">Téléphone :</div>
        <div class="flex-card-element" [formGroup]="ajoutFormAdmin"><input id="telUser" #telUser type="text" formControlName="telUser" placeholder="Téléphone" /></div>
      </div>
      <div class="flex-card-container-row">
        <div class="flex-card-element labelFormRight">Droits :</div>
        <div class="flex-card-element" [formGroup]="ajoutFormAdmin"><select id="droitUser" formControlName="idDroitUser">
          <option value="" selected="Droit"></option>
          <ng-container *ngFor="let unDroit of lesDroitsAdmins">
            <option value="{{ unDroit.idDroit }}>{{ unDroit.libelleDroit }}</option>
          </ng-container>
        </select>
      </div>
      <div class="flex-card-container-row">
        <div class="flex-card-element"><form [formGroup]="ajoutFormAdmin" (ngSubmit)="validAjoutAdmin()"><button class="button-blue" type="submit">Valider</button></form></div>
        <div class="flex-card-element"><button class="button-red" (click)="annuler()">Annuler</button></div>
      </div>
    </div>
  </ng-container>
  <ng-template #normalAjout>
    <div class="flex-card-container">
      <div class="flex-card-container-row premiere-ligne-ajout">
        <div class="flex-card-element"><a href="javascript:void(0); " (click)="createAdmin()">+<span class="ajout-text"><br/>Ajouter un nouvel administrateur</span></a></div>
      </div>
    </div>
  </ng-template>
</div>

```

Interface complète d'ajout d'un nouvel administrateur.

Résultats :

Liste des administrateurs :

The screenshot shows a list of administrators with two entries:

- ID : 4**
Nom : yrdy
Prénom : Admin
E-Mail : suadmin@gmail.com
Mot de passe : *****
Téléphone : 1
Droit : superadmin
- ID : 13**
Nom : Admin
Prénom : Boutique
E-Mail : adminbout@gmail.com
Mot de passe : *****
Téléphone : 0658941563
Droit : admin_boutique

Buttons at the bottom: **Modifier** (blue) and **Supprimer** (red).



Ajouter un nouvel administrateur

Affichage normal de l'interface.

Liste des administrateurs :

The screenshot shows a list of administrators with two entries. The entry for ID 4 is highlighted and has its details displayed in a modal-like overlay:

- ID : 4**
Nom : yrdy
Prénom : Admin
E-Mail : suadmin@gmail.cc
Mot de passe : *****
Téléphone : 1
Droit : superadmin

Buttons at the bottom: **Valider** (blue) and **Annuler** (red).



Ajouter un nouvel administrateur

Affichage de l'interface lors d'une modification.

Liste des administrateurs :

The screenshot shows a list of administrators with two entries. The entry for ID 4 is highlighted and has its details displayed in a modal-like overlay. A confirmation dialog box is overlaid on the right side:

Voulez-vous supprimer cet administrateur ?

Oui (blue) **Non** (red)



Ajouter un nouvel administrateur

Affichage de l'interface lors d'une suppression.

Liste des administrateurs :

The screenshot shows a list of administrators with two entries. The entry for ID 4 is highlighted and has its details displayed in a modal-like overlay. A new entry form is overlaid on the right side:

Nom : Nom
Prénom : Prénom
E-Mail : E-Mail
Mot de passe : *****
Téléphone : Téléphone
Droit : Droit

Buttons at the bottom: **Valider** (blue) and **Annuler** (red).



Ajouter un nouvel administrateur

Affichage de l'interface lors d'un ajout.

Le code CSS de cette interface est présent en Annexe 6.

J'ai ensuite créé un nouveau composant (avec la commande « « *ng generate component* ») pour faire le filtre afin de trier les comptes administrateur en fonction de critères spécifiques (en recherchant par nom, prénom, e-mail, numéro de téléphone ou droit).

Un formulaire est donc créé pour la saisie et la récupération des informations.

```
this.filtreForm = this.formBuilder.group({
  nom: '',
  prenom: '',
  email: '',
  tel: '',
  libelleDroit: ''
});
```

Puis le template HTML est codé de manière à afficher le formulaire ainsi que deux boutons, un pour « *Rechercher* » qui formatera les données saisies en JSON et les enverra hors du composant via un « *Output* ». Un autre « *Annuler* » qui permettra d'annuler la recherche en cours et de vider le filtre des données saisies.

```
src > app > filtre-administrateurs > filtre-administrateurs.component.html > h3
1  <h3>Filtre :</h3>
2
3  <div class="flex-filtre-container-column" [formGroup]="filtreForm">
4    <div class="flex-filtre-element"><input id="nom" #nom type="text" formControlName="nom" placeholder="Nom" /></div>
5    <div class="flex-filtre-element"><input id="prenom" #prenom type="text" formControlName="prenom" placeholder="Prénom" /></div>
6    <div class="flex-filtre-element"><input id="email" #email type="text" formControlName="email" placeholder="E-mail" /></div>
7    <div class="flex-filtre-element"><input id="tel" #tel type="text" formControlName="tel" placeholder="Téléphone" /></div>
8    <div class="flex-filtre-element"><select id="libelleDroit" #droit formControlName="libelleDroit">
9      <option value="" selected>Droit</option>
10     <ng-container *ngFor="let unDroit of lesDroits">
11       <option value="{{ unDroit.libelleDroit }}">{{ unDroit.libelleDroit }}</option>
12     </ng-container>
13   </select></div>
14
15   <div class="flex-filtre-element">
16     <button class="button-right button-blue" (click)="validRechercheFiltreAdmin()">Rechercher</button>
17     <button class="button-left button-red" (click)="annulerRechercheFiltreAdmin()">Annuler</button>
18   </div>
19 </div>
```

Interface du filtre dans le template HTML.

Lorsqu'on appuie sur « *Rechercher* », on vérifie s'il y a des données qui sont « *null* » si c'est le cas on remplace par vide (' ') sinon on laisse la valeur saisie. Puis on formate en JSON et on l'envoi. Si aucune donnée n'est saisie alors un message erreur apparait car, comme pour la modification, on doit au moins saisir une information pour lancer une recherche.

```
validRechercheFiltreAdmin()
{
  if (this.filtreForm.value.nom == "" && this.filtreForm.value.prenom == "" && this.filtreForm.value.email == "" && this.filtreForm.value.tel == "" && this.filtreForm.value.libelleDroit == "")
  {
    alert("Pour réaliser une recherche, veuillez au moins saisir un champs.");
  }
  else
  {
    if (this.filtreForm.value.nom == null)
    {
      this.filtreForm.value.nom = '';
    }
    if (this.filtreForm.value.prenom == null)
    {
      this.filtreForm.value.prenom = '';
    }
    if (this.filtreForm.value.email == null)
    {
      this.filtreForm.value.email = '';
    }
    if (this.filtreForm.value.tel == null)
    {
      this.filtreForm.value.tel = '';
    }
    if (this.filtreForm.value.libelleDroit == null)
    {
      this.filtreForm.value.libelleDroit = '';
    }
    let json;
    if (this.filtreForm.value.nom == "" && this.filtreForm.value.prenom == "" && this.filtreForm.value.email == "" && this.filtreForm.value.tel == "" && this.filtreForm.value.libelleDroit == "")
    {
      json = null;
    }
    else
    {
      json = JSON.parse("{\"nom\": " + this.filtreForm.value.nom + ", \"prenom\": " + this.filtreForm.value.prenom + ", \"email\": " + this.filtreForm.value.email + ", \"tel\": " + this.filtreForm.value.tel + ", \"libelleDroit\": " + this.filtreForm.value.libelleDroit + "}");
    }
    this.filtreOutAdmin.emit(json);
  }
}
```

Procédure appelé lors de l'appui sur le bouton « Rechercher ».

Lorsqu'on appuie sur « Annuler » si aucun champ n'est saisi, alors un message d'erreur apparaît et cela signifie qu'il n'y a aucune recherche en cours. Sinon, on envoie une variable « null » dans l'« Output » de manière à annuler la recherche précédente et les champs du filtre sont vidés en utilisant les « *ViewChild* » pour vider les inputs et le « *.reset()* » pour vider le formulaire.

```
annulerRechercheFiltreAdmin()
{
  if (this.filtreForm.value.nom == "" && this.filtreForm.value.prenom == "" && this.filtreForm.value.email == "" && this.filtreForm.value.tel == "" && this.filtreForm.value.libelleDroit == "")
  {
    alert("Aucune recherche en cours.");
  }
  else
  {
    this.filtreForm.reset();

    this.filtreForm.value.nom = '';
    this.filtreForm.value.prenom = '';
    this.filtreForm.value.email = '';
    this.filtreForm.value.tel = '';
    this.filtreForm.value.libelleDroit = '';

    this.nomInput.nativeElement.value = '';
    this.prenomInput.nativeElement.value = '';
    this.emailInput.nativeElement.value = '';
    this.telInput.nativeElement.value = '';
    this.droitInput.nativeElement.value = '';

    let json = null;

    this.filtreOutAdmin.emit(json);
  }
}
```

Procédure appelée lors de l'appui sur le bouton « Annuler ».

```
@ViewChild('nom') nomInput : any;
@ViewChild('prenom') prenomInput : any;
@ViewChild('email') emailInput : any;
@ViewChild('tel') telInput : any;
@ViewChild('droit') droitInput : any;
```

*Déclarations des « *ViewChild* » pour le filtre.*

```
@Output() filtreOutAdmin = new EventEmitter<any>();
```

*Déclaration de l'« *Output* » qui va permettre d'envoyer le filtre en dehors du composant, soit à un autre composant.*

Résultat :

Filtre :

| | |
|--|---|
| Nom | |
| Prénom | |
| E-mail | |
| Téléphone | |
| Droit | |
| Rechercher | Annuler |

Le code du style est présent à l'annexe 7.

J'ai ensuite réalisé un gestionnaire uniquement pour les autres utilisateurs (sans les admin). Il est réalisé de la même manière que celui précédemment (pour les administrateurs). Ce qui a uniquement changé c'est les données que l'on va manipuler.

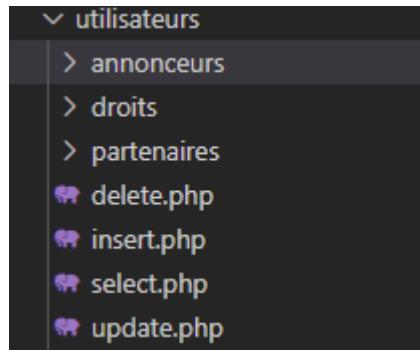
J'ai donc créé un nouveau modèle pour les utilisateurs.

```
src > app > model > Users > Users.ts
```

```
1  export class Users {
2    idUser? : number;
3    nomUser? : string;
4    prenomUser? : string;
5    emailUser? : string;
6    passwordUser? : string;
7    telUser? : string;
8    tokenInsc? : string;
9    idDroitUser? : number
10
11   constructor (idUser:number, unNomUser:string, unPrenomUser:string, unEmailUser:string, unPasswordUser:string, unTelUser:string, unTokenInsc:string, unIdDroitUser:number)
12   {
13     this.idUser = idUser;
14     this.nomUser = unNomUser;
15     this.prenomUser = unPrenomUser;
16     this.emailUser = unEmailUser;
17     this.passwordUser = unPasswordUser;
18     this.telUser = unTelUser;
19     this.tokenInsc = unTokenInsc;
20     this.idDroitUser = unIdDroitUser
21   }
22 }
```

Fichier « Users.ts ».

Puis j'ai modifié le CRUD en reprenant tout ce qui a été fait auparavant mais en réadaptant pour ne pas avoir les administrateurs.



CRUD pour les utilisateurs.

```
SELECT iduser, nomUser, prenomUser, emailUser, passwordUser, telUser, tokenInsc, libelleDroit
FROM user INNER JOIN droits ON idDroit = idDroitUser
WHERE libelleDroit NOT IN ('superadmin','admin_boutique','admin_logistique')
```

Requête SQL pour récupérer les utilisateurs non administrateurs.

```
"SELECT idDroit, libelleDroit FROM droits WHERE libelleDroit NOT IN ('superadmin','admin_boutique','admin_logistique')";
```

Requête SQL permettant de récupérer les droits non admin.

Enfin j'ai rajouté les nouvelles requêtes HTTP dans le service « pannel-administration.service.ts ».

```
getUsers()
{
    return this.http.get<Users[]>(this.baseUrl+'utilisateurs/select.php');
}

getUnUser(id:any)
{
    return this.http.get<Users[]>(this.baseUrl+'utilisateurs/select.php?id='+id);
}

modifUser(user:any)
{
    return this.http.put(this.baseUrl+'utilisateurs/update.php', user);
}

supprUser(id:any)
{
    return this.http.delete(this.baseUrl+'utilisateurs/delete.php?id='+id);
}

ajoutUser(user:any)
{
    return this.http.post(this.baseUrl+'utilisateurs/insert.php', user);
}

getDroits()
{
    return this.http.get<Droits[]>(this.baseUrl+'utilisateurs/droits/select.php');
}
```

J'ai aussi réalisé un filtre conçu de la même manière que celui utilisé pour les administrateurs.

Résultat :

The screenshot shows a user management interface. At the top, there is a 'Filtre:' section with input fields for Nom, Prénom, E-mail, and Téléphone, and a dropdown menu for Droit. Below it is a 'Rechercher' button and an 'Annuler' button. The main area is titled 'Liste des utilisateurs:' and displays three user cards:

- User 1:**
ID : 3
Nom : Test
Prénom : Marketplace
E-Mail : client@gmail.com
Mot de passe : *****
Téléphone : 443534234
Droit : client
- User 2:**
ID : 1
Nom : Annonceur
Prénom : 1
E-Mail : anno1@gmail.com
Mot de passe : *****
Téléphone : 0673424323
Droit : annonceur
- User 3:**
ID : 2
Nom : Annonceur
Prénom : 2
E-Mail : anno2@gmail.com
Mot de passe : *****
Téléphone : 042372392
Droit : annonceur

At the bottom right of the interface is a card with a blue plus sign and the text "ajouter un nouvel utilisateur".

Gestionnaire des utilisateurs mais sans les administrateurs.

Lors d'un ajout ou d'une modification d'un administrateur ou d'un utilisateur, il est vérifié si les données saisies (nom et prénom ou adresse e-mail ou numéro de téléphone) ne correspondent pas à un utilisateur (administrateur ou utilisateur) déjà existant.

```
lesAdministrateurs : any;
lesUtilisateurs : any;
```

On vient automatiquement récupérer dans les deux gestionnaires tous les utilisateurs et les administrateurs afin d'avoir l'ensemble des comptes présents en BDD.

```
for (let unAdministrateur of this.lesAdministrateurs)
{
  if (unAdministrateur.idUser != this.unAdministrateurVerifExiste.idUser)
  {
    if ((unAdministrateur.nomUser == this.modifFormAdmin.value.nomUser && unAdministrateur.prenomUser == this.modifFormAdmin.value.prenomUser) || (unAdministrateur.emailUser == this.modifFormAdmin.value.emailUser) || (unAdministrateur.telUser == this.modifFormAdmin.value.telUser))
    {
      verifAdministrateurExiste = true;
    }
  }
}

if (verifAdministrateurExiste)
{
  for (let unUtilisateur of this.lesUtilisateurs)
  {
    if ((unUtilisateur.nomUser == this.modifFormAdmin.value.nomUser && unUtilisateur.prenomUser == this.modifFormAdmin.value.prenomUser) || (unUtilisateur.emailUser == this.modifFormAdmin.value.emailUser) || (unUtilisateur.telUser == this.modifFormAdmin.value.telUser))
    {
      verifAdministrateurExiste = true;
    }
  }
}
```

Puis, que ce soit pour la modification ou l'ajout, on vérifie sur l'ensemble des comptes si les données saisies ne correspondent pas à un utilisateur ou à un administrateur déjà existant en BDD. En revanche, dans le cadre de l'ajout, cette vérification est faite uniquement s'il y a des utilisateurs présents dans la BDD (puisque'on récupère précédemment les données, on vérifie si la variable contenant ces données est vide ou « undefined »).

Par la suite, j'ai réalisé un nouveau composant qui contiendra une petite barre de navigation qui sera interne à la page de gestion des utilisateurs permettant ainsi de naviguer entre le gestionnaire des utilisateurs et le gestionnaire des administrateurs. J'utilise donc deux « *Outputs* » de manière à pouvoir émettre un événement afin de pouvoir changer de gestionnaire. Deux variables booléennes sont présentes principalement pour changer le style de la barre.

```
src > app > nav-gestion-utilisateurs > nav-gestion-utilisateurs.component.ts > ...
1   import { Component, Output, EventEmitter } from '@angular/core';
2
3   @Component({
4     selector: 'app-nav-gestion-utilisateurs',
5     templateUrl: './nav-gestion-utilisateurs.component.html',
6     styleUrls: ['./nav-gestion-utilisateurs.component.scss']
7   })
8   export class NavGestionUtilisateursComponent {
9     @Output() gestionUtilisateurs = new EventEmitter();
10    @Output() gestionAdministrateurs = new EventEmitter();
11
12    gestionUtilClick : boolean = true;
13    gestionAdminClick : boolean = false;
14
15    constructor()
16    {
17    }
18
19    lienAccesGestionUtilisateurs()
20    {
21      this.gestionUtilisateurs.emit();
22      this.gestionAdminClick = false;
23      this.gestionUtilClick = true;
24    }
25
26    lienAccesGestionAdministrateurs()
27    {
28      this.gestionAdministrateurs.emit();
29      this.gestionAdminClick = true;
30      this.gestionUtilClick = false;
31    }
32  }
```

Partie logique du composant « nav-gestion-utilisateurs.component.ts ».

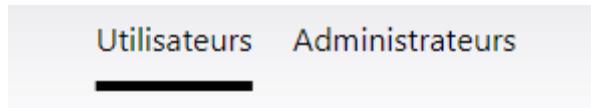
```
src > app > nav-gestion-utilisateurs > nav-gestion-utilisateurs.component.html > div.nav-bar
1  <div class="nav-bar">
2    <ul>
3      <li>
4        <ng-container *ngIf="gestionUtilClick; else normalAffichageUtil">
5          <a class="lienFocus" href="javascript:void(0);" (click)="lienAccesGestionUtilisateurs()">Utilisateurs</a>
6        </ng-container>
7
8        <ng-template #normalAffichageUtil>
9          <a class="lienDefaut" href="javascript:void(0);" (click)="lienAccesGestionUtilisateurs()">Utilisateurs</a>
10         </ng-template>
11      </li>
12      <li>
13        <ng-container *ngIf="gestionAdminClick; else normalAffichageAdmin">
14          <a class="lienFocus" href="javascript:void(0);" (click)="lienAccesGestionAdministrateurs()">Administrateurs</a>
15        </ng-container>
16
17        <ng-template #normalAffichageAdmin>
18          <a class="lienDefaut" href="javascript:void(0);" (click)="lienAccesGestionAdministrateurs()">Administrateurs</a>
19        </ng-template>
20      </li>
21    </ul>
22  </div>
```

Template HTML du composant « nav-gestion-utilisateurs.component.ts ».

```
src > app > nav-gestion-utilisateurs > nav-gestion-utilisateurs.component.scss > ul
1  .nav-bar
2  {
3    margin-top: 100px;
4  }
5
6  ul
7  {
8    text-align: center;
9    padding: 0;
10   margin: 0;
11  }
12
13 li
14 {
15   display: inline;
16   margin-left: 10px;
17   margin-right: 10px;
18 }
19
20 .lienDefaut
21 {
22   position: relative;
23   color: black;
24   text-decoration: none;
25 }
26
27 .lienDefaut::after
28 {
29   content: '';
30   position: absolute;
31   width: 100%;
32   transform: scaleX(0);
33   height: 5px;
34   top: 32px;
35   left: 0;
36   background-color: black;
37   transform-origin: bottom right;
38   transition: transform 0.25s ease-out;
39 }
40
41 .lienDefaut:hover::after
42 {
43   transform: scaleX(1);
44   transform-origin: bottom left;
45 }
46
47 .lienFocus
48 {
49   text-decoration: underline;
50   text-underline-offset: 15px;
51   text-decoration-color: black;
52   color: black;
53   text-decoration-style: solid;
54   text-decoration-thickness: 5px;
55 }
```

Style de la barre de navigation.

Résultat :



Tous les composant sont maintenant créés, il ne reste plus qu'à créer le composant « gestion-utilisateurs » qui sera notre page réunissant tous nos composants. Il permettra aussi de servir de support d'échange entre le filtre et la liste ainsi que d'afficher le bon gestionnaire au moment de cliquer sur un lien de la barre de navigation précédemment créée.

```
filtreRecu : any = null;
filtreRecuAdmin : any = null;
```

Déclaration des variables qui vont récupérer les données envoyées par les « Outputs » des filters utilisateurs et administrateurs.

```
gestionUtilisateurs : boolean = true;
gestionAdministrateurs : boolean = false;
```

Grâce aux « EventEmitter » de la barre de navigation, ces deux booléens vont permettre de définir vers quel gestionnaire l'utilisateur veut se rediriger. Par défaut, c'est le gestionnaire des utilisateurs qui est affiché donc la valeur de la variable le représentant est à « true ».

```
accesGestionUtilisateurs()
{
    this.gestionUtilisateurs = true;
    this.gestionAdministrateurs = false;
}

accesGestionAdministrateurs()
{
    this.gestionUtilisateurs = false;
    this.gestionAdministrateurs = true;
}

async receptionFiltre(filtre:any)
{
    this.filtreRecu = filtre;
}

async receptionFiltreAdmin(filtre:any)
{
    this.filtreRecuAdmin = filtre;
}
```

Procédure permettant de récupérer les filters et de rediriger vers le bon gestionnaire.

```

ngOnInit(): void {
  if (this.authService.isAuthenticated())
  {
    this.connected = true;
    this.user = JSON.parse(localStorage.getItem('authUser')!);

    if (this.user.roles != 'superadmin')
    {
      this.connected = false;
      this.router.navigate(['/405']);
    }
  }
  else
  {
    this.connected = false;
    this.router.navigate(['/login']);
  }
}

```

Partie du code qui permet de vérifier si l'utilisateur connecté est bien un super administrateur et s'il est bien toujours connecté, on met aussi cette partie du code dans le ngOnChanges pour qu'à la moindre modification, la vérification se fasse automatiquement et déconnecte l'utilisateur si le token est expiré.

```

src > app > gestion-utilisateurs > gestion-utilisateurs.component.html > ng-container
1  <ng-container *ngIf="connected == true && user.roles == 'superadmin'">
2    <app-nav-gestion-utilisateurs
3      (gestionUtilisateurs)="accesGestionUtilisateurs()"
4      (gestionAdministrateurs)="accesGestionAdministrateurs()"
5    ></app-nav-gestion-utilisateurs>
6
7    <ng-container *ngIf="gestionUtilisateurs === true">
8      <app-filtre-utilisateurs
9        (filtreOut)="receptionFiltre($event)"></app-filtre-utilisateurs>
10     <app-list-utilisateurs
11       [filtreIn]="filtreRecu"></app-list-utilisateurs>
12   </ng-container>
13
14   <ng-container *ngIf="gestionAdministrateurs === true">
15     <app-filtre-administrateurs
16       (filtreOutAdmin)="receptionFiltreAdmin($event)"></app-filtre-administrateurs>
17     <app-list-administrateurs
18       [filtreInAdmin]="filtreRecuAdmin"></app-list-administrateurs>
19   </ng-container>
20 </ng-container>

```

Template HTML où l'on voit que la barre de navigation va faire appel aux procédures pour indiquer vers quel gestionnaire l'utilisateur souhaite se rediriger. On voit aussi que les filtres font appels à leurs fonctions adéquates permettant ainsi de transmettre les données du filtre au composant de la page.

Ce dernier retransmet les données reçus aux listes adéquates.

Enfin, il suffit de reprendre le code de la liste des administrateurs et des utilisateurs pour ajouter un « *Input* » afin de récupérer les données envoyées par le filtre. Puisqu'il s'agit d'un objet JSON, il suffit de regarder pour chaque propriété non vide si elle correspond bien aux données présent en BDD. Si ce qui est saisi à une taille inférieure à ce qui est en BDD, alors je réduis la taille des données comparées en la mettant à la même longueur, ce qui permet alors de ne pas tout taper. Pour pouvoir réaliser cette comparaison, il faut utiliser un nouveau cycle de vie d'un composant : le « *OnChanges* ». Il permet lors d'un changement réalisé dans le composant, de recharger/remonter celui-ci. Il fonctionne notamment avec les « *Inputs/Outputs* », permettant ainsi lors de la réception de nouvelles données par l'« *Input* » ou par l'envoi de nouvelles données via l'« *Output* » tout au long du fonctionnement du composant (changements pouvant intervenir à tout moment), de réagir et recharger le composant en conséquence et réaliser de nouvelles actions. On l'importe et on l'implémente dans la classe du composant de la même manière que le « *OnInit* ». La liste des administrateurs est vidée avant la vérification. Si un administrateur dispose des mêmes données que celles saisies dans le filtre, alors on l'ajoute dans la liste des administrateurs. Le code de la comparaison se trouve à l'annexe 8.

Résultat final :

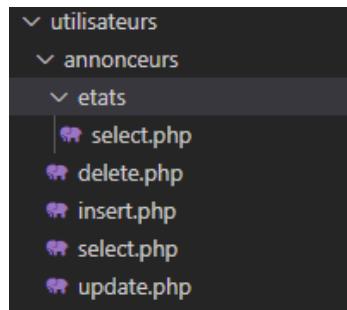
| ID | Nom | Prénom | E-mail | Mot de passe | Téléphone | Rôle |
|----|-------------|-----------|------------------|--------------|------------|----------|
| 3 | Marketplace | Test | client@gmail.com | ***** | 443534234 | client |
| 1 | annoncer | Annonceur | anno1@gmail.com | ***** | 0673424323 | annoncer |
| 2 | anno2 | Annonceur | anno2@gmail.com | ***** | 042372392 | annoncer |
| | | | | | | |

II) Application Web :

3. Administration.
- b. La gestion des annonceurs.

Le gestionnaire des annonceurs est conçu de la même manière que les interfaces précédentes pour les utilisateurs et les administrateurs. Ce qui change c'est les données affichées et manipulées ainsi que les critères de filtrage. Pour le reste du code, tout reste inchangé.

Dans un premier temps, on vient ajouter au CRUD les fichiers et les codes nécessaires à l'affichage, la modification, la suppression et l'ajout des données des annonceurs en BDD.



Puis on vient ajouter au service du panel d'administration, les requêtes HTTP nécessaires pour interagir avec le CRUD.

```
getLesAnnonceurs()
{
  return this.http.get<Annonceurs[]>(this.baseUrl+'utilisateurs/annonceurs/select.php');
}

getUnAnnonceur(id:any)
{
  return this.http.get<Annonceurs[]>(this.baseUrl+'utilisateurs/annonceurs/select.php?id='+id);
}

modifAnno(anno:any)
{
  return this.http.put(this.baseUrl+'utilisateurs/annonceurs/update.php', anno);
}

supprAnno(id:any)
{
  return this.http.delete(this.baseUrl+'utilisateurs/annonceurs/delete.php?id='+id);
}

ajoutAnno(anno:any)
{
  return this.http.post(this.baseUrl+'utilisateurs/annonceurs/insert.php', anno);
}

getLesEtatsAnnonceurs()
{
  return this.http.get<etatsComptes[]>(this.baseUrl+'utilisateurs/annonceurs/etats/select.php');
}

getUnEtatAnnonceurs(id:number)
{
  return this.http.get<etatsComptes[]>(this.baseUrl+'utilisateurs/annonceurs/etats/select.php?id='+id);
}
```

Ensuite, une liste est créée pour afficher les annonceurs présents en base de données. La seule particularité est qu'il y a une image de profil à afficher.

```
<div class="flex-card-container-row premiere-ligne">
    <div class="flex-card-element"></div>
```

Je vais devoir réaliser un uploader de fichier sur le serveur afin d'importer une image.

Pour cela, je réalise le code suivant qui va me permettre d'envoyer un nom de fichier, un chemin et un fichier afin d'importer l'image de profile sur le serveur.

```
crud > uploadeur > uploadeur.php
1  <?php
2      if ($_POST['nvNomFichier'] != "")
3      {
4          $filename = $_POST['nvNomFichier'];
5      }
6      else
7      {
8          $filename = $_FILES['fichier']['name'];
9      }
10
11     $location = "../../".$_POST['chemin']."/".$filename;
12
13     if (move_uploaded_file($_FILES['fichier']['tmp_name'], $location))
14     {
15         echo 'Success';
16     }
17     else
18     {
19         echo 'Failure';
20     }
21 ?>
```

Puis je viens ajouter la requête HTTP dans le service du panel d'administration afin de pouvoir interagir avec l'uploader.

```
async uploadeur(fichier:any,chemin:any,nvNomFichier:any)
{
    let formData = new FormData();

    formData.append("fichier", fichier.files[0]);
    formData.append("chemin", chemin);
    formData.append("nvNomFichier", nvNomFichier);

    await fetch(this.baseUrl+'uploadeur/uploadeur.php', {method: "POST", body: formData});
}
```

Enfin, il me suffit juste de réaliser dans le formulaire d'ajout et de modification un input HTML de type « file » qui va me permettre de saisir un fichier et de vérifier lors de la validation si un fichier a bien été saisi.

```
const logo : any = (<HTMLInputElement>document.getElementById("logoImgModif"));
```

Pour la vérification, on vient récupérer le fichier saisi.

```
if (typeof logo.files[0] != "undefined")
{
    const chemin = "src/assets/img/logo";
    let nomFichier;

    if (this.modifAnnoForm.value.libelleNomAnnonceur == "")
    {
        nomFichier = this.nomAnnonceur + logo.files[0].name.substr(logo.files[0].name.indexOf('.'),logo.files[0].name.length);
    }
    else
    {
        nomFichier = this.modifAnnoForm.value.libelleNomAnnonceur + logo.files[0].name.substr(logo.files[0].name.indexOf('.'),logo.files[0].name.length);
    }

    this.pannelAdmin.uploadateur(logo,chemin,nomFichier);

    this.modifAnnoForm.value.logoAnnonceur = chemin + "/" + nomFichier;
}
```

Puis on vient saisir le chemin et le nouveau nom du fichier afin d'importer ce dernier au chemin avec le nom voulu. La particularité ici est que l'on vient récupérer l'extension du fichier présent dans le nom du fichier saisi.

Pour le reste, rien ne change. L'interface permettra de modifier, de supprimer ou d'ajouter un annonceur. Les données sont vérifiées et, comme pour les interfaces précédentes, lors de la modification on n'est pas obligé de saisir toutes les champs mais il faut au moins en saisir un et pour l'ajout, on est obligé de saisir tous les champs. Les messages d'erreurs sont exactement les mêmes et le filtre ainsi que la page de gestion sont conçus de la même manière sauf que pour le filtre, on ne pourra trier uniquement que par état du compte. Une barre de navigation est réalisée aussi de la même manière afin de pouvoir par la suite ajouter le gestionnaire des annonces.

Résultat :

The screenshot shows the 'ANNONCEUR' section of the Marketplace application. It lists three advertisers:

- Advertiser 1:** Compte: Annonceur 1 (anno1@gmail.com), Nom: hoff hgf, Adresse: (), Etat: Refusé. Actions: Modifier, Supprimer.
- Advertiser 2:** Compte: Annonceur 2 (anno2@gmail.com), Nom: fkdqsjnfldko, Adresse: fkdslm 4 Erevous (27190) Azerbaïdjan, Etat: En Attente. Actions: Modifier, Supprimer.
- Advertiser 3:** Compte: Test Marketplace (client@gmail.com), Nom: vhjzgs, Adresse: zara 4 zaza (27190) Fidji, Etat: Refusé. Actions: Modifier, Supprimer.

A large blue plus sign icon is visible in the fourth slot on the right, indicating where to click to add a new advertiser. The URL in the browser is localhost:4200/gestionAnnonceurs.

II) Application Web :

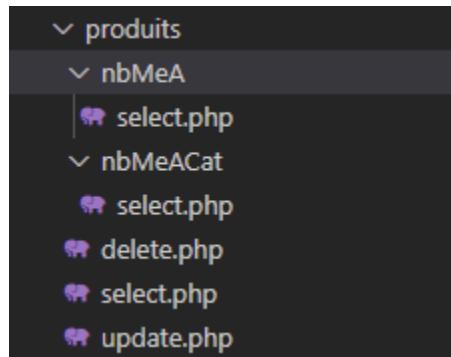
3. Administration.
- c. La gestion des produits.

Le gestionnaire des produits est conçu de la même manière que les interfaces précédentes. La particularité est que l'on ne peut pas ajouter de nouveau produit ni modifier les données contenues en BDD, on ne peut qu'en supprimer. Néanmoins, un bouton « Modifier » est quand même présent car on va pouvoir modifier la mise en avant global et par catégorie d'un produit.

Il faut savoir que la page d'accueil permettra d'afficher 10 produits mis en avant globalement et 10 par catégorie.

Notre tuteur nous a fait part du besoin que l'administrateur puisse choisir ces produits mis en avant. Il est donc de mon rôle de m'en occuper.

Dans un premier temps, j'ai modifié mon CRUD afin de pouvoir récupérer les informations des produits, supprimer un produit et mettre à jour les booléens en BDD gérant la mise en avant global et par catégorie.



La particularité ici est que j'ai dû réaliser deux fichier « *select.php* » qui vont me permettre de récupérer le nombre de produit mis en avant globalement et par catégorie.

```
$sql = "SELECT COUNT(*) AS nbMiseEnAvant FROM produits WHERE miseEnAvant = 1";
```

Requête SQL permettant de récupérer le nombre de produit mis en avant.

```
"SELECT idCategorieProduit, COUNT(*) AS nbMiseEnAvantCat FROM produits WHERE idCategorieProduit = $categorie_id AND miseEnAvantCat = 1 GROUP BY idCategorieProduit"
```

Requête SQL permettant de récupérer le nombre de produit mis en avant pour une catégorie.

Puis j'ai ajouté les nouvelles requêtes HTTP me permettant d'interagir avec le CRUD.

```
getNbMeACat(id:number)
{
  return this.http.get(this.baseUrl+'produits/nbMeACat/select.php?id='+id);
}

getNbMeA()
{
  return this.http.get(this.baseUrl+'produits/nbMeA/select.php');
}

getLesProduits()
{
  return this.http.get<Produits[]>(this.baseUrl+'produits/select.php');
}

getUnProduits(id:number)
{
  return this.http.get<Produits[]>(this.baseUrl+'produits/select.php?id='+id);
}

modifProduit(produit:any)
{
  return this.http.put(this.baseUrl+'produits/update.php', produit);
}

supprProduit(id:any)
{
  return this.http.delete(this.baseUrl+'produits/delete.php?id='+id);
}
```

Ensute, dans le template HTML et sur l'interface de modification, j'utilise des checkbox afin de saisir si un produit sera mis en avant par catégorie et/ou globalement ou non.

```
<div class="flex-card-container-row">
  <div class="flex-card-element"><span class="labelForm labelCheckBow">Mise en avant ?</span></div>
  <div class="flex-card-element checkFlex"><input type="checkbox" id="miseEnAvant" #miseEnAvant /></div>
</div>

<div class="flex-card-container-row">
  <div class="flex-card-element"><span class="labelForm labelCheckBowLong">Mise en avant par catégorie ?</span></div>
  <div class="flex-card-element checkFlexLong"><input type="checkbox" id="miseEnAvantCat" #miseEnAvantCat /></div>
</div>
```

Lorsque l'utilisateur appuiera sur le bouton « Modifier » d'un produit, le composant viendra récupérer les informations du produit à modifier afin qu'il puisse dans un premier temps vérifier si le produit n'est pas déjà mis en avant globalement et/ou pour sa catégorie, si c'est le cas alors les cases adéquates seront cochées.

```
this.pannelAdmin.getUnProduits(id).subscribe(
  (result:any)=>{
    this.changeDetector.detectChanges();

    if (result.data.miseEnAvant == 1)
    {
      this.checkMeA.nativeElement.checked = true;
    }

    if (result.data.miseEnAvantCat == 1)
    {
      this.checkMeACat.nativeElement.checked = true;
    }
  }
);
```

Puis le composant viendra récupérer le nombre de produit mis en avant globalement et pour la catégorie du produit afin de vérifier s'il n'y a pas déjà 10 produits sélectionnés pour ces deux cas. Si oui, alors la ou les cases qui ne sont pas cochées pour ce produit seront désactivées pour que l'utilisateur ne puisse pas ajouter un autre produit mis en avant. Mais il pourra toujours modifier les cases des produits déjà mis en avant pour les désactiver par exemple.

```
this.pannelAdmin.getNbMeACat(result.data.idCategorieProduit).subscribe(  
  (result:any)=>{  
    if (typeof result.data != "undefined")  
    {  
      if (result.data.nbMiseEnAvantCat >= 10)  
      {  
        if (this.checkMeACat.nativeElement.checked == false)  
        {  
          this.checkMeACat.nativeElement.disabled = true;  
        }  
      }  
    }  
  );  
  
this.pannelAdmin.getNbMeA().subscribe(  
  (result:any)=>{  
    if (typeof result.data != "undefined")  
    {  
      if (result.data.nbMiseEnAvant >= 10 && typeof result.data != "undefined")  
      {  
        if (this.checkMeA.nativeElement.checked == false)  
        {  
          this.checkMeA.nativeElement.disabled = true;  
        }  
      }  
    }  
  );
```

Par la suite, lors de la validation, on vient attribuer en valeur des champs « *miseEnAvant* » et « *miseEnAvantCat* » 0 ou 1 en fonction de si la case est cochée ou non.

```
if (this.checkMeA.nativeElement.checked)  
{  
  this.produitModifForm.value.miseEnAvant = 1;  
}  
else  
{  
  this.produitModifForm.value.miseEnAvant = 0;  
}  
  
if (this.checkMeACat.nativeElement.checked)  
{  
  this.produitModifForm.value.miseEnAvantCat = 1;  
}  
else  
{  
  this.produitModifForm.value.miseEnAvantCat = 0;  
}
```

Puis on vient enregistrer en base de données les choix pour la mise en avant globale et par catégorie du produit voulu de la même manière que l'on mettrait les données saisies des interfaces précédentes en BDD mais sans vérification puisque les valeurs retournées seront forcément 0 ou 1. L'avantage ici est que l'on n'a pas de message d'erreur à gérer. Un bouton annuler est quand même prévu pour annuler l'affichage de l'interface de modification. La suppression et le filtre fonctionnent de la même manière que les interfaces précédentes. En revanche pour le filtre, on ne peut trier que par annonceur ou par catégorie.

Résultat :

| ID | Announceur | Nom | Quantité | Poids | Prix Unitaire (HT) | Catégorie | TVA | Mise en avant? | Mise en avant par catégorie? |
|----|------------|-------------------|----------|---------|--------------------|-----------|-------------|----------------|------------------------------|
| 12 | hgff hgff | Iphone 13 Pro Max | 20 | 1.00 KG | 1129.00 € | iPhone | TVA Normale | Non | Non |
| 9 | vhjgzs | ffswqlqd | 10 | 3.00 KG | 3.00 € | iPhone | TVA Normale | checked | checked |

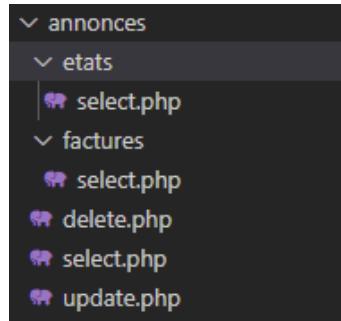
II) Application Web :

3. Administration.
- d. La gestion des annonces.

Le gestionnaire des annonces a été conçu de la même manière que les autres interfaces. Mais comme pour le gestionnaire des produits, l'utilisateur ne pourra pas ajouter une nouvelle annonce ni modifier les informations présentes en BDD. Il pourra uniquement supprimer une annonce. Un bouton « Modifier » est quand même présent afin de pouvoir modifier uniquement l'état de l'annonce.

Lorsqu'un annonceur mettra une annonce en ligne sur la marketplace, son état sera automatiquement mis « En Attente ». Cela signifie que l'annonce existe en BDD mais elle ne sera pas affichée sur le site. C'est alors que l'administrateur pourra décider si l'annonce est valide afin de pouvoir l'afficher sur le site ou non.

Dans un premier temps, je viens créer le CRUD qui va me permettre de modifier, supprimer et récupérer les informations d'une annonce. On vient récupérer les états des annonces et les factures pour récupérer des informations dont l'annonceur.



Puis je viens déclarer dans le panel d'administration les requêtes HTTP afin d'interagir avec le CRUD pour l'administration des annonces.

```
getLesAnnonces()
{
    return this.http.get<Annonces[]>(this.baseUrl+'annonces/select.php');
}

getUneAnnonce(idProduit:number,idFacture:number)
{
    return this.http.get<Annonces[]>(this.baseUrl+'annonces/select.php?idProduit='+idProduit+'&idFacture='+idFacture);
}

getLesAnnoncesFactures()
{
    return this.http.get<facturesAnnonces[]>(this.baseUrl+'annonces/factures/select.php');
}

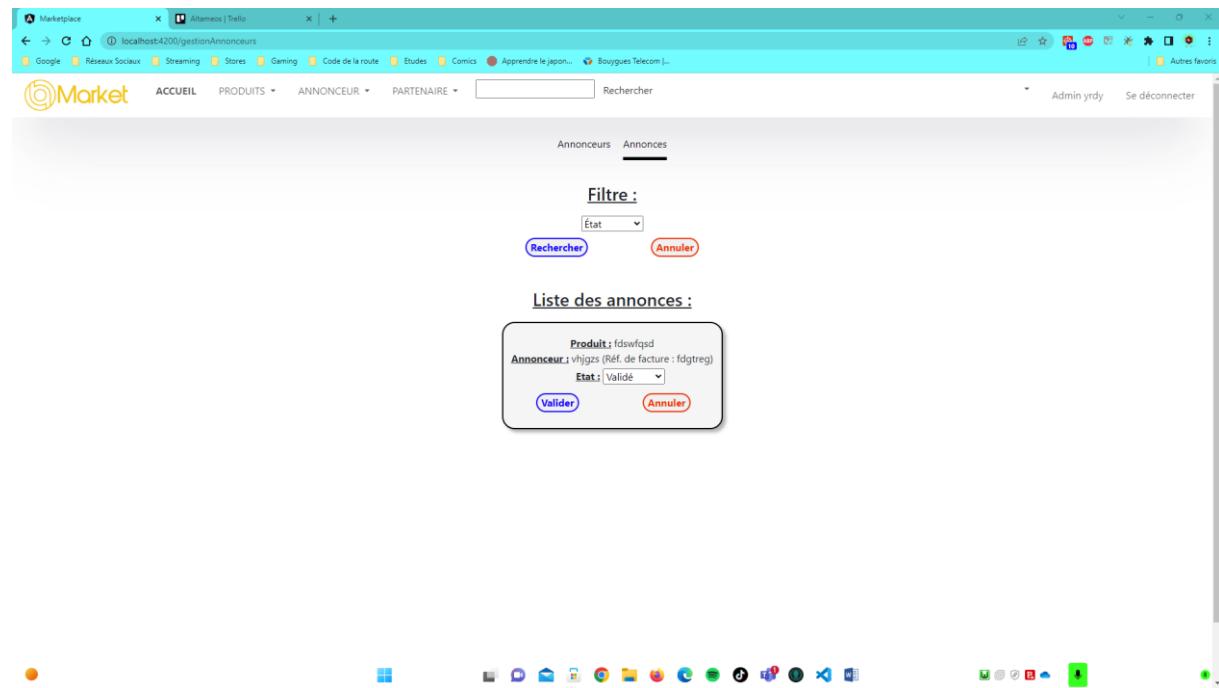
getLesEtatsAnnonces()
{
    return this.http.get<etatsAnno[]>(this.baseUrl+'annonces/etats/select.php');
}

modifEtatAnnonce(Anno:any)
{
    return this.http.put(this.baseUrl+'annonces/update.php', Anno);
}

supprAnnonce(idProduit:number,idFacture:number)
{
    return this.http.delete(this.baseUrl+'annonces/delete.php?idProduit='+idProduit+"&idFacture="+idFacture);
}
```

Le bouton « *Modifier* » permet d'accéder à l'interface de modification de l'état de l'annonce. Il s'agit tout simplement d'une comboBox répertoriant tous les états possibles pour une annonce. Le bouton supprimer ainsi que l'interface de suppression sont conçus de la même manière que précédemment. Le filtre l'est aussi à la différence que l'utilisateur ne pourra trier les annonces que par état. L'accès au gestionnaire des annonces a été ajouté dans la barre de navigation de la page du gestionnaire des annonceurs.

Résultat :



II) Application Web :

4. Annonceurs.
 - a. Inscription.

Lors de la réalisation de l'inscription de l'annonceur, mon binôme et moi, nous nous sommes rendu compte que l'inscription des annonceurs et des partenaires étaient la même tâche : proposer une interface permettant à un utilisateur de saisir ses informations et une autre permettant à l'administrateur de valider ou refuser l'inscription de l'utilisateur en tant que partenaire ou annonceur.

Étant donné que mon binôme avait déjà fait l'interface d'inscription du partenaire du côté client et que moi j'avais déjà fait l'interface de gestion des annonceurs, nous nous sommes échangés les tâches : mon binôme devait s'occuper de l'interface d'inscription des annonceurs côté client en plus de celle pour les partenaires et moi je devais m'occuper de l'interface où l'administrateur validerait ou non l'inscription d'un utilisateur en tant qu'annonceur et, en plus, en tant que partenaire.

Dans un premier temps j'ai donc repris mon gestionnaire des annonceurs qui permet donc déjà de modifier l'état du compte (s'il est validé, refusé ou en attente).

Ce qu'il faut c'est que tant que le compte n'est pas validé, l'utilisateur ne pourra donc pas se connecter avec son compte en tant qu'annonceur ou partenaire.

Pour cela, nous allons utiliser le token d'inscription réalisé par un autre binôme. Ce token est enregistré en BDD et tant que sa valeur n'est pas à 0, l'utilisateur ne peut pas se connecter avec ses identifiants.

J'ai donc réalisé une requête HTTP dans le panel d'administration qui me permet de générer un token d'inscription via l'adresse e-mail du compte s'inscrivant en tant que partenaire ou annonceur.

```
getTokenInscr(email:string)
{
  return this.http.get(this.baseUrl+'generatetoken.php?email=' + email);
}
```

Lors de la validation sur l'interface de modification, le contrôleur regarde si l'état choisi est « Validé », dans ce cas là le token sera mis à 0, sinon un token sera générer avec l'e-mail du compte. Dans tous les cas, le token sera enregistrer en BDD pour le compte voulu.

```
this.pannelAdmin.getUnEtatAnnonceurs(this.modifAnnoForm.value.idEtat).subscribe(
  (result:any)>>{
    clearUserForm(this.modifUserForm);

    if (result.data.libelleEtat == "Validé")
    {
      this.pannelAdmin.modifUser(this.modifAnnoForm.value).subscribe(
        (result:any)>{
          this.modifUserForm.value.idUser = Number(this.modifAnnoForm.value.idAnnonceur);
          this.modifUserForm.value.tokenInsc = "0";

          this.pannelAdmin.modifUser(this.modifUserForm.value).subscribe(
            (result:any)>{}
          );
        }
      );
    }
    else if (result.data.libelleEtat == "En Attente" || result.data.libelleEtat == "Refusé")
    {
      this.modifUserForm.value.idUser = Number(this.modifAnnoForm.value.idAnnonceur);

      this.pannelAdmin.getidUser(this.modifUserForm.value.idUser).subscribe(
        (result:any)>{
          this.pannelAdmin.getTokenInscr(result.data.emailuser).subscribe(
            (result:any)>{
              this.modifUserForm.value.tokenInsc = result.message;

              this.pannelAdmin.modifUser(this.modifUserForm.value).subscribe(
                (result:any)>{}
              );
            }
          );
        }
      );
    }
  }
)
```

En revanche, lors de l'ajout d'un nouvel annonceur, il n'y a pas la possibilité de choisir l'état puisqu'il sera mis par défaut à « En Attente » et qu'un token sera automatiquement générer avec l'adresse e-mail de l'annonceur saisie lors de l'ajout.

```
        this.pannelAdmin.getUnUser(this.ajoutAnnoForm.value.idAnnonceur).subscribe(
            (result:any)=>{
                this.pannelAdmin.getTokenInscr(result.data.emailuser).subscribe(
                    (result:any)=>{
                        clearUserForm(this.modifUserForm);

                        this.modifUserForm.value.idUser = this.ajoutAnnoForm.value.idAnnonceur;
                        this.modifUserForm.value.tokenInsc = result.message;

                        this.pannelAdmin.modifUser(this.modifUserForm.value).subscribe(
                            (result:any)=>{}
                        );
                    }
                );
            }
        );
    
```

Résultats :

The screenshot shows the 'Marketplace' application interface. At the top, there's a navigation bar with links like 'Google', 'Réseaux Sociaux', 'Streaming', 'Stores', 'Gaming', 'Code de la route', 'Etudes', 'Comics', 'Apprendre le japon...', 'Bouygues Telecom ...', and 'Autres favoris'. Below the navigation, there's a header with the 'Marketplace' logo, 'ACCUEIL', 'PRODUITS', 'ANNONCEUR', 'PARTENAIRE', a search bar, and user info ('Admin yrdy' and 'Se déconnecter'). The main area has tabs 'Annonceurs' and 'Annonces'. A 'Filtre:' dropdown is set to 'État' with buttons 'Rechercher' and 'Annuler'. Below this, a section titled 'Liste des annonceurs:' displays three cards:

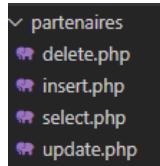
- Card 1 (Left):** Placeholder image, 'Compte: Annonceur 1 (anno1@gmail.com)', 'Nom: hgff hgff', 'Adresse: ()', 'État: Refusé', with 'Modifier' and 'Supprimer' buttons.
- Card 2 (Middle):** Placeholder image, 'Compte: Annonceur 2', 'Nom: fsdqds', 'Adresse: 48 fsdqds', 'Ville: fdfdf', 'Code Postal: 12345', 'Pays: Algérie', 'État: Valide' (highlighted in blue), with 'Modifier' and 'Supprimer' buttons. A dropdown menu is open over the 'État' field, showing options: 'Validé', 'En Attente', 'Valide', and 'Refusé'.
- Card 3 (Right):** Placeholder image, 'Compte: Test Marketplace (client@gmail.com)', 'Nom: vjhjzs', 'Adresse: zaza 4 zaza (27190) Fidji', 'État: Refusé', with 'Modifier' and 'Supprimer' buttons.

Interface de modification de l'état d'un annonceur

| | État | motdepasse |
|---|--------|---|
| 1 | Refusé | 042372392 eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2N... |
| 2 | Valide | 042372392 eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2N... |

Résulta en BDD, on voit bien qu'à la fin le token d'inscription charge. Il passe à 0 si on valide le compte, sinon un token est généré.

Le gestionnaire des partenaires a été conçu exactement de la même manière pour gérer l'état du compte, il y a juste les informations affichées et les données manipulées qui changent pour prendre uniquement celles des partenaires.



Le CRUD pour gérer les données des partenaires.

```
getLesPartenaires()
{
    return this.http.get<Partenaires[]>(this.baseUrl+'utilisateurs/partenaires/select.php');
}

getUnPartenaire(id:any)
{
    return this.http.get<Partenaires[]>(this.baseUrl+'utilisateurs/partenaires/select.php?id='+id);
}

modifPart(part:any)
{
    return this.http.put(this.baseUrl+'utilisateurs/partenaires/update.php', part);
}

supprPart(id:any)
{
    return this.http.delete(this.baseUrl+'utilisateurs/partenaires/delete.php?id='+id);
}

ajoutPart(part:any)
{
    return this.http.post(this.baseUrl+'utilisateurs/partenaires/insert.php', part);
}
```

Les requêtes HTTP ajouté dans le panel d'administration.

Tout le reste du code du gestionnaire des partenaires, est un copier-coller du code que j'ai réalisé pour le gestionnaire des annonceurs.

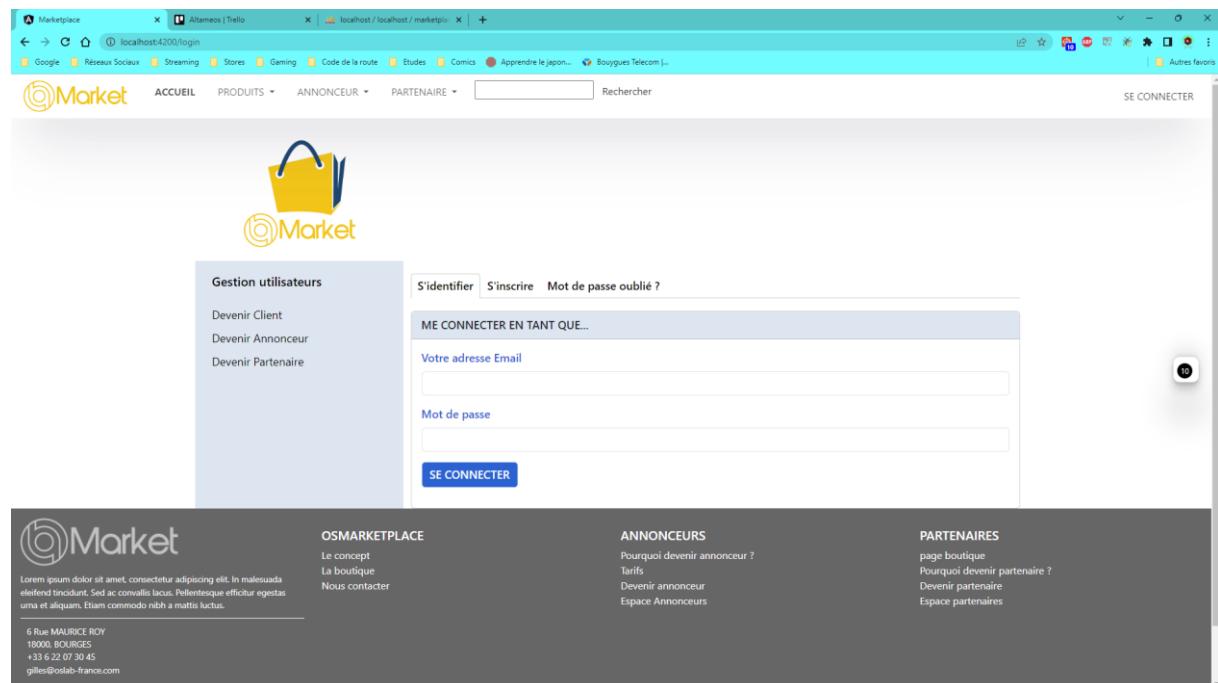
Résultat :

II) Application Web :

4. Annonceurs.
- b. Identification.

A l'origine, je devais m'occuper de l'identification des annonceurs. Mais après consultation du tuteur de stage avec le binôme qui s'est occupé de l'identification général du site, on s'est mis d'accord sur le fait que les annonceurs se connecteront via l'identification générale. Je n'ai donc aucune nouvelle interface à faire et aucune modification de code à réaliser.

Résultat :



Page de login réalisé par un autre binôme, les annonceurs peuvent bien se connecter par cette interface.

II) Application Web :

5. Tutoriels et actualités.

a. BACK.

Dans un premier temps, j'ai ajouté à la BDD la table « cattutoactu » qui va permettre de catégoriser les tutoriels et des actualités, la table « tutoactu » qui contiendra les tutoriels et les actualités ainsi que la table « imgtutoactu » qui va permettre de sauvegarder les images rattachées aux tutoriels et aux actualités.

| idCategorie | libelleCategorie | couleurCategorie |
|-------------|------------------|------------------|
|-------------|------------------|------------------|

La table « cattutoactu ».

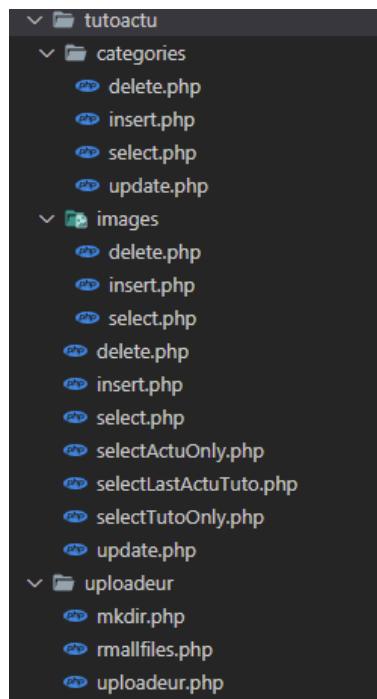
| idActuTuto | titreActuTuto | contenuActuTuto | resumeActuTuto | datePublication | dateModification | imgPreview | boolActuTuto | idCatActuTuto |
|------------|---------------|-----------------|----------------|-----------------|------------------|------------|--------------|---------------|
|------------|---------------|-----------------|----------------|-----------------|------------------|------------|--------------|---------------|

La table « tutoactu ».

| idlImg | idTutoActuImg | lienImg |
|--------|---------------|---------|
|--------|---------------|---------|

La table « imgtutoactu ».

J'ai ensuite ajouté au CRUD les éléments qui permettront la gestion des tutoriels et des actualités ainsi que la gestion des catégories et des images de manière à pouvoir afficher, ajouter, modifier ou supprimer une image, une catégorie, un tutoriel ou une actualité. Deux fichiers ont été ajoutés à l'uploader afin de pouvoir créer des répertoires ou supprimer des fichiers.



Répertoire du CRUD pour les tutoriels et les actualités ainsi que pour les modifications apportées à l'uploader de fichiers.

```
crud > uploadeur > mkdir.php
1  <?php
2      $chemin = "../../".$_POST['chemin']."/";
3      $repertoire = $_POST['nomRep'];
4
5      mkdir($chemin.$repertoire);
6  ?>
```

Code du fichier « mkdir.php » permettant la création de répertoire.

```
crud > uploadeur > rmallfiles.php
1  <?php
2      $chemin = "../../".$_POST['chemin'];
3
4      $fichiers = glob($chemin.'/*');
5
6      foreach($fichiers as $unFichier)
7      {
8          if(is_file($unFichier))
9          {
10              unlink($unFichier);
11          }
12      }
13  ?>
```

Code du fichier « rmallfiles.php » qui permet la suppression des fichiers d'un répertoire spécifique.

Je viens ensuite ajouter les requêtes HTTP nécessaires à la gestion des tutoriels et des actualités, des catégories et des images reliées aux tutoriels ou aux actualités dans le service « pannel-administration ».

```
getUnTutoActu(id:number)
{
  return this.http.get(this.baseUrl+'tutoactu/select.php?id='+id);
}

ajoutActuTuto(actututo:any)
{
  return this.http.post(this.baseUrl+'tutoactu/insert.php', actututo);
}

modifActuTuto(actututo:any)
{
  return this.http.put(this.baseUrl+'tutoactu/update.php', actututo);
}

getLesCatTutoActu()
{
  return this.http.get(this.baseUrl+'tutoactu/categories/select.php');
}

supprImgTutoActu(id:any)
{
  return this.http.delete(this.baseUrl+'tutoactu/images/delete.php?id='+id);
}

ajoutImgTutoActu(img:any)
{
  return this.http.post(this.baseUrl+'tutoactu/images/insert.php', img);
}

getLastIdTutoActu()
{
  return this.http.get(this.baseUrl+'tutoactu/selectLastActuTuto.php');
}

ajoutCatActuTuto(cat:any)
{
  return this.http.post(this.baseUrl+'tutoactu/categories/insert.php', cat);
}

modifCatActuTuto(cat:any)
{
  return this.http.put(this.baseUrl+'tutoactu/categories/update.php', cat);
}

supprCatActuTuto(id:any)
{
  return this.http.delete(this.baseUrl+'tutoactu/categories/delete.php?id='+id);
}

getLesTuto()
{
  return this.http.get(this.baseUrl+'tutoactu/selectTutoOnly.php');
}

getLesActu()
{
  return this.http.get(this.baseUrl+'tutoactu/selectActuOnly.php');
}

supprActuTuto(id:any)
{
  return this.http.delete(this.baseUrl+'tutoactu/delete.php?id='+id);
}

getLesImgActuTuto(id:any)
{
  return this.http.get(this.baseUrl+'tutoactu/images/select.php?id='+id);
```

Code du fichier « pannel-administration.service.ts ».

II) Application Web :

5. Tutoriels et actualités.
 - b. Partie gestionnaire.

Pour la partie gestionnaire, elle est composée de quatre composants : l'éditeur de texte, le formulaire d'ajout et de modification, le gestionnaire des catégories et l'affichage qui correspond à la page réunissant les trois précédents composants.

Pour l'éditeur de texte, j'ai utilisé la bibliothèque « *ckeditor* » qui permet de réaliser un éditeur de texte complet et de ressortir en résultat un code HTML qui permettra d'utiliser ce résultat directement pour l'afficher en page.

Voici les différentes étapes qui m'ont permis de réaliser l'éditeur de texte :

1^{ère} étape : Il faut installer les prérequis via le terminal, en exécutant les commandes suivantes :

```
npm install --save @ckeditor/ckeditor5-angular @ckeditor/ckeditor5-watchdog
npm install --save @ckeditor/ckeditor5-build-classic
npm install --save-dev @types/ckeditor__ckeditor5-build-classic
npm install --save @ckeditor/ckeditor5-alignment
```

2^{ème} étape : Dans « *app.module.ts* », il faut ajouter :

...

```
import { CKEditorModule } from '@ckeditor/ckeditor5-angular';
@NgModule({
  ...
  imports: [
    ...
    CKEditorModule,
  ]
  ...
})
```

3^{ème} étape : Dans le composant où va apparaître l'éditeur, il faut ajouter :

...

```
import * as ClassicEditor from '@ckeditor/ckeditor5-build-classic';
```

...

```
export class [nom composant] {
```

...

```
    public Editor:any = ClassicEditor;
```

...

```
}
```

4^{ème} étape : Il faut ajouter la ligne suivante dans le template HTML pour faire apparaître l'éditeur :

```
<ckeditor [editor]="Editor" data="

Hello, world!

"></ckeditor>
```

S'il y a une erreur dans le build avec une certaine variable nommé "CKEDITOR_VERSION", il faut se rendre dans le fichier « *node_modules\@ckeditor\ckeditor5-angular\ckeditor.component.d.ts* » et mettre en commentaire la déclaration de cette variable présente à la ligne 12 : « *// CKEDITOR_VERSION?: string;* ».

Pour importer un build custom de la toolbar/editor (<https://ckeditor.com/ckeditor-5/online-builder/>), il faut suivre les étapes suivantes :

1^{ère} étape : On vient créer et télécharger le build, puis on vient créer un dossier qui contiendra le dossier "build" et les fichiers "LICENSE.md", "package.json" et "README.MD" du build précédemment téléchargé.

2^{ème} étape : On vient récupérer le custom editor mais pas en faisant un « *import* » comme pour le « *classicEditor* », mais en créant une variable qui va venir le récupérer avec le « *require* ».

...

```
const customEditor = require('../text-editor/customckeditor/build/ckEditor');
```

...

3^{ème} étape : On vient attribuer la variable de l'étape précédente (« *customEditor* ») à la variable "Editor" qui permet d'afficher l'éditeur de texte.

Un problème rencontré lors de la réalisation de l'éditeur de texte est la lecture du code HTML et CSS résultant de celui-ci. En effet, les navigateurs ont une sécurité, ce nommant « DOM », qui permet de ne pas lire du code HTML et CSS venant « de l'utilisateur ». Il faut donc pouvoir « stériliser » et « purifier » le code résultant de l'éditeur afin qu'il soit accepté par le navigateur et affiché dans le site.

Pour utiliser le résultat venant de l'éditeur de texte, il faut réaliser les étapes suivantes :

1^{ère} étape : On vient installer le prérequis qui va permettre de « purifier » et confirmer au navigateur que le code HTML venant de l'éditeur est sain et sécurisé.

```
npm install isomorphic-dompurify
```

2^{ème} étape : On vient réaliser le code suivant dans le composant où le résultat de l'éditeur sera récupéré pour purifier et confirmer au DOM du navigateur que le code est sain afin de l'afficher dans le template HTML de ce même composant.

```
import { DomSanitizer } from '@angular/platform-browser';
import * as DOMPurify from 'isomorphic-dompurify';
import { SafeHtml } from '@angular/platform-browser';

voirPlusTutoActu : SafeHtml = "";

constructor(protected sanitizer: DomSanitizer)
{
    ...
}

const sanitizedContent = DOMPurify.sanitize(data.data.contenuActuTuto);
this.voirPlusTutoActu = this.sanitizer.bypassSecurityTrustHtml(sanitizedContent);
...
```

En réadaptant les étapes précédentes, j'ai réalisé un composant gérant uniquement l'éditeur de texte afin de pouvoir utiliser ce même éditeur sur toutes les pages nécessitant celui-ci. Grâce à un « *Output* » et au cycle « *OnChanges* » du composant, à chaque fois que l'éditeur est utilisé, le contenu de celui-ci est exporté de manière à être recueilli par une variable. Un « *Input* » est aussi utilisé de manière à remplir le contenu de l'éditeur lors de la modification.

```

src > app > text-editor > text-editor.component.ts > ...
1 import { Component, Output, Input, EventEmitter } from '@angular/core';
2 import { ChangeEvent } from "@ckeditor/ckeditor5-angular/ckeditor.component";
3
4 const customEditor = require('../text-editor/customckeditor/build/ckEditor');
5
6 @Component({
7   selector: 'app-text-editor',
8   templateUrl: './text-editor.component.html',
9   styleUrls: ['./text-editor.component.scss']
10 })
11
12 export class TextEditorComponent {
13   @Output() textEditorOut = new EventEmitter<string>();
14   @Input() dataRecup : string = "";
15
16   public Editor:any = customEditor;
17
18   retrieveddata : string = "";
19
20   constructor()
21   {
22   }
23
24
25   public onChange({ editor }: ChangeEvent)
26   {
27     const data = editor.getData();
28
29     if (data !== null)
30     {
31       this.retrieveddata = data;
32     }
33
34     this.textEditorOut.emit(this.retrieveddata);
35   }
36 }

```

Code du composant « text-editor ».

J'ai ensuite créé un composant qui contiendra le formulaire permettant de saisir les informations nécessaires à la création d'un tutoriel ou d'une actualité et d'insérer les données dans la BDD, s'il s'agit d'un ajout, ou de modifier un existant, s'il s'agit d'une modification.

En effet, ce formulaire est modulable en fonction de si c'est une actualité ou un tutoriel ou s'il s'agit d'une modification ou d'un ajout. Cela est possible grâce à des « *Inputs* » qui, en fonction de leurs valeurs, vont permettre de réadapter le formulaire.

Cela me permet de ne faire qu'un seul composant pour gérer toute la partie gestionnaire des tutoriels ou des actualités.

```

@Input() isModif! : boolean;
@Input() idActuTuto! : number;
@Input() typeActuTuto! : number;

```

« Inputs » permettant la modulation du formulaire.

Le formulaire permettra donc d'ajouter au maximum 4 images à afficher ainsi qu'une pour la « *preview* », de saisir un titre, un résumé et un contenu pour l'actualité ou le tutoriel.

Des vérifications sont faites pour s'assurer de la bonne saisie des données et, dans le cadre de la modification, s'assurer qu'il y ait au moins une donnée de saisie ou, dans le cadre de l'ajout, s'assurer que toutes les données sont saisies. Que ce soit pour la modification ou l'ajout, il n'est pas obligatoire de saisir 4 images à afficher (cela peut être moins voir aucune).

Ensuite j'ai réalisé un autre composant qui va permettre de gérer (ajouter, supprimer et modifier) les catégories et aussi filtrer les tutoriels ou les actualités affichés.

Résultats :

- Formulaire d'ajout et de modification d'un tutoriel ou d'une actualité :

Bienvenue dans votre espace d'administration

Espace Administration ▾

Ajout d'une actualité

Titre:

Image de preview: Aucun fichier choisi

Images (4 maximum): Aucun fichier choisi

Catégorie:

Résumé:

Contenu:

Paragraphe ▾ | **B** U *I* ~~O~~ ^A _A

- #### - Gestionnaire des catégories :

Ajouter Modifier Supprimer Trier

Libellé



Ajouter

Ajout d'une catégorie.

Ajouter Modifier Supprimer Trier

Catégories ▾

Libellé



Modifier

Modification d'une catégorie.

Ajouter Modifier Supprimer Trier

Catégories ▾

Supprimer

Suppression d'une catégorie.

Ajouter Modifier Supprimer Trier

Catégorie ▾

Rechercher

Annuler

Filtre permettant de trier les tutoriels ou les actualités par catégories qui seront affiché. Il fonctionne de la même manière que les filtres réalisés précédemment.

Pour finir, j'ai réalisé deux nouveaux composants qui seront mes pages de gestion des tutoriels et des actualités. Il s'agit dans les deux cas de réaliser un affichage en cartes permettant de lister les tutoriels ou les actualités avec des boutons d'ajout, de suppression ou de modification ainsi que l'affichage du gestionnaire des catégories. En premier lieu, les cartes afficheront l'image de « preview », le titre, le

résumé, la catégorie avec sa couleur, la date d'ajout et la date de la dernière modification s'il y en a eu une. Un bouton « Voir plus » permet d'aller voir le contenu de la page. Ces deux composants permettent de faire la modulation du formulaire pour ajouter ou modifier un tutoriel ou une actualité.

Pour pouvoir enregistrer le contenu récupéré de l'éditeur de texte en BDD il faut encoder les données saisies, et pour pouvoir les récupérer depuis la BDD, il faut les décoder de manière à lire les caractères spécifiques au code HTML.

```
$contenuActuTuto = htmlentities(htmlspecialchars(trim($data->contenuActuTuto)));
```

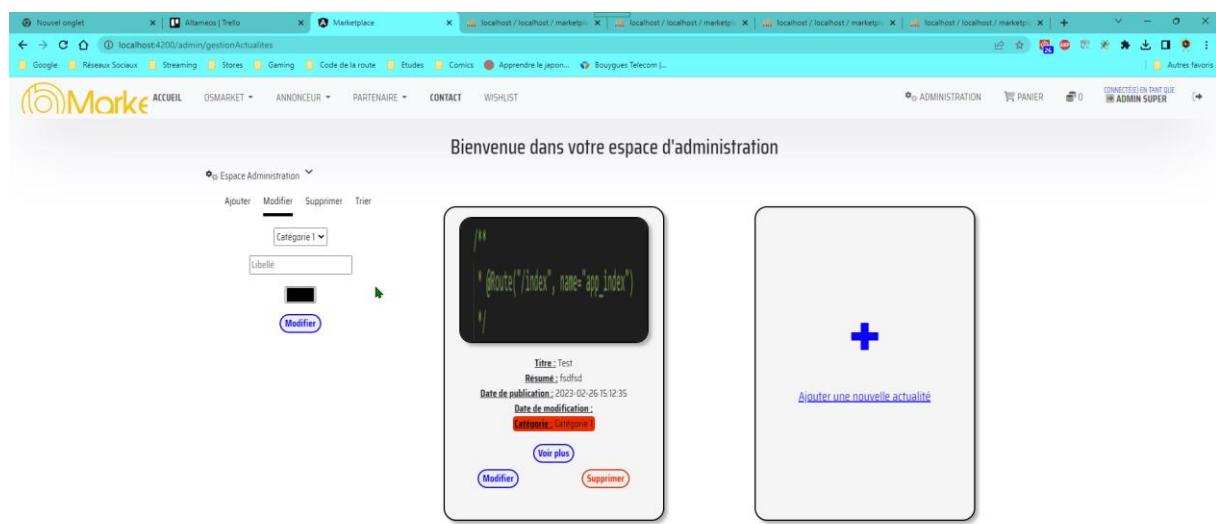
Ligne présente dans le fichier « insert.php » et « update.php » permettant d'encoder et d'enregistrer le contenu de l'éditeur en BDD.

```
$data['contenuActuTuto'] = html_entity_decode(htmlspecialchars_decode($data['contenuActuTuto']));
```

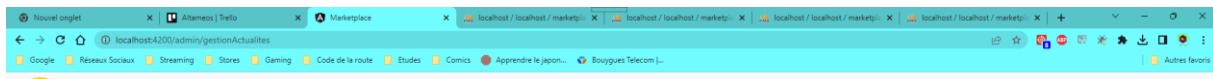
Ligne présente dans le fichier « select.php » permettant de décoder le contenu présent en BDD pour l'utiliser et l'afficher sur le site Web.

Résultat final des deux gestionnaires :

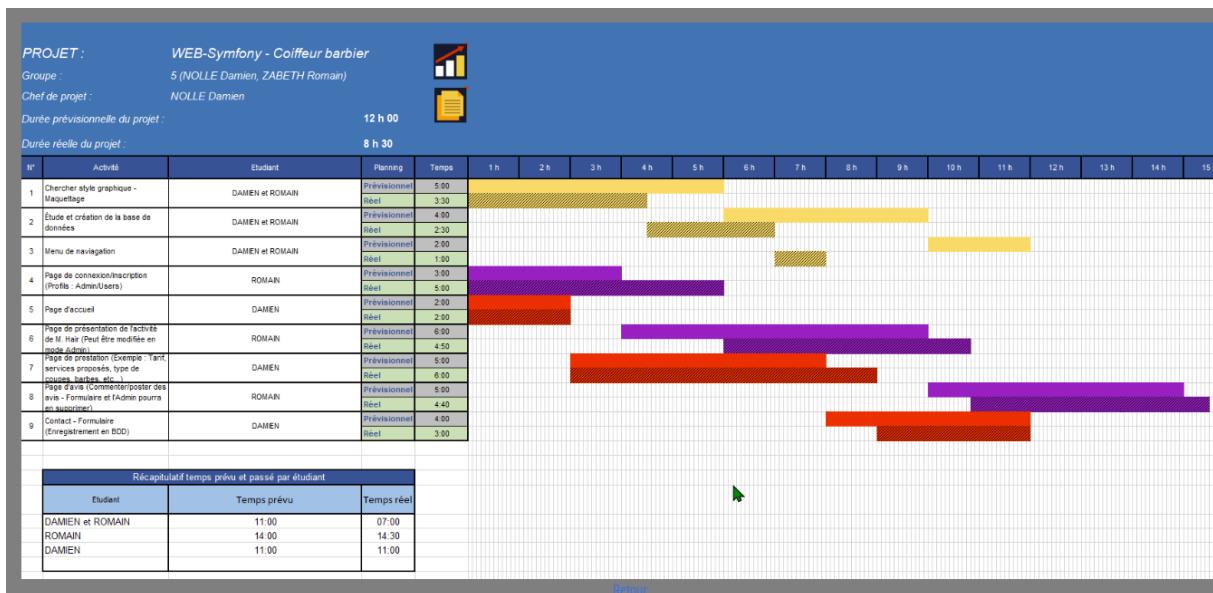
- Gestionnaires des actualités :



Affichage standard du gestionnaire.

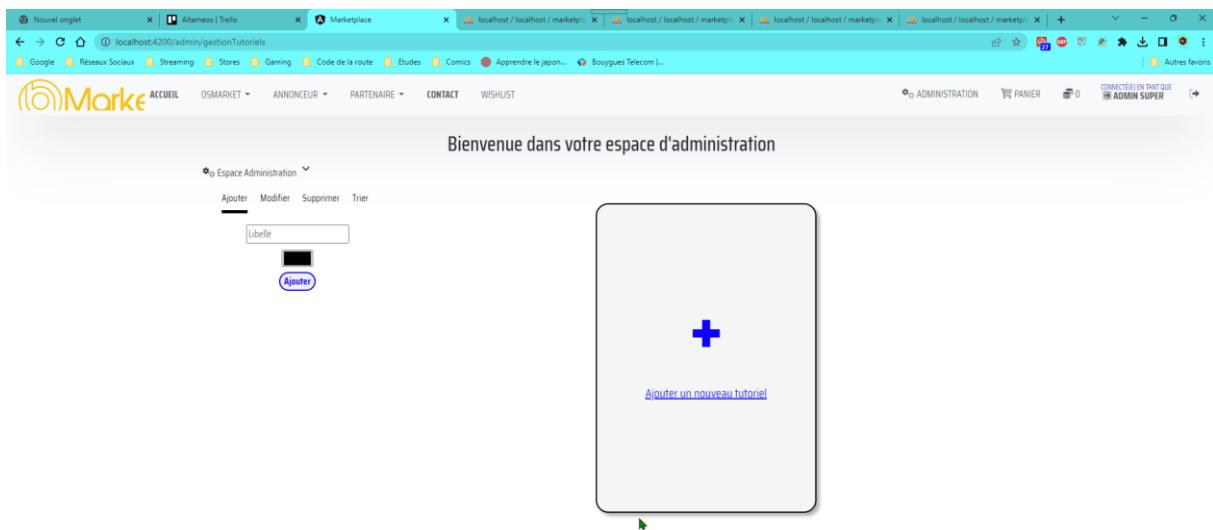


Affichage du contenu d'une actualité en appuyant sur le bouton « Voir plus ». On peut y voir le contenu saisi dans l'éditeur de texte et les images importés par l'utilisateur depuis le formulaire.



Lorsque l'utilisateur clique sur une des images présentes, elle se retrouve zoomée.

- Gestionnaires des tutoriels :



Ce gestionnaire fonctionne de la même manière que le précédent mais en réadaptant la modulation pour les tutoriels.

II) Application Web :

5. Tutoriels et actualités.
 - c. Partie affichage.

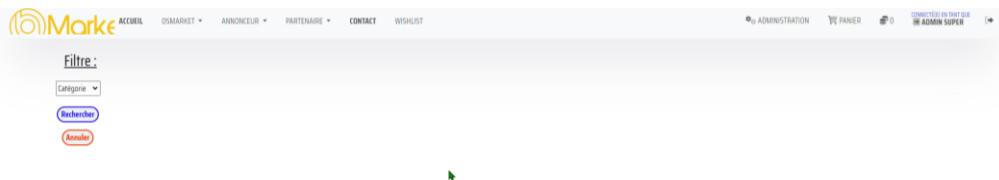
Il s'agit du même fonctionnement d'affichage des tutoriels et des actualités que les gestionnaires précédents sauf que contrairement à ces gestionnaires, accessibles uniquement pour les super-administrateurs, il n'y a pas la possibilité de gérer les tutoriels, les actualités et les catégories (il s'agit donc uniquement d'un affichage) et cette interface est accessible à tous.

Résultat :

- Affichage des actualités :



- Affichage des tutoriels :



II) Application Web :

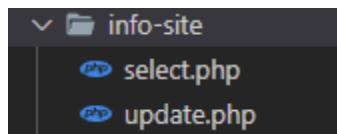
6. Gestion des informations de la boutique.

Il s'agit ici de dynamiser les informations du site dont le logo, la description, les moyens de contact (numéro de téléphone et adresse e-mail), l'adresse du siège social (rue, code postal, ville et pays), le copyright, la page « Qui sommes-nous ? » et les réseaux sociaux.

Dans un premier temps, j'ai ajouté une table qui contiendra des informations par défaut. Cet enregistrement sera présent afin de pouvoir les modifier.

| logoSite | logoEnteteSite | descriptionSite | emailSite | adresseSite | cpSite | villeSite | paysSite | numeroSite | copyright | qsnSite | facebookSite | instagramSite | linkedinSite |
|--------------------------------------|--|-----------------|-----------|-------------|--------|-----------|------------|------------|-----------|--|-------------------------|----------------------------|--------------------------|
| src/assets/img/logo/globale/logo.png | src/assets/img/logo/header/logo-header.png | description | email | adresse | 12345 | ville | Azerbaijan | numero | copyright | &#amp;lt;p style=&#amp;quot;text-align:center;&#amp;q... | https://m.facebook.com/ | https://www.instagram.com/ | https://fr.linkedin.com/ |

Dans un second temps, j'ai modifié le CRUD afin de pouvoir récupérer les données présentes en BDD et les modifier.



Puis j'ai ajouté les requêtes HTTP nécessaires au service du panel d'administration de manière à pouvoir interagir avec le CRUD précédent et réaliser les actions nécessaires à la gestion des informations du site.

```
getInfoSite()
{
    return this.http.get<InfoSite[]>(this.baseUrl+'info-site/select.php');
}

modifInfoSite(InfoSite:any)
{
    return this.http.put(this.baseUrl+'info-site/update.php', InfoSite);
}
```

Enfin, j'ai créé un composant qui contiendra le formulaire de modification des informations du site, fonctionnant de la même manière que le formulaire de modification des tutoriels ou des actualités. Je fais donc appel à l'éditeur de texte pour modifier la page « Qui-sommes-nous » ainsi qu'à l'uploadeur pour le logo du site. Bien évidemment les données saisies sont vérifiées avec, pour certains, le format via des RegEx. Et, comme pour toutes les interfaces de modification que j'ai réalisées, il n'est pas obligatoire de saisir tous les champs mais il faut qu'il y en ai au moins un de saisi pour réaliser une modification, sinon il y aura l'apparition d'un message d'erreur. Cette interface est uniquement accessible aux super-administrateurs du site.

Résultat final :

Bienvenue dans votre espace d'administration

Espace Administration ▾

Logo: Aucun fichier choisi

Logo d'en-tête: Aucun fichier choisi

Description:
description

Adresse e-mail:

Adresse:

Code postal:

Ville:

Pays:

Numeró:

Copyright:

Qui sommes-nous:

Paragraphe ▾ | **B** U *I* ∂ Δ \forall \exists \vdash \vdash | \equiv \equiv | \equiv \equiv | \leftrightarrow \rightarrow \wedge \vee \neg \exists \forall \hookleftarrow \hookrightarrow

Qui sommes-nous ?

Facebook:

Instagram:

LinkedIn:

[Modifier](#)

III) Conclusion :

Ce stage s'est réalisé avec succès et toutes mes tâches ont été terminées à temps. Il m'a permis de revoir beaucoup de notions vues lors de ma première et deuxième année de BTS SIO et d'apprendre une nouvelle technologie (Angular) et de nouveaux langages (TypeScript et SCSS) ainsi que de renforcer mes connaissances sur la réalisation de sites Web mono-pages et notamment sur la technologie React, cette dernière ayant été vue en cours.

Il m'a permis aussi de me mettre en condition réelle de développement en réalisant un projet « *on scratch* » (depuis le début) et de découvrir le travail en équipe dans un pôle de développement composé de notre tuteur de stage, des 5 autres étudiants venant du même BTS et moi-même.

Malgré quelques problèmes rencontrés dans notre équipe notamment sur l'organisation, la réalisation de la base de données et la compréhension de certaines tâches, tout le projet a pu être mené à bien grâce au travail de toute notre équipe et grâce à notre tuteur de stage.

Je suis très satisfait de ce stage qui m'a apporté beaucoup en terme de connaissances et d'expériences et qui fut agréable en la compagnie de notre tuteur de stage et de notre équipe d'étudiants venant du même BTS.

IV) Annexes

Exemple de réalisation d'un CRUD :

Annexe 1 : Fichier PHP de connexion à la base de données contenant nos étudiants
(« connect_bdd.php »).

```
connect_bdd.php
1 <?php
2     class Operations {
3         private $db_host = 'localhost';
4         private $db_name = 'test_etudiants';
5         private $db_username = 'root';
6         private $db_password = '';
7
8         public function dbConnect()
9         {
10             try
11             {
12                 $cnx = new PDO('mysql:host='.$this->db_host.';dbname='.$this->db_name, $this->db_username, $this->db_password);
13                 $cnx->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
14                 return $cnx;
15             }
16             catch (PDOException $e)
17             {
18                 echo "Connection error ".$e->getMessage();
19                 exit;
20             }
21         }
22     }
23 ?>
```

Annexe 2 : Fichier PHP contenant le SELECT qui va permettre soit de récupérer l'ensemble des étudiants, soit uniquement l'étudiant qui aura son id placé dans l'URL (« *view.php* »).

```
view.php
1  <?php
2  header("Access-Control-Allow-Origin: *");
3  header("Access-Control-Allow-Headers: access");
4  header("Access-Control-Allow-Methods: GET");
5  header("Access-Control-Allow-Credentials: true");
6  header("Content-Type: application/json; charset=UTF-8");
7  error_reporting(E_ERROR);
8
9  if ($_SERVER['REQUEST_METHOD'] !== 'GET') :
10     http_response_code(405);
11     echo json_encode([
12         'success' => 0,
13         'message' => 'Bad Request Detected! Only get method is allowed',
14     ]);
15     exit;
16 endif;
17
18 require 'connect_bdd.php';
19 $db = new Operations();
20 $cnx = $db->dbConnect();
21
22 $id = null;
23
24 if (isset($_GET['id'])) {
25     $students_id = filter_var($_GET['id'], FILTER_VALIDATE_INT, [
26         'options' => [
27             'default' => 'all_students',
28             'min_range' => 1
29         ]
30     ]);
31 }
32
33 try {
34     $sql = is_numeric($students_id) ? "SELECT * FROM `tb_etudiants` WHERE etd_id='$students_id'" : "SELECT * FROM `tb_etudiants`";
35     $stmt = $cnx->prepare($sql);
36     $stmt->execute();
37
38     if ($stmt->rowCount() > 0) :
39
40         $data = null;
41         if (is_numeric($students_id)) {
42             $data = $stmt->fetch(PDO::FETCH_ASSOC);
43         } else {
44             $data = $stmt->fetchAll(PDO::FETCH_ASSOC);
45         }
46
47         echo json_encode([
48             'success' => 1,
49             'data' => $data,
50         ]);
51
52     else :
53         echo json_encode([
54             'success' => 0,
55             'message' => 'No Record Found!',
56         ]);
57     endif;
58 }
59 catch (PDOException $e) {
60     http_response_code(500);
61     echo json_encode([
62         'success' => 0,
63         'message' => $e->getMessage(),
64     ]);
65     exit;
66 }
67
68 ?>
```

Annexe 3 : Fichier PHP permettant de mettre à jour une ou plusieurs données d'un étudiant
 (« *update.php* »).

```

update.php
1   <?php
2   header("Access-Control-Allow-Origin: *");
3   header("Access-Control-Allow-Headers: access");
4   header("Access-Control-Allow-Methods: PUT");
5   header("Content-Type: application/json; charset=UTF-8");
6   header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
7
8   $method = $_SERVER['REQUEST_METHOD'];
9
10  if ($method == "OPTIONS")
11  {
12      die();
13  }
14
15  if ($_SERVER['REQUEST_METHOD'] !== 'PUT') :
16      http_response_code(405);
17      echo json_encode([
18          'success' => 0,
19          'message' => 'Bad Request detected! Only PUT method is allowed',
20      ]);
21      exit;
22  endif;
23
24  require 'connect_bdd.php';
25  $db = new Operations();
26  $cnx = $db->dbConnect();
27
28  $data = json_decode(file_get_contents("php://input"));
29
30  if (!isset($data->id))
31  {
32      echo json_encode(['success' => 0, 'message' => 'Please enter correct Students id.']);
33      exit;
34  }
35
36  try
37  {
38      $fetch_post = "SELECT * FROM `tb_etudiants` WHERE ctd_id=:id";
39      $fetch_stmt = $cnx->prepare($fetch_post);
40      $fetch_stmt->bindValue(':id', $data->id, PDO::PARAM_INT);
41      $fetch_stmt->execute();
42
43      if ($fetch_stmt->rowCount() > 0) :
44          $row = $fetch_stmt->fetch(PDO::FETCH_ASSOC);
45          $nom = isset($data->nom) ? $data->nom : $row['ctd_nom'];
46          $prenom = isset($data->prenom) ? $data->prenom : $row['ctd_prenom'];
47
48          $update_query = "UPDATE `tb_etudiants`"
49          SET ctd_nom = :nom,
50              ctd_prenom = :prenom
51              WHERE ctd_id = :id";
52
53          $update_stmt = $cnx->prepare($update_query);
54
55          $update_stmt->bindValue(':nom', htmlspecialchars(strip_tags($nom)), PDO::PARAM_STR);
56          $update_stmt->bindValue(':prenom', htmlspecialchars(strip_tags($prenom)), PDO::PARAM_STR);
57
58          $update_stmt->bindValue(':id', $data->id, PDO::PARAM_INT);
59
60
61          if ($update_stmt->execute())
62          {
63              echo json_encode([
64                  'success' => 1,
65                  'message' => 'Record updated successfully'
66              ]);
67              exit;
68          }
69
70          echo json_encode([
71              'success' => 0,
72              'message' => 'Did not update. Something went wrong.'
73          ]);
74          exit;
75
76      else :
77          echo json_encode(['success' => 0, 'message' => 'Invalid ID. No record found by the ID.']);
78          exit;
79      endif;
80  }
81  catch (PDOException $e)
82  {
83      http_response_code(500);
84      echo json_encode([
85          'success' => 0,
86          'message' => $e->getMessage()
87      ]);
88      exit;
89  }
90  ?>
```

Annexe 4 : Fichier PHP permettant d'ajouter un nouvel étudiant (« *insert.php* »).

```
❶ insert.php
 1  <?php
 2   header("Access-Control-Allow-Origin: *");
 3   header("Access-Control-Allow-Headers: access");
 4   header("Access-Control-Allow-Methods: POST");
 5   header("Content-Type: application/json; charset=UTF-8");
 6   header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
 7
 8   $method = $_SERVER['REQUEST_METHOD'];
 9
10  if ($method == "OPTIONS")
11  {
12  |   die();
13  }
14
15
16  if ($_SERVER['REQUEST_METHOD'] !== 'POST') :
17  |   http_response_code(405);
18  |   echo json_encode([
19  |     'success' => 0,
20  |     'message' => 'Bad Request! Only POST method is allowed',
21  |   ]);
22  |   exit;
23  | endif;
24
25  require 'connect_bdd.php';
26  $db = new Operations();
27  $cnx = $db->dbConnect();
28
29  $data = json_decode(file_get_contents("php://input"));
30
31  if (!isset($data->id) || !isset($data->nom) || !isset($data->prenom)) :
32
33  | echo json_encode([
34  |     'success' => 0,
35  |     'message' => 'Please enter compulsory fields | id, nom and prenom',
36  |   ]);
37  | exit;
38
39  elseif (empty(trim($data->id)) || empty(trim($data->nom)) || empty(trim($data->prenom))) :
40
41  | echo json_encode([
42  |     'success' => 0,
43  |     'message' => 'Field cannot be empty. Please fill all the fields.',
44  |   ]);
45  | exit;
46
47  | endif;
48
49  try
50  {
51    $id = htmlspecialchars(trim($data->id));
52    $nom = htmlspecialchars(trim($data->nom));
53    $prenom = htmlspecialchars(trim($data->prenom));
54
55    $query = "INSERT INTO `tb_etudiants` (etd_id, etd_nom, etd_prenom)
56    VALUES(:id,:nom,:prenom)";
57
58    $stmt = $cnx->prepare($query);
59
60    $stmt->bindValue(':id', $id, PDO::PARAM_INT);
61    $stmt->bindValue(':nom', $nom, PDO::PARAM_STR);
62    $stmt->bindValue(':prenom', $prenom, PDO::PARAM_STR);
63
64
65    if ($stmt->execute()) {
66
67      http_response_code(201);
68      echo json_encode([
69      |     'success' => 1,
70      |     'message' => 'Data Inserted Successfully.'
71      |   ]);
72      | exit;
73    }
74
75    echo json_encode([
76      |     'success' => 0,
77      |     'message' => 'There is some problem in data inserting'
78      |   ]);
79    | exit;
80
81  }
82  catch (PDOException $e)
83  {
84
85    http_response_code(500);
86    echo json_encode([
87      |     'success' => 0,
88      |     'message' => $e->getMessage()
89      |   ]);
90    | exit;
91  }
?
```

Annexe 5 : Fichier PHP permettant de supprimer un étudiant via son id (« *delete.php* »).

```
delete.php
1  <?php
2      header("Access-Control-Allow-Origin: *");
3      header("Access-Control-Allow-Headers: access");
4      header("Access-Control-Allow-Methods: DELETE");
5      header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
6
7      $method = $_SERVER['REQUEST_METHOD'];
8
9      if ($method == 'OPTIONS')
10     {
11         die();
12     }
13
14     if ($_SERVER['REQUEST_METHOD'] != 'DELETE') :
15         http_response_code(405);
16         echo json_encode([
17             'success' => 0,
18             'message' => 'Bad Request detected. HTTP method should be DELETE',
19         ]);
20         exit;
21     endif;
22
23     require 'connect_bdd.php';
24     $db = new Operations();
25     $cnx = $db->dbConnect();
26
27     $data = json_decode(file_get_contents("php://input"));
28
29     $id = $_GET['id'];
30
31     if (!isset($id)) {
32         echo json_encode(['success' => 0, 'message' => 'Please provide the post ID.']);
33         exit;
34     }
35
36     try
37     {
38         $fetch_post = "SELECT * FROM `tb_students` WHERE ctd_id=:id";
39         $fetch_stmt = $cnx->prepare($fetch_post);
40         $fetch_stmt->bindValue(':id', $id, PDO::PARAM_INT);
41         $fetch_stmt->execute();
42
43         if ($fetch_stmt->rowCount() > 0) :
44
45             $delete_post = "DELETE FROM `tb_students` WHERE ctd_id=:id";
46             $delete_post_stmt = $cnx->prepare($delete_post);
47             $delete_post_stmt->bindValue(':id', $id, PDO::PARAM_INT);
48
49             if ($delete_post_stmt->execute())
50             {
51                 echo json_encode([
52                     'success' => 1,
53                     'message' => 'Record Deleted successfully'
54                 ]);
55                 exit;
56             }
57
58             echo json_encode([
59                 'success' => 0,
60                 'message' => 'Could not delete. Something went wrong.'
61             ]);
62             exit;
63
64         else :
65             echo json_encode(['success' => 0, 'message' => 'Invalid ID. No posts found by the ID.']);
66             exit;
67         endif;
68     }
69     catch (PDOException $e)
70     {
71         http_response_code(500);
72         echo json_encode([
73             'success' => 0,
74             'message' => $e->getMessage()
75         ]);
76         exit;
77     }
78 }
79 ?>
```

Annexe 6 : Style du composant « list-administrateurs » qui est le style de la majorité des composants du même type.

```
list-administrateurs.component.scss X
src > app > list-administrateurs > list-administrateurs.component.scss > .grid-container
1  .grid-container
2  {
3    display: grid;
4    grid-template: 1fr / 1fr 1fr 1fr 1fr;
5    grid-gap: 30px;
6    grid-template-columns: repeat(auto-fit, minmax(350px, 1fr));
7    margin: 20px;
8  }
9
10 .flex-card-container
11 {
12   display: flex;
13   flex-direction: column;
14 }
15
16 h3
17 {
18   margin-top: 50px;
19   text-align: center;
20   text-decoration: underline;
21 }
22
23 .flex-card-container-row
24 {
25   display: flex;
26   flex-direction: row;
27 }
28
29 .flex-card-element
30 {
31   flex-basis: 100%;
32   text-align: center;
33 }
34
35 .grid-element
36 {
37   margin-left: auto;
38   margin-right: auto;
39   border: solid 2px black;
40   border-radius: 20px;
41   background-color: #rgb(245, 245, 245);
42   box-shadow: 5px 5px 5px #rgb(180, 180, 180);
43   width: 350px;
44   height: 350px;
45 }
46
47 button
48 {
49   display: block;
50   margin-left: auto;
51   margin-right: auto;
52 }
53
54 .labelForm
55 {
56   text-decoration: underline;
57   font-weight: bold;
58 }
59
```

```
60 .labelFormRight
61 {
62     margin-top: 4px;
63     text-decoration: underline;
64     font-weight: bold;
65     text-align: right !important;
66     margin-bottom: 4px;
67 }
68
69 button
70 {
71     display: block;
72     margin-top: 30px;
73     margin-bottom: 30px;
74 }
75
76 .premiere-ligne
77 {
78     margin-top: 50px;
79 }
80
81 .premiere-ligne-form
82 {
83     margin-top: 30px;
84 }
85
86 .premiere-ligne-suppr
87 {
88     margin-top: 130px;
89 }
90
91 .premiere-ligne-ajout
92 {
93     margin-top: 20px;
94 }
95
96 #nomUser
97 {
98     margin-top: 2px;
99     margin-bottom: 2px;
100    width: 80px;
101    display: block;
102    margin-left: 5px;
103 }
104
105 #prenomUser
106 {
107     margin-top: 2px;
108     margin-bottom: 2px;
109     width: 75px;
110     display: block;
111     margin-left: 5px;
112 }
```

```
114 #emailUser
115 {
116     margin-top: 2px;
117     margin-bottom: 2px;
118     width: 140px;
119     display: block;
120     margin-left: 5px;
121 }
122 #passwordUser
123 {
124     margin-top: 2px;
125     margin-bottom: 2px;
126     width: 75px;
127     display: block;
128     margin-left: 5px;
129 }
130 #telUser
131 {
132     margin-top: 2px;
133     margin-bottom: 2px;
134     width: 95px;
135     display: block;
136     margin-left: 5px;
137 }
138 #idDroitUser
139 {
140     margin-top: 5px;
141     width: 100px;
142     display: block;
143     margin-left: 5px;
144 }
145 .labelFormCenter
146 {
147     text-align: center;
148 }
149 .titreId
150 {
151     text-decoration: underline;
152     font-weight: bold;
153 }
154 button
155 {
156     border-radius: 100px;
157     border: solid 2px;
158     font-weight: bold;
159 }
160 .button-blue
161 {
162     color: #rgb(0, 0, 255);
163     background-color: none;
164 }
```

```
173 .button-blue:hover
174 {
175     background-color: ■rgb(160, 160, 255)
176 }
177
178 .button-blue:active
179 {
180     background-color: ■rgb(0, 0, 255);
181     border-color: ■rgb(160, 160, 255);
182     color: ■rgb(160, 160, 255);
183 }
184
185 .button-red
186 {
187     color: ■rgb(255, 0, 0);
188     background-color: none;
189 }
190
191 .button-red:hover
192 {
193     background-color: ■rgb(255, 160, 160)
194 }
195
196 .button-red:active
197 {
198     background-color: ■rgb(255, 0, 0);
199     border-color: ■rgb(255, 160, 160);
200     color: ■rgb(255, 160, 160);
201 }
202
203 a
204 {
205     text-decoration: none;
206     font-weight: bold;
207     font-size: 120px;
208     color: ■rgb(0, 0, 255);
209 }
210
211 a:hover
212 {
213     color: ■rgb(160, 160, 255);
214 }
215
216 a:active
217 {
218     color: ■rgb(0, 0, 255);
219 }
220
221
222 .ajout-text
223 {
224     text-decoration: underline;
225     font-weight: normal;
226     font-size: 20px;
227 }
```

Annexe 7 : Style du composant « filtre-administrateurs » qui est le style de la majorité des composants du même type.

```
.flex-filtre-container
{
    display: flex;
    flex-direction: row;
}

.flex-filtre-container-column
{
    display: flex;
    flex-direction: column;
}

h3
{
    margin-top: 50px;
    text-align: center;
    text-decoration: underline;
}

.button-left
{
    margin-left: 50px;
}

.button-right
{
    margin-right: 50px;
}

.flex-filtre-element
{
    margin-left: auto;
    margin-right: auto;
}

.flex-filtre-element
{
    margin-top: 10px;
}

button
{
    border-radius: 100px;
    border: solid 2px;
    font-weight: bold;
}

.button-blue
{
    color: #rgb(0, 0, 255);
    background-color: none;
}

.button-blue:hover
{
    background-color: #rgb(160, 160, 255);
}

.button-blue:active
{
    background-color: #rgb(0, 0, 255);
    border-color: #rgb(160, 160, 255);
    color: #rgb(160, 160, 255);
}

.button-red
{
    color: #rgb(255, 0, 0);
    background-color: none;
}

.button-red:hover
{
    background-color: #rgb(255, 160, 160);
}

.button-red:active
{
    background-color: #rgb(255, 0, 0);
    border-color: #rgb(255, 160, 160);
    color: #rgb(255, 160, 160);
}
```

Annexe 8 : Code de la comparaison des données saisies dans le filtre des administrateurs avec ce qu'il y a dans la BDD. Ce code est semblable à tous les autres composants utilisant un filtre et réalisant une comparaison similaire.

```

ngOnChanges()
{
  if (this.filtreInAdmin != null)
  {
    this.lesAdministrateurs = [];

    this.pannelAdmin.getLesAdministrateurs().subscribe(
      (result:any)=>{
        for (let unAdministrateur of result.data)
        {
          let verifNom = null;
          let verifPrenom = null;
          let verifEmail = null;
          let verifTel = null;
          let verifIdDroit = null;

          if (this.filtreInAdmin.nom != "")
          {
            if (unAdministrateur.nomUser.substr(0,Object.keys(this.filtreInAdmin.nom).length) == this.filtreInAdmin.nom)
            {
              verifNom = true;
            }
            else
            {
              verifNom = false;
            }
          }

          if (this.filtreInAdmin.prenom != "")
          {
            if (unAdministrateur.prenomUser.substr(0,Object.keys(this.filtreInAdmin.prenom).length) == this.filtreInAdmin.prenom)
            {
              verifPrenom = true;
            }
            else
            {
              verifPrenom = false;
            }
          }

          if (this.filtreInAdmin.email != "")
          {
            if (unAdministrateur.emailUser.substr(0,Object.keys(this.filtreInAdmin.email).length) == this.filtreInAdmin.email)
            {
              verifEmail = true;
            }
            else
            {
              verifEmail = false;
            }
          }

          if (this.filtreInAdmin.tel != "")
          {
            if (unAdministrateur.telUser.substr(0,Object.keys(this.filtreInAdmin.tel).length) == this.filtreInAdmin.tel)
            {
              verifTel = true;
            }
            else
            {
              verifTel = false;
            }
          }

          if (this.filtreInAdmin.libelleDroit != "")
          {
            if (unAdministrateur.libelleDroit == this.filtreInAdmin.libelleDroit)
            {
              verifIdDroit = true;
            }
            else
            {
              verifIdDroit = false;
            }
          }

          if ((verifNom === true || verifNom === null) && (verifPrenom === true || verifPrenom === null) && (verifEmail === true || verifEmail === null) && (verifTel === true || verifTel === null) && (verifIdDroit === true || verifIdDroit === null))
          {
            this.lesAdministrateurs.push(unAdministrateur);
          }
        }
      }
    );
  }
  else
  {
    this.lesAdministrateurs = null;
  }
}

this.pannelAdmin.getLesAdministrateurs().subscribe(
  (result:any)=>{
    this.lesAdministrateurs = result.data;
  }
);
}

```