



Région académique  
NORMANDIE



# RAPPORT DE STAGE

Réalisé par :

Oliwer SKWERES

Encadrant :

Lucas SCHNEKENBURGER

Stage effectué chez Altameos pour une durée de 6 semaines.

BTS SIO - Année : 2022/2023



# SOMMAIRE

<b>0/Remerciment.....</b>	<b>4</b>
<b>1/Présentation de l'entreprise.....</b>	<b>5</b>
A/L'entreprise.....	5
B/Les moyens informatique .....	5
C/Accueil.....	5
<b>2/Présentation du projet .....</b>	<b>6</b>
A/Le projet.....	6
B/Outils utilisés.....	6
<b>3/Mise en place du projet.....</b>	<b>7</b>
A/L'apprentissage.....	7
B/Les tâches confiées .....	7
C/La base de données .....	7
<b>4/Le projet .....</b>	<b>10</b>
A/Gestion des produits.....	10
B/Le premier merge .....	17
C/Paiement des annonces.....	18
E/Le deuxième merge.....	22
F/Reformater l'affichage .....	23
G/Affichage des produits via offres.....	26
H/Correction d'erreurs.....	27
I/Le troisième merge .....	32
J/Les dernières tâches .....	32
<b>5/Bilan.....</b>	<b>33</b>

## 0/Remerciment

Je tiens à remercier Monsieur Lucas SCHNEKENBURGER qui m'a accueilli dans son entreprise, Altameos, pendant une période de 6 semaines. J'ai été en mesure de réaliser un projet très intéressant et d'apprendre une nouvelle technique. Il a pu nous accompagner avec mon équipe dans ce projet pour nous guider, en apportant ses connaissances et son opinion sur notre travail.

J'aimerais également remercier mon équipe, qui était en formation avec moi au cours de ces 6 semaines. J'ai pu les aider et me faire aider, apprendre avec eux, ce qui a aussi permis, je pense, de finir le travail à temps, car tout seul, ce ne serait pas possible.

Enfin, je voudrais remercier Mme Hellard, responsable du BTS SIO au Lycée Privé Saint Adjutor de Vernon, de m'avoir mis en contact avec Altameos au cours de ma première année de BTS. J'ai réussi à revenir dans la même entreprise pour mon stage de deuxième année.

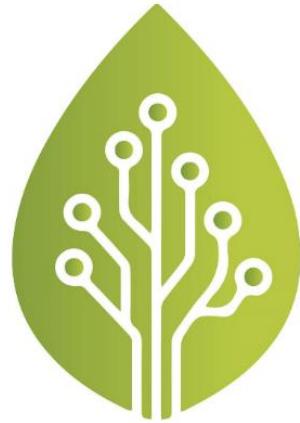
## 1/Présentation de l'entreprise

### A/L'entreprise

Altameos Multimédia est une agence web située à Evreux, dirigée par Monsieur Lucas Schnekenburger depuis 2003. L'entreprise regroupe les graphistes, programmeurs et pros du web afin de répondre aux besoins des entreprises sur Internet. L'entreprise Altameos propose plusieurs de ses services comme le web design, le référencement, l'hébergement, conseils informatiques et le développement internet.

L'entreprise travaille des grandes entreprises comme VINCI et Renault, mais aussi avec des particuliers.

Le travail dans cette société s'effectue en distanciel, via Teams, étant donné que l'entreprise ne possède pas de locaux à elle seule.



### B/Les moyens informatique

Pour satisfaire ses clients, Altameos utilise plusieurs logiciels tels que Visual Studio Code, un IDE permettant de développer les sites, FileZilla permettant de transférer les fichiers locaux vers un serveur afin d'assurer l'hébergement des sites web ou encore Teams afin de rester en communication avec les stagiaires de celle-ci, de se réunir et de discuter sur des projets.

Elle utilise plusieurs langages de développement (HTML, SCSS, JAVASCRIPT, PHP) et s'appuie sur les frameworks comme Angular, Symfony pour réaliser les projets. Ces frameworks, qui apportent une facilité à travailler, permettent aussi d'importer des bibliothèques en plus, tel que JQuery, AJAX, Stripe ou encore JWT. L'utilisation de MYSQL n'est pas exclue, afin de stocker des données sur un serveur.

### C/Accueil

L'accueil dans l'entreprise c'est fait directement sur Teams avec les autres stagiaires. Notre tuteur de stage a pu nous donner des détails sur le projet à faire avant de commencer, nous expliquer ce qu'il attendait de nous, et nous a laissé-nous former pendant une semaine sur la technologie utilisée dans le cadre du projet.

Deux points quotidiens sur notre avancement seront fait, un le matin pour savoir ce que l'on allait faire, et répondre à nos questions, et un le soir, afin de soulever les problèmes qu'ont rencontré et les tâches finalisées.

## 2/Présentation du projet

### A/Le projet

Le but de ce projet était de réaliser un site de marketplace, permettant à plusieurs personnes de vendre du matériel pour personne atteinte d'handicapé. Grâce à ce site, les utilisateurs pourraient commander des produits liés au domaine médical, de suivre leurs commandes et gérer leurs comptes. Une partie pour des annonceurs serait développer, leurs permettant de vendre des produits, de gérer des produits en les mettant en avant grâce à des offres payantes. Une partie sera réservée pour les partenaires qui pourront afficher des publicités sur le site en échange d'une rémunération.

Le projet doit être réalisé avec Angular, un framework basé sur Typescript, pour le côté client, et une API PHP qui sera hébergée sur un serveur. Comme Angular était quelque chose de nouveau, je devais me former d'abord dessus pendant la première semaine de mon stage.

Pour la mise en commun, on a décidé d'utiliser GitHub ainsi que GitKraken pour pouvoir mieux gérer ce projet et de pouvoir mutualiser le projet à tout moment assez facilement.

Nous étions 6 pendant la réalisation de ce stage, les tâches seront séparées entre 3 groupes de 2 personnes. Nous verrons ça plus tard dans ce rapport.

### B/Outils utilisés

Le site étant développé avec Angular, nous avons besoin de plusieurs outils. Pour la partie écriture du code, nous avons décidé d'utiliser Visual Studio Code, comme tout le monde se sentait à l'aise avec ce logiciel.

Pour le serveur local, on a utilisé soit XAMPP, soit WAMP, deux logiciels permettant de simuler en local un serveur phpMyAdmin, nous permettant d'exploiter une base de données. Pour réaliser la base de données, nous avons utilisé MySQL Workbench car ce logiciel permet de générer un script que l'on peut directement importer sur notre serveur pour créer une base de données.

Pour les langages utilisés, on utilise l'HTML, accompagné du framework Angular qui permet de rajouter des éléments en plus dans le fichier. Le style du site se fera via des fichiers SCSS, un langage qui sera ensuite compilé et interprété en CSS, permettant d'écrire du code CSS plus facilement. On utilisera ensuite Angular et donc TypeScript, afin de récupérer nos données via l'API, créer des composants pour pouvoir les réutiliser sur d'autres pages. Pour installer d'éventuelles dépendances en plus, on utilisera NodeJS. Pour finir, comme mentionné plutôt, pour l'API on va utiliser PHP via le logiciel XAMPP pour ma part.

### 3/Mise en place du projet

#### A/L'apprentissage

Après le premier entretien qui a duré une matinée, on nous a fait part du projet dans les grandes lignes, une boutique en ligne, permettant aux utilisateurs de vendre des objets liés à la médecine. On nous a appris que le projet sera réalisé en Angular, donc notre tuteur nous a laissé une semaine pour nous former sur cette technologie via des vidéos tutos ou la documentation directement sur le site d'Angular.

Les différentes vidéos nous ont permis de comprendre Angular, comprendre comment il marche, sur quoi se base ce framework, la connexion à une base de données, le routing ou l'utilisation des services.

Pour la faire courte, on va générer un component, le rajouter dans un fichier qui gère les routes pour pouvoir l'afficher. Si nous avons besoin de faire une requête API, on va passer par un service, qui va stocker nos fonctions de requêtes API.

Au bout de cette semaine, quelques personnes rencontraient encore des petites difficultés avec Angular, mais on pouvait enfin commencer le projet.

#### B/Les tâches confiées

Nous étions 6 sur le projet et il y avait beaucoup de tâches à faire. On avait environ 50 tâches à réaliser, globalement le site en entier. Nous nous étions repartis les tâches ensemble assez équitablement.

On m'a aussi informé que je serais chef de projet, je devrais donc gérer les problèmes des membres du groupe et le projet dans sa globalité.

Mes tâches à moi étaient :

- Création de la base de données (Faite avec tout le monde pour avoir un maximum d'idée)
- Création du repository GitHub
- Gestion des produits (Ajout, Suppression, Modification, Affichage)
- Classer les produits (Plus vendus, plus vues)
- Gestion des annonces (Ajout, Suppression, Modification, Affichage)
- Gestion des offres d'annonces (Ajout, Suppression, Modification, Affichage, Paiement)
- Gestion du carrousel de la page d'accueil (Tâche partagé avec mon binôme)
- Gestion de la page d'accueil (Tâche partagé avec mon binôme)
- Gestion de la page état des commandes de produits

Pour suivre ce qui a pu être fait, on a utilisé le site Trello, qui permet de classer nos tâches et de pouvoir les séparer en plusieurs tableaux.

#### C/La base de données

Première chose qu'on avait décidé de faire, c'était la base de données. Vous pouvez retrouver plus bas le schéma qu'on a pu faire. Globalement c'était une première version, on savait qu'on serait obligé de changer la base au fur et à mesure que le projet avance.

Je ne vais expliquer toute la base de données, sinon ça prendrait énormément de temps, mais dans les grandes lignes, on aura un utilisateur qui aura différents droits (Admin, Partenaire, Annonceur, Client). S'il est annonceur, il faudra qu'il renseigner des informations en plus. L'utilisateur peut laisser des avis sur le site ou, commander des produits, s'il est annonceur, il peut vendre des produits, acheter des offres d'annonces pour mettre en avant ses produits.

Les tables un peu complexes sont la table Commander et Annonce. La table commander est relié à CommandeFacture, qui est la facture de commande pour l'utilisateur, et la table Produits, où il y a les produits vendus sur le site. On a décidé de créer une table Commander entre Produits et CommandeFacture afin de pouvoir référencer les produits achetés pour une commande dans une table. Cette table contient en clé primaire numFactureCommande qui fait référence à CommandeFacture et idProduitCommande qui fait référence à la table Produits. On a aussi une colonne quantité, qui permet de renseigner la quantité de produit commandé. Cette manière permet de stocker assez facilement les informations d'une commande, le mieux c'est un tableau qui peut illustrer cet exemple :

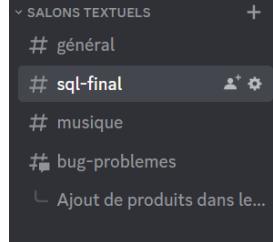
NumFactureCommande	idProduitCommande	qtProduitCommande
0000001	1	2
0000001	2	4
0000002	1	1
0000003	3	1

Un produit peut être acheté plusieurs fois, pour différentes commandes.

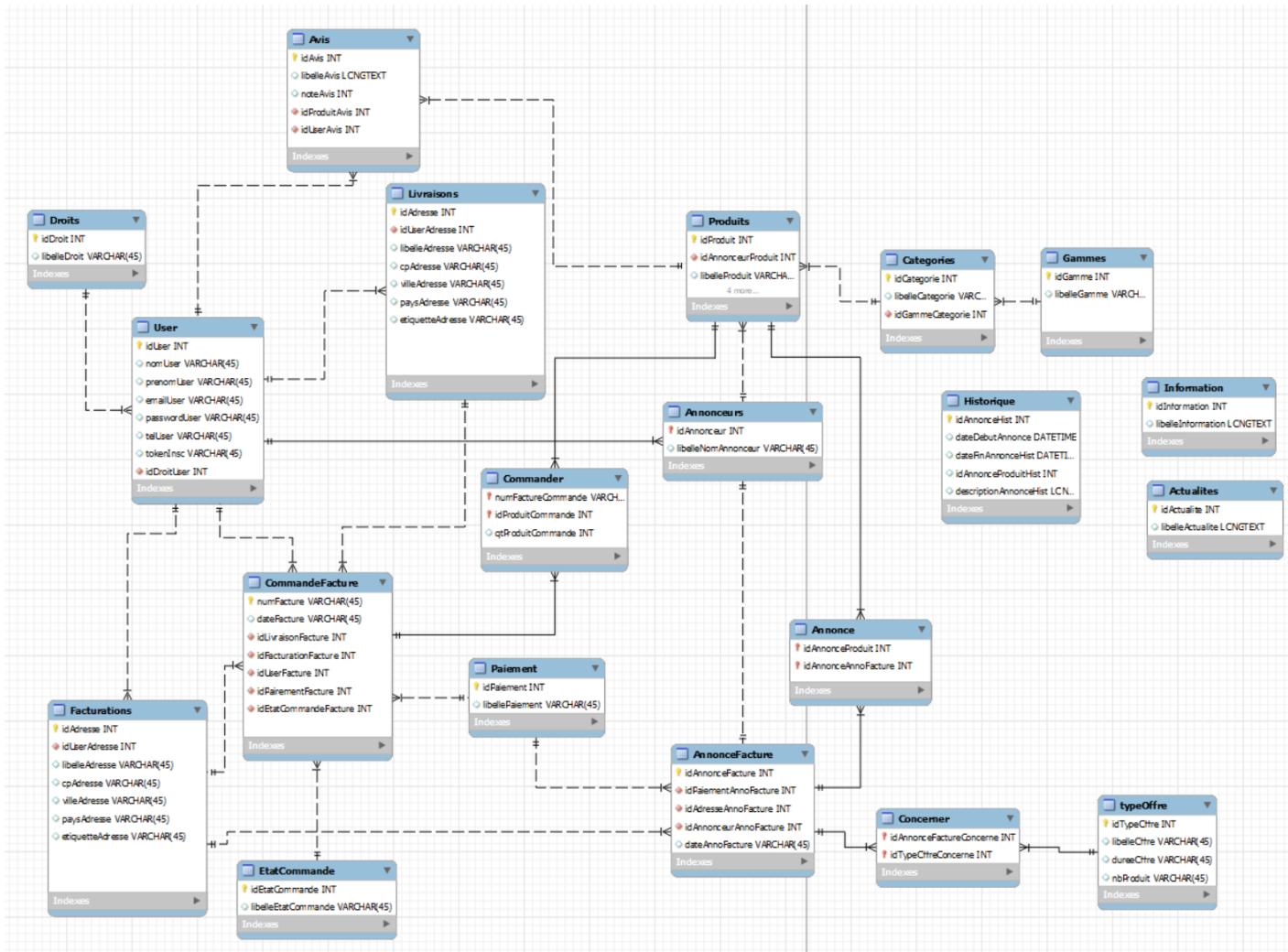
La table Annonce relie la table AnnonceFacture et la table Produits. Lorsqu'un annonceur va acheter une offre, il va pouvoir choisir, parmi les produits qu'il met en vente, des produits à mettre en avant sur le site. Tous les produits mis en vente par l'annonceur ne seront pas affichés, il faut pouvoir acheter une offre d'annonce. La table Annonce permettra alors de récupérer les produits qui pourront être affichés sur le site et acheté.

On a décidé de créer un serveur Discord qui nous permettrait de communiquer plus facilement, se séparer en petit groupe et de faire remonter des problèmes.

Le channel sql-final nous permettra de suivre les modifications à effectuer lorsqu'on fera la mise en commun du projet.



La base de données se situe sur la page suivante.



## 4/Le projet

### A/Gestion des produits

Après avoir créé la base de données, j'ai créé un repo GitHub et ajouté les membres de mon groupe. On pouvait avoir le même début de projet.

Dans cette première version on pouvait déjà commander à travailler. On avait aussi décidé d'y rajouter plus tard un dossier « crud » où on placera tous nos fichiers PHP afin que notre tuteur de stage puisse récupérer l'API en entier avec la base de données.

On a ensuite créé plusieurs branches pour travailler :

- Branch main
- Branch release
- Branch features
- Branch features-g1
- Branch features-g2
- Branch features-g3
- Puis une branche par personne. (Donc 6 en plus)



J'ai décidé alors sur ma branche de m'attaquer directement à la gestion des produits. Je vais expliquer comment j'ai fait pour la création des fichiers, le fonctionnement d'Angular et ce sera globalement la même chose pour les autres tâches que j'aurais fait.

Pour commencer, j'ai créé un component « gestion-produit » avec la commande suivante :

```
swerk@KARM ~ % cd marketplace/feature-oliwer
swerk@KARM ~ % ng g c view-produits
```

J'ai aussi créé un service produit ou je vais stocker mes fonctions de requêtes API avec la commande suivante :

```
swerk@KARM ~ % cd marketplace/feature-oliwer
swerk@KARM ~ % ng g s produits
```

Puis pour finir, j'ai créé, dans un dossier externe, mon API avec mes premiers fichiers pour les requêtes.

Dans mon fichier app-routing-module, qui regroupe toutes les routes du site, j'ai intégré le fichier la route vers le component créé :

```
import { ViewProduitsComponent } from './view-produits/view-produits.component';

const routes: Routes = [
  {path: 'view-product', component: ViewProduitsComponent}, // Voir tous Les produits (Admin)
];
```

Lorsque je démarre mon application avec la commande « ng serve » et que je me rends sur l'url <http://localhost:4200/view-product>, j'arrive sur une page blanche.



Je vais pouvoir afficher sur cette page mes produits. Notre tuteur de stage nous a dit de ne pas nous soucier du CSS car c'est lui qui le fera.

Le component se divise en 4 fichiers avec 4 extensions différentes :

- .html : Ecrire ce qu'on veut afficher sur la page
- .scss : Faire le design de la page
- .spec.ts : Effectuer des tests
- .ts : Appeler notre API et écrire du JS



Ce component permettra d'afficher gérer les produits d'un annonceur. Il aura accès seulement à ses produits mis en avant et non ceux des autres annonceurs.

Pour commencer, j'ai fait un affichage simple pour dire que quel page on se trouve :

```
<div class="container">
  <h2 class="text-center">Annonces (annonce acheté pour affichage)</h2>
  <div class="d-flex flex-row justify-content-around">
    </div>
</div>
```

J'ai créé ensuite le composant add-produit, pour ajouter un produit. Mon binôme a créé un service catégorie qui permettra de gérer les catégories, j'ai donc repris le sien.

Pour renseigner un produit, il faut renseigner un libellé, la quantité en stock, jusqu'à 3 images, une description, un prix et une catégorie. On doit alors utiliser un formulaire. Pour créer un formulaire avec Angular, on doit utiliser un « formBuilder » et le déclarer dans le constructeur comme ceci :

De même pour les services, ils doivent être importés dans le constructeur, afin de pouvoir les utiliser ensuite. Le formbuilder étant importé, je dois déclarer une variable, ici « addProduitForm » qui sera la référence pour le formulaire d'ajout de produit. Ensuite pour créer notre formulaire, on doit définir différents inputs, donc le libelleProduit, la quantité ect...

Notre formulaire est prêt.

Ensuite, je dois récupérer toutes les catégories qui existent en base de données pour pouvoir les afficher dans un input type select.

```
addProduitForm:any;

constructor(
  private formBuilder: FormBuilder,
  private router: Router,
  private ProduitsService: ProduitsService,
  private CategoriesService: CategoriesService,
  private http: HttpClient
) {
  this.addProduitForm = this.formBuilder.group({
    libelleProduit: ['', Validators.required],
    qtStock: ['', Validators.required],
    imageProduit: ['', Validators.required],
    descriptionProduit: ['', Validators.required],
    prixUnitHT: ['', Validators.required],
    idCategorieProduit: ['', Validators.required],
  })
}
```

Pour ça, on crée un autre variable, nommé catégorie, qui contiendra toutes ces informations. Pour remplir cette variable à l'initialisation, on utilise la fonction ngOnInit(). Cette fonction permet de faire des actions lorsque la page est prête, comme récupérer des informations.

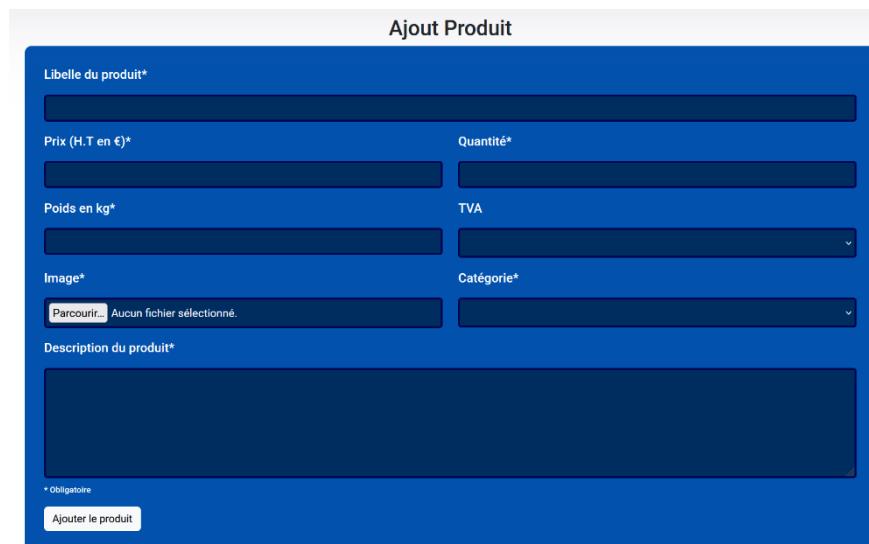
```
ngOnInit(): void {
  this.CategoriesService.getCategories().subscribe((data: any) => {
    this.categories = data.data;
  })
}
```

Maintenant, je pouvais intégrer mon formulaire dans le site. En HTML, on doit utiliser les « `formGroup` » d'Angular pour stocker les valeurs des inputs dans notre variable créés auparavant. Pour le faire, il faut rentrer ce code :

```
<div class="text-center">Ajoute un produit</div>
<form [formGroup]="addProduitForm" novalidate class="form">
  <div class="box-form">
    <div class="row justify-content-center">
      <div class="col d-flex flex-column">
        <p>Libelle du produit*</p>
        <input type="text" name="libelleProduit" formControlName="libelleProduit" required/>
      </div>
    </div>
    <div class="row justify-content-center">
      <div class="col d-flex flex-column mt-3">
        <p>Prix (H.T en €)*</p>
        <input type="text" name="prixUnitHT" formControlName="prixUnitHT" required/>
      </div>
    </div>
    <div class="col d-flex flex-column mt-3">
      <p>Quantité*</p>
      <input type="text" name="qtStock" formControlName="qtStock" required/>
    </div>
  </div>
</form>
```

On peut continuer le code en remplaçant avec les inputs notre formulaire. Il faut noter qu'il faut renseigner le nom de notre input avec « `formControlName` ».

Une fois fini, avec un peu de CSS, le résultat est :



Le poids et la TVA ont été ajouté plus tard dans le projet.

J'ai ensuite ajouté une fonction « `onSubmit()` » sur le click du bouton qui permet de valider le formulaire.

```
<button type="submit" class="btn btn-light" (click)="onSubmit()">Ajouter le produit</button>
```

De retour sur mon fichier TS, je déclare ma fonction `onSubmit()` et dedans je déclare un `formData`, qui va m'aider à envoyer des données dans l'API avec la méthode POST.

```
onSubmit() {
  if(this.selectedImages.length > 0) {
    const formData = new FormData();
    // Assigné une image à une variable puis l'ajouter dans le formData
    for (let i = 0; i < this.selectedImages.length; i++) {
      formData.append('image' + i, this.selectedImages[i], this.selectedImages[i].name);
    }
    // Ajouter tous les autres champs du formulaire
    formData.append('libelleProduit', this.addProduitForm.value.libelleProduit);
    formData.append('qtStock', this.addProduitForm.value.qtStock);
    formData.append('descriptionProduit', this.addProduitForm.value.descriptionProduit);
    formData.append('prixUnitHT', this.addProduitForm.value.prixUnitHT);
    formData.append('idCategorieProduit', this.addProduitForm.value.idCategorieProduit);
    console.log(formData)
    // Envoie via le service la requête vers le serveur
    this.ProduitsService.addProduit(formData).subscribe((data:any) => {
      console.log(data);
    })
    this.router.navigate(['/view-product']);
  }
}
```



Dans notre service, on déclare alors la fonction « addProduit » pour et on y précise le fichier PHP cible

```
export class ProduitsService {  
  constructor(private http:HttpClient) {}  
  
  baseUrl: string = 'http://localhost/api/';  
  
  addProduit(formData: FormData) {  
    return this.http.post(this.baseUrl + 'add-produit.php', formData)  
  }  
}
```

On va ensuite créer un fichier « add-produit.php » où on mettra notre requête SQL pour insérer le produit.

Dans le 1er screen, on déclare toutes les variables pour faciliter l'insertion dans la base de données.

Sur le 2ème screen, on exécute notre requête SQL qui permet l'insertion dans la base de données, le nouveau produit. Les images étant stockées dans une table différentes, on doit traiter ça avec une autre requête.

Sur le 3ème screen, on récupère le dernier produit ajouté dans la base de données, notamment son id, afin de le renseigner comme clé étrangère dans la table « images ».

Avec cette méthode, si on ajoute plusieurs images, on peut les stocker pour un produit

```
$libelleProduit = $_POST['libelleProduit'];  
$qtStock = $_POST['qtStock'];  
$descriptionProduit = $_POST['descriptionProduit'];  
$prixUnitHT = $_POST['prixUnitHT'];  
$poidsProduit = $_POST['poidsProduit'];  
$idCategorieProduit = $_POST['idCategorieProduit'];  
$idAnnonceurProduit = $_POST['idAnnonceurProduit'];  
$idTvaProduit = $_POST['idTvaProduit'];  
str_replace("", " ", $descriptionProduit);  
  
$images = array();  
  
if(isset($_FILES['image0'])) {  
  array_push($images, $_FILES['image0']);  
}  
if(isset($_FILES['image1'])) {  
  array_push($images, $_FILES['image1']);  
}  
if(isset($_FILES['image2'])) {  
  array_push($images, $_FILES['image2']);  
}  
  
try {  
  $req = $conn->prepare("INSERT INTO produits(  
    libelleProduit,  
    qtStock,  
    descriptionProduit,  
    prixUnitHT,  
    poidsProduit,  
    idCategorieProduit,  
    idAnnonceurProduit,  
    idTvaProduit  
) VALUES (  
    '$libelleProduit',  
    '$qtStock',  
    '$descriptionProduit',  
    '$prixUnitHT',  
    '$poidsProduit',  
    '$idCategorieProduit',  
    '$idAnnonceurProduit',  
    '$idTvaProduit'  
)");  
  $req->execute();  
  
  $req = $conn->prepare("SELECT max(idProduit) as nbProduit FROM produits");  
  $req->execute();  
  $data = $req->fetch();  
  
  $nb = $data['nbProduit'];  
  
  for ($i = 0; $i < count($images); $i++) {  
    if ($images[$i]['error'] == UPLOAD_ERR_OK) {  
      $tmp_name = $images[$i]['tmp_name'];  
      $name = $images[$i]['name'];  
  
      $file_ext = explode('.', $name);  
      $file_ext = strtolower(end($file_ext));  
  
      $name = $nb . '-' . $i . '.' . $file_ext;  
  
      move_uploaded_file($tmp_name, "photos/produits/$name");  
  
      $req = $conn->prepare("INSERT INTO image(libelleImage, idProduitImage) VALUES ('$name', '$nb')");  
      $req->execute();  
    }  
  }  
}  
catch (Exception $e) {  
  http_response_code(405);  
  echo json_encode([  
    'success' => 0,  
    'data' => $e->getMessage()  
  ]);  
  exit;  
}  
}
```

Maintenant, nos produits sont ajoutés dans la base de données. On apportera des modifications par la suite pour les lier à un annonceur. Pour ça, il faut qu'un autre groupe réussi à faire la connexion au site. Pendant ce temps, je suis revenu sur le component « view-produit » pour gérer l'affichage des produits pour l'annonceur connecté au site via son id. On va juste afficher les produits de l'annonceur 1.

Le problème de l'insertion est qu'on ne peut pas ajouter de virgule, je vais m'en occuper après dans mon stage.

De la même manière que pour les catégories, on récupère les produits d'un annonceur.

Ici je n'ai pas vraiment optimisé le code pour le moment, vu que je récupère toutes les images en même temps, alors que pour l'affichage des produits, une seule image suffit. Je reviendrais sur ce problème plus tard dans le stage.

J'attache chaque image à son produit, puis je peux ensuite commencer à faire l'affichage.

```
ngOnInit() {
  this.ProduitsService.getProduits(1).subscribe((data:any) => {
    data = data.data;
    for(let i = 0; i < data.images.length; i++) {

      if(this.images.has(data.images[i].idProduitImage)) {
        this.images.get(data.images[i].idProduitImage).push(data.images[i]);
      } else {
        this.images.set(data.images[i].idProduitImage, []);
        this.images.get(data.images[i].idProduitImage).push(data.images[i]);
      }
    }
    for (let y = 0; y < Object.keys(data).length -1; y++) {
      data[y].images = this.images.get(data[y].idProduit);
      this.produits.push(data[y]);
    }
  })
}
```

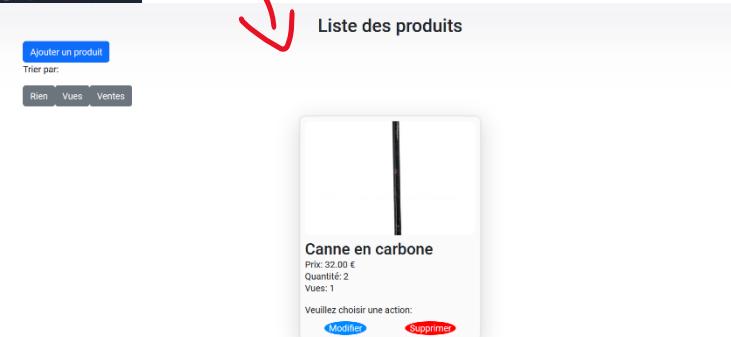
```
<div class="container">
  <h2>Liste des produits (Pour le vendeur numéro 1)</h2>
  <div class="d-flex flex-row justify-content-around">
    <div *ngFor="let produit of produits">
      <div class="card">
        
        <div class="infos-prod">
          <h3>{{produit.libelleProduit}}</h3>
          <p>Prix: {{produit.prixUnitaire}}</p>
          <p>Quantité: {{produit.qteStock}}</p>
          <p>Vues: {{produit.NbVues}}</p>
        </div>
        <div class="actions">
          <p>Veuillez choisir une action:</p>
          <div class="d-flex flex-row justify-content-around items-center">
            <div class="update">
              <button type="button" (click)="edit(produit.idProduit)"><mat-icon>create</mat-icon></button>
            </div>
            <div class="delete">
              <button type="button" (click)="deleteProduct(produit.idProduit)"><mat-icon>delete_sweep</mat-icon></button>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Je n'avais pas expliqué ça avant, mais on peut faire des conditions et des boucles dans notre fichier HTML avec Angular en utilisant des « \*ngIf » qui permettent de faire une condition (par exemple si un utilisateur est connecté, on va faire un \*ngIf='user' pour vérifier si cet objet existe) mais on peut aussi faire des boucles \*ngFor qui permettent d'afficher un à un, des données récupérées avec les requêtes. Par exemple, dans ce cas, je fais un \*ngFor='let produit of produits', sachant que notre variable « produits » a été déclarée côté TS, je peux la réutiliser dans mon HTML. Donc ce qu'il y a dans la div va se passer pour tous les produits, donc chaque produit aura sa card, avec un bouton de modification et un bouton de suppression

Ensuite j'ai créé une page « view-annonce » qui permettra d'afficher tous les produits de tous les annonceurs, une page avec tous les produits qui peuvent être achetés. J'ai modifié un peu la card pour l'affichage du produit sur cette page, sachant qu'on n'a pas besoin de toutes les informations comme le nombre de vues par exemple.

La fonction « ajouter au panier » et « souhait » seront ajoutées lorsqu'un autre binôme aura fait la tâche. Pour le moment, on peut cliquer sur « Voir » pour consulter le produit.

Avant de faire la page « Voir », j'ai décidé de finir avec la gestion des produits, c'est-à-dire les boutons suppression et modification.





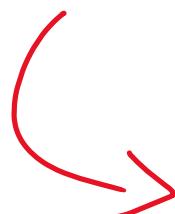
La suppression ce n'est rien de compliqué, lorsque je clique sur le bouton, je fais appel à une fonction « onDelete() » du service des produits, et je supprime avec une requête SQL le produit.

Pour l'edit, j'ai copié la page « view-produit » et j'ai créé un component « edit-produit » et coller l'HTML dedans. Ensuite grâce à la fonction « PatchValues » dans mon edit-produit.component.ts, j'ai pu remplir automatiquement les données des inputs, pour qu'on ne puisse modifier que les erreurs entrées et pas modifier le produit en entier. Pour savoir le produit à modifier, on fait passer dans l'URL l'id du produit, que l'on va ensuite récupérer, puis on exécute une requête pour récupérer ce produit. Pour pouvoir passer un paramètre dans l'URL, il faut le préciser dans notre app.routing.module :

```
{path: 'edit-product/:id', component: EditProduitComponent}, // Edit un produit
```

L'id récupéré, je peux afficher les données du produit :

```
ngOnInit(): void {
  this.idProduit = this.url.snapshot.params['id'];
  this.ProduitsService.getSingleProduits(this.idProduit).subscribe((data:any) => {
    this.addProduitForm.patchValue(data.data)
  })
}
```



Modifier un produit

Libelle du produit*	Canne en carbone	
Prix (H.T en €)*	32.00	Quantité*
Poids en kg*	3.00	TVA
Catégorie*	1 - Homme	
Description du produit*	ZEGZEG	
* Obligatoire		
<input type="button" value="Modifier le produit"/>		

Maintenant que j'ai fait ceci, j'ai décidé de rendre la fonction « Voir » disponible.

J'ai donc créé un nouveau component « view-single-produit ». Avec un id passé en paramètres dans l'URL, je récupère le produit et je l'affiche. Plus tard je mettrai une sécurité pour que les produits qui ne doivent pas être vendu, ne soient pas accessible.

Globalement la page se divisera en 2 parties :

- Le produit
- Les commentaires

Plus tard, on pourra rajouter d'autres informations sur cette page, pour le moment j'ai fait le strict minimum pour afficher un produit. Pour le produit, on a donc son nom, l'entreprise qui le vend, son prix, sa notation par les utilisateurs, sa description ainsi qu'un bouton pour l'ajouter au panier et ajouter à la wishlist, cependant les icônes ne veulent pas très bien marcher, j'ai décidé de les laisser ici pour montrer qu'il faudra des icônes pour mieux guider l'utilisateur.

**Canne en carbone**

Vendu par: Entreprise

**32.00 €**

(5/5) 1 avis

Description:

ZEGZEG

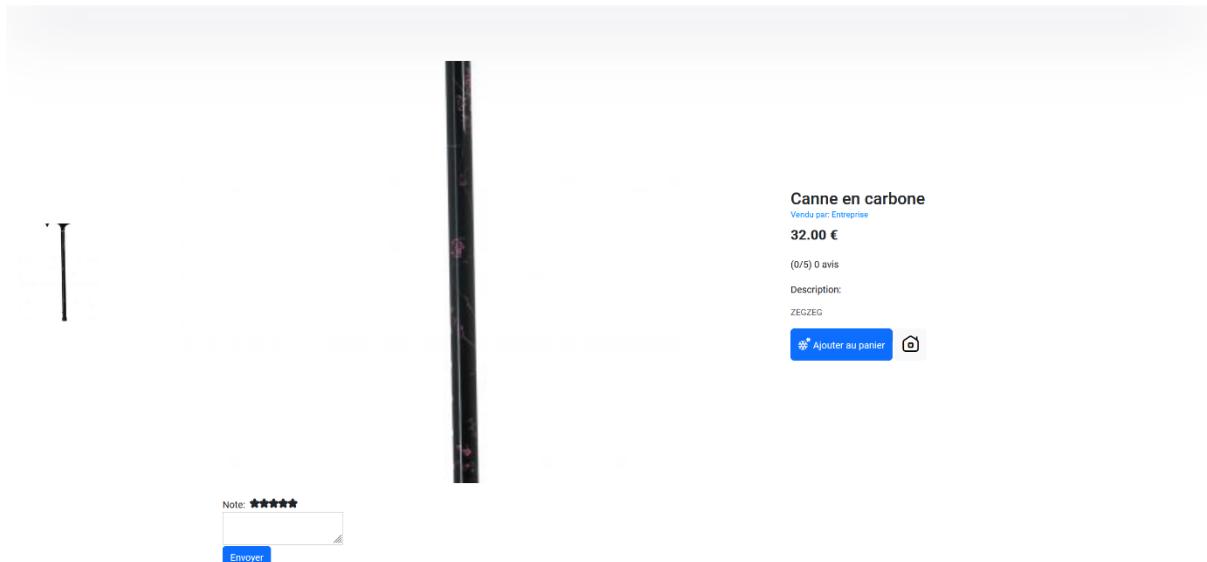
Ajouter au panier



En scrollant un peu plus bas, j'ai fait une petite section de commentaires avec l'avis de chaque utilisateur sur le produit ainsi que la note attribuée par un utilisateur au produit.

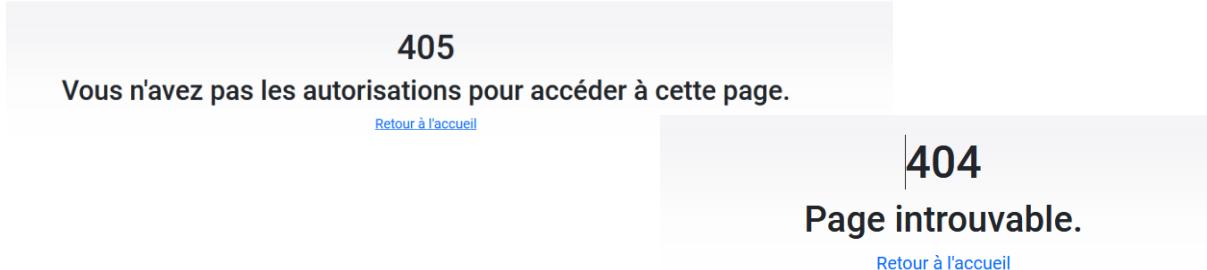
Pour faire en sorte ne pas pouvoir laisser plein de commentaires, ils doivent être validés par un administrateur avant d'être affiché. Avec une colonne « valid » dans la table « avis », si le commentaire est à 0, il ne sera pas affiché, sinon il sera à 1.

Les personnes peuvent aussi déposer seulement 1 commentaire par produit.



La page ressemble à ça, avec en plein milieu la photo du produit, sur la gauche éventuellement d'autres photos. Le CSS sera fait par mon tuteur de stage, donc je n'avais pas vraiment besoin de le faire.

Avant le merge, j'ai décidé de faire 2 pages, une qui renvoie l'erreur 404 et une 405.



Pendant que je faisais ces tâches, j'ai pu aider mes coéquipiers à résoudre leurs problèmes dans le code.

Pour la connexion au site, on rencontrait un problème car certains modules ne pouvaient pas être installés sous Angular comme JWT par exemple, qui était censé assurer le token de connexion de notre utilisateur sur le site. On a essayé différentes manières pour faire marcher le module mais sans succès. La solution qu'on a adaptée était de générer le token côté serveur pas client. Avec Composer, on a pu

installer JWT sur le serveur PHP et faire appel à lui. Après pour la connexion, on a regardé encore ensemble car lorsque l'utilisateur se connectait, le site ne voulait pas le registrer comme connecté.

Mon binôme avait besoin d'aide pour l'insertion des catégories, car son code ne marchait pas, je lui ai montré une autre technique pour faire fonctionner sa tâche.

Sinon globalement, c'était de l'aide pour des petits bugs.

## B/Le premier merge

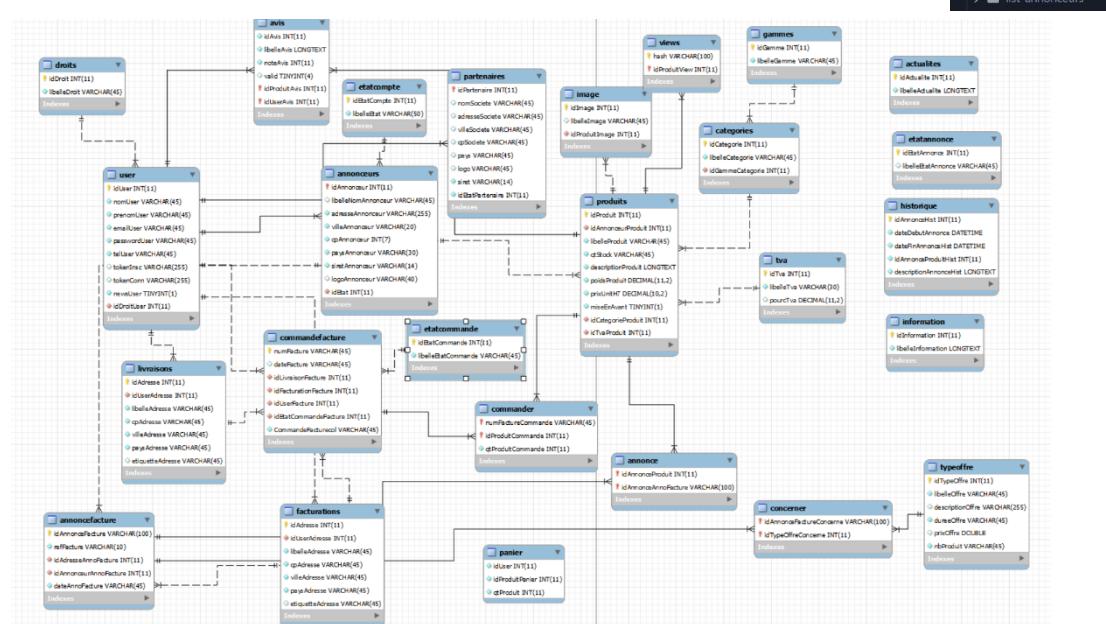
Après une semaine de travail, on a décidé de merge pour que notre tuteur puisse commencer à faire le front du site. J'ai décidé de merge pour tout le monde sachant que je suis à l'aise avec GitKraken et pour éviter les problèmes. A chaque merge, il faut modifier les fichiers et choisir ce que l'on souhaite garder et supprimer. Les merges en plus n'en sont presque pas automatiquement, certains bouts de code se chevauchent, donc ça prenait un peu plus de temps.

Le projet commençait alors à prendre un peu de forme, on a été surpris de voir le nombre de fichiers totaux générés par le merge. On a eu une incompréhension des tâches lors du projet, un membre a empiété sur les tâches des autres ce qui a généré énormément de fichier en plus. On a décidé de garder pour le moment ces fichiers car ils pourraient nous être utile pour la suite. Si jamais ces fichiers finalement sont inutiles, on les supprimera du projet.

On a passé une journée à merge le projet et avancer sur notre rapport de stage, pour corriger des bugs et incohérence, recréer le script de la base de données qui a évolué.

Finalement, on a réussi à mettre en place le projet. On a donc pu récupérer la version sur nos branches respectives et continuer à travailler.

Notre tuteur a lui aussi pu commencer à travailler le front du site.



Ce merge a aussi permis à commencer à faire les vérifications pour voir si l'utilisateur est connecté, s'il a les droits d'accéder à une page. C'est indispensable au bon fonctionnement du site sinon un client pourrait mettre en vente son propre produit. Grâce à ce code, on peut voir si l'utilisateur connecté est un annonceur ou non.

```
if(!localStorage.getItem('authUser')) {
  this.router.navigate(['/login']);
} else {
  this.user = JSON.parse(localStorage.getItem('authUser')!);
  if(this.user.roles != "annonceur") {
    this.router.navigate(['/405']);
  } else {
```

## C/Paiement des annonces

Maintenant les principaux components étant fait, tous les produits ne peuvent pas être affichés sur le site. Afin de pouvoir vendre un produit, l'annonceur doit acheter une offre d'annonce, qui lui permet d'afficher X nombre de produit. Sans payer d'offre d'annonce, l'annonceur ne peut pas vendre de produit, juste en ajouter au site.

Pour commencer, il fallait trouver un module de paiement. Je pensais que je devrais sauvegarder les données comme la carte bancaire dans la base de données pour le paiement, sauf que ce n'était pas obligatoire vu que mon maître de stage m'a conseillé d'utiliser le module de paiement Stripe. Stripe est un module très connu, utiliser pour différents sites et applications comme Deliveroo ou l'école IPSSI, il est facile à mettre en place sans vraiment de grosse difficulté.

Tout d'abord j'avais essayé d'installer ce module sur Angular encore une fois, sauf que Stripe a besoin de modules qui ne sont pas disponibles sous Angular. Comme pour JWT, on a décidé de passer côté serveur, avec une page hébergée sur celui-ci, qui nous mènera sur la page de paiement.

Une fois le plan décidé, j'ai décidé de me mettre au travail.

J'ai commencé par créer un panel pour l'administration des offres d'annonces.

Pour commencer j'ai fait une page simple qui les offre d'annonces. J'ai ajouté manuellement une offre dans la base de données pour avoir une preview de l'affichage.

Cette page est accessible pour les annonceurs uniquement.

Ensuite, j'ai créé une page pour l'administration de ces offres d'annonces. Comme pour les produits, j'ai créé un ajout, suppression et modification.

## Offres annonces

4.99 €

Offre Standard

Durée: 1 mois

Nombre produits: 3

Acheter

## Gestion des annonces

Ajouter une offre

### Offre Standard

Durée de l'offre: 1 Mois

Prix de l'offre: 4.99

Nombre de produit: 3

Modifier Supprimer

Pour les formulaires, j'ai utilisé NG Material est une bibliothèque de component. C'est très utilisé surtout pour les appareils Google qui l'utilise énormément.

J'ai dû utiliser NG Matériaux car ce sera plus simple pour notre tuteur de stage de faire la partie graphique du site. Je devrais donc modifier aussi les inputs que j'ai pu mettre dans les pages pour la modification, ajout et suppression des produits.

J'ai décidé cependant de me concentrer sur le module de paiement étant donné que j'étais lancé dessus.

Les components d'NG Materials sont plus beaux, donc le formulaire est déjà un peu plus beau juste avec ça.

Pour ajouter une offre d'annonce, il faut saisir son libelle, une durée, c'est-à-dire au bout de combien de temps elle expire, un prix pour cette offre, le nombre de produits que je peux vendre l'annonciateur grâce à cette offre ainsi qu'une description rapide de l'offre.

Je n'avais pas encore géré la possibilité de rentrer des virgules dans la base de données, j'ai donc profité de cette tâche pour le faire. Pour pallier à ce problème on peut utiliser la fonction « replace » qui remplace des caractères dans une phrase.

```
FormData.append('prixOffre', (this.addForm.value.prixOffre).replace(',', '.'));
```

Ici on remplacera la virgule par un point pour que ce soit acceptable en base de données.

## Ajout d'une offre

Libellé de l'offre*	Durée de l'offre en mois*
Prix de l'offre en €*	Nombre de produit*
Description du produit*	

**Ajouter l'offre**

Front-end : [HTML](#) [React](#) [Next.js](#) Back-end : [Ruby](#) [Node](#) [PHP](#) [Python](#) [Go](#) [.NET](#) [Java](#)

### Page Checkout préconfigurée

[Page Checkout préconfigurée](#) Tunnel de paiement personnalisé  
Explorez un échantillon de code complet et fonctionnel d'une intégration avec Stripe Checkout. Le code côté client et côté serveur redirige vers une page de paiement préconfigurée hébergée sur Stripe.

Avant de démarrer, confirmez les moyens de paiement que vous souhaitez proposer dans vos paramètres de moyen de paiement. Par défaut, nous activons les cartes bancaires et d'autres moyens de paiement courants pour vous, et nous vous recommandons d'activer des moyens de paiement supplémentaires pertinents pour votre entreprise et vos clients.

[Télécharger l'application complète](#)

Vous ne codez pas ? Utilisez les options no-code de Stripe ou contactez nos partenaires pour obtenir de l'aide.

#### Configurer le serveur

##### Installer la bibliothèque PHP

Installez la bibliothèque avec composer et initialisez-la avec votre clé API secrète. Si vous partez de zéro et que vous avez besoin d'un fichier composer.json, vous pouvez également télécharger les fichiers à l'aide du lien dans l'éditeur de code.

[Composer](#) [GitHub](#)

##### Installez la bibliothèque :

```
$ composer require stripe/stripe-php
```

Version beta

Stubborn Attachments 20.00 \$US

**Checkout**

checkout.php secrets.php checkout.html success.html cancel.html Télécharger ↴

```
1 <?php
2
3 require_once '../vendor/autoload.php';
4 require_once '../secrets.php';
5
6 \Stripe\Stripe::setApiKey($stripeSecretKey);
7 header('Content-Type: application/json');
8
9 $YOUR_DOMAIN = "http://localhost:4242";
10
11 $checkout_session = \Stripe\Checkout\Session::create([
12   'line_items' => [
13     # Provide the exact Price ID (e.g. pr_1234) of the product you want to sell
14     'price' => "{{PRICE_ID}}",
15     'quantity' => 1,
16   ],
17   'mode' => 'payment',
18   'success_url' => "$YOUR_DOMAIN /success.html",
19   'cancel_url' => "$YOUR_DOMAIN /cancel.html",
20 ]);
21
22 header("HTTP/1.1 303 See Other");
23 header("Location: " . $checkout_session->url);
```

On arrive sur cette page-là lorsqu'on souhaite payer. Etant donné que cette page est stockée sur le même serveur que l'API, on peut faire directement des requêtes SQL depuis cette page. En passant en paramètres les informations telles que l'utilisateur, l'id de l'annonce et l'id de l'offre, on peut récupérer et stocker pas mal d'information.

 <localhost/api/paiement/public/checkout.html?id=1&uid=1&idFact=1>

Je me suis dit que sur le site de Stripe, il serait possible de consulter qui a commandé cette offre par exemple, donc j'ai passé aussi l'id de l'utilisateur.

Pour faire des tests de paiement, on peut utiliser la carte 4242 4242 4242 4242, en date d'expiration et CVC on peut mettre n'importe quoi.

Pour finaliser le paiement, on clique sur le bouton pour confirmer le paiement. L'application se charge alors de contacter son API à elle, puis d'enregistrer sur son serveur le paiement ainsi que plusieurs informations passé dans le metadata.

Pour pouvoir utiliser l'API de Stripe, il faut une clé privée et clé publique. La clé privée ne doit surtout jamais sortir en dehors du serveur, sinon des personnes malveillantes pourraient avoir accès à notre compte. Pour faire ça, on peut stocker directement la clé dans un fichier secrets.php qui contiendra notre clé, et pourra être appelé si besoin.

Une fois le paiement accepté, j'ai créé une page « checkout-success » qui va faire un récapitulatif de la commande passé.

**Paiement**

4.99 €  
E-mail

Carte bancaire  EPS  giropay  IDEAL

Numéro de carte  
1234 1234 1234 1234

Date d'expiration  
MM / AA      CVC

Pays  
France

**Confirmer le paiement**

Après avoir confirmé le paiement, vous serez redirigé.  
Powered by stripe

```
// Create a PaymentIntent with amount and currency
$paymentIntent = \Stripe\PaymentIntent::create([
    'amount' => calculateOrderAmount(),
    'currency' => 'eur',
    'metadata' => [
        'userid' => $_GET['uid'],
        'prdid' => $_GET['id'],
        'idFact' => $_GET['idFact']
    ],
    'automatic_payment_methods' => [
        'enabled' => true,
    ],
]);
```

**MERCI POUR VOTRE ACHAT !**

La référence de votre commande est : #1ADF22348D

Vous pouvez consulter vos commandes en cliquant [ici](#)

**Recapitulatif**

Adresse : erh	Prénom : Annonceur
Code postal : 23532	Nom : 1
Ville : erh	Email : anno1@gmail.com
Pays : FDGR	

Téléchargez votre facture ici: [#1ADF22348D](#)

Sur le site de Stripe, on peut voir un nouveau paiement effectué :

**Paiements**

Tous	Réussis	Remboursés	Non capturés	En échec
<input type="checkbox"/> Date	<input type="checkbox"/> Montant	<input checked="" type="checkbox"/> État   Réussi	<input type="checkbox"/> Moyen de paiement	
<input type="button" value="Effacer les filtres"/>				
MONTANT	DESCRIPTION	CLIENT	DATE	...
4.99 €	Réussi ✓ pi_3MYsijGKuCg1RS3J1y7mLJ6G	oliver721@gmail.com	7 févr. à 16:16	...

Lorsque l'on clique dessus, on a des informations supplémentaires sur la commande

**PAIEMENT**

**4,99 € EUR** Réussi ✓

Dernière mise à jour 7 févr. à 16:16	Client Aucun	Moyen de paiement **** 4242	Évaluation des risques Risque normal
---	-----------------	--------------------------------	---

**Chronologie**

- Paiement réussi  
7 févr. 2023 à 16:16
- Paiement démarré  
7 févr. 2023 à 16:04

**Détails du paiement**

Libellé de relevé bancaire	ALTAMEOS MULTIMEDIA
Montant	4,99 €
Frais	0,39 €
Net	4,60 €
État	Réussi
Description	Aucune description

De notre côté, on enregistre aussi ce paiement sur notre base de données avec quelques informations

←→	idAnnonceFacture	refFacture	idAdresseAnnoFacture	idAnnonceurAnnoFacture	dateDebutAnnonce	dateAnnoFacture
<input type="checkbox"/> Copier	pi_3MYsijGKuCg1RS3J1y7mLJ6G	1ADF22348D	1	1	NULL	07-02-23

L'idAnnonceFacture a le même ID que le site, la refFacture est un hash md5 de longueur 10 effectué sur l'ID de l'annonce, le dateDebutAnnonce permet de ne pas activer instantanément l'offre, ce sera à l'utilisateur de l'activé.

Une fois ceci fait, je devais générer une facture pour le paiement. Pendant ma première année de BTS, on a déjà pu utiliser très rapidement FPDF, j'ai donc décidé de partir sur la même solution. FPDF est une classe PHP qui permet de générer des fichiers PDF. Pour l'installer, il suffit de le télécharger et de le mettre sur notre serveur et de l'importer dans le dossier souhaité.

Une fois installé, je devais regarder la documentation pour faire le PDF. Il faut déclarer chaque chose que l'on veut imprimer sur ce PDF avec des fonctions

Par exemple avec ce code, on peut créer l'entête du tableau, il faut définir la position à la main et comparer avec ce qui se passe sur le fichier PDF.

A la fin, on peut enregistrer le fichier en local puis le récupérer côté Angular avec un lien qui pointe le fichier.

```
$pdf->Output();
// Sauvegarder le fichier pdf en local
$nomfacture = $_GET['hash'];
$pdf->Output('F', '../factures/'.$nomfacture.'.pdf');
```

```
// header
$this->SetFont('Arial','b',10);
$this->SetFillColor(255,0,0);
$this->SetDrawColor(189,189,189);
$this->Cell(40,7,'Produit',1);
$this->Cell(30,7,'Quantite',1);
$this->Cell(40,7,'Prix HT',1);
$this->Cell(30,7,'TVA',1);
$this->Cell(40,7,'Prix TTC',1);
$this->Ln(10);
```

En cliquant sur ce lien, je suis redirigé vers le fichier pdf.

Téléchargez votre facture ici: #F22F49C179

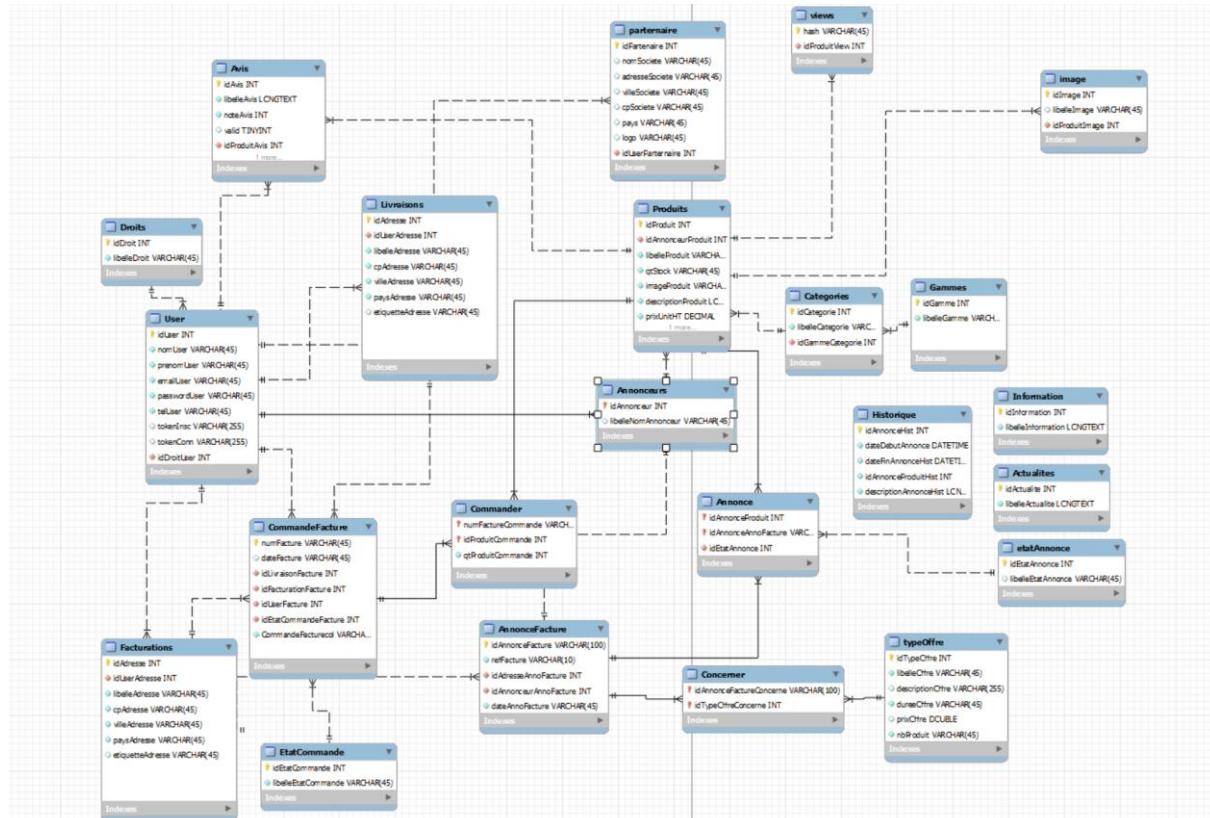
J'ai décidé de faire le calcul de la TVA, vu que j'en aurais besoin plus tard normalement, je n'ai pas encore fixé ça avec mon tuteur de stage. Pour le moment j'ai mis donc 20% sur le prix HT du produit.

Le problème maintenant est que si l'utilisateur change son adresse de facturation, ça changera celui de l'adresse de la facture. Je vais voir pour régler ce problème après, sachant qu'on avait un deuxième merge à faire et commencer à travailler la page d'accueil.

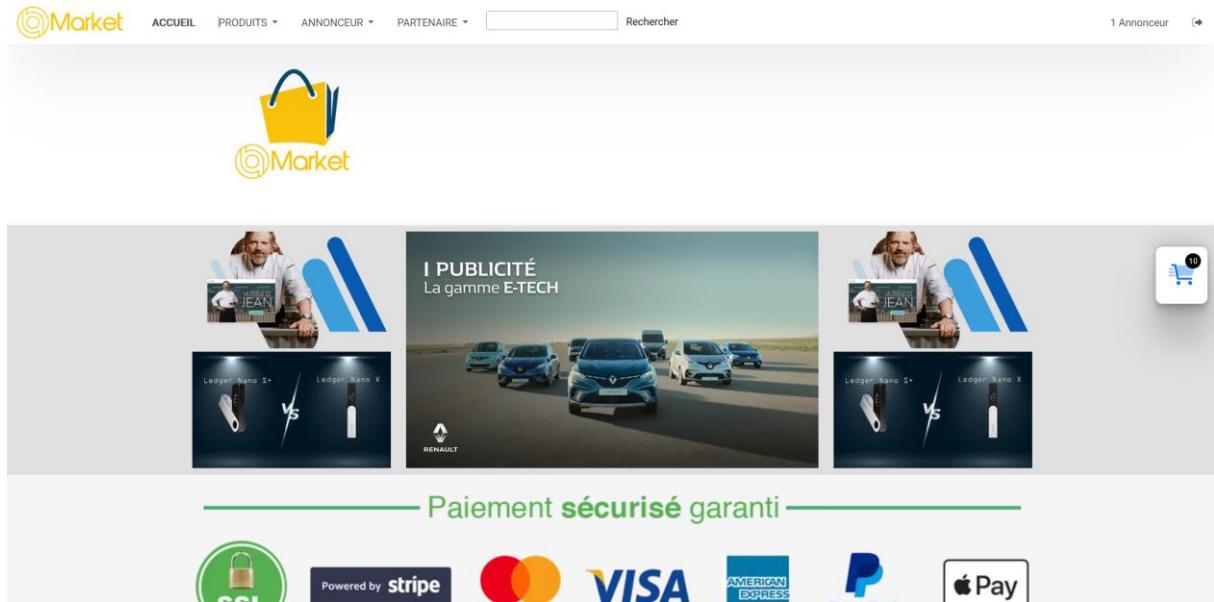
Dans la globalité, on pouvait déjà acheter des annonces. Il ne restera plus qu'à lier les produits à une offre et les gérer.

## E/Le deuxième merge

Pour le 2<sup>ème</sup> merge, notre tuteur a mis en place plus ou moins la forme de la page d'accueil et quelques éléments en plus. De notre côté, le merge nous a pris la journée afin de corriger les erreurs dans le code et assurer la compatibilité. On a aussi mis en commun les changements dans la base de données.



On a par exemple supprimé la table « paiement » car on n'en a finalement pas besoin, Stripe gère déjà le paiement tout seul. Sinon globalement, ce sont des ajouts dans des tables déjà existantes.



La page d'accueil lorsqu'on arrive sur le site ressemble à ça. La barre de recherche devra être ajouté au niveau du logo du site. Lorsqu'on scroll vers le bas, on retrouvera les produits du site.

## F/Reformater l'affichage

Maintenant que nous avons reçu les premiers bouts du front, on peut commencer à mettre en place tout ce qu'on a préparé.

Tout d'abord, je devais afficher les produits à afficher sur le site. Une catégorie « A la une » sera d'abord présente, puis on affichera ensuite des produits mis en avant par catégorie. Pour la mise en avant des produits, j'ai décidé de créer 2 colonnes en plus dans la base de données au niveau du produit. Une colonne « miseEnAvant » et une « miseEnAvantCat ». Si la valeur est à un, on peut mettre le produit en avant, selon le scénario. Je devais aussi gérer des produits « administrateurs », car des admins pouvaient ajouter leur propre produit, qui seront mis en avant. En plus de ça, faut que les produits afficher soit contenu dans une offre, sinon on ne les afficher pas. C'est une tâche qui peut paraître facile mais qui est assez compliquée au final.

Je devais donc afficher, pour les produits « A la une », les produits :

- En vente par un annonceur via une offre d'annonce
- Administrateur mis en avant
- Qui sont validés à la main par un administrateur

Et pour les produits mis en avant par catégorie, les produits :

- En vente par un annonceur via une offre d'annonce qui font partie de la catégorie
- Les produits administrateurs mis en avant et qui font partie de la catégorie
- Qui sont validés à la main par un administrateur

J'aurais pu faire ça en plusieurs requêtes SQL, mais ça voudrait dire faire travailler le serveur un peu plus qu'il ne le faut.

Il faut aussi modifier les permissions d'accès à certaines pages à l'administrateur :

```
if(this.user.roles != "annonceur" && this.user.roles != "superadmin") {
  this.router.navigate(['/405']);
```



Pour l'affichage des produits, j'ai d'abord créé un component « produit-mis-en-avant » qui permettrait de le réutiliser sur d'autres pages s'il le faut. J'ai repris les blocs des produits créé par le tuteur de stage et je les ai formatés pour qu'ils puissent recevoir des données. Par ailleurs, pour l'affichage, on devra afficher les produits dans un carrousel. Mon binôme avait déjà fait le carrousel, donc je lui ai demandé de m'expliquer comment ça marchait.

Ce code permet d'afficher un produit dans le carrousel, on y affiche sa note, le nom du produit, le prix ainsi que la description de celui-ci.

On a différentes options comme l'ajout à la wishlist, l'ajout au panier ou encore allez consulter le produit sur sa page.

Ensuite, pour récupérer les données à afficher, il faut faire une requête SQL assez spécifique.

```
<div class="row slideProduits carouselEnAvant">
  <owl-carousel-o [options]="customOptions">
    <ng-container *ngFor="let produit of produitsEnAvant">
      <ng-template carouselSlide [id]="produit.idProduit">
        <div class="itemProduit" style="width:300px; height: 480px;">
          
          <h3>{{produit.libelleProduit}}</h3>
          <div class="bt1">
            <label "ngIf":index <= produit.averageNote><i class="fa-solid fa-star"></i></label>
            <label "ngIf":index > produit.averageNote><i class="fa-light fa-star"></i></label>
          </div>
          <div class="presentation">
            {{produit.descriptionProduit}}
          </div>
        </div>
        <div class="note">{{produit.prixProduit}} </div>
        <div class="actions">
          <a class="btn btn-primary" routerLink="/view-annonce/{{produit.idProduit}}" title="Voir la fiche produit"><i class="fa-duotone fa-eye"></i><span>Voir</span></a>
          <a class="btn btn-primary" title="Ajouter au panier" (click)="addToCart(produit.idProduit)"><i class="fa-duotone fa-cart-plus"></i><span>Panier</span></a>
          <a class="btn btn-primary" title="Ajouter à la liste de souhaits" (click)="addToWishlist(produit.idProduit)"><i class="fa-duotone fa-file-plus"></i><span>Souhait</span></a>
        </div>
      </ng-template>
    </ng-container>
    <ng-template carouselSlide [id]="">
      <div class="itemProduit" style="width:300px; height: 480px;">
        ...
      </div>
    </ng-template>
  </owl-carousel-o>
</div>
```

```
$sql = "SELECT Produits.idProduit, descriptionProduit, prixUnitHT as prixProduit, Produits.libelleProduit,
AVG(CASE WHEN Avis.valid = 1 THEN Avis.noteAvis ELSE NULL END) as averageNote, MAX(Image.libelleImage) as image
FROM Produits
LEFT JOIN Avis ON Produits.idProduit = Avis.idProduitAvis
LEFT JOIN Image ON Produits.idProduit = Image.idProduitImage
WHERE (adminProduit = 1 AND miseEnAvant = 1)
OR (miseEnAvant = 1 AND idProduit IN (SELECT idAnnonceProduit FROM Annonce WHERE idEtatAnnonce = 2))
GROUP BY Produits.idProduit, Produits.libelleProduit;"
```

Cette requête permet de répondre aux contraintes un peu plus en haut. Les produits s'affichent ensuite sous cette forme. Ici ce sont des produits mis en avant par l'annonceur et qui appartiennent à une offre.

A la Une

+Vus +Vendus +Notés

**Déambulateur**

★★★★★

Description

23.00 €

[Voir](#) [Panier](#) [Souhait](#)

**Fauteuil Roulant**

★★★★★

Description produit

234.00 €

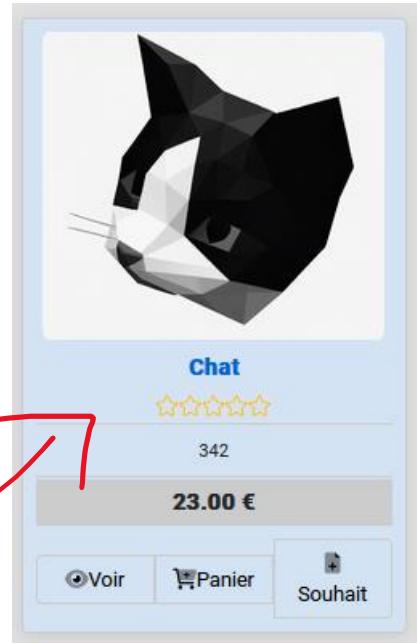
[Voir](#) [Panier](#) [Souhait](#)

Voir tous les produits

...

Ces produits sont dans un carrousel, ils vont défiler automatiquement. A côté de ce carrousel, je devais placer les derniers produits qui ont été mis en vente. Pour ça j'ai fait une requête qui récupère le dernier produit et qui le place dans une case.

```
$sql = "SELECT Produits.idProduit, descriptionProduit, prixUnitHT as prixProduit,
Produits.libelleProduit,
AVG(CASE WHEN Avis.valid = 1 THEN Avis.noteAvis ELSE NULL END) as averageNote,
MAX(Image.libelleImage) as image
FROM Produits
LEFT JOIN Avis ON Produits.idProduit = Avis.idProduitAvis
LEFT JOIN Image ON Produits.idProduit = Image.idProduitImage
WHERE idProduit IN (SELECT idProduit FROM Produits WHERE miseEnAvant = 2)
OR adminProduit = 1
GROUP BY Produits.idProduit, Produits.libelleProduit
ORDER BY Produits.idProduit DESC
LIMIT 1;";
$stmt = $conn->prepare($sql);
$stmt->execute();
```



Pour finir l'affichage de ces produits, j'ai fait l'affichage des produits mis en avant pour les catégories. J'ai repris le même principe que pour les produits à la une, j'ai créé un component, puis une requête et le formatage pour l'affichage :

```
<div class="row slideProducts carrouselEnAvant">
<owl-carousel>{options}</owl-carousel>
<ng-template carrouselSlide [id]="produit.idProduit">
<div class="itemProduct" style="width: 300px; height: 480px;">

<h3>{{produit.libelleProduit}}</h3>
<div class="btt">
<ngform>let star of [1,2,3,4,5], let index</ngform>
<label >ngIf<index < produit.averageNote><i class="fa-solid fa-star"></i></label>
<label >ngIf<index > produit.averageNote><i class="fa-light fa-star"></i></label>
</label>
<div class="presentation">
{{produit.descriptionProduit}}
</div>
<div class="note">{{produit.prixProduit}}</div>
<div class="actions">
<a class="btn bts-primary" routerLink="/view-annonce/{{produit.idProduit}}" title="Voir la fiche"><i class="fa-eye"></i>Voir</a>
<a class="btn bts-primary" title="Ajouter au panier" (click)=addCart(produit.idProduit)"><i class="fa-duotone fa-cart-plus"></i>Panier</a>
<a class="btn bts-primary" title="Ajouter à la liste de souhaits" (click)=addToWishlist(produit.idProduit)"><i class="fa-duotone fa-file-plus"></i>Souhait</a>
</a>


```

Catégorie: Mobilité

+Vus +Vendus +Notés

**OFFRE EXCEPTIONNELLE**  
**PACK DOMAINE**  
+ HÉBERGEMENT WEB OFFERT  
à partir de  
**0,99**

Voir tous les produits

**Fauteuil Roulant**  
★★★★★  
Description produit  
234.00 €  
Voir Panier Souhait

Dans les catégories, on peut voir une publicité, qui sera mise en place par mon binôme. Lorsqu'il aura fini de travailler dessus, on mettra la pub dynamiquement et aléatoirement. De même pour trier en fonction des +vendus, +vues et +notés, ce sera à lui de s'en occuper.

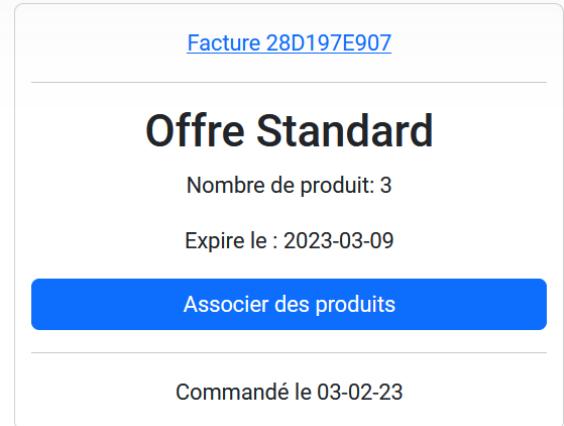
Maintenant cette partie étant faite, j'ai décidé de gérer les produits liés à une offre. Un autre binôme s'occupera de l'acceptation par l'administrateur pour valider ou non l'offre, moi je ferai la partie où l'annonceur peut choisir le produit qu'il souhaite mettre en vente via une annonce.

## G/Affichage des produits via offres

Dans cette partie, il faut que l'annonceur puisse associer des produits à afficher via une offre. J'ai donc créé un component « view-offres-annonceur » qui permet de voir l'intégralité des offres de l'annonceur connecté. J'avais donc fait cet affichage-là :

L'annonceur peut avoir plusieurs informations sur son offre, comme sa date d'expiration, le nombre de produits, avoir accès à sa facture, et associer ses produits. Cependant, mon tuteur de stage m'a demandé de modifier cet affichage, afin que l'offre ne s'active pas directement, mais quand l'annonceur le souhaite. J'ai rajouté une colonne pour la date de début de l'offre dans la base de données, qui sera d'abord null et quand on cliquera pour activer, la date sera remplacée avec celle de la date d'aujourd'hui + la durée de l'offre, pour savoir quand est-ce que cette offre ne serait plus disponible.

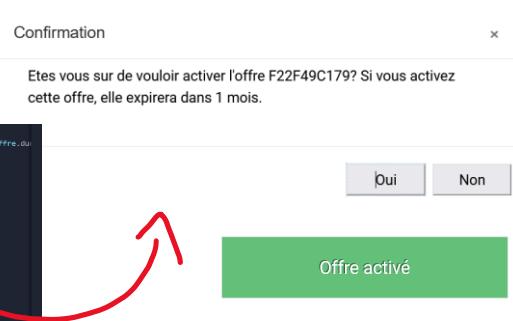
Une fois les modifications apportées, on a cet affichage :



The screenshot shows three cards for offers with IDs Facture 28D197E907, Facture E05C8D2F11, and Facture F22F49C179. All three are titled "Offre Standard". Each card shows the number of products (3), the expiration date (2023-03-09 or "Offre non activé"), and a blue "Associer des produits" button. Below each card, it says "Commandé le 03-02-23" or "Commandé le 07-02-23".

Grâce au module « alertifyjs » trouvé par mon binôme, on peut faire des boîtes de confirmation, succès ou erreur assez simplement.

```
activateOffer(offre:any) {
  alertify.confirm("Confirmation", "Etes vous sur de vouloir activer l'offre ${offre.refFacture}? Si vous activez cette offre, elle expirera dans ${offre.dur} mois.", function(data:any) => {
    console.log(data);
    if(data.success == "200") {
      alertify.success(data.data.date);
      setTimeout(() => {
        window.location.reload();
      },1000)
    } else {
      alertify.error(data.data);
    }
  });
},()>{
  $().set({transition:'zoom', resizable:'true', mouvable:'false', labels:{ok:'Oui', cancel:'Non'}});
}
```



Quand on valide, l'offre active et la date d'expiration sont renseignées. L'annonceur peut ensuite cliquer sur le bouton pour associer des produits. Il redirigera vers un component « associer-offre-produit ».

Sur cette page, on fait un affichage classique : l'image du produit et une checkbox. Lorsqu'on cliquera sur cette checkbox, notre produit sera mis en attente de la validation de l'administrateur.



Une problématique qui est arrivée c'est qu'un utilisateur pourrait lier un produit à plusieurs offres, je devais gérer ce petit problème côté SQL et renvoyer une erreur.

```
updateStatus(idProduit: any) {
  let element = document.getElementById(idProduit) as HTMLInputElement;
  if(element.checked) {
    this.annoncesService.modifAffichageProduit(idProduit, this.offre.refFacture, 1).subscribe((data:any) => {
      if(data.success == 500) {
        element.checked = false;
        alertifyjs.error(data.data);
      } else {
        alertifyjs.success(data.data);
      }
    })
  } else {
    this.annoncesService.modifAffichageProduit(idProduit, this.offre.refFacture, 0).subscribe((data:any) => {
      if(data.success == 500) {
        alertifyjs.error(data.data);
      } else {
        alertifyjs.success(data.data);
      }
    })
  }
}
```

Ajout effectué avec succès. Il sera vérifié par un administrateur dans les 24h.

Ce produit est déjà dans une facture.

Il fallait aussi afficher, dans une offre, les produits qui sont déjà liés à celle-ci. Pour ça, j'ai traité les données après les avoirs récupérés depuis la base de données. On cochera automatiquement la case :

```
this.annoncesService.getProduitAnnonce(ref).subscribe((data3:any) => {
  data3.data.forEach(element:<any> => {
    let element2 = document.getElementById(element.idAnnonceProduit) as HTMLInputElement;
    element2.checked = true;
  });
});
```

au chargement de la page



## H/Correction d'erreurs

Lors du projet, on s'est rendu compte que l'insertion des apostrophes ne passait pas dans la base de données. Un autre binôme a trouvé la solution pour pouvoir les insérer. Côté SQL, on devait modifier les informations ou il y aurait potentiellement des apostrophes et faire cette ligne de code :

```
$libelleProduit = str_replace("'", "\'', $libelleProduit);
$descriptionProduit = str_replace("'", "\'', $descriptionProduit);
```

Et modifier notre requête SQL :

```
//Update SQL
$sql = 'UPDATE produits
SET libelleProduit = "'.$libelleProduit.'"
, qtStock = "'.$qtStock.'"
, descriptionProduit = "'.$descriptionProduit.'"
, prixUnitHT = "'.$prixUnitHT.'"
, poidsProduit = "'.$poidsProduit.'"
, idCategorieProduit = "'.$idCategorieProduit.'"
, idTvaProduit = "'.$idTvaProduit.'"
WHERE idProduit = "'.$idProduit.''";
$stmt = $conn->prepare($sql);
$stmt->execute();
```

Un autre souci était que nous n'avions pas de d'id de session de paiement. La personne pouvait donc changer des informations et faire des paiements pour d'autres offres d'annonces pour payer (payer 5€ au lieu de 20€ par exemple). J'ai donc mis en place un ID de session, qui est généré lorsque l'on clique sur le paiement et est lié à un utilisateur. Il regroupe la date, l'user id et l'id du produit.

```
//Session
let date = new Date();
let dateString = date.getFullYear() + '-' + (date.getMonth() + 1) + '-' + date.getDate() + ' ' + date.getHours() + ':' + date.getMinutes() + ':' + date.getSeconds();
let ciphertext = MD5(uid.toString() + id.toString() + dateString).toString();
```

Si la personne quitte la page et revient, l'id de paiement aura changé et on devra refaire un paiement. J'ai aussi profité de cette tâche pour chiffrer les données dans l'url.

```
payForAnnonce(id:any, idFact:any) {
  if(!localStorage.getItem('authUser')) {
    window.location.href = 'http://localhost:4200/login'
  } else {
    let uid = JSON.parse(localStorage.getItem('authUser')).id;
    let date = new Date();
    let dateString = date.getFullYear() + '-' + (date.getMonth() + 1) + '-' + date.getDate() + ' ' + date.getHours() + ':' + date.getMinutes() + ':' + date.getSeconds();
    let ciphertext = MD5(uid.toString() + id.toString() + dateString).toString();

    this.http.get(this.baseUrl + 'new-session-paiement.php?session=' + ciphertext + '&user=' + uid).subscribe(data => {
      uid = MD5(uid.toString()).toString();
      idFact = SHA256(idFact.toString()).toString();
      window.location.href = 'http://localhost/api/paiement/public/checkout.html?session=' + ciphertext + '&id=' + id + '&idFact=' + idFact + '&uid=' + uid;
    });
  }
}
```

Avant, l'url ressemblait à `localhost/api/paiement/public/checkout.html?id=1&idFact=1&uid=1`, maintenant il ressemble à ça :

Accept a payment | host/api/paiement/public/checkout.html?session=62596121821927c2ecadd2e98aca4c51&id=1&idFact=6b86b273ff34fce19d6b804eff5a3f5747ada4ea22f1d49c01e52ddb7875b4b&uid=c4ca2

En étant sur les factures, j'ai décidé de corriger qui faisait changer l'adresse de facturation. Avant, lors de la génération de la facture, je faisais une requête SQL pour récupérer celle de la base de données une fois le produit payé. J'ai décidé de le faire avant et de rajouter des métadonnées dans mon paiement, ce qui me permettrait de le récupérer plus tard via une requête API Stripe.

```
$uid = $data['idUserSession'];
$sql = "SELECT idAdresse, libelleAdresse, cpAdresse, villeAdresse, paysAdresse FROM facturations WHERE idUserAdresse = '$uid'";
$stmt = $conn->prepare($sql);
$stmt->execute();
$data = $stmt->fetch(PDO::FETCH_ASSOC);

// Create a PaymentIntent with amount and currency
$paymentIntent = \Stripe\PaymentIntent::create([
    'amount' => calculateOrderAmount(),
    'currency' => 'eur',
    'metadata' => [
        'userid' => $uid,
        'prdid' => $_GET['id'],
        'idFact' => $data['idAdresse'],
        'adresse' => $data['libelleAdresse'],
        'cp' => $data['cpAdresse'],
        'ville' => $data['villeAdresse'],
        'pays' => $data['paysAdresse'],
    ],
    'automatic_payment_methods' => [
        'enabled' => true,
    ],
]);

```

```
$this->Cell(40,7,'27000 Evreux',0,0,'L');
$this->Cell(142,7,utf8_decode($paymentIntent['metadata'][['adresse']]),0,0,'R');
$this->Ln(5);
$this->Cell(40,7,'France',0);
$this->Cell(142,7,utf8_decode($paymentIntent['metadata'][['cp']] . ' ' . $paymentIntent['metadata'][['ville']]),0,0,'R');
$this->Ln(5);
$this->Cell(182,7,utf8_decode($paymentIntent['metadata'][['pays']]),0,0,'R');
$this->Ln(30);
$this->Cell(40,7,utf8_decode("Vous trouverez ci dessous le récapitulatif de votre commande:"),0,0,'L');
$this->Ln(10);
```

Pour la partie facture, tout était fait pour ma part.

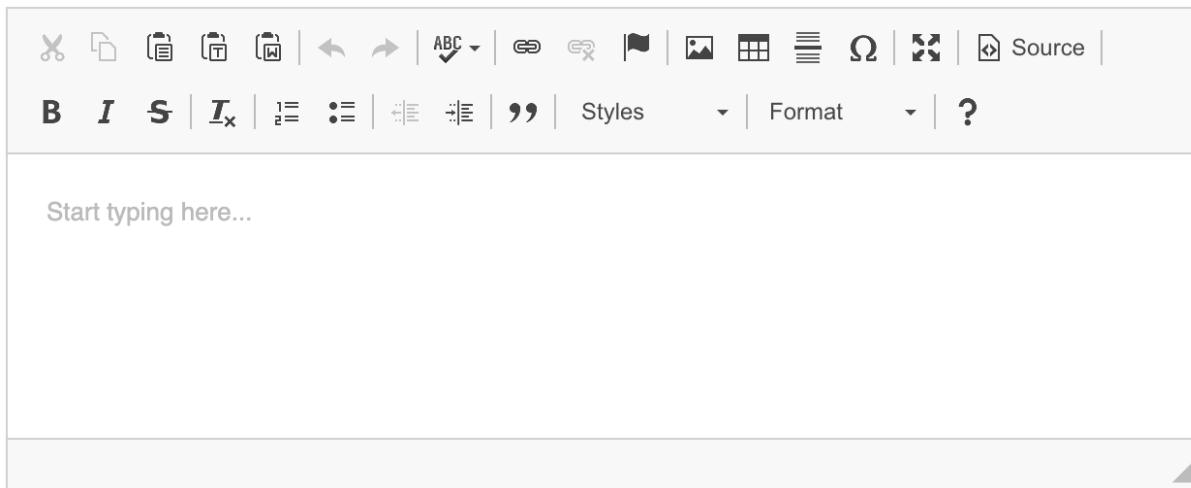
Une autre erreur que j'ai fait était que je n'avais pas fait la partie administration des avis. Seulement l'administrateur du site pouvait gérer les avis sur le site. Les avis devaient d'abord être accepté avant d'être affichés. J'ai donc fait une page assez simple qui me permettrait d'avoir les avis validés, non validés et l'ensemble des avis. Une fois un avis validé, il ne pourra être que supprimé par l'administrateur.

```
ngOnInit(): void {
  let url = this.url.snapshot.params['valid'];
  if(url == 0) {
    this.produitsService.getAvisValid(0).subscribe((data:any) => {
      this.avis = data.data;
      console.log(this.avis)
    })
  } else if(url == 1) {
    this.produitsService.getAvisValid(1).subscribe((data:any) => {
      this.avis = data.data;
      console.log(this.avis)
    })
  } else {
    this.produitsService.getAvis().subscribe((data:any) => {
      this.avis = data.data;
      console.log(this.avis)
    })
  }
}
```



The image contains two screenshots of a web application interface. Both screenshots feature a header with the text 'Super Admin'. Below the header, there is some descriptive text and a rating indicator (e.g., '4/5'). A main section displays a list of reviews. In the first screenshot, a specific review is highlighted, showing a green button labeled 'Valider l'avis'. In the second screenshot, the same review is shown again, but the green button has been replaced by a red button labeled 'Supprimer l'avis', indicating that the validation status has been changed.

L'erreur suivante à corriger était celle des descriptions ou des textes trop long. On avait mis au début des textarea ce qui permettait de faire des textes longs, mais les textes n'étaient pas formatés. On a trouvé une alternative, une extension nommée « CKEditor ».



Maintenant on n'a pas un texte qui est retourné lorsque l'on valide le formulaire mais du HTML. Derrière, pour afficher ces informations formatées, on peut utiliser « innerHTML » ce qui va convertir notre string en code HTML :

```
<p>Description:</p>
<p class="desc-prod" [innerHTML]="annonce[4][1]"></p>
<div class="d-flex flex-row">
```

**Sauvage**  
Vendu par: SwerkiNC

**2323.00 €**

(0/5) 0 avis

Description:

Produit test  
CECI EST UN TEST DE CKEDITOR  
*ITALIQUE*

En base de données :

qtStock	descriptionProduit	p
433	<p>Produit test <strong>CECI EST UN TEST DE CKE...</strong></p>	

La dernière chose à corriger pour ma part était de faire en sorte de pouvoir changer les images des produits déjà présents sur le site.

J'ai décidé de faire en sorte que l'image se changera directement au l'upload ou suppression. J'ai décidé aussi de faire en sorte que si on modifie une image, l'administrateur devra revérifier le produit.

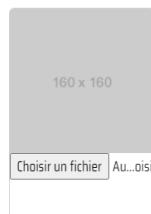
J'ai donc fait 3 boxes qui contiennent 3 images ou des placeholders avec une action soit d'ajouter, soit de supprimer :



Supprimer l'image



Choisir un fichier Au...oisi



Choisir un fichier Au...oisi

Note: Les changements des images ont apportés directement. Si vous décidez de modifier une image, votre produit ne sera plus mis en vente et devra être validé par un administrateur.

C'était très long à faire, car les extensions des images peuvent être des png ou jpg, je devais faire en sorte de réattribuer la place de l'image supprimée aux autres images.

Exemples :

Si nous avons 3 images (31-0.jpg, 31-1.png, 31-2.jpg), et que nous supprimons la première, les images deviennent : (31-0 .png, 31-1.jpg).

C'est à cause des extensions des images que cette tâche m'a pris pas mal de temps car je devais changer le nom des fichiers et dans la base de données, ce qui m'a pris toute une après-midi. Je pense que c'était la tâche la plus compliquée. Ceci est un bout du code, qui est assez long, pour remplacer l'image 31-1 par 31-0 :

```
if(isset($images[1])) {
    $image1 = explode('-', $images[1]['libelleImage']);
    $nom = $image1[0];
    $fileToRemove = $images[1]['libelleImage'];
    $ext = explode('.', $image1[1]);
    unlink("photos/produits/".$images[0]['libelleImage']);
    rename("photos/produits/".$images[1]['libelleImage'], "./photos/produits/".$nom."-0.".$ext[1]);
    $renamedFiles = $nom."-0.".$ext[1];
    $sql = "UPDATE image SET libelleImage = '$renamedFiles' WHERE libelleImage ='$fileToRemove'";
    $stmt = $conn->prepare($sql);
    $stmt->execute();
    echo json_encode([
        'success' => 200,
        'message' => "Image supprimée avec succès.",
    ]);
    exit;
}
```



Supprimer l'image



Supprimer l'image



Choisir un fichier Au...oisi

Note: Les changements des images ont apportés directement. Si vous décidez de modifier une image, votre produit ne sera plus mis en vente et devra être validé par un administrateur.

## I/Le troisième merge

Le troisième merge est le dernier merge globale de ce projet.

Lorsqu'on a merge le projet, chacun avait à peu près fini leurs tâches sauf le binôme qui travaillait sur le module de paiement. Ce merge s'est fait le dernier jour, en début d'après-midi. Il me restait une tâche à faire, et pendant ce temps, les autres vont reformater le code et une fois que j'ai fini, je vais les aider. On a réglé les 2-3 soucis du merge et on a pu relancer le travail.

J/Les dernières tâches

Pour ma dernière tâche, je devais gérer le statut des commandes. Une fois une commande passée, l'administrateur du site peut changer le statut de la commande. Il peut aussi laisser un message qui sera envoyé par e-mail au client. Une tâche pas très compliquée et très rapide à faire une fois 6 semaines passées à développer avec Angular.

## Gestion des commandes

Facture numéro: 6350AE2A28

Nom: Client Prénom: Marketplace

Status: En livraison

**Changer le status:**

Contenu de votre message:

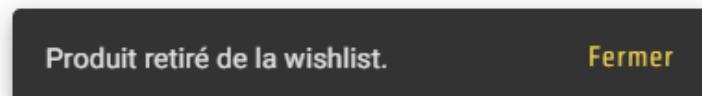
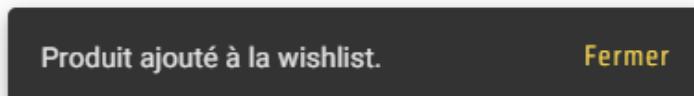
Paragraphe ▼ B U I Ø À ▼ A! ▼ ✖ ▼ := ▼ = ▼ ≡ ▼ |> ▼ “ ” ▼ ■ ▼ □ ▼ ↔ ▼ ↪ ▼

Statut:

1 - Préparation

Note: Le changement s'applique directement après avoir sélectionné un statut.

La dernière tâche à faire était sur les erreurs qui s'affichent. Toutes les erreurs étaient d'abord faites avec alertifyjs, on a décidé de mettre à la place les snackbars d'NGMaterials ce qui donne ceci :



Toutes mes tâches étaient maintenant finies. J'ai décidé d'aider le groupe dans leurs tâches, par exemple pour le module de paiement, expliquer un peu plus la logique ou encore sur la page d'accueil où on avait beaucoup d'erreurs dans la console. J'ai aussi enlevé tous les console.log qui étaient présents dans le code et j'ai rajouté la vérification sur chaque page pour voir si la personne connectée avait les droits d'accès à la page.

## 5/Bilan

Pour ce qui est du stage, j'ai bien aimé apprendre une nouvelle technologie web et apprendre une nouvelle méthode de travail aussi. J'ai trouvé que ce stage était bien mieux que mon précédent que j'ai pu effectuer chez Altameos car j'ai acquis plus d'expérience entre temps et j'ai eu moins de mal à avancer. J'ai bien aimé travailler en équipe, ça permet d'avoir la vision de plusieurs personnes sur une même question et avoir un peu d'aide. J'ai bien aimé travailler avec Angular aussi, ça rend le travail plus simple grâce à des commandes et à une bonne structure de projet. Je suis sûr qu'il est possible de faire plein de chose encore à découvrir. Ce stage m'a donné un peu plus envie à apprendre les autres Framework existant style React ou VueJS.

Pour ce qui est du projet, je trouve assez dommage qu'il n'a pas pu être finalisé à 100%. Il restait 2-3 problèmes dans l'application qu'on n'a pas pu régler par manque de temps, mais le projet en lui-même était intéressant à faire. Au début du stage, je pensais qu'on devait faire nous même le module de paiement, puis on nous a proposé Stripe, j'étais étonné qu'un module comme ça existe directement et disponible pour les entreprises.

A titre personnel, je pense avoir progressé, j'ai compris comment marchent des components ce qui peut être utile pour apprendre React. J'ai aussi pu un peu développer mes compétences et ma réflexion sur les problèmes et j'ai pu tester un peu le métier de lead dev pour tout ce qui est de la gestion du projet et l'aide au sein du groupe.