Armand HURAULT, Ouriel MAMAN, Nabil EL BARKAOUI

Github du projet :
https://github.com/Armantos/Projet_eCommerce

Sommaire:

I)Rappel du projet :
II)Présentation technique :
III)Architecture du projet :
IV)Diagramme UML du projet :
V)Installation du projet (nécessite PHP, MySQL, Apache, Symfony et Composer d'installés) :
VI)Code du projet :
VII)Fonctionnalités du site :
VIII)Répartition du travail :
IX)Problèmes rencontrés :
X)Fonctionnalités non implémentées par manque de temps :

I)Rappel du projet :

Le but du projet étant de développer un site internet pour vendre des produits. Le site doit disposer d'un frontend pour les utilisateurs finaux et d'un backend pour les administrateurs du site. Le frontend est accessible sans authentification, mais pour réaliser un achat l'utilisateur doit créer un compte et remplir un panier puis payer sa commande . L' administrateur doit avoir à disposition une liste des commandes en cours afin de les préparer etc...

Le projet doit implémenter les fonctionnalités suivantes :
1. Authentification (Administrateurs / Utilisateurs)
2. Gestion des utilisateurs (Création, modification et suppression)
3. Statistiques : Nombre d'articles vendus etc.
4. Gestion des stock (produit disponible à l'achat)
5. Un système de paiement par exemple paypal ou autre
6. Toutes autres fonctions que vous jugez utiles

II)Présentation technique :

Le site web est codé en PHP à l'aide du framework Symfony (documentation : https://symfony.com/doc/current/index.html)

La base de données tourne sous MySQL et le serveur sous Apache .

Le projet commence avec peu de dossiers. Chaque fonctionnalité a été ajoutée individuellement par la suite grâce au système de bundle de symfony (fonctions déjà codées pour ne pas avoir à réinventer la roue).

La majorité des bundles sont téléchargeables en ligne (partie login, inscription, admin etc…). Les dépendances des bundles sont installées automatiquement grâce à Symfony Flex (intégré à Symfony de base) et à Composer (logiciel d'installation de dépendances) dans le dossier /vendor.

Le front est réalisé à partir du template Bootstrap suivant : https://bootswatch.com/lux/

Le html est généré grâce à twig qui est un générateur de template html fonctionnant avec Symfony (documentation : https://twig.symfony.com/)

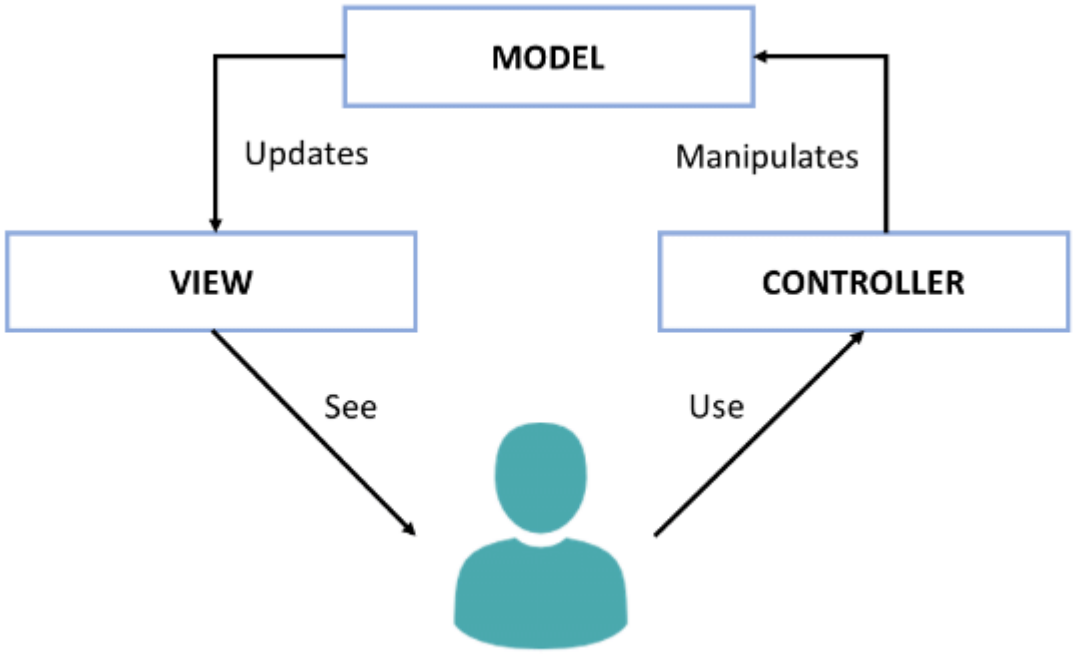
Le système de paiement (pas totalement intégré) fonctionne grâce à la plateforme Stripe via une API (documentation https://stripe.com/fr)

La répartition des tâches et le suivi de l'avancement du projet a été fait grâce à Trello.

Le versionning du code à été réalisé sur Github.

III)Architecture du projet :

Le projet est construit avec une architecture MVC (model-view-controller) afin d'encapsuler des parties du code pour limiter l'accès à certaines fonctions à l'utilisateur pour des raisons de sécurité.



Model : Traite la gestion des données de la base de données.
Les entités de la base de données sont manipulées grâce à l'ORM Doctrine.
C'est-à-dire une surcouche permettant l'abstraction des données.
Doctrine va récupérer les données de la BDD et les convertir en objet.
Les données manipulées par les contrôleurs sont des objets (et non pas juste des lignes de données récupérées par des SELECT). Les méthodes CRUD (CREATE, READ, UPDATE, DELETE) sont générées automatiquement par Symfony en utilisant la console (documentation Doctrine : https://symfony.com/doc/current/doctrine.html).

Par exemple, pour récupérer la liste de tous les articles, il suffit d'utiliser les lignes suivantes:
$repo = $this->getDoctrine()->getRepository(Article::class);
$articles = $repo->findAll();

findAll() correspond pour Doctrine à SELECT * FROM Article;
(extrait de la fonction index() de HomeController.php)

View: Le front-end (partie IHM) utilise Bootstrap qui est un framework pour faciliter le développement du front et le rendre responsive (documentation bootstrap : https://getbootstrap.com/docs/4.1/getting-started/introduction/).
Les parties en HTML utilisent Twig (un moteur de template permettant entre autres d'utiliser de l'héritage de code pour éviter les duplication ou de conditions d'affichage par exemple afficher le panier en fonction de si l'utilisateur est connecté ou non).

Controller :
Les contrôleurs reçoivent les actions de l'utilisateur, demandent au modèle de récupérer des infos dans la BDD (via l'ORM Doctrine) et les envoient à la vue pour afficher ces données .
Chaque route (exemple /login dans l'url) renvoie à un contrôleur.
Celui-ci renvoie un objet Response (qui peut demander une redirection de route, l'affichage d'un template Twig ou transmettre des données etc…)

L'utilisateur peut donc voir la vue et interagir avec le contrôleur, mais il ne peut pas faire de modification directement dans le modèle (par exemple, il ne peut pas faire directement des insertions dans la bdd sinon cela poserait un problème de sécurité).

Exemple de fonctionnement du code :
>En arrivant sur la page d'accueil, la fonction index() dans le contrôleur /src/HomeController.php est appelée.
>Le contrôleur va demander à Doctrine (modèle) de récupérer tous les articles de la BDD
>Le contrôleur va demander à Twig (vue) de faire un rendu de la page en lui envoyant les

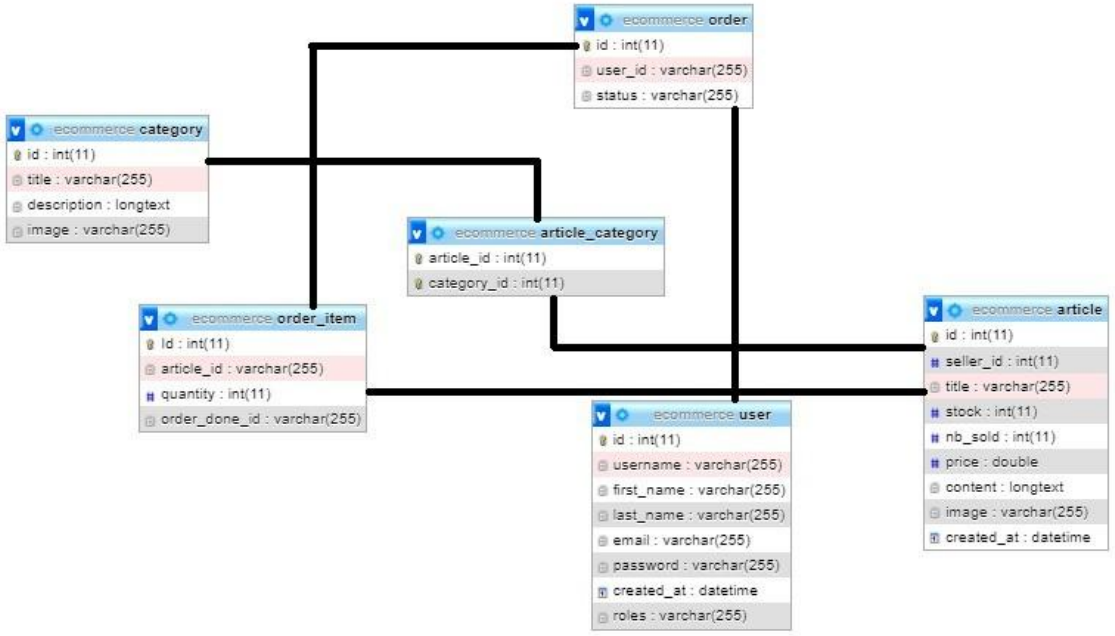
données reçues de la BDD (template situé dans le dossier /templates/home/index.html.twig)

```php
class HomeController extends AbstractController
{

    #[Route('/', name: 'home')]
    public function index(): Response
    {
        $repo = $this->getDoctrine()->getRepository( persistentObject: Article::class);

        $articles = $repo->findAll();

        //dd($articles); //debug pour afficher le tableau d'articles

        return $this->render( view: "/home/index.html.twig",
            ['articles' => $articles,
            ]);
    }
```

#[Route('/', name: 'home')]
>La route de la page d'accueil est "/" , name correspond au nom que nous avons donné à cette route pour l'utiliser dans d'autres fonctions .

Autre exemple : #[Route('/show/{id}', name: 'show')]
>La fonction show est utilisée dans le template de la page d'accueil.
>Elle est liée au bouton "En savoir plus" qui va lancer la fonction show($id) en utilisant l'id de l'article (passé en paramètre dans l'url) puis rediriger l'utilisateur vers la route de l'article (par exemple /show/1 pour l'article numéro 1)

IV)Diagramme UML du projet :



V)Installation du projet (nécessite PHP, MySQL, Apache, Symfony et Composer d'installés) :

Apache + MySQL + PHP : https://www.apachefriends.org/download.html
Symfony : https://symfony.com/download
Composer : https://getcomposer.org/download/
(Git windows pour récupérer automatiquement le code : https://git-scm.com/downloads)

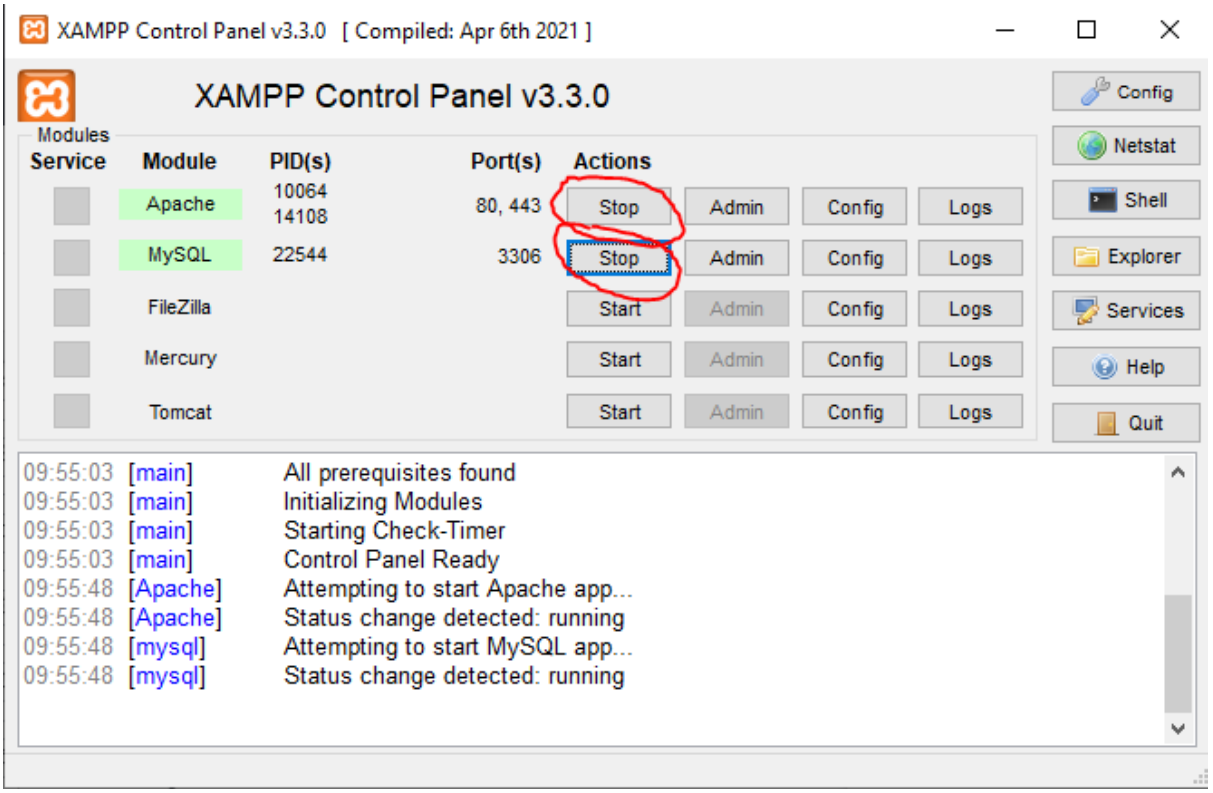
Récupération du code + installation automatique OU manuelle
Ouvrir le terminal dans un dossier pour installer le projet :
> **git clone https://github.com/Armantos/Projet_eCommerce**
> **cd Projet_eCommerce**
> **composer install**

OU

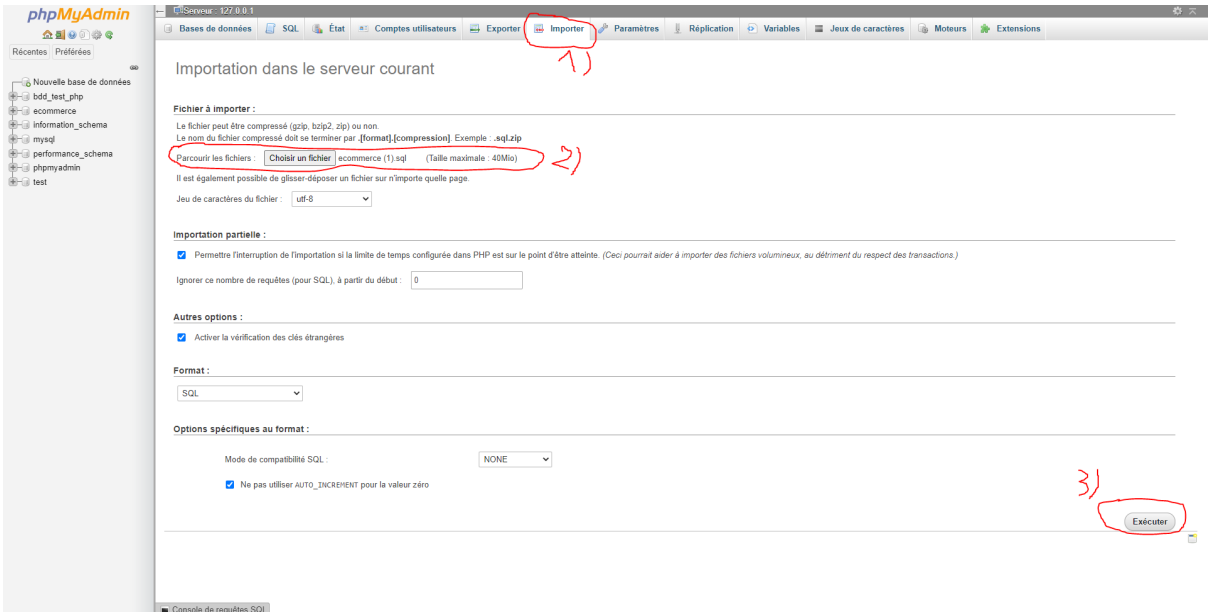
>Récupérer le code sur moodle
>Le dézipper
>Se placer à la racine du projet
>Ouvrir un terminal et taper :
> **composer install**

La commande “composer install” permet d’installer les bundles. Sans la commande, certaines fonctionnalités ne fonctionnent pas (login, partie admin …)

Importation de la base de données :
> Récupérer le fichier “ecommerce.sql” situé à la racine du dossier du projet
(> Lancer le serveur et la base de données sur XAMPP)



> Aller sur phpmyadmin : http://localhost/phpmyadmin
> Cliquer sur l'onglet "Importer"
> Cliquer sur le bouton "Choisir un fichier"
> Sélectionner le fichier "ecommerce.sql"
> Cliquer sur le bouton "Exécuter" pour finaliser l'importation.



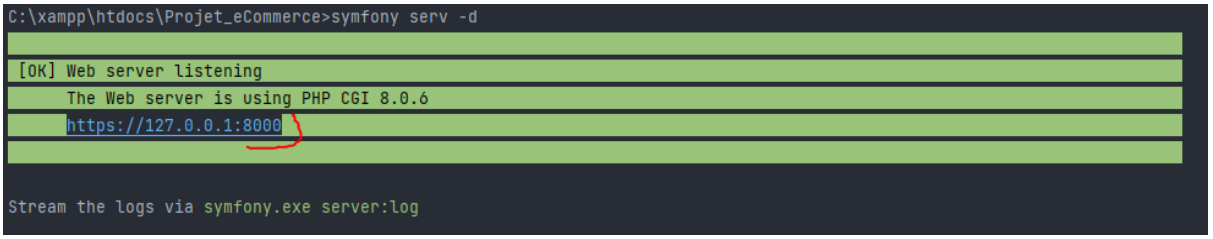
Lancer le serveur local de Symfony :

> Se placer à la racine du projet
> Ouvrir un terminal
> Taper la commande suivante :
> **symfony server:start -d**
(> Pour éteindre le serveur :
> **symfony server:stop**)

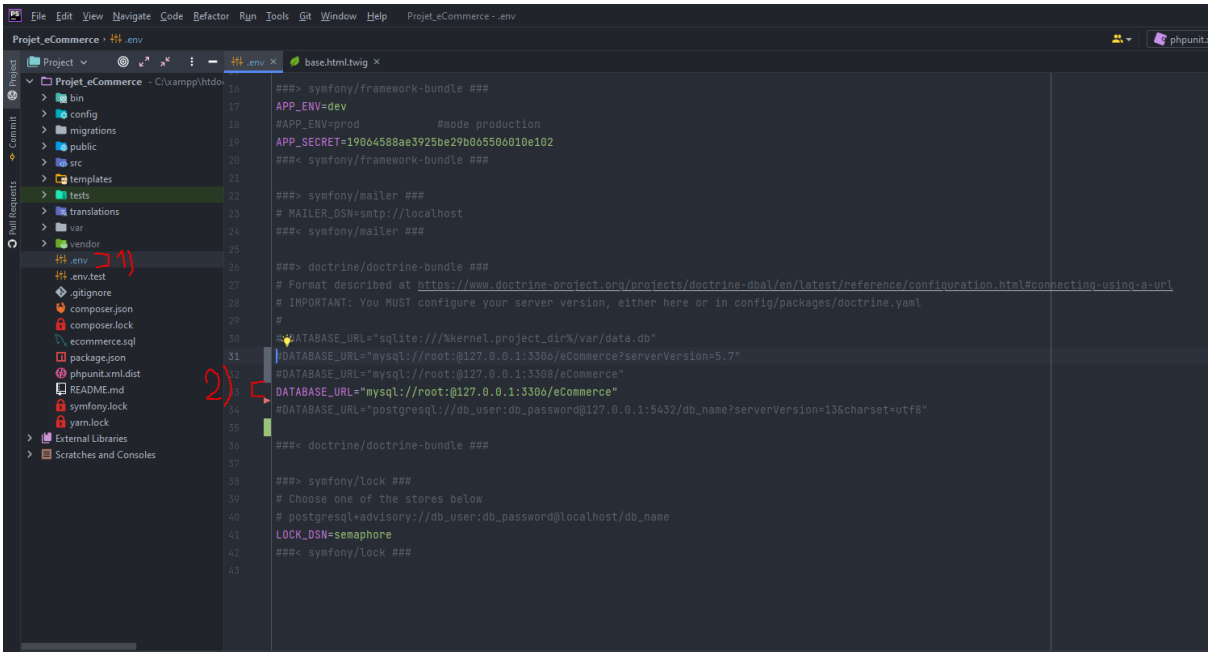
Le site est maintenant disponible à l'adresse https://localhost:8000/ ou https://127.0.0.1:8000/

Erreurs possible au chargement du site :

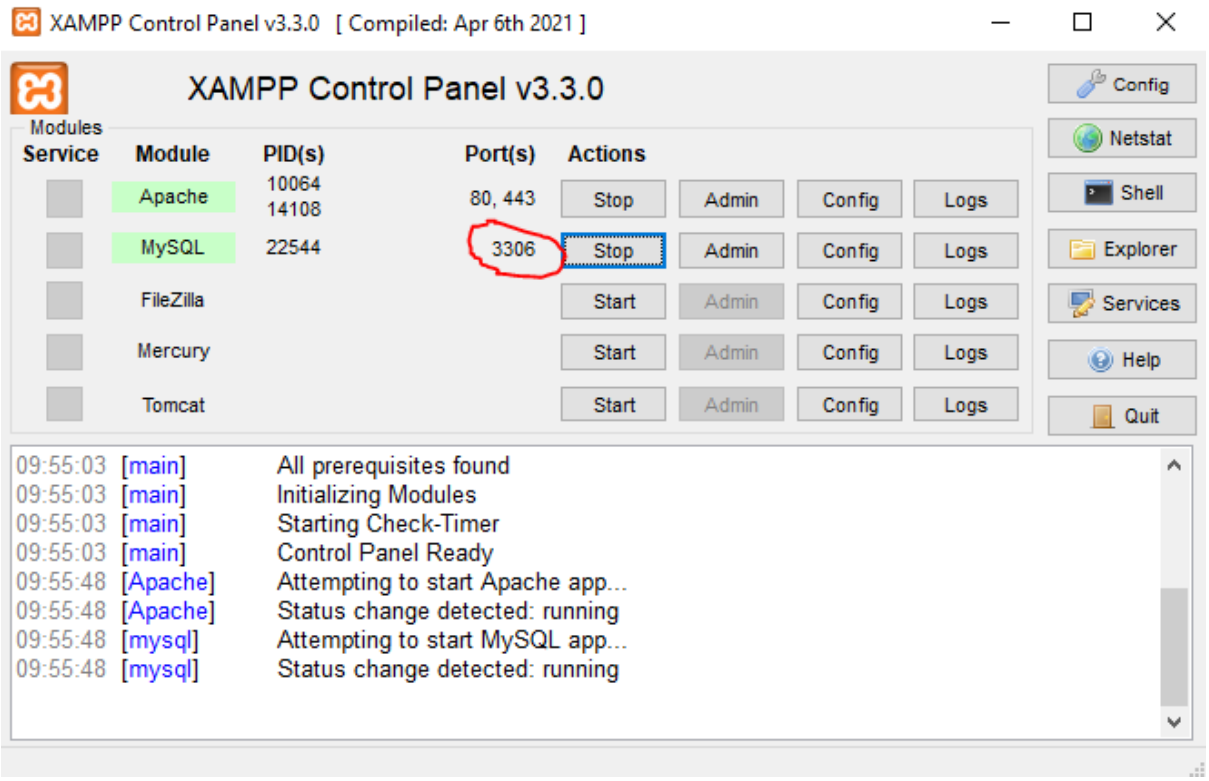
La page ne s'affiche pas et le navigateur indique comme erreur "ERR_CONNECTION_REFUSED"
>Le port 8000 est peut-être déjà utilisé .
>Modifier l'url par celui du port utilisé par Symfony (le port est indiqué au lancement du serveur local dans le terminal)



Symfony affiche en rouge le message : "An exception occurred in driver: SQLSTATE[HY000] [1045] Access denied for user 'root'@'localhost' (using password: NO)"
>La connexion à la bdd n'a pas pu être établie
>Aller dans le fichier .env à la racine du projet.



>Modifier dans le fichier .env la ligne
“DATABASE_URL="mysql://root:@127.0.0.1:3306/eCommerce" avec les bons paramètres de connexion :
-Base de données : mysql OU sqlite OU postgresql ...
-Identifiants de la BDD : root par défaut, aucun mdp par défaut.
En cas d’utilisation d’un id ou d’un mdp différents, les indiquer après les “:” .
Exemple : mysql://user:password@127.0.0.1:3306/eCommerce
-Adresse utilisée dans l’url : 127.0.0.1 par défaut
-Port utilisé par la BDD : 3306 par défaut (indiqué sur XAMPP)



-Nom de la bdd : donnée à l’importation du fichier ecommerce.sql

VI)Code du projet :

Les dossiers/fichiers importants ayant été codés/modifiés :

- /config/security.yaml : contient les données concernant la sécurité du site (login, droit d'accès avec les firewall, hashage des mdp, logout …)

- /public : contient les fichiers accessibles publiquement (index, html, css, js, images etc…)

- /src : dossier le plus important du back-end, contient les contrôleurs et les entités (user, article etc…)
- /src/Controller : contient les contrôleurs de chaque page.
- /src/Controller/admin: contient les contrôleurs de la partie admin.

- /templates : contient les templates Twig pour générer le code html pour le front-end.
>Chaque sous-dossier correspond à une page .
(exemple : /templates/login/login.html.twig contient la page de formulaire de connexion).

>Toutes les pages visibles par l'utilisateur (sauf la partie admin) héritent de la page /templates/base.html.twig qui contient entre autres la navbar et le footer.

- /.env : contient les informations de connexion à la bdd

VII)Fonctionnalités du site :

Navbar : celle-ci est présente sur toutes les pages (sauf la partie admin).
Si l'utilisateur n'est pas connecté , les liens affichés sont :
-Home | login | Créer un compte
Si l'utilisateur est connecté, les liens affichés sont :
-Home | Panier | Profil | Se déconnecter
Si l'utilisateur est un admin, les liens affichés sont :
-Home | Panier | Profil | Se déconnecter | Admin

Liste des pages
-Accueil : (/)
>en arrivant sur le site, l'utilisateur arrive sur la page d'accueil où il peut voir tous les produits en vente ainsi que leur caractéristiques (nom, prix, image, date d'ajout, vendeur, quantité disponible, nombre de ventes)
>Il peut afficher les détails du produit en cliquant sur le bouton "En savoir plus", ce qui va le rediriger vers la page de l'article.
>Le bouton "ajouter au panier" va ajouter l'article dans une variable de session.

-Article : (/show/{id})
>Affiche les détails du produit (nom, description, nombre de ventes, nombre en stock etc…)
>Les articles sont créés dans la partie Admin.
>Le bouton "ajouter au panier" va ajouter l'article dans une variable de session.

-Créer un compte :
>L'utilisateur peut remplir le formulaire pour s'inscrire.
>Le mdp doit être 2 fois le même.
>Le mdp doit faire minimum 6 caractères (ce nombre peut être modifié dans /Controller/Form/RegistrationFormType.php)
>Le mdp va être hashé avant d'être inséré dans la bdd (algorithme de hashage bcrypt)
>Après inscription, l'utilisateur est redirigé vers la page d'accueil (mais il n'est pas connecté automatiquement)

-Login : (/login)
>Après le login, l'utilisateur est redirigé vers la page d'accueil

-Se déconnecter : (/logout)
>Dirige vers la route logout, déconnecte l'utilisateur et vide la session.

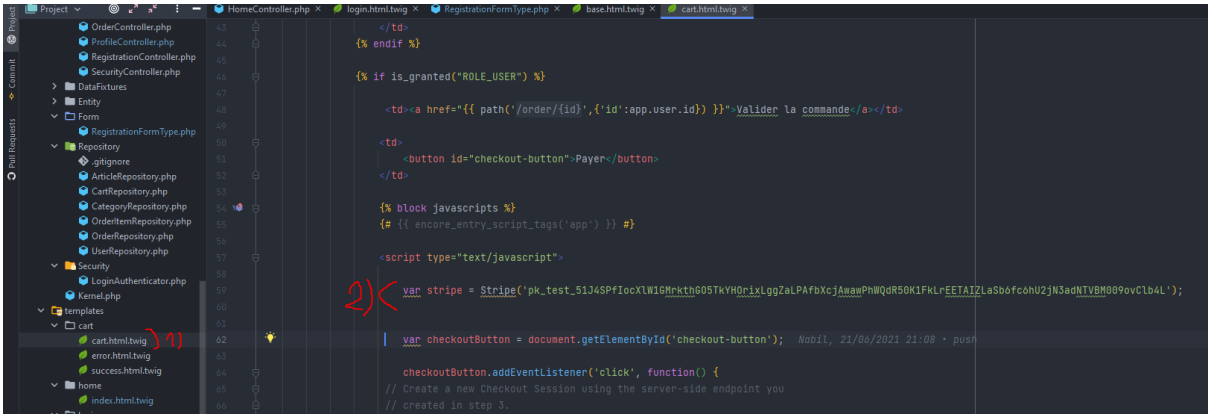
-Profil : (/profile)
>Affiche les infos de l'utilisateur (nom, prénom, pseudo, mail)
>Permet à l'utilisateur de modifier les infos de son compte (le formulaire n'a pas encore été implémenté, les infos sont entrées en brut à partir de ProfileController.php =>updateProfile())

>Permet à l'utilisateur à supprimer son compte
>Permet à l'utilisateur de voir ses commandes en cours

-Panier : (/cart)
>L'article ajouté est inscrit dans une variable de session
>Si il existe déjà, le compteur incrémente de 1.
>Le bouton rouge permet de retirer l'objet du panier.
>Le prix total est calculé automatiquement.
>Le bouton "valider la commande" dirige vers la page commande
>Le bouton "payer" mène vers la page de paiement Stripe mais celle-ci n'est pas entièrement fonctionnelle.
>Le fonctionnement normal aurait été de remplir le formulaire de livraison, accéder à la page de paiement Stripe puis être redirigé vers la page commande.

-Commandes : (/order)
>La page de commande permet de créer la commande et de l'insérer à la bdd.

-Paiement : (API stripe)
>Nécessite la création d'un compte Stripe .
>Dans la partie développeur, récupérer la clé privé et la clé publique.
>La clé publique est à entrer dans /templates/cart/cart.html.twig



>La clé privé est à entrer dans CheckoutController.php

>La liste des paiement est dans affichée dans Stripe

-Admin: (/admin)
Pour accéder à la partie admin
>Se connecter au compte Admin :
les identifiants de connexion sont :

id : admin@admin.fr
mdp : admin12
>1 seul admin a été créé (Pour créer un autre admin, sur phpmyadmin, sur la table user, sous l'attribut role, ajouter ["ROLE_ADMIN"])
>La partie admin est bloquée par un firewall, si une personne n'ayant pas le rôle admin essaye d'accéder à la zone, une erreur est lancée .
>Toutes les commandes CRUD sont accessibles pour modifier le contenu de la BDD.

Base de données :
Quasiment toutes les entités créés dans la BDD (user, article etc…) sont des fausses données insérées grâce à un bundle de Symfony appelé Fixtures (documentation https://symfony.com/doc/current/bundles/DoctrineFixturesBundle/index.html) et du package appelé faker (https://packagist.org/packages/fzaninotto/faker).
Les plus anciennes données n'ont pas été renouvelées par manque de temps et ont certains attributs vides ou incorrects (certains utilisateurs n'ont pas le mdp hashé ou ont un rôle null par exemple) mais cela ne pose pas de problèmes pour les données nouvellement créées.

VIII)Répartition du travail :

Armand :
-Partie admin avec le bundle Symfony easyAdmin
-Partie login/inscription avec le bundle Symfony security-bundle
-Partie profil utilisateur

Ouriel :
-Maquette du site sur Adobe XD
-Front général (bootstrap, formulaires, template twig)

Nabil :
-Partie affichage des articles
-Partie panier / commandes
-Partie paiement

La mise en place de la base de données, la mise en place des entités , la répartition des tâches sur Trello ont été faites ensemble.

IX)Problèmes rencontrés :
>Aucun de nous n'avait utilisé Symfony ou Bootstrap auparavant, la majorité du temps a été passée à lire de la documentation plutôt qu'à écrire du code au début, ainsi que l'installation des différents outils.
>Certains membres du groupe n'avaient jamais fait de dev web ou utilisé github auparavant.
>Symfony étant relativement récent et évoluant rapidement, un grand nombre de fonctionnalités trouvées en ligne (surtout sur les sites comme stackoverflow) sont dépassées. La documentation n'étant pas encore assez référencée sur les moteurs de recherche, il était nécessaire de lire en détail la documentation pour trouver les informations importantes.

X)Fonctionnalités non implémentées par manque de temps :

>Sécuriser intégralement le site
    >Eviter les bruteforces en limitant le nombre de tentatives de login pendant une période de temps (ligne à décommententer dans security.yaml)
    >Vérification des saisies utilisateur (utilisation de regex dans les formulaire)
    >Vérification URL de toutes les entrées GET.
    >Pages 403 en cas de paramètres incorrects ou de droits incorrects.
>Formulaire pour la modification des données de l'utilisateur dans la partie "Paramètres"
>Traduire le site entierement en français (notamment le formulaire d'inscription et la partie admin)
>Barre de recherche d'un produit avec proposition et auto-complétée
>Système de pagination (afficher 10 articles sur la page 1, 10 sur la page 2 etc …)
>Charger plus rapidement le site en ajoutant de l'asynchrone (Notamment pour la page d'accueil pour ne pas avoir à attendre que tous les articles chargent)
>Héberger le site en ligne
>Récupération de mots de passe
>Envoie de mail
>Formulaire de livraison
>Afficher les statistiques de produits (le plus vendu, le plus cher etc…) implémentable dans Symfony en utilisant un query builder (https://symfony.com/doc/current/doctrine.html#querying-with-the-query-builder)