Une image contenant texte, téléphone

Description générée automatiquement **Floriane IZARD**

**RAPPORT DE STAGE DE DEUXIEME ANNEE BTS SIO (SLAM)**



Tuteur pendant le stage : *Nicola Bonaccorso*

Tuteur Académique : *Pierron Vialard*

Etablissement / Formation : Lycée Général Turgot

BTS SIO (Spécialité SLAM)

**Du 29 novembre 2021 au 21 janvier 2022**

**SOMMAIRE**

1. Remerciements………………………………………………………………………………………3
2. Introduction……………………………………………………………………………………………4
3. Organigramme……………………………………………………………………………………….5
4. Présentation de l’entreprise…………………………………………………………………..6
5. Déroulement du stage……………………………………………………………………………7
6. Conclusion……………………………………………………………………………………………25

**REMERCIEMENTS**

Je tiens tout d’abord à remercier le lycée Turgot, et mes professeurs pour m’avoir fait confiance.

À ce titre, je souhaiterais remercier tout particulièrement mon tuteur de stage Nicola Bonaccorso.

Ce stage m’a permis d’apprendre de nouvelles choses pour réussir mon BTS et signe l’aboutissement de ma deuxième année au lycée Turgot en BTS SIO option SLAM.

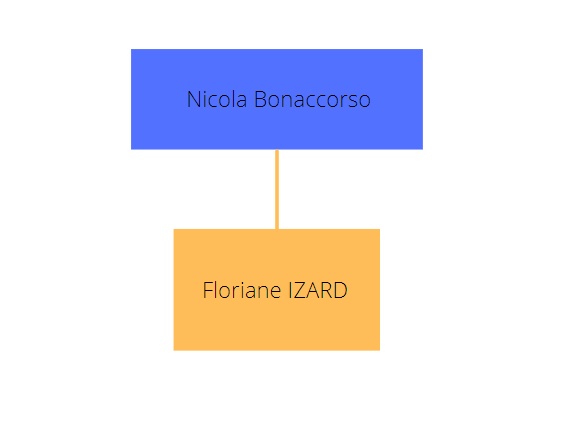
Je n’oublie pas non plus mes proches qui m’ont sans cesse soutenu dans l’élaboration de mon projet professionnel et m’ont aidée à chaque étape de ce rapport de stage.

**INTRODUCTION**

Dans le cadre de ma formation en BTS SIO option SLAM, j’ai effectué mon stage avec Nicola Bonaccorso, un auto-entrepreneur, directeur général et chef de projet Digital, du 29 novembre 2021 au 21 janvier 2022.

Mon rôle durant ce stage a été de créer un site dynamique avec un Framework. Dans une première partie, je vais parler de mon environnement de travail et les activités faite puis par la suite j’expliquerais mes missions, mon expérience et ce que cela m’a apporté.

Ce stage m’a permis d’assimiler quelques nouvelles connaissances, et d’en approfondir d’autres. Il m’a aussi appris à travailler à la maison et d’organiser mon temps.

**ORGANIGRAMME**

**PRESENTATION DE L’ENTREPRISE**

Auto-entrepreneur c’est quoi ?

* C’est un travailleur indépendant simplifiant la gestion administrative grâce à un versement de cotisations unique et proportionnel au chiffre d’affaires.
* Il s'applique aux personnes physiques qui créent ou possèdent déjà une entreprise individuelle pour exercer une activité commerciale, artisanale ou libérale.

Mon tuteur Nicola Bonaccorso est chef de projet dans le digital, ce qui représente une partie de son activité est :

* De fournir des prestations de Dev UX et DA comme une agence web.
* Il fait aussi de la stratégie commerciale.
* Et du Web marketing.

**DEROULEMENT DU STAGE**

1. Le déroulement :

Le stage s’est déroulé entièrement chez moi, j’ai donc travaillé sur mon ordinateur personnel. J’avais pour projet de créer un site web dynamique avec l’aide d’un Framework de mon choix. J’ai donc commencé mon stage avec des recherches internet sur un Framework qui pourrait me correspondre et être intéressant pour mon apprentissage.

Puis, de trouver un thème captivant et ludique pour le développement de mon site.

Ensuite, il a fallu élaborer un cahier des charges pour la création du site et cela m’a permis de suivre un plan tout le long de mon stage.

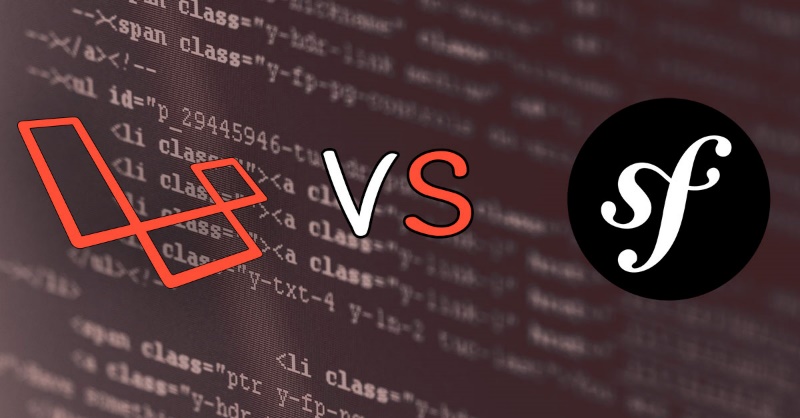
1. Les travaux effectués :

Tout d’abord j’ai hésité entre deux Framework Laravel et Symfony.

* Symfony est un ensemble de composants PHP ainsi qu'un Framework MVC libre écrit en PHP. Il fournit des fonctionnalités modulables et adaptables qui permettent de faciliter et d’accélérer le développement d'un site web.
* Laravel est un Framework web open-source écrit en PHP respectant le principe modèle-vue-contrôleur et entièrement développé en programmation orientée objet. Laravel est distribué sous licence MIT, avec ses sources hébergées sur GitHub.

Les 2 Framework utilisent tous les 2 un ORM (Object Relational Mapping) pour gérer les requêtes de manière native, il s’agit d’une technique de programmation informatique qui permet de simplifier l’accès à une base de données en proposant à l’informaticien des « objets » plutôt que d’accéder directement à des données relationnelles.

Mais ils utilisent chacun un ORM différent. Doctrine pour Symfony et Eloquent pour Laravel.



J’ai donc choisi de prendre Symfony et de plus Symfony dispose d’une documentation complète qui m’a beaucoup aidé.

Le thème :

Un site de vente e-commerce avec des interactions utilisateurs.

Cahier des charges :

-Création des articles

-Afficher des articles

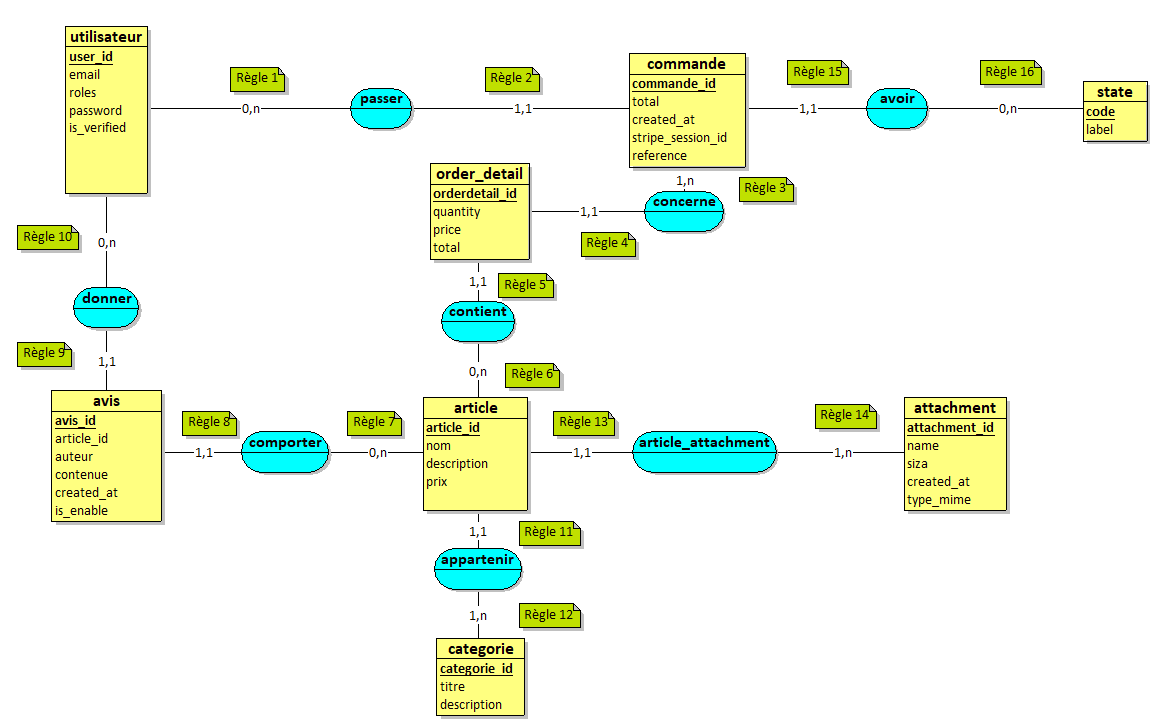
-Modification des articles

-Inscription avec vérification de courriel

-Connexion -> accès au site avec sa création et modification d’articles

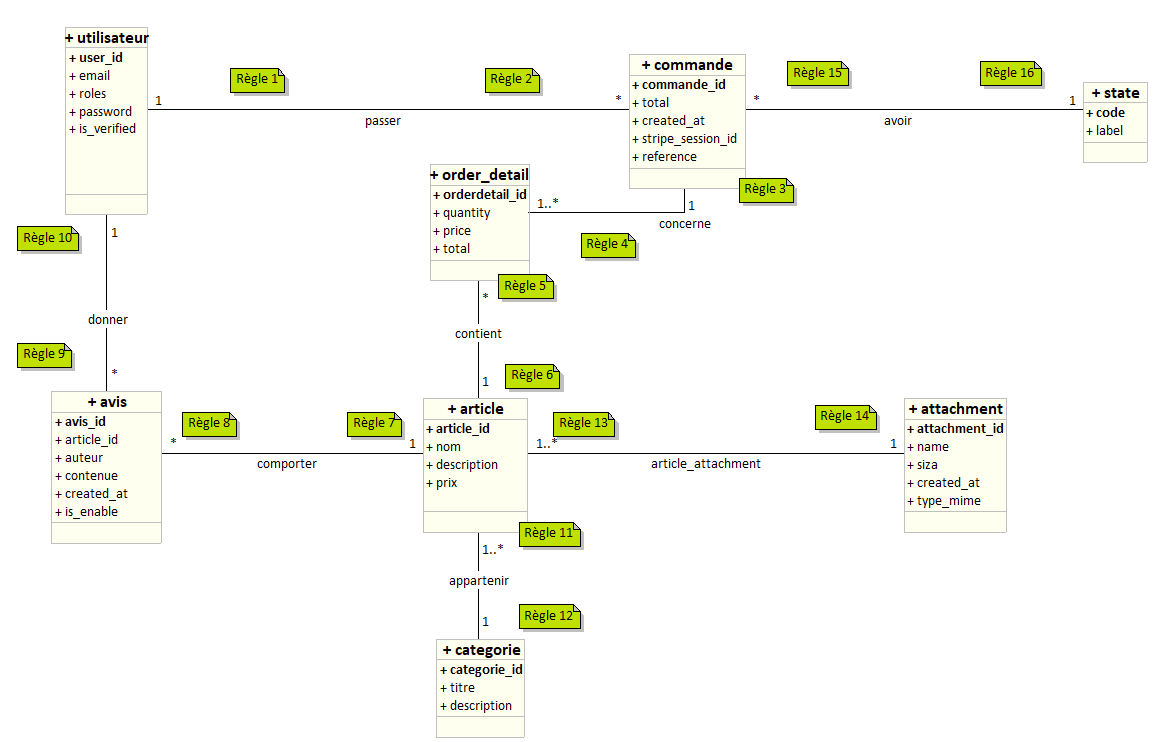
-Déconnexion

-Créer un/plusieurs avis

-Modération des avis

MCD

Pour les règles voir annexe



UML

**1ére étape :**

*Installer et configurer le Framework Symfony :*

Il faut tout d’abord installer PHP et les packages de PHP (Install Composer).

Puis installer le Symfony CLI, c'est un binaire qui fournit tous les outils dont vous avez besoin pour développer et exécuter votre application Symfony localement.

Dans l’invite de commande, j’ai exécuté la commande :

$ symfony new my\_project\_name --full

Elle permet de créer une nouvelle application Symfony avec tous les composants nécessaires pour créer un site internet.

**2éme étape :**

**Installation de Doctrine :**

C'est un composant logiciel qui permet de transformer les données contenues dans une table MySQL en objets utilisables dans votre application, et vice-versa. Cette procédure est qualifiée de mapping.

$ composer require symfony/orm-pack

composer require --dev symfony/maker-bundle

Il a fallu également personnaliser mon fichier .env (situé à la racine du projet) qui gère la configuration des variables d’environnement. Il faut définir la DATABASE\_URL.

DATABASE\_URL="mysql://db\_user:db\_password@127.0.0.1:3306/db\_name?serverVersion=5.7"

Le « .env » est le fichier de référence, qui peut être commiter sur le repo. Il ne doit pas contenir les vraies informations de connexion. C’est simplement une sorte de référentiel pour savoir quelles variables d’environnement sont utilisées par notre application.

C’est dans le « .env.local » qu’il faudra renseigner les vraies informations et notamment celles de connexions à la base de données, pour éviter tous vols de données (compte utilisateurs par exemple, même si les mots de passe sont sécurisés comme on le verra par la suite) dans ma base de données.

**3éme étape :**

*Faire démarrez le serveur web local :*

Dans l’invite de commande :

$ symfony server:start

Dans mon navigateur je dois accédez à <http://localhost:8000/>

Mais j’ai fait le choix d’utiliser un environnement Docker pour être isométrique (colle à) au serveur de production.

Donc l’installation se fait avec la commande :

$ docker compose up --build

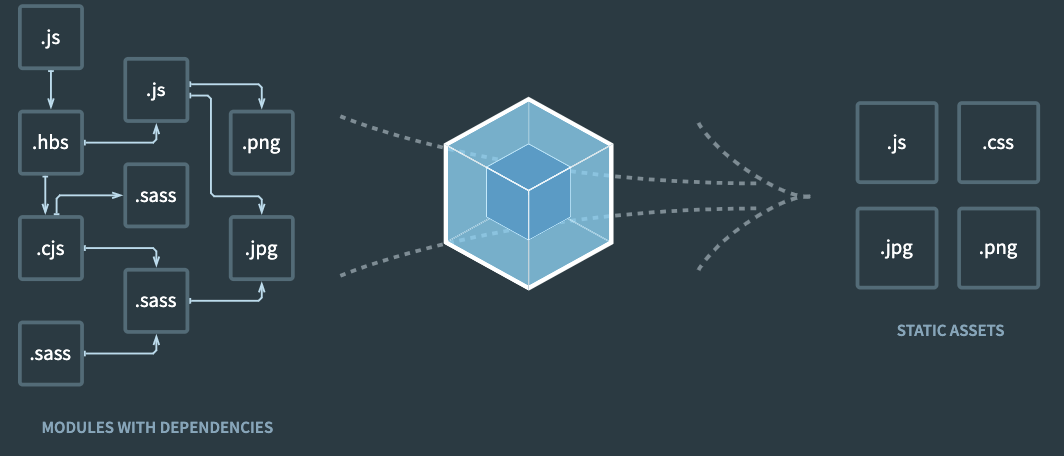
**4éme étape:**

*Installation de packages :*

Twig : Twig est un moteur de templates pour le langage de programmation PHP.

Webpack-encore-bundle : Permet simplement d’intégrer Webpack, il gère les assets (css, js, images, fonts, …). Il offre une API puissant pour compiler et réduire.

Webpack est un groupeur de modules (module bundler) écrits en javascript et open source.



Verify-email-bundle : sert à vérifier les emails à la création d’un compte par un utilisateur.

LiipImagineBundle : Il permet de faire la gestion d’affichage des images via des filtres prédéfinie.

**5éme étape:**

**Création d'une classe d'entité :**

$ php bin/console make:entity

**Cette commande créer automatiquement une classe avec les méthodes getter et setter.**

**Création d’un Controller :**

$ php bin/console make:controller

Il va aussi créer une Template dans un dossier qui contient le nom qu’on a donné et un fichier twig pour la vue.

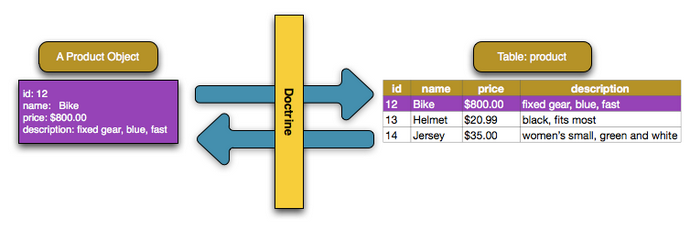
**Migrations : création des tables de la base de données**

$ php bin/console make:migration

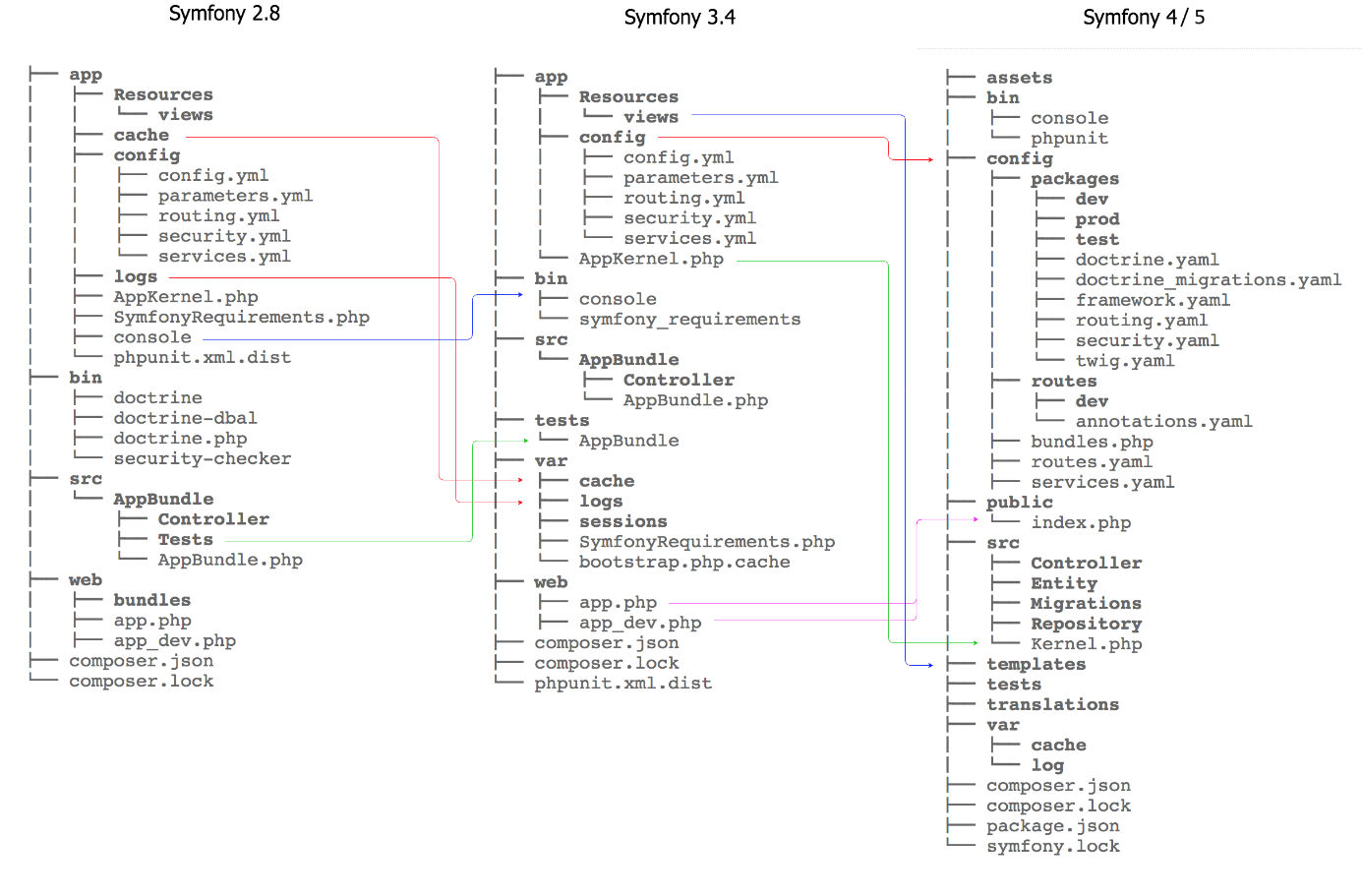
On obtient un fichier, qui contient le SQL nécessaire pour mettre à jour la base de données. Pour exécuter ce SQL, il faut exécuter les migrations :

$ php bin/console doctrine:migrations:migrate

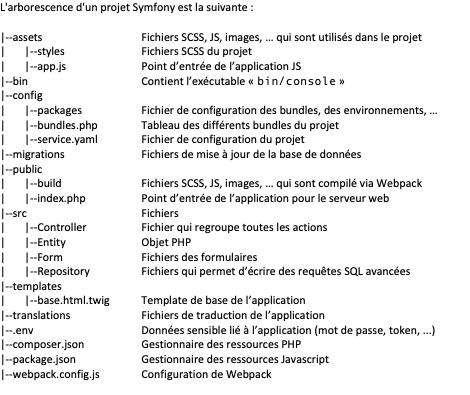
Cette commande exécute tous les fichiers de migration qui n'ont pas encore été exécutés sur la base de données. On exécute cette commande en production lors du déploiement pour maintenir à jour votre base de données de production.



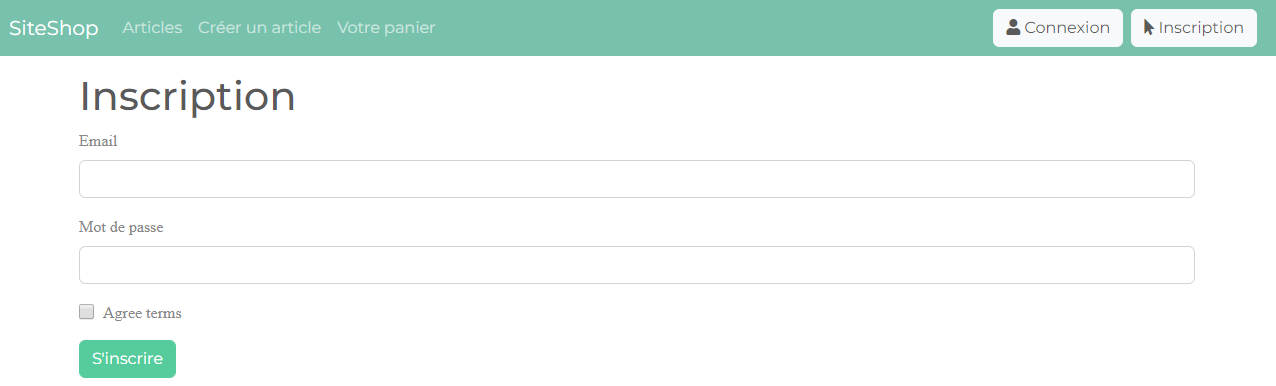
**Arborescence de Symfony :**



**Explications :**



**CONNEXION/INSCRIPTION :**



Il faut créer tout d’abord une entité User avec tous les champs nécessaires (email, mot de passe).

Puis, Symfony nous fournit une commande, création d’un formulaire d’inscription :

$ php bin/console make :form RegistrationFormType

Avec Symfony, le composant formulaire vous permet de créer, traiter et réutiliser des formulaires.

Il créer donc les champs qui sont dans l’entité User comme ceci :

Une image contenant texte

Description générée automatiquement

Il est possible d’ajouter des champs supplémentaires qui ne font pas partie de l’entité. Ce sont des champs non mappés, donc il faut ajouter de l’attribut ‘mapped’=>false. C’est une case pour accepter les termes (booléen).

Toujours passer en paramètre les entités qu’on va utiliser

Une image contenant texte

Description générée automatiquementPuis RegistrationController.php est le Controller qui va permettre de traiter les données du formulaire.

Il set et hash le mot de passe, si le formulaire est soumis et valider

Créer le formulaire et renvoie un objet Form

Créer un nouvel objet de type User

Puis il traite l’info et enregistre dans la base de données

On envoie ensuite notre formulaire à notre vue puis la fonction createView() qui permet de générer la vue du formulaire.

La vue register.html.twig : Fonction de Twig

Une image contenant texte

Description générée automatiquementPermets de commencer et d’afficher le formulaire avec le bouton pour envoyer mes informations.

Pour afficher chacun des champs indépendamment, on utilise des balises twig qui commencent par ‘form\_’.

La form\_row() est une fonction qui génère l'intégralité du contenu du champ, y compris l'étiquette, le message d'aide, les éléments HTML et les messages d'erreur.

Pour la sécuriser des mots de passe dans la base de données, il existe une commande :

$ composer require symfony/password-hasher

Puis de configurer dans le dossier config, le fichier security.yaml :

Une image contenant texte

Description générée automatiquement

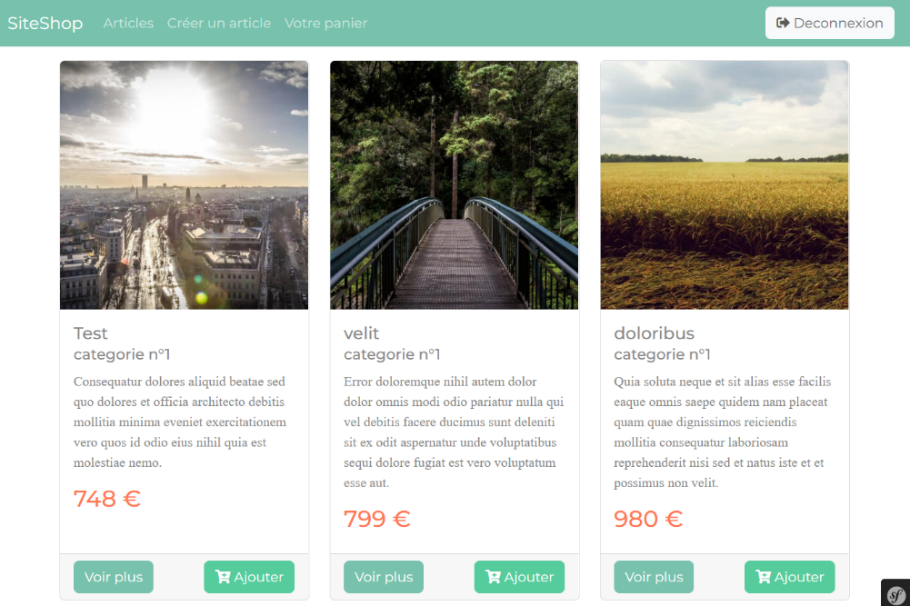
L’algorithme "auto" est utilisé car ce hacheur sélectionne automatiquement l'algorithme le plus sécurisé disponible sur le système.

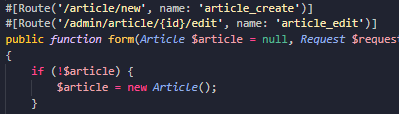


Il s’affichera comme ceci dans la base de données.

Une image contenant texte

Description générée automatiquementPour se connecter :

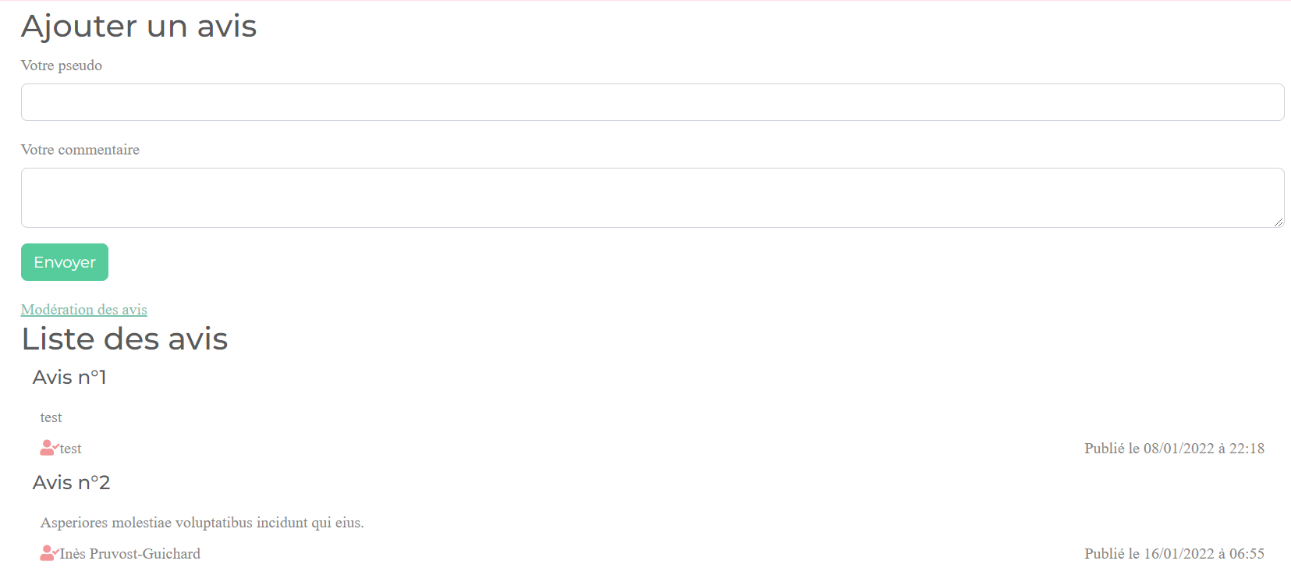
**ARTICLES :**

Créer un article :

Nous avons 2 routes différentes pour la même action. Une route sera utilisée pour l’ajout et l’autre pour la modification (qui sera accessible que pour l’admin).

Une image contenant texte

Description générée automatiquement

 Les avis :

Une image contenant texte

Description générée automatiquementDans ArticleController.php (voir //) Dans la vue show.html.twig :

Balise qui permet d’afficher les autres champs du formulaire qui n’ont pas été déclarés.

Une image contenant texte

Description générée automatiquementLe formulaire d’avis

Une image contenant texte, capture d’écran, écran

Description générée automatiquementLa liste des avis

Les utilisateurs ont accès au formulaire mais le commentaire doit être activé pour être affiché :



Dans le controller :

Une image contenant texte

Description générée automatiquementL’avis est automatiquement isEnable = 0

Une image contenant texte

Description générée automatiquementDans l’entité

Donc on créer un if quand l’avis est à 0 on le met à 1 sinon l’inverse.

Une image contenant texte

Description générée automatiquement

Dans la vue :

**Les Fixtures :**

* Créer des jeux de fausses données
* Exécutable à souhait
* Réutilisable par les autres

$ php bin/console make:fixtures

Une image contenant texte

Description générée automatiquementOn créer un nouveau ArticleFixtures.php dans un nouveau dossier DataFixtures :

Cela créer 100 fixtures d’articles qui sont généré avec des faker.

Puis après il faut charger toutes les fixtures dans la base de données

$ php bin/console doctrine :fixtures :load

Comme ceci, en lorem ipsum:

Une image contenant table

Description générée automatiquement

**GESTION DES ROLES :**

Il existe deux rôles sur mon site, rôle USER et ADMIN.

Le rôle ADMIN a accès à la modération des avis et la modification d’un article.

Dans l’entité User.php : On crée une constante qui définit le rôle admin.



On déclare que chaque utilisateur a un rôle user au départ de son inscription :

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquementDans un nouveau dossier Command, ont créé un fichier UserAdminCommand.php, qui permet de créer une commande pour créer un utilisateur admin :

On déclare le nom de la commande qui sera appelé via la commande php bin/console app :user-admin.

Une image contenant texte

Description générée automatiquement

On déclare la fonction qui permet de configurer notre commande, dans ce cas ce sera l’email et le mot de passe.

Une image contenant texte

Description générée automatiquement

Création de l’user avec un rôle « ROLE\_ADMIN »

Vérification si on a bien tous les paramètres nécessaires pour continuer l’exécution de la commande => en cas d’erreur on affiche une erreur à l’utilisateur

Récupère les arguments passés en paramètre de la commande

Dans le dossier config/packages puis le fichier security.yaml. Seuls les utilisateurs ayant le rôle ROLE\_ADMIN peuvent accéder aux routes commençant par ^/admin :

Une image contenant texte

Description générée automatiquement

Une fois le rôle admin créer, on peut donc l’utiliser, il existe plusieurs moyens de refuser l’accès à un rôle user :

Dans l’avisController.php :

Une image contenant texte

Description générée automatiquement

Si l’user n’est pas admin, il génère une exception (throw) avec un message d’erreur.

IsGranted, permet de restreindre l’accès. Un message d’erreur sera donc affiché si l’user veut accéder à cette page.

Dans la vue show.html.twig :

Une image contenant texte

Description générée automatiquementSi on a le rôle admin le lien est affiché sinon on le voit pas.

**PANIER DE COMMANDE :**

Symfony fournit un objet de session et plusieurs utilitaires qui permettent de stocker des informations sur l'utilisateur entre les requêtes.

J’ai donc créé un controller CartController.php :

J’utilise la SessionInterface de Symfony, il ne faut pas oublier d’ajouter le namespace :

use Symfony\Component\HttpFoundation\Session\*SessionInterface*;

Une image contenant texte

Description générée automatiquement

Parcourt mon panier et récupère l’id et la quantité d’un article

On ajoute au total le prix d’un article multiplié par la quantité

Récupération du panier ou d’un tableau vide

L’ajout d’un article au panier :

Si ce n’est pas vide on l’incrémente

Sinon je l’initialise a 1

Une image contenant texte

Description générée automatiquement

On sauvegarde dans la session

Une image contenant table

Description générée automatiquement

Vue sur mon site



Une image contenant texte

Description générée automatiquementPour supprimer une quantité d’articles :

Si le panier n’est pas vide

S’il est plus grand que 1, on enlève une quantité

Sinon on supprime la ligne



Pour supprimer la ligne :

Une image contenant texte

Description générée automatiquement

Si le panier n’est pas vide on supprime toute la ligne

Pour supprimer tout le panier : $session->remove('panier');

**STRIPE** **plateforme de traitement des paiements :**

Stripe permet de transférer de l’argent du compte bancaire d’un client vers le compte de l’entreprise au moyen d’une transaction par carte de crédit.

Pour mon cas, Stripe utilise un mode test pour les développeurs :

Sur mon site, ça donne ceci :

Une image contenant texte

Description générée automatiquement

Tout d’abord, il faut installer les composer Stripe avec la commande :

$ composer require stripe/stripe-php

Ensuite, Stripe utilise les clés API dans le compte pour authentifier les requêtes API. Si on n’inclut pas les clés lors d’une requête API ou qu’on utilise une clé non valide ou périmée, Stripe renvoie un code de réponse HTTP 401 - Unauthorized.

Dans la racine de mon projet, je dois ajouter mes clés dans le fichier .env.local :



Une image contenant texte, intérieur, écran

Description générée automatiquementUne image contenant texte

Description générée automatiquementPuis dans le dossier config, fichier services.yaml (point d’entrée pour configurer mes propres services) :

Cela est plus pratique pour taper le code dans le controller, les variables $, sont en privé et dans une fonction « \_\_construct » publique.

J’ai donc créé un StripeController.php. Puis deux entités Order.php et OrderDetail.php pour les commandes et une entité State.php pour le statut des commandes.

Une image contenant texte

Description générée automatiquementDans le controller :

Création d’un objet Order (commande)

Parcourt mon panier et récupère l’article et sa quantité pour créer un objet OrderDetail.

Une image contenant texte

Description générée automatiquement

Créer un tableau de produit avec le prix en euro et le nom de l’article.

Une image contenant texte

Description générée automatiquement

Créer une Session avec l’email de l’user, le mode de paiement et l’url générer si c’est le paiement est réussi ou non.

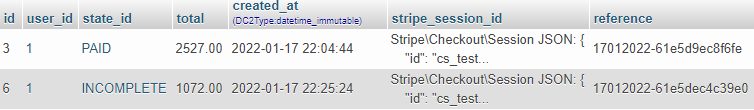
Dans le cas où l’user réussit sa commande :



Dans le cas où l’user ne réussit pas sa commande :



Exemple dans la base de données :



Et dans Stripe :

Une image contenant texte

Description générée automatiquement

**CONCLUSION**

Ce stage m'a permis d'apprendre de nouvelles choses notamment l'usage d'un Framework et de créer un projet de A à Z avec un cahier des charges.

J’ai appris à gérer les problèmes seuls et de trouver la solution à mes problèmes rencontrer.

Finalement, cela m'a donné l'opportunité d’être autonome et de bien géré mon temps.

**ANNEXE :**

**Les règles du MCD :**

R001 un client peut passer plusieurs commandes

R002 une commande a un et un seul client

R003 une commande concerne au moins un détail de commande

R004 un détail de commande concerne une et une seule commande

R005 un détail de commande contient un et un seul article

R006 un article peut contenir un détail de commande

R007 un article peut comporter plusieurs avis

R008 un avis comporte un et un seul article

R009 un avis est donné par un et un seul utilisateur

R010 un utilisateur peut donner plusieurs avis

R011 un article appartient à une et une catégorie

R012 une catégorie peut appartenir à un et plusieurs articles

R013 un article peut avoir une et une seule collection

R014 une collection peut avoir une et plusieurs collections

R015 une commande peut avoir un et un seul statut

R016 un statut peut avoir 0 ou plusieurs commandes

**Lien de la documentation Symfony :**

[Symfony, High Performance PHP Framework for Web Development](https://symfony.com/doc/current/index.html)

**Lors de l’inscription, envoie un mail de confirmation avec MailHog (**<http://localhost:8025/>**) :**

Une image contenant texte

Description générée automatiquement