

Erika Druenes

Rapport de stage

du 26 juillet au 22 octobre 2021

Formation Développement Web Et Web
Mobile

Sommaire

Introduction.....	p.3
Présentation de Village Green.....	p.3
Cahier des charges.....	p.4
Charte Graphique.....	p.6
L'existant.....	p.7
Dictionnaire des données et MCD.....	p.8
Dictionnaire des données.....	p.8
Modèle Conceptuel de Données.....	p.9
Symfony.....	p.13
Architecture globale de Symfony.....	p.13
Le bundle de Sécurité.....	p.14
Les contrôleurs.....	p.14
Authentification.....	p.17
Base de données et relations.....	p.18
Doctrine.....	p.18
Création de l'entité User.....	p.19
GETTERS ET SETTERS.....	p.23
Relations entre les entités avec Doctrine.....	p.23
Formulaire d'inscription.....	p.26
Javascript.....	p.30
Rôles et privilèges.....	p.35
Easy Admin.....	p.37
CRUD.....	p.37
Front.....	p.43
Page home.....	p.50
Page produits.....	p.52
Conclusion.....	p.55

Introduction

Présentation de Village Green:

Village Green est une entreprise fictive qui distribue du matériel musical. Cette entreprise a comme principale activité la revente de matériel musical aux magasins spécialisés dans la musique. Elle aimeraient élargir sa clientèle, et donc vendre son matériel à des clients particuliers via un site internet.



Cahier des charges :

•Créer un module de gestion des produits :

◦Ajouter des produits et leurs caractéristiques (libellé, caractéristiques, prix etc.)

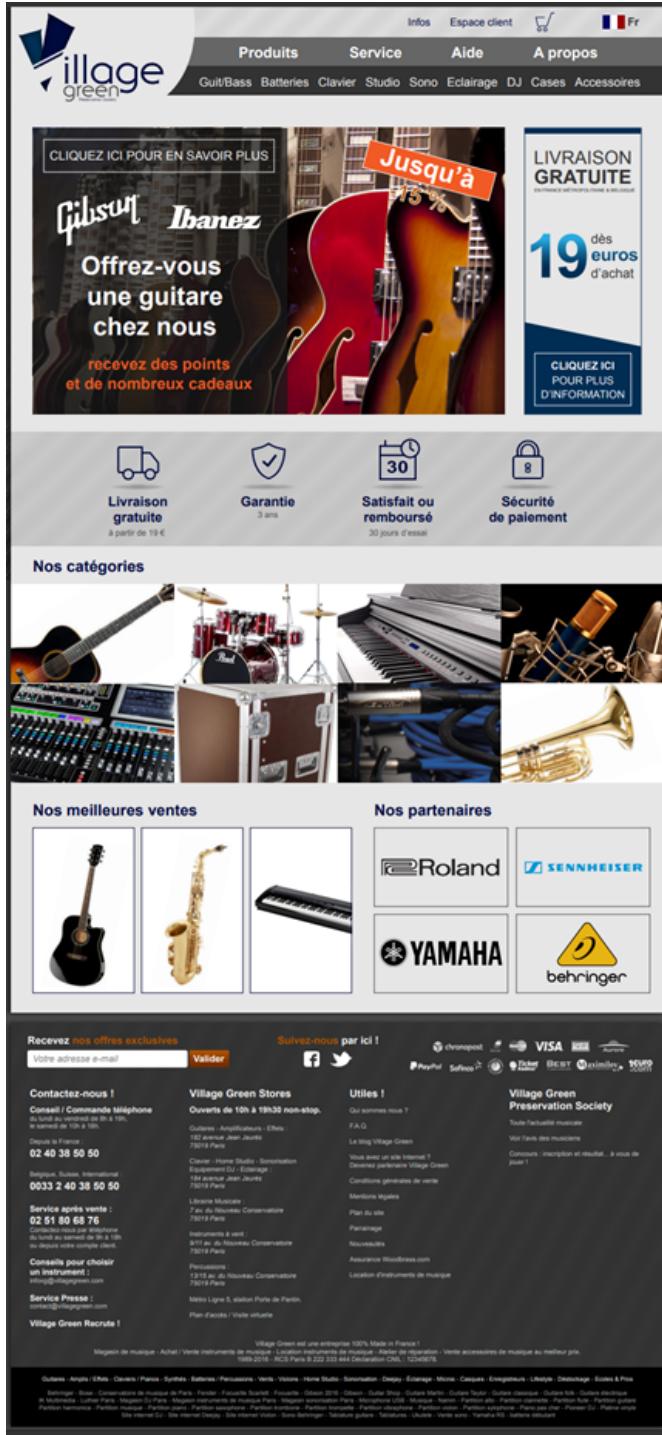
◦Modifier des produits

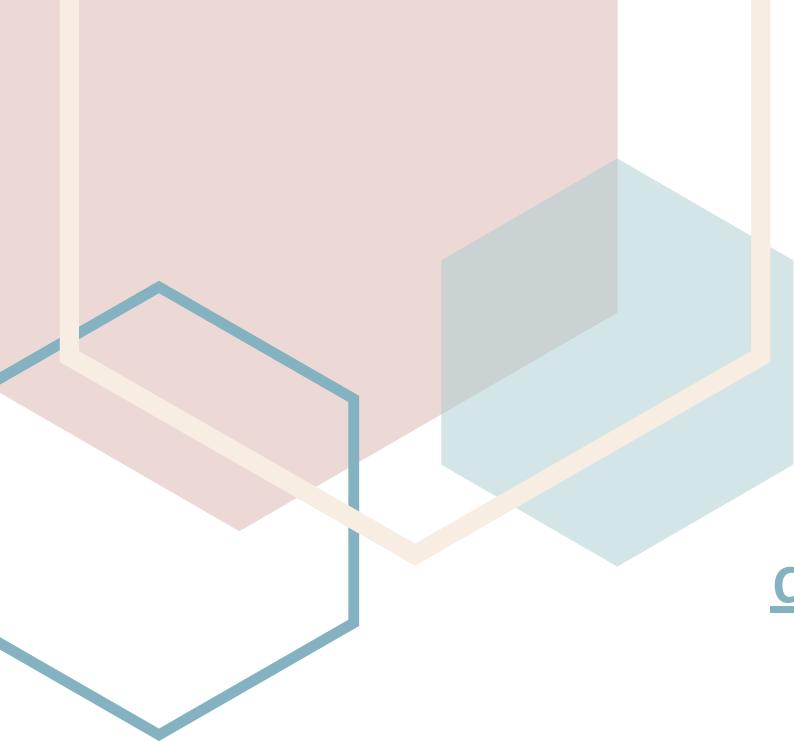
◦Supprimer des produits

- Elaborer le dictionnaire des données à partir du cahier des charges :
 - Les descriptions devront être claires et réalistes.
 - Les informations devront être typées
- Réaliser le MCD du site e-commerce en tenant compte de toutes les contraintes fonctionnelles énoncées dans le cahier des charges (partie L'existant notamment).
- Ecrire le script de création de la base de données (vous pouvez utiliser le script de génération de la base précédemment obtenu).
- Pour la sécurité, prévoir plusieurs profils de connexion et décliner les autorisations nécessaires.
 - Profil visiteur : lecture sur le catalogue
 - Profil client : lecture sur toute la base (insertion et mise à jour dans commande et client)
 - Profil gestion : lecture/écriture dans la base
 - Profil administrateur (ou développeur) : gestion + création et suppression d'objet
- Réaliser un site e-commerce pour les clients particuliers qui permet de consulter le catalogue

- Alimenter la base de tests : créez un script d'insertion des données dans l'ensemble des tables de la base de données. Ces données seront compréhensibles par un utilisateur de base et devront donc avoir des valeurs en cohérence avec le domaine fonctionnel.
- Réaliser une page d'accueil pour votre projet. Vous devez réaliser l'intégration HTML/VSS de votre page d'accueil à partir des éléments qui vous sont fournis dans la charte graphique.
 - Front-office : contient la partie publique du site (dont la page d'accueil) et un accès à la liste de produits et accès au formulaire d'inscription. Vous devez intégrer au mieux les éléments de la charte graphique.
 - Back-office : c'est la partie privée du site, réservée à l'administrateur, elle permet de gérer les produits et les commandes.
- Intégrer des scripts clients (Javascript)
 - Votre application web doit comporter un formulaire d'inscription pour le client.
 - Vous devez empêcher l'utilisateur d'envoyer des informations erronées et lui indiquer les erreurs.
- Mettre en œuvre la gestion CRUD sur une table de votre choix. Ces pages devront être accessibles à partir de votre menu d'accueil.
- Votre interface doit permettre d'afficher la liste des éléments, l'ajout, la modification et la suppression.
- Utiliser une architecture MVC pour réaliser ce travail.

Charte graphique :





L'existant pour la mise en place de la base de données :

- Les produits référencés sont achetés directement auprès des fournisseurs (constructeurs ou importateurs).
- Dans le catalogue de l'entreprise tous les produits sont organisés en catégories, elles-mêmes divisées en sous-catégories.
- Chaque produit doit être décrit par un libellé court et un libellé long (description), une référence fournisseur, un prix d'achat et une photo.
- L'équipe qui gère les relations avec les fournisseurs tient à jour le catalogue. Elle met à jour le stock, valide ou pas la publication de nouveaux produits et désactive d'anciens produits.
- Tous les prix sont notés hors taxes.
- A chaque client est attribué un commercial. Un commercial spécifique s'occupe des clients particuliers
- Quand un client passe une commande, il peut être appliqué une réduction supplémentaire sur le total, cette réduction est négociée par le service commercial.
- Au moment de la commande, il faut prendre en compte l'adresse de livraison et l'adresse de facturation.
- Pour les clients particuliers, un paiement à la commande est exigé.
- Pour les clients professionnels, le paiement se fait en différé.

Dictionnaire des données

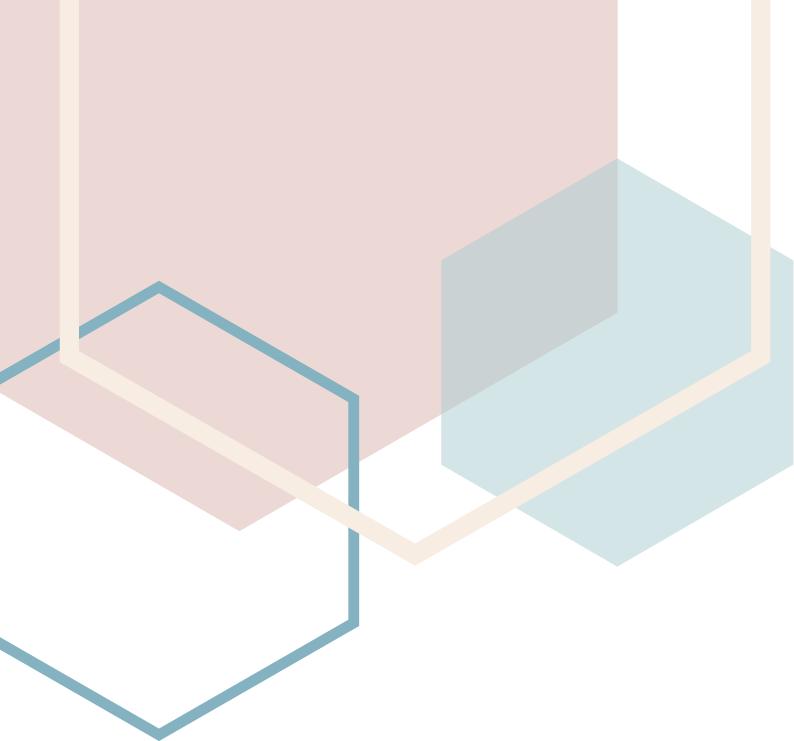
et MCD:

Le dictionnaire des données est la liste de toutes les données qui serviront à structurer la base de données de notre projet, elle permet à toute une équipe de pouvoir comprendre l'organisation, mais aussi le vocabulaire ou les noms donnés aux propriétés par exemple. On précise également à chaque fois à quoi feront référence chaque propriété, et de quel type celles-ci seront. C'est grâce à ce dictionnaire des données, que l'on pourra créer plus facilement un modèle conceptuel de données pour notre projet.

(Annexe 1)

J'ai décidé de créer un tableau de dictionnaire des données pour chaque entité pour que ce soit plus clair, et mieux organisé. Dans la première colonne on trouve de nom de la propriété que l'on trouvera sur le MCD. C'est dans la deuxième colonne que je précise à quoi correspondent les propriétés de chaque entité. Enfin, dans la dernière colonne je donne le type de chaque propriété, qui sera également précisé dans le MCD.

Ce n'est pas expliqué dans mon dictionnaire des données, mais il est recommandé de préfixer chaque propriété des trois premières lettres de l'entité dont elles appartiennent. Cela permet d'éviter de mélanger les id des entités qui permettront d'identifier celles-ci dans les tables de relations (les relations entre les tables seront expliquées dans la partie MCD).



Modèle Conceptuel de Données :

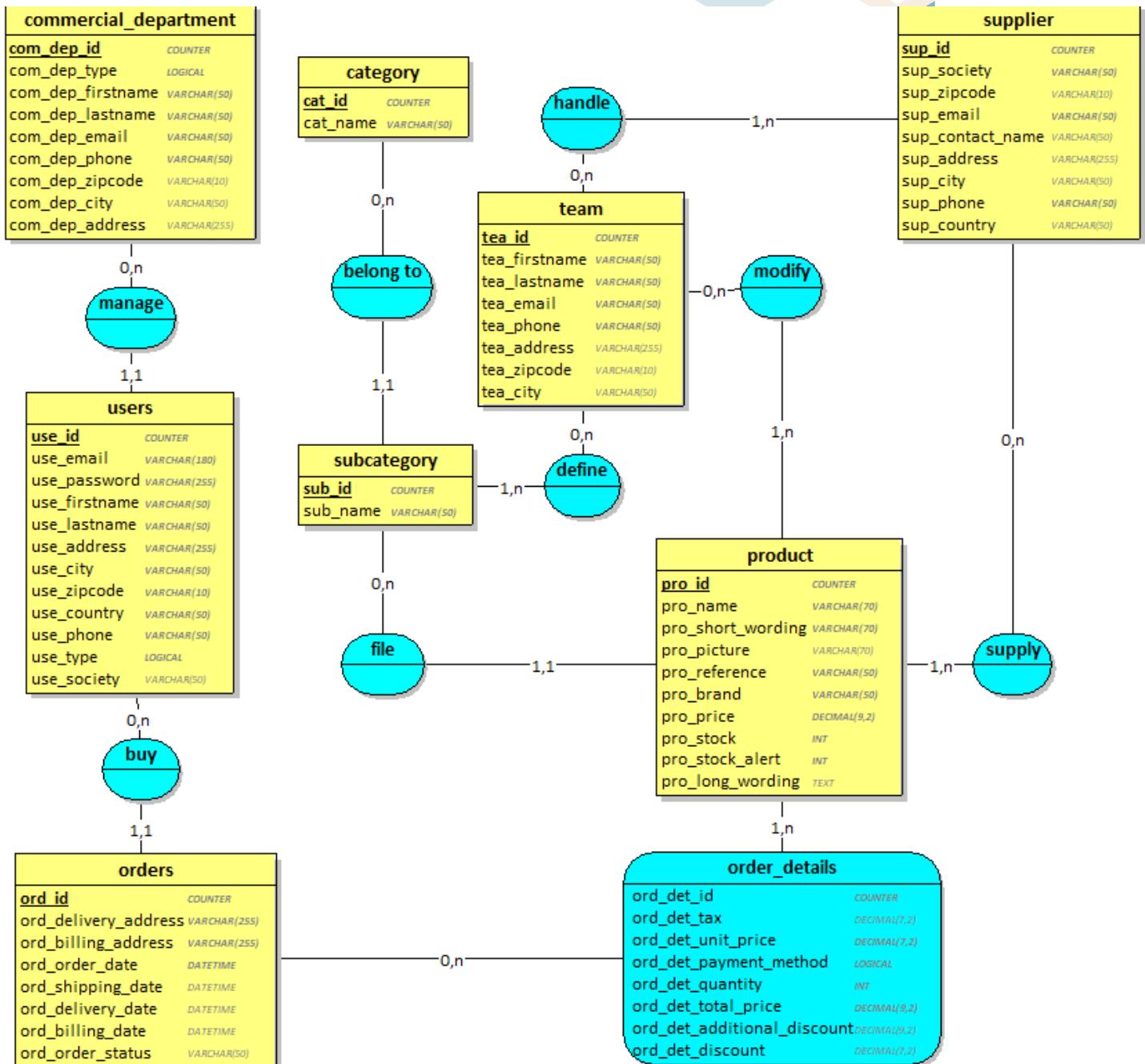
Le modèle conceptuel de données est une représentation graphique et structurée de notre future base de données. Elle a pour but de faciliter la compréhension des relations entre les entités.

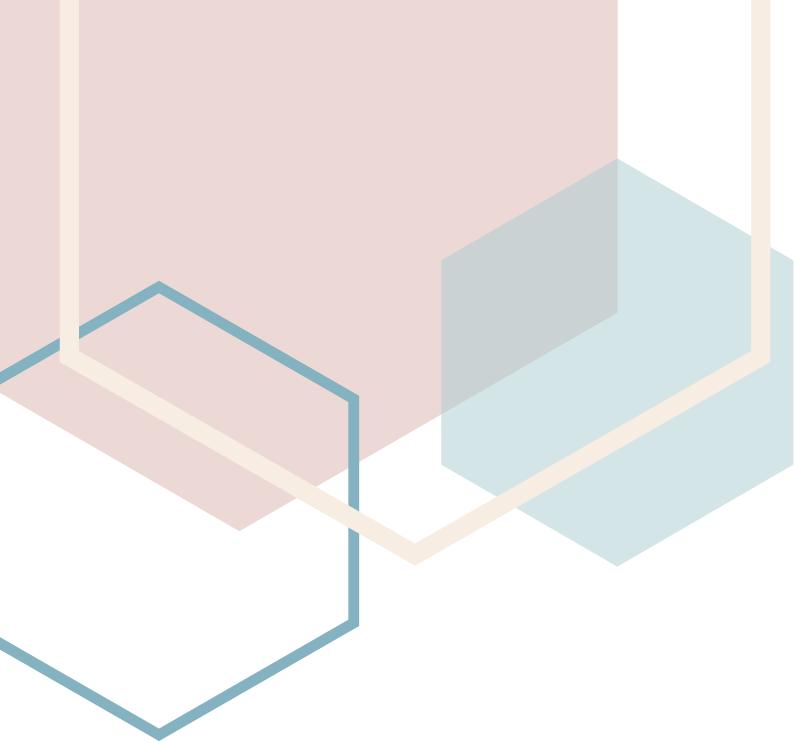
Les entités seront les futures tables de notre base de données, elles sont représentées en jaune. Les entités comportent des propriétés et leur type. Les associations sont en bleues, elles servent à créer la relation entre deux entités. Les associations peuvent devenir de futures tables ayant des clés étrangères dans notre base de données, en fonction de la relation qu'elles ont avec les entités. Les associations sont nommées par un verbe pour décrire l'action qui relie les entités entre elles (sauf order_details qui est un cas à part). En revanche contrairement aux associations, les entités et les propriétés sont toujours nommées par un nom ou par un groupe nominal.

Afin de joindre les entités aux associations, et donc de rendre plus claire la relation entre deux entités, on utilise des liens dans le MCD pour les relier.

Ces liens sont représentés par des traits noirs, et possèdent des cardinalités. Les cardinalités servent à définir le type de relation entre deux Entités. Voici les différents types de cardinalités possibles :

- 0,1 : au minimum 0, au maximum 1 ;
- 0,n : au minimum 0, au maximum plusieurs ;
- 1,1 : au minimum 1, au maximum 1 ;
- 1,n : au minimum 1, au maximum plusieurs ;



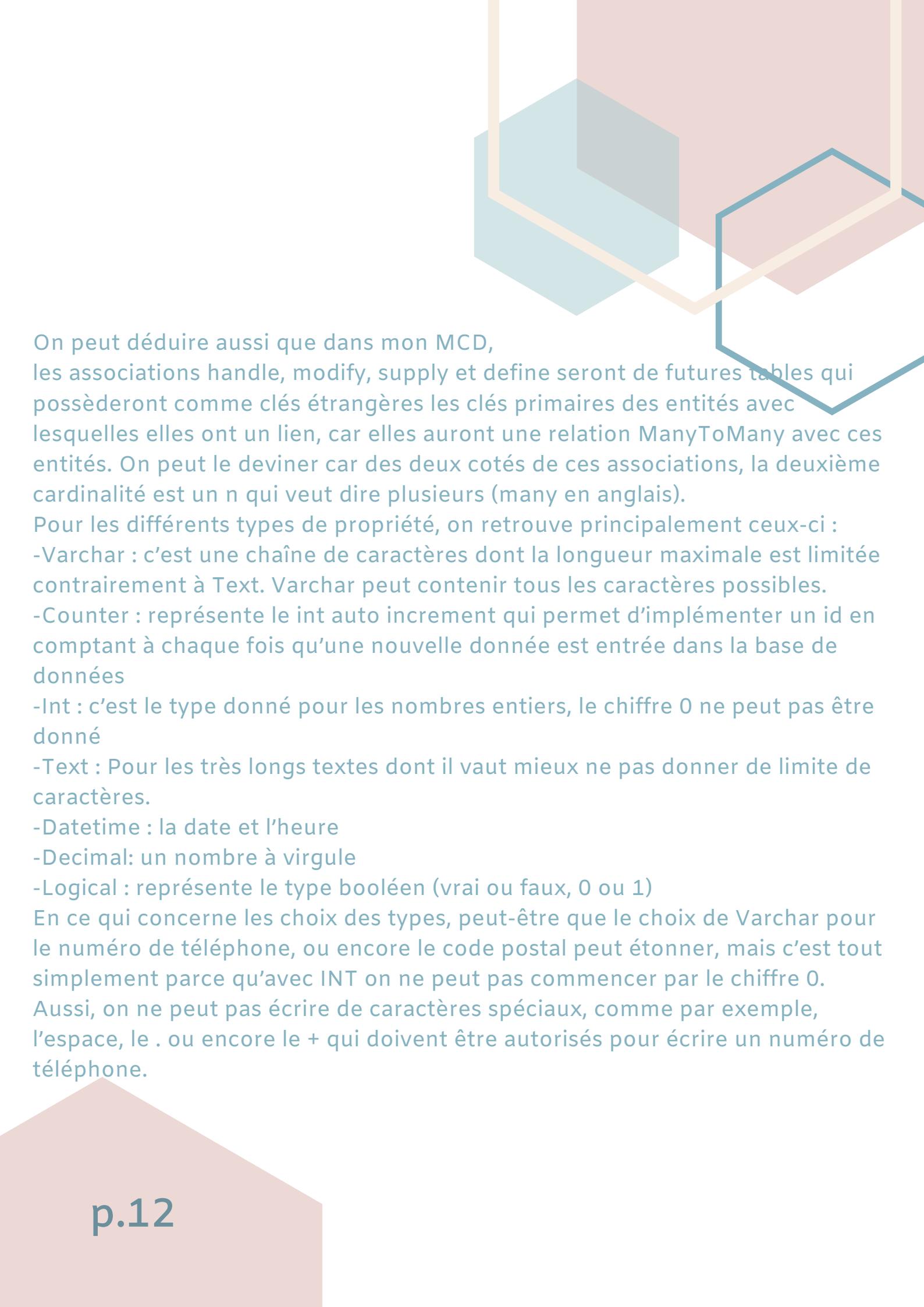


J'ai décidé d'écrire mes entités, mes associations et mes propriétés tout en anglais, car avec Symfony l'entité des utilisateurs doit s'appeler User pour des raisons de sécurité et de formalités. Et donc, pour rester cohérente, j'ai décidé de tout écrire en anglais.

Pour comprendre le fonctionnement des cardinalités, je vais vous expliquer comment j'ai procédé :

Pour les produits, je suis partie du principe qu'un produit (product) peut être rangé (file) à une seule et même (1,1) sous-catégorie (subcategory), mais qu'une sous-catégorie (subcategory) peut contenir 0 ou plusieurs (0,n) produits (product). Et avec le même schéma, une sous-catégorie (subcategory) peut appartenir (belong to) à une seule et même (1,1) catégorie (category), mais une catégorie (category) peut avoir 0 ou plusieurs (0,n) sous-catégories (subcategory).

Prenons exemple maintenant avec les fournisseurs, l'équipe (team) peut s'occuper de (handle) 0 ou plusieurs (0,n) fournisseurs (supplier), et un fournisseur (supplier) est forcément pris en charge par 1 ou plusieurs (1,n) membres de l'équipe (team). Le fournisseur (supplier) fournit (supply) 0 ou plusieurs (0,n) produits (product), alors que le produit (product) est forcément fourni par un ou plusieurs (1,n) fournisseurs (supplier), car le produit n'est pas arrivé tout seul.

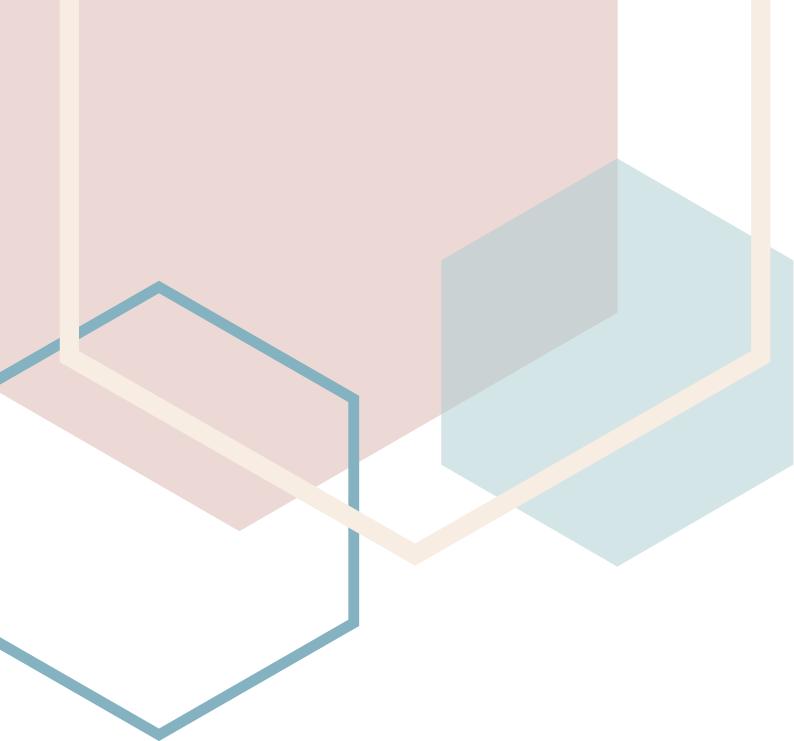


On peut déduire aussi que dans mon MCD, les associations handle, modify, supply et define seront de futures tables qui possèderont comme clés étrangères les clés primaires des entités avec lesquelles elles ont un lien, car elles auront une relation ManyToMany avec ces entités. On peut le deviner car des deux cotés de ces associations, la deuxième cardinalité est un n qui veut dire plusieurs (many en anglais). Pour les différents types de propriété, on retrouve principalement ceux-ci :

- Varchar : c'est une chaîne de caractères dont la longueur maximale est limitée contrairement à Text. Varchar peut contenir tous les caractères possibles.
- Counter : représente le int auto increment qui permet d'implémenter un id en comptant à chaque fois qu'une nouvelle donnée est entrée dans la base de données
- Int : c'est le type donné pour les nombres entiers, le chiffre 0 ne peut pas être donné
- Text : Pour les très longs textes dont il vaut mieux ne pas donner de limite de caractères.
- Datetime : la date et l'heure
- Decimal: un nombre à virgule
- Logical : représente le type booléen (vrai ou faux, 0 ou 1)

En ce qui concerne les choix des types, peut-être que le choix de Varchar pour le numéro de téléphone, ou encore le code postal peut étonner, mais c'est tout simplement parce qu'avec INT on ne peut pas commencer par le chiffre 0. Aussi, on ne peut pas écrire de caractères spéciaux, comme par exemple, l'espace, le . ou encore le + qui doivent être autorisés pour écrire un numéro de téléphone.

Symfony



J'ai décidé d'utiliser Symfony pour développer le site de Village Green. Symfony est un puissant framework MVC (Modèle-Vue-Contrôleur) écrit en PHP qui possède tout un ensemble de composants (librairies), pour développer des sites internet dynamiques, et très sécurisés.

Architecture globale de Symfony:

bin/ : contient la console, ainsi que phpunit

config/ : contient les fichiers de configuration des bundles, des routes, de la sécurité et des services

public/ : contient le contrôleur frontal « index.php » et les fichiers css, javascript et les images

src/ : contient les contrôleurs (C de MVC), les entités (M de MVC) et les formulaires (à vérifier)

templates/ : contient les vues (V de MVC, la partie visuelle du site)

tests/ : contient les fichiers qui permettent de tester le site

var/ : contient les données de cache, le log et les sessions, Symfony l'utilise pendant l'exécution

vendor/ : contient les fichiers sources de Symfony et de ses bundles

Pour résumer la méthode MVC de notre projet, Doctrine servira à gérer l'accès aux données (M), Twig se chargera d'afficher les données (V), et les contrôleurs géreront le traitement des données (C).

Afin d'installer certains bundles sur mon projet, j'ai avant tout installé Composer qui est un logiciel de gestionnaire de dépendances écrites en PHP.

Le bundle de Sécurité:

Un bundle est un ensemble de fichiers de code créés par des développeurs, qui permet de réaliser une ou plusieurs fonctionnalités. Dans mon projet, j'ai installé les bundles en entrant des lignes de commandes (que l'on trouve dans la documentation Symfony) dans le terminal.

Le premier bundle que j'ai installé sur mon projet Village Green est le bundle Security. Il permet de configurer des fonctionnalités comme l'authentification des utilisateurs, ou encore l'autorisation basée sur les rôles.

Security a installé le SecurityController qui permet de créer le système de connexion (login) et de déconnexion (logout), et de vérifier l'authentification du username de l'utilisateur.

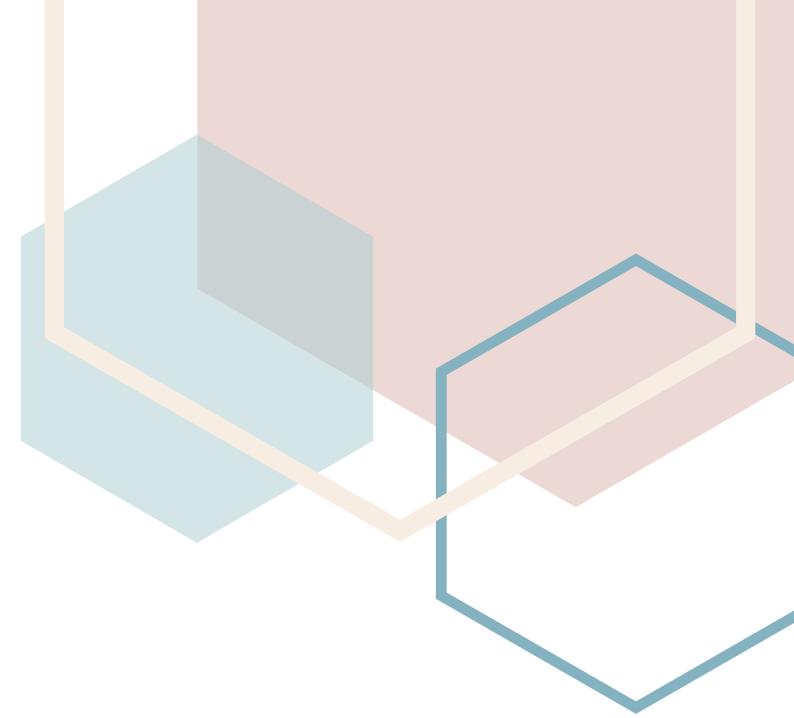
(Annexe 2)

Les contrôleurs:

Dans les contrôleurs (Controller) on crée des fonctions qui permettent de récupérer les informations d'objets et de les retourner avec les routes.

Dans chaque contrôleur, au début il faut toujours ajouter le namespace qui précise la route du fichier, mais aussi les use qui servent à importer les namespace des classes ou des paramètres utilisés.

```
src > Controller > VillageGreenController.php > ...
1  <?php
2
3  namespace App\Controller;
4
5  use App\Entity\Product;
6  use App\Repository\ProductRepository;
7  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
8  use Symfony\Component\HttpFoundation\Response;
9  use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;
10 use Symfony\Component\Routing\Annotation\Route;
11
12 class VillageGreenController extends AbstractController
13 {
14     /**
15      * @Route("/", name="home")
16      */
17     public function home(): Response
18     {
19         return $this->render('villagegreen/home.html.twig', [
20             'controller_name' => 'VillageGreenController',
21         ]);
22     }
23
24     /**
25      * @Route("/inscription", name="inscription")
26      */
27     public function inscription()
28     {
29         return $this->render('villagegreen/inscription.html.twig');
30     }
31
32     /**
33      * @Route("/produits", name="produits")
34      */
35     public function produits(ProductRepository $repo): Response
36     {
37         $repo = $this->getDoctrine()->getRepository(Product::class);
38
39         $products = $repo->findAll();
40
41         return $this->render('villagegreen/produits.html.twig', [
42             'controller_name' => 'VillageGreenController',
43             'products' => $products
44         ]);
45     }
46
47     /**
48      * @Route("/show/{id}", name="show")
49      */
50     public function show($id, ProductRepository $productRepo) : Response
51     {
52         $product = $productRepo->findOneBy(['id' => $id]);
53
54         if(!$product){
55             throw new NotFoundHttpException('Produit introuvable');
56         }
57
58         return $this->render('villagegreen/show.html.twig', [
59             'controller_name' => 'VillageGreenController',
60             'product' => $product
61         ]);
62     }
63 }
```



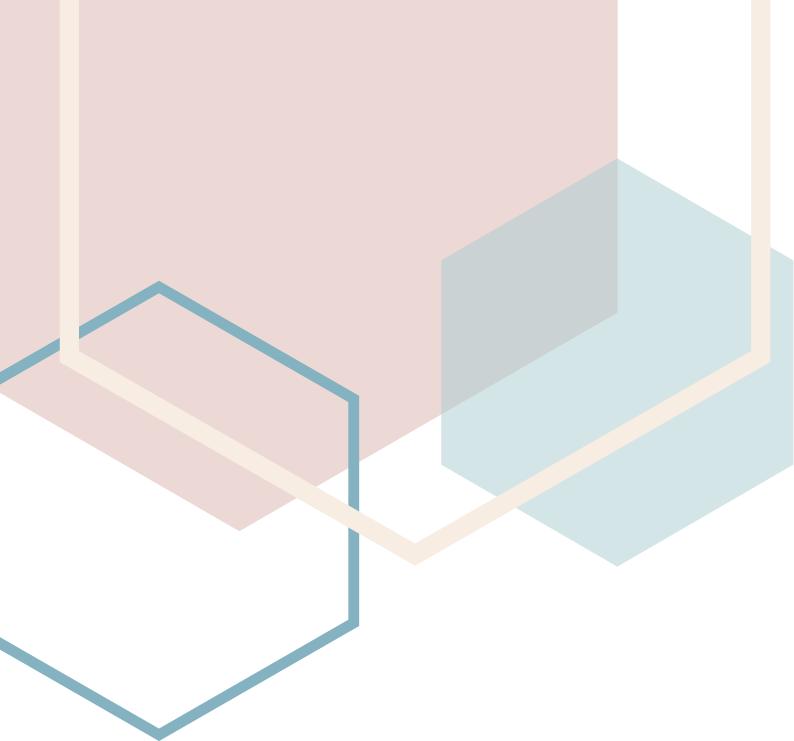
Dans le VillageGreenController, la classe VillageGreenController va contenir plusieurs fonctions qui vont servir à créer les routes qui retourneront les informations qui se trouvent dans les paramètres des fonctions. A noter que les templates de ces routes se trouveront donc dans le dossier VillageGreen des templates.

14 à 16 : C'est ici que l'on annote la route.

17 à 21 : Dans la fonction home, on demande à ce que la réponse renvoyée soit le template soit home.html.twig qui se trouve dans le dossier VillageGreen, et que donc son contrôleur est VillageGreenController.

35 à 39 : Le dépôt de l'entité Produit est mis en paramètre, on demande à Doctrine de récupérer tous les produits pour pouvoir tous les afficher.

52 à 56 : On demande à ce qu'il n'y ait qu'un produit qui soit affiché et identifié par son id. Puis on écrit comme condition que si le produit n'existe pas, alors on renvoie comme erreur « Produit introuvable ».



Authentification:

Ensuite j'ai installé le composant Guard qui permet de combiner plusieurs couches d'authentification. Puis j'ai installé Make:auth qui va permettre d'authentifier un utilisateur, il va vérifier si l'adresse email (qui est utilisé comme identifiant pour l'utilisateur) est unique ou non. Ainsi, il est impossible de créer deux comptes d'utilisateur avec la même adresse email.

```
config/packages/security.yaml
1 security:
2     encoders:
3         App\Entity\User:
4             algorithm: auto
5
6     # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
7     providers:
8         # used to reload user from session & other features (e.g. switch_user)
9         app_user_provider:
10            entity:
11                class: App\Entity\User
12                property: email
```

1 à 4 : le bundle Security a installé un algorithme d'encodage qui servira pour les mots de passe des utilisateurs.

7 à 12 : l'utilisateur sera identifié et reconnu par son adresse email.

Base de données

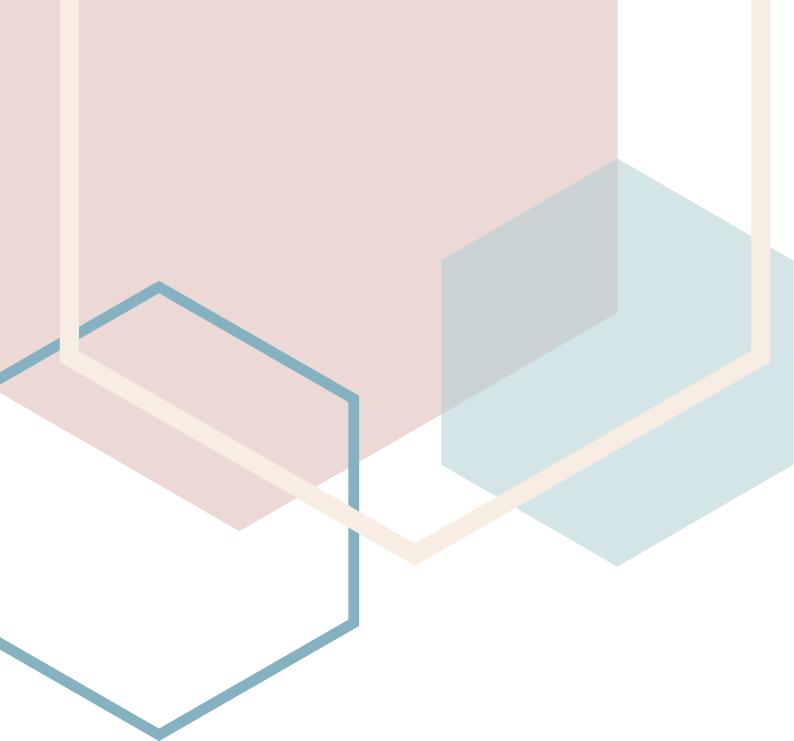
et relations

Doctrine:

Pour créer mes entités et relations, j'ai utilisé l'ORM (mapping objet-relationnel) Doctrine qui permet d'interagir avec la base de données sans écrire de requête SQL. Par conséquent, les tables de notre base de données seront utilisées comme des objets.

```
23  ##> doctrine/doctrine-bundle ###
24  # Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connecting-using-a-url
25  # IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
26  #
27  # DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
28  DATABASE_URL="mysql://root:@127.0.0.1:3306/village_green?serverVersion=5.7"
29  # DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=13&charset=utf8"
30  ##< doctrine/doctrine-bundle ###
```

Afin de relier ma base de données à mon code, dans le fichier .env, j'ai décommenté le lien mysql, et je l'ai modifié en précisant le nom de ma base de données (village_green), ainsi que mon identifiant de connexion (root).



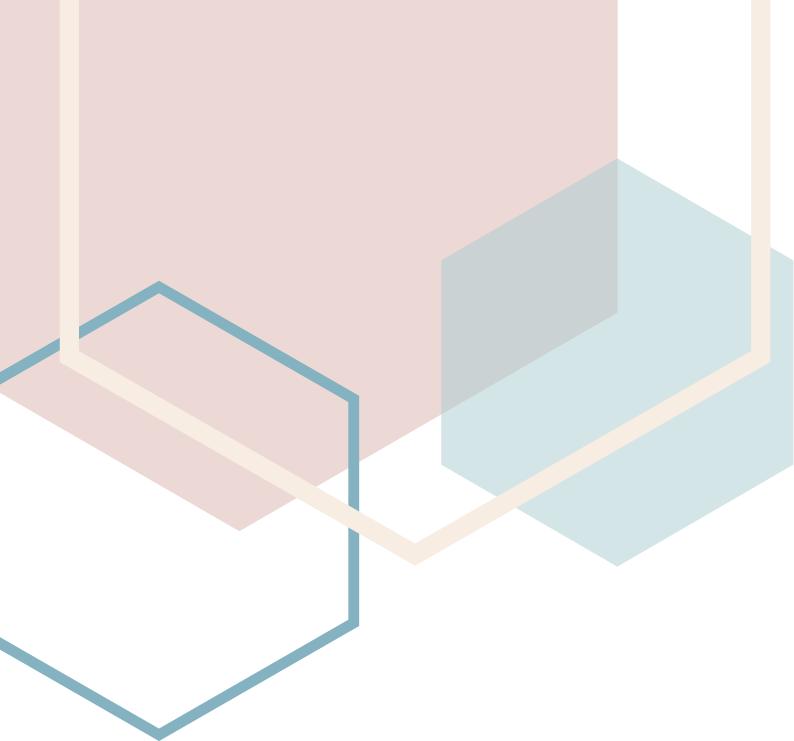
Création de l'entité User:

Make :user sert à créer une entité User (Utilisateur) avec comme propriété par défaut l'adresse mail qui servira d'identifiant et le mot de passe qui sera hashé et caché. Le mot de passe sera vérifié s'il correspond au bon mot de passe et aux règles imposés.

```

13 /**
14 * @ORM\Entity(repositoryClass=UserRepository::class)
15 * @UniqueEntity(fields={"email"}, message="Il existe déjà un compte possèdant cette adresse email")
16 */
17 class User implements UserInterface
18 {
19     /**
20      * @ORM\Id
21      * @ORM\GeneratedValue
22      * @ORM\Column(type="integer")
23      */
24     private $id;
25
26     /**
27      * @ORM\Column(type="string", Length=180, unique=true)
28      */
29     private $email;
30
31     /**
32      * @ORM\Column(type="json")
33      */
34     private $roles = [];
35
36     /**
37      * @var string The hashed password
38      * @ORM\Column(type="string")
39      * @Assert\Length(min="8", minMessage="Votre mot de passe doit faire minimum 8 caractères")
40      */
41     private $password;
42
43     /**
44      * @ORM\Column(type="string", Length=50)
45      */
46     private $firstname;
47
48     /**
49      * @ORM\Column(type="string", Length=50)
50      */
51     private $lastname;
52
53     /**
54      * @ORM\Column(type="string", Length=255)
55      */
56     private $address;
57
58     /**
59      * @ORM\Column(type="string", Length=50)
60      */
61     private $city;
62
63     /**
64      * @ORM\Column(type="string", Length=10)
65      */
66     private $zipcode;
67
68     /**
69      * @ORM\Column(type="string", Length=50)
70      */
71     private $country;
72
73     /**
74      * @ORM\Column(type="string", Length=50, nullable=true)
75      */
76     private $phone;
77
78     /**
79      * @ORM\Column(type="boolean", nullable=true)
80      */
81     private $type;
82
83     /**
84      * @ORM\Column(type="string", Length=50, nullable=true)
85      */
86     private $society;
87

```



14 à 15 : On va chercher le dépôt de l'entité User grâce à l'ORM Doctrine. Ensuite on précise qu'un compte utilisateur est unique grâce à l'adresse email, et donc si une personne tente de s'inscrire avec une adresse email déjà utilisée pour un autre compte, un message d'erreur s'affichera avec écrit « Il existe déjà un compte possédant cette adresse email ».

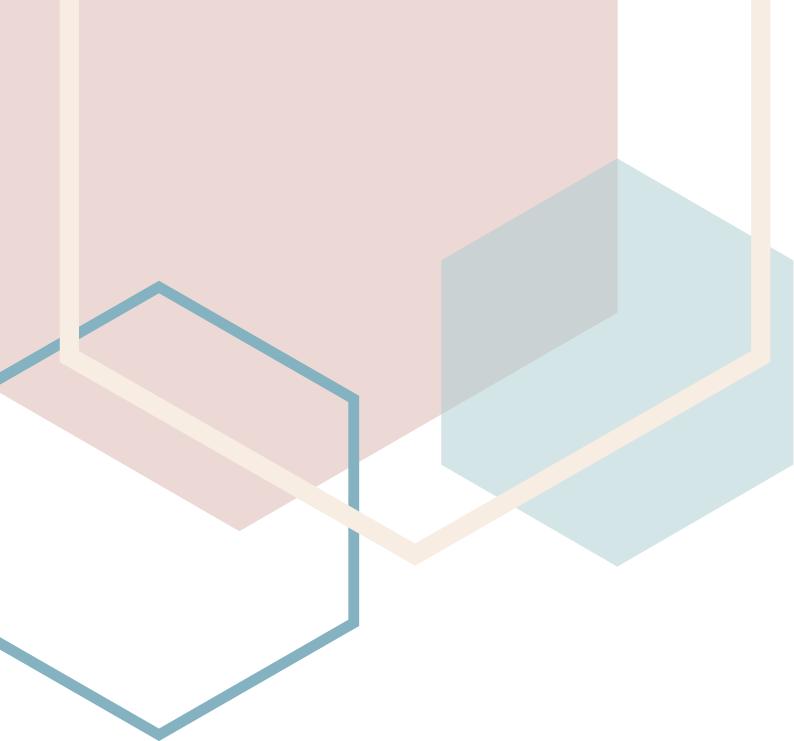
17 : On crée la classe User et à l'intérieur les propriétés sont déclarées avec des annotations au-dessus.

19 à 23 : Ces annotations auto-génèrent un id de type integer (nombre entier INT) pour chaque utilisateur.

37 à 39 : Il est précisé que le mot de passe sera hashé dans la base de données, qu'il sera de type string (VARCHAR), qu'il faudra qu'il fasse minimum 8 caractères sinon un message d'erreur s'affichera : « Votre mot de passe doit faire minimum 8 caractères ».

Les Assert sont un autre moyen de sécuriser le formulaire d'inscription, ils permettent par exemple de mettre des contraintes d'insertion aux utilisateurs. J'ai pu les utiliser grâce au composant Validator de Symfony.

```
97
98     public function __construct()
99     {
100         $this->orders = new ArrayCollection();
101     }
102
103     public function getId(): ?int
104     {
105         return $this->id;
106     }
107
108     public function getEmail(): ?string
109     {
110         return $this->email;
111     }
112
113     public function setEmail(string $email): self
114     {
115         $this->email = $email;
116
117         return $this;
118     }
119
120     /**
121      * A visual identifier that represents this user.
122      *
123      * @see UserInterface
124      */
125     public function getUsername(): string
126     {
127         return (string) $this->email;
128     }
129
130     /**
131      * @see UserInterface
132      */
133     public function getRoles(): array
134     {
135         $roles = $this->roles;
136         // guarantee every user at least has ROLE_USER
137         $roles[] = 'ROLE_USER';
138
139         return array_unique($roles);
140     }
141
142     public function setRoles(array $roles): self
143     {
144         $this->roles = $roles;
145
146         return $this;
147     }
148
```



GETTERS ET SETTERS :

Les getters et setters servent à récupérer les variables privées. Grâce aux getters on peut utiliser (lire) ces variables, et grâce aux setters on peut les créer ou modifier.

Les getters sont les fonctions qui utilisent la méthode get, et les setters sont les fonctions qui utilisent la méthode set.

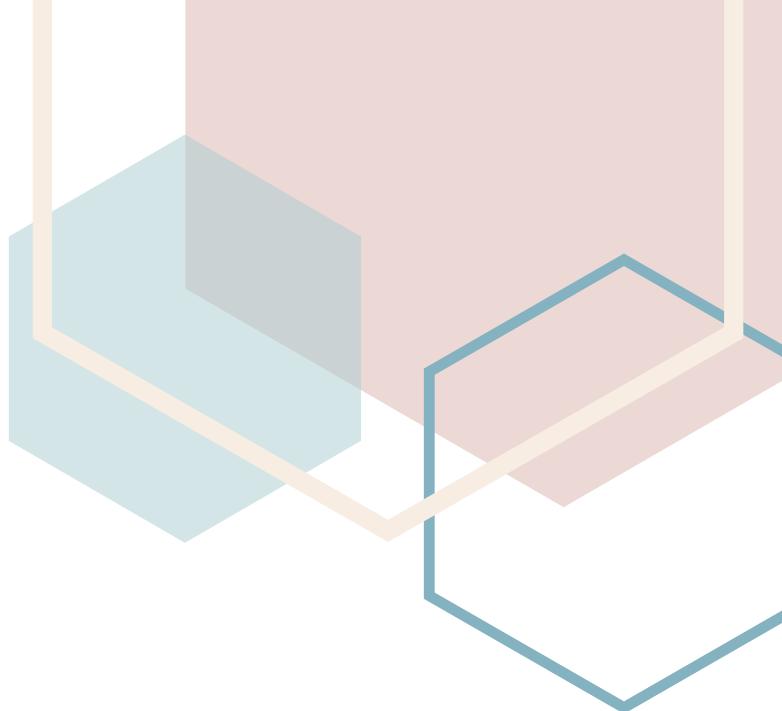
120 à 128 : L'adresse email est déclarée comme l'identifiant de l'utilisateur.
133 à 140 : On déclare le rôle utilisateur (ROLE_USER) comme le rôle par défaut, c'est-à-dire que chaque utilisateur possède au moins le rôle utilisateur.

Relations entre les entités avec Doctrine :

Pour comprendre les relations entre les entités, il faut savoir faire la différence entre les entités propriétaires et les entités inverses. Une relation est faite entre une entité propriétaire et une entité inverse. L'entité propriétaire est l'entité qui possède la référence de l'autre entité qui est l'entité inverse.

Les 4 différents types de relation :

Les 4 différents types de relation entre les entités sont OneToOne, OneToMany, ManyToOne, ManyToMany.



Afin d'expliquer à quoi correspondent les différents types de relations, j'ai donné pour chacune d'entre elle l'exemple de la relation entre la catégorie et la sous-catégorie. Dans cet exemple, on part du principe que la sous-catégorie est l'entité propriétaire, et la catégorie l'entité inverse.

OneToOne : chaque sous-catégorie est liée à une seule catégorie, et chaque catégorie possède une seule sous-catégorie.

OneToMany : chaque sous-catégorie peut être liée à plusieurs catégories, et chaque catégorie possède une seule sous-catégorie.

ManyToOne : chaque sous-catégorie est liée à une seule catégorie, et chaque catégorie peut posséder plusieurs sous-catégories.

ManyToMany : chaque sous-catégorie peut être liée à plusieurs catégories, et chaque catégorie peut posséder plusieurs sous-catégories.

Voici les relations de ma base de données:

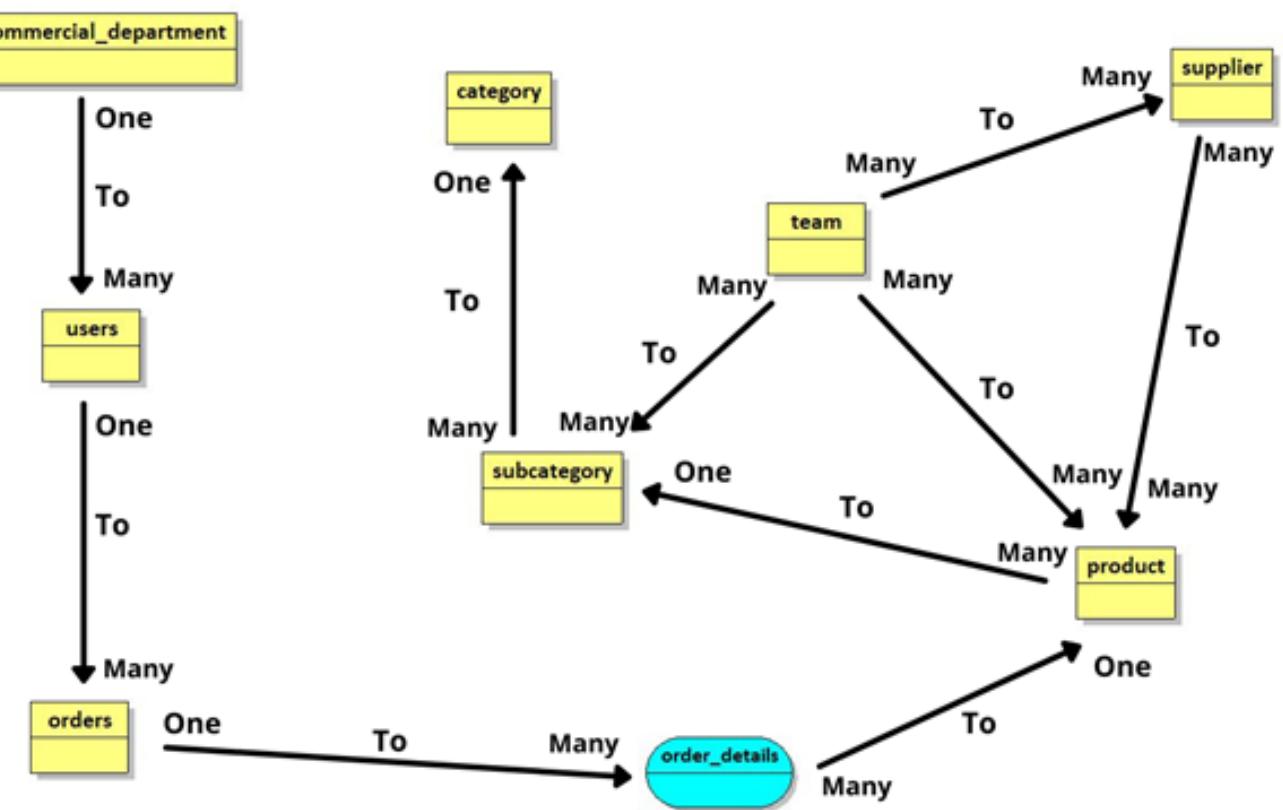


Table	Action
category	Parcourir Structure Rechercher Insérer Vider Supprimer
commercial_department	Parcourir Structure Rechercher Insérer Vider Supprimer
doctrine_migration_versions	Parcourir Structure Rechercher Insérer Vider Supprimer
order	Parcourir Structure Rechercher Insérer Vider Supprimer
order_details	Parcourir Structure Rechercher Insérer Vider Supprimer
product	Parcourir Structure Rechercher Insérer Vider Supprimer
subcategory	Parcourir Structure Rechercher Insérer Vider Supprimer
supplier	Parcourir Structure Rechercher Insérer Vider Supprimer
supplier_product	Parcourir Structure Rechercher Insérer Vider Supprimer
team	Parcourir Structure Rechercher Insérer Vider Supprimer
team_product	Parcourir Structure Rechercher Insérer Vider Supprimer
team_subcategory	Parcourir Structure Rechercher Insérer Vider Supprimer
team_supplier	Parcourir Structure Rechercher Insérer Vider Supprimer
user	Parcourir Structure Rechercher Insérer Vider Supprimer
14 tables	Somme

Comme prévu, de nouvelles tables se sont créées (supplier_product, team_product, team_subcategory, team_supplier) grâce à la relation ManyToMany.

Formulaire

d'inscription

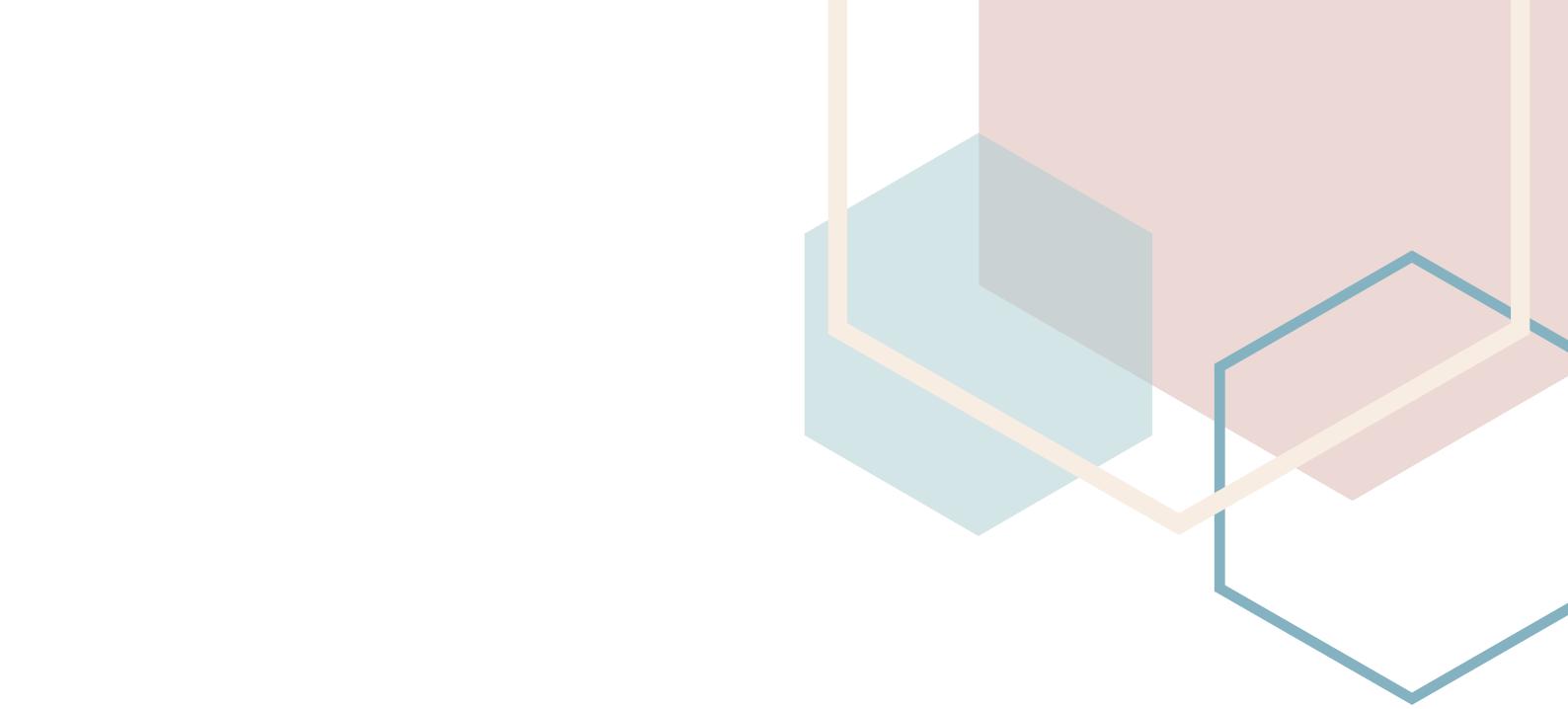
Afin de créer le formulaire d'inscription, j'ai installé le composant registration-form. Ce composant fournit différents outils qui facilite la sécurisation du formulaire.

(Annexe 3)

```

src > Form > RegistrationFormType.php > ...
1  <?php
2
3  namespace App\Form;
4
5  use App\Entity\User;
6  use Symfony\Component\Form\AbstractType;
7  use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
8  use Symfony\Component\Form\Extension\Core\Type\PasswordType;
9  use Symfony\Component\Form\Extension\Core\Type\RepeatedType;
10 use Symfony\Component\Form\FormBuilderInterface;
11 use Symfony\Component\OptionsResolver\OptionsResolver;
12 use Symfony\Component\Validator\Constraints\IsTrue;
13 use Symfony\Component\Validator\Constraints\Length;
14 use Symfony\Component\Validator\Constraints\NotBlank;
15
16 class RegistrationFormType extends AbstractType
17 {
18     public function buildForm(FormBuilderInterface $builder, array $options)
19     {
20         $builder
21             ->add('email')
22             ->add('firstname')
23             ->add('lastname')
24             ->add('address')
25             ->add('city')
26             ->add('zipcode')
27             ->add('country')
28             ->add('phone')
29             ->add('agreeTerms', CheckboxType::class, [
30                 'mapped' => false,
31                 'constraints' => [
32                     new IsTrue([
33                         'message' => 'Veuillez cocher cette case.',
34                     ]),
35                 ],
36             ])
37             ->add('plainPassword', RepeatedType::class, [
38                 // instead of being set onto the object directly,
39                 // this is read and encoded in the controller
40                 'type'=> PasswordType::class,
41                 'invalid_message' => 'Le mot de passe doit être le même.',
42                 'options' => ['attr' => ['class' => 'password-field']],
43                 'required' => true,
44                 'first_options' => ['label' => 'Password'],
45                 'second_options' => ['label' => ' '],
46                 'mapped' => false,
47                 'attr' => ['autocomplete' => 'new-password'],
48                 'constraints' => [
49                     new NotBlank([
50                         'message' => 'Entrez un mot de passe',
51                     ]),
52                     new Length([
53                         'min' => 8,
54                         'minMessage' => 'Le mot de passe doit contenir au moins {{ limit }} caractères',
55                         // max Length allowed by Symfony for security reasons
56                         'max' => 4096,
57                     ]),
58                 ],
59             ])
60         ;
61     }
62
63     public function configureOptions(OptionsResolver $resolver)
64     {
65         $resolver->setDefaults([
66             'data_class' => User::class,
67         ]);
68     }
69 }
70

```



16 à 62 : dans la classe `RegistrationFormType`, on crée une fonction qui permet de construire le formulaire.

20 à 36 : on ajoute les propriétés au constructeur, on ajoute également une case à cocher (`CheckboxType`) pour les conditions générales (`AgreeTerms`). A la suite, on fait en sorte que l'utilisateur coche bien la case avec une contrainte. Si la case n'est pas cochée, alors le message « Veuillez cocher cette case » s'affiche.

37 à 60 : on demande à ce que le mot de passe soit répété (pour être sûr que l'utilisateur est bien tapé le bon mot de passe) avec le `RepeatedType`. `PasswordType` fait en sorte que le mot de passe soit caché avec des étoiles au moment de son écriture. Un message s'affiche si le mot de passe n'est pas le même. « `required=>true` » rend le champ obligatoire. On ajoute comme contraintes d'obligatoirement écrire un mot de passe, il doit faire au moins 8 caractères et peut faire au maximum 4096 caractères (pour des questions de sécurité). Ces contraintes sont encore une fois possible grâce au composant `Validator` de `Symfony`.

64 à 69 : permet de valider les paramètres passés en GET sur une action en testant leur présence.

```

templates > registration > register.html.twig
1  (% extends "base.html.twig" %)
2
3  (% block title %)Inscription(% endblock %)
4
5
6  (% block body %)
7  <article>
8    <h1>
9      <br>
10     <br>
11     <br>
12     <form method="POST" action="#">
13       <h1 class="text-center">Créer vos identifiants</h1>
14       <br>
15       <br>
16       {{ form_start(registrationForm) }}
17       <div class="container form-group row">
18         <div class="col-md-4 offset-md-2 ">
19           {{ form_row(registrationForm.email, { "label": "Email", "id": "email", "attr":{("class": "form-control")}}) }}
20           <span id="missEmail"></span>
21           {{ form_row(registrationForm.plainPassword.first, { "label": "Mot de passe", "id": "motdepasse", "attr":{("class": "form-control")}}) }}
22           <span id="missPassword"></span>
23         </div>
24         <div class="col-md-4 offset-md-1 ">
25           {{ form_row(registrationForm.plainPassword.second, { "attr":{("class": "form-control confirmation_motdepasse")}}) }}
26           <label class="mdp confi_mdp" for="mdp">Confirmation de mot de passe</label>
27         </div>
28         <span class="col-md-4 offset-md-2 " id="missMotdepasse"></span>

```

12 : La méthode POST permet de récupérer les données du formulaire au moment de l'inscription de manière masquée (mais non cryptée).

On utilise Twig pour le template de notre formulaire d'inscription.

16 : le formulaire d'inscription commence ici.

```

html.twig
<div class="col-md-4 container-fluid"><div class="form-group float-left inscription">
{{ form_row(registrationForm.firstname, { "label": "Prenom", "id": "prenom", "attr":{("placeholder": "Prenom", "class": "form-control")}}) }}
<span id="missPrenom"></span>
{{ form_row(registrationForm.lastname, { "label": "Nom", "id": "nom", "attr":{("class": "form-control")}}) }}
<span id="missNom"></span>
{{ form_row(registrationForm.address, { "label": "Adresse", "id": "adresse", "attr":{("class": "form-control")}}) }}
<span id="missAdresse"></span>
{{ form_row(registrationForm.city, { "label": "Ville", "id": "ville", "attr":{("class": "form-control")}}) }}
<span id="missVille"></span>
{{ form_row(registrationForm.zipcode, { "label": "Code postal", "id": "codepostal", "attr":{("class": "form-control")}}) }}
<span id="missCodepostal"></span>
{{ form_row(registrationForm.country, { "label": "Pays", "id": "pays", "attr":{("class": "form-control")}}) }}
<span id="missPays"></span>
{{ form_row(registrationForm.phone, { "label": "Numero de telephone", "id": "telephone", "attr":{("class": "form-control")}}) }}
<span id="missTelephone"></span>
<br>
{{ form_row(registrationForm.agreeTerms, { "label": "J'accepte les conditions", "id": "checkbox", "attr":{("class": "form-check-input")}}) }}
<span id="missCheckbox"></span>
<br>
<button type="submit" class="btn btn-submit" id="boutonenvoi">Valider</button>
{{ form_end(registrationForm) }}
<br>
<br>
```

19 à 60 : j'ai décidé d'utiliser form_row car il permet de personnaliser notre formulaire d'inscription comme on le souhaite en ayant la possibilité de rajouter des options. Les données sont récupérées grâce à registrationForm.propriété qui est lié au RegistrationFormType, qui est lui-même lié à l'entité User (Utilisateur) de notre base de données. En paramètres, on retrouve l'id qui nous servira pour notre javascript (que l'on verra dans la partie suivante), mais également les attributs (attr). Les balises span serviront à afficher les messages d'erreur javascript au cas où l'utilisateur n'ait pas rempli correctement le formulaire d'inscription.

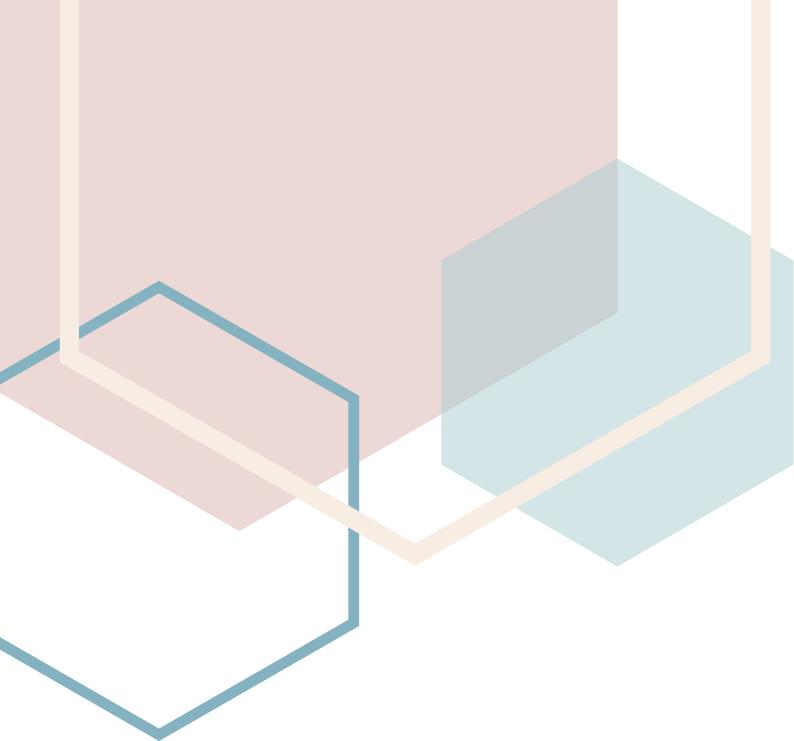
59 à 60 : le bouton de type submit va permettre d'envoyer les données dans notre base de données au moment de l'inscription (au moment où l'utilisateur va appuyer sur ce bouton).

Javascript:

Une autre façon de sécuriser le formulaire d'inscription et de forcer l'utilisateur à respecter les contraintes d'insertion est de créer des regex (expressions régulières) et des fonctions Javascript.

Pour chaque propriété, on crée une variable où l'on récupère l'élément grâce à l'id qui se trouve dans le template. Ensuite on crée une deuxième variable qui est la valeur valide de la propriété qui correspond à l'expression régulière donnée. Puis on crée une troisième variable qui correspondra à la valeur manquante (Miss).

```
public > build > JS app.js > ...
1 var formValid = document.getElementById("boutonenvoi");
2 var nom = document.getElementById("nom");
3 var nomValid = /^[A-Za-zÀ-ÿ]{1,13}[- ]{0,1}[A-Za-zÀ-ÿ]{0,12}[ ']{0,1}[A-Za-zÀ-ÿ]{1,13}[- ]{0,1}[A-Za-zÀ-ÿ]{0,12}$/;
4 var missNom = document.getElementById("missNom");
5 var prenom = document.getElementById("prenom");
6 var prenomValid = /^[A-Za-zÀ-ÿ]{1,13}[- ]{0,1}[A-Za-zÀ-ÿ]{0,12}[ ']{0,1}[A-Za-zÀ-ÿ]{1,13}[- ]{0,1}[A-Za-zÀ-ÿ]{0,12}$/;
7 var missPrenom = document.getElementById("missPrenom");
8 var codepostal = document.getElementById("codepostal");
9 var codepostalValid = /^[a-zA-Z]{1,10}$/;
10 var missCodepostal = document.getElementById("missCodepostal");
11 var email = document.getElementById("email");
12 var emailValid = /^[a-zA-Z]{1,}[.]{0,1}[a-zA-Z]{1,}@{1,}[a-zA-Z]{1,}[.]{1,}[a-zA-Z]{1,5}$/;
13 var missEmail = document.getElementById("missEmail");
14 var motdepasse = document.getElementById("motdepasse");
15 var motdepasseValid = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@#$!%*?&]{8,50}$/;
16 var missMotdepasse = document.getElementById("missMotdepasse");
17 var checkbox = document.getElementById("checkbox");
18 var missCheckbox = document.getElementById("missCheckbox");
19 var adresse = document.getElementById("adresse");
20 var adresseValid = /^[a-zA-ZÀ-ÿ\d\s',-]{5,250}$/;
21 var missAdresse = document.getElementById("missAdresse");
22 var ville = document.getElementById("ville");
23 var villeValid = /^[A-Za-zÀ-ÿ]{1,13}[- ]{0,1}[A-Za-zÀ-ÿ]{0,12}[ ']{0,1}[A-Za-zÀ-ÿ]{1,13}[- ]{0,1}[A-Za-zÀ-ÿ]{0,12}$/;
24 var missVille = document.getElementById("missVille");
25 var pays = document.getElementById("pays");
26 var paysValid = /^[A-Za-zÀ-ÿ]{1,13}[- ]{0,1}[A-Za-zÀ-ÿ]{0,12}[ ']{0,1}[A-Za-zÀ-ÿ]{1,13}[- ]{0,1}[A-Za-zÀ-ÿ]{0,12}$/;
27 var missPays = document.getElementById("missPays");
28 var telephone = document.getElementById("telephone");
29 var telephoneValid = /^[\\d+-]{2,50}$/;
30 var missTelephone = document.getElementById("missTelephone");
31 var checkbox = document.getElementById("checkbox");
32 var missCheckbox = document.getElementById("missCheckbox");
33
```



3 : `/^[\u00c1-Z\u00e2-\u00e3]{1,13}` on autorise entre 1 et 13 lettres de l'alphabet (a à z) minuscule ou majuscule, ainsi que les accents.

`[-]{0,1}` on autorise 0 ou 1 espace ou tiret (pour les noms composés)

`[\u00c1-Z\u00e2-\u00e3]{0,12}` on autorise entre 0 ou 12 lettres de l'alphabet minuscule ou majuscule, ainsi que les accents.

`[']{0,1}` on autorise 0 ou une apostrophe.

`[\u00c1-Z\u00e2-\u00e3]{1,13}` on autorise entre 1 et 13 lettres de l'alphabet (a à z) minuscule ou majuscule, ainsi que les accents.

`[-]{0,1}` on autorise 0 ou 1 tiret ou espace.

`[\u00c1-Z\u00e2-\u00e3]{0,12}$`; on autorise entre 0 et 12 lettres de l'alphabet minuscule ou majuscule, ainsi que les accents.

Je suis partie du principe qu'en général un nom non composé ne dépasse pas les 13 lettres. J'ai fait en sorte que le total ne dépasse pas les 50 caractères autorisés. J'ai mis la possibilité qu'il puisse y avoir un tiret, ou un espace ou une apostrophe en cas de nom composé.

9 : `/^[\u00e1-z\u00c1-Z\u00d7]{1,10}$`; on autorise entre 1 à 10 caractères. Les caractères autorisés sont les lettres minuscules et majuscules sans accent, ainsi que les chiffres. J'ai préféré inclure les lettres en cas de code postal international. Et pour la même raison j'ai mis une limite de 10 caractères à ne pas dépasser.

12 : /^[a-z\d]{1,} on autorise uniquement les lettres minuscules sans accent, ainsi que les chiffres. Il peut y avoir au minimum 1 caractère sans limite maximale.

[-]{0,1} on autorise 0 ou 1 point ou tiret.

[a-z\d]{1,} on autorise 1 ou plusieurs lettres minuscules sans accent ou chiffres.

[@]{1} Un seul et unique @ est obligatoire.

[a-z]{1,} on autorise 1 ou plusieurs lettres minuscules sans accent ou chiffres.

[-]{0,1} on autorise 0 ou 1 tiret.

[a-z.]{1,} on autorise 1 ou plusieurs lettres minuscules sans accent ou chiffres.

[.]{1} un point est ici obligatoire.

[a-z]{1,5}\$; on autorise entre 1 et 5 lettres minuscules sans accent.

15 : /^(?=.*[a-z]) on attend à ce qu'il y ait au moins une lettre minuscule.

(?=.*[A-Z]) on attend à ce qu'il y ait au moins une lettre majuscule.

(?=.*\d) on attend à ce qu'il y ait au moins un chiffre.

(?=.*[@\$!%*?&]) on attend à ce qu'il y ait au moins un caractère spécial parmi ceux-ci @\$!%*?&

[A-Za-z\d@\$!%*?&]{8,50}\$; On autorise les lettres minuscules et majuscules, les chiffres et les caractères spéciaux @\$!%*?&, et qu'il y ait au moins 8 caractères et 50 caractères maximum.

20 : /^[a-zA-ZÀ-ÿ\d\s',-]{5,250}\$; on autorise les lettres minuscules, les lettres majuscules, les chiffres, les espaces, les apostrophes, les virgules et les tirets. On attend au minimum 5 caractères, et maximum 250 caractères.

```

public > build > JS app.js > validation
102     if (paysValid.test(pays.value) == false) {
103         event.preventDefault();
104         missPays.textContent = "Pays manquant ou incorrect";
105         missPays.style.color = "red";
106     } else {
107         missPays.textContent = "";
108     }
109
110     if (telephoneValid.test(telephone.value) == false) {
111         event.preventDefault();
112         missTelephone.textContent = "numéro de téléphone manquant ou incorrect";
113         missTelephone.style.color = "red";
114     } else {
115         missTelephone.textContent = "";
116     }
117
118     if (checkbox.validity.valueMissing) {
119         event.preventDefault();
120         missCheckbox.textContent = "Cochez cette case pour valider le formulaire";
121         missCheckbox.style.color = "red";
122
123     }
124     if (checkbox.checked) {
125         missCheckbox.style.display = "none";
126     }
127
128 }
129 formValid.addEventListener("click", validation);

```

Ensuite on crée une fonction validation qui permet de créer des évènements en fonction des conditions.

Pour chaque propriété on crée une condition qui dit que si la propriété n'a pas de valeur ou ne respecte pas la regex, alors un message en rouge s'affiche et annule la validation du formulaire. Si l'utilisateur rentre les bonnes données correctement en respectant les regex, alors aucun message d'erreur ne s'affiche et la validation du formulaire a lieu au moment du clique sur le bouton d'envoi.

Créer vos identifiants

Email

Adresse email manquante ou incorrecte, votre adresse email doit être écrite sans espace avec obligatoirement un @ et un .

Mot de passe

Mot de passe manquant ou incorrect, le mot de passe doit contenir au moins 8 caractère, 1 majuscule, 1 minuscule et 1 caractère spécial (@\$!%*?&)

Confirmation de mot de passe

Créer vos identifiants

Prénom

Prénom

Prénom manquant ou incorrect, écrivez votre prénom en lettres (majuscules, minuscules, espace, ' et - autorisés)

Nom

Nom manquant ou incorrect, écrivez votre nom en lettres (majuscules, minuscules, espace, ' et - autorisés)

Adresse

Adresse manquante ou incorrecte

Ville

Ville manquante ou incorrecte

Code postal

Code postal manquante ou incorrecte

Pays

Pays manquante ou incorrecte

Numéro de téléphone

numéro de téléphone manquante ou incorrect

J'accepte les conditions

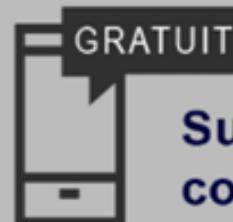
Cochez cette case pour valider le formulaire

Valider

Vos numéros de téléphone

Numéro de Portable

Numéro de téléphone fixe



GRATUIT
Suivez vos commandes par SMS

Rôle et

privilèges

Première méthode :
Je crée d'abord un utilisateur qui aura pour identifiant admin_villagegreen, identifié avec le mot de passe AZEnbv9 ?
On lui attribue tous les privilèges sur toutes les tables de la base de données villagegreen.

```
1 CREATE USER 'admin_villagegreen'@'localhost' IDENTIFIED BY 'AZEnbv9?';
2
3 GRANT ALL PRIVILEGES ON villagegreen.* 
4 TO 'admin_villagegreen'
5 IDENTIFIED BY 'AZEnbv9?'
6
7
8 -- Ne fonctionne qu'avec MySQL 8:
9
10 CREATE ROLE 'r_villagegreen_admin'@'localhost'
11
12
13 GRANT ALL
14 ON villagegreen.*
15 TO 'r_villagegreen_admin'@'localhost'
16
17 GRANT r_villagegreen_admin
18 TO 'admin_villagegreen'@'localhost'
19
20
21
22
23 UPDATE 'user' SET 'roles' = '[\"ROLE_ADMIN\"]'
24 WHERE 'user'. 'email' = 'admin@villagegreen.com';
```

Deuxième méthode (ne fonctionne qu'avec MySQL8) :
Je crée d'abord un rôle, se sera le rôle de l'administrateur. Ensuite j'attribue tous les privilèges au rôle d'administrateur. Ensuite on attribue le rôle à l'utilisateur admin_villagegreen.

Pour finir, dans notre base de données on rentre cette requête SQL :

```
UPDATE 'user' SET 'roles' = '[\"ROLE_ADMIN\"]'
WHERE 'user'. 'email' = 'admin@villagegreen.com';
```

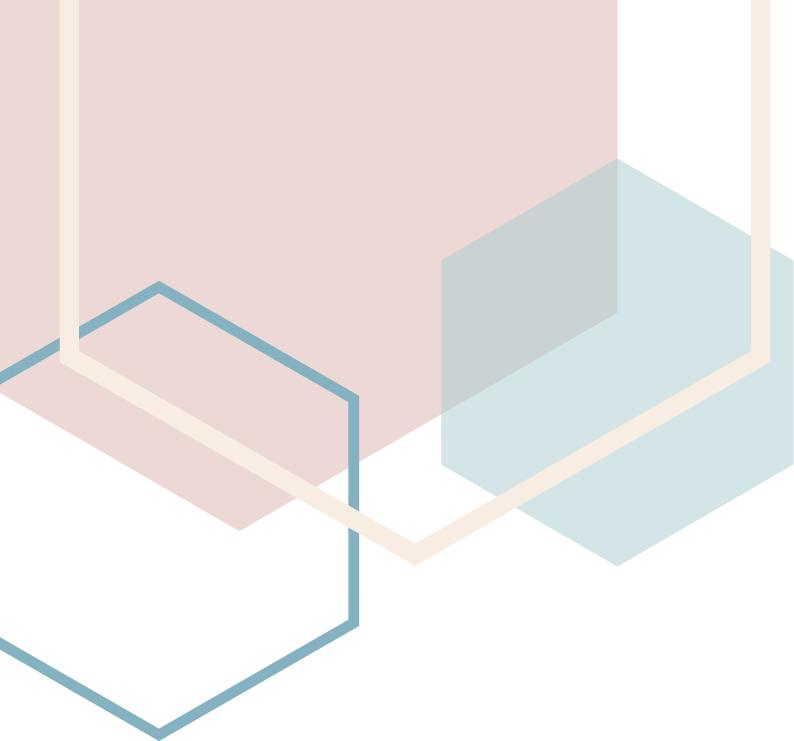
On met à jour le statut de l'utilisateur qui possède comme email « admin@villagegreen.com », on lui donne le rôle de ADMIN.

```

config/packages/!security.yaml
1 security:
2     encoders:
3         App\Entity\User:
4             algorithm: auto
5
6     # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
7     providers:
8         # used to reload user from session & other features (e.g. switch_user)
9         app_user_provider:
10            entity:
11                class: App\Entity\User
12                property: email
13     firewalls:
14         dev:
15             pattern: ^/(_profiler|wdt)|css|images|js/
16             security: false
17         main:
18             anonymous: true
19             lazy: true
20             provider: app_user_provider
21             guard:
22                 authenticators:
23                     - App\Security\LoginFormAuthenticator
24             logout:
25                 path: app_logout
26                 # where to redirect after Logout
27                 # target: app_any_route
28
29             # activate different ways to authenticate
30             # https://symfony.com/doc/current/security.html#firewalls-authentication
31
32             # https://symfony.com/doc/current/security/impersonating_user.html
33             # switch_user: true
34
35     # Easy way to control access for Large sections of your site
36     # Note: Only the *first* access control that matches will be used
37     access_control:
38         - { path: ^/admin, roles: ROLE_ADMIN }
39         - { path: ^/profil, roles: ROLE_USER }

```

Dans `security.yaml`, avec le `access_control` on sécurise l'accès des chemins (path) mentionnés, ainsi pour toutes les pages `/admin` seront accessibles seulement à l'utilisateur qui aura le rôle d'admin.



Easy Admin

EasyAdmin est un bundle qui permet de mettre en place un tableau de bord d'administration. J'ai choisi d'utiliser EasyAdmin pour son côté fonctionnel et non esthétique. Il est aussi rapide et performant, offrant donc une navigation fluide. C'est avec ce bundle que je vais vous montrer la gestion du CRUD sur nos entités Doctrine. EasyAdmin nous offre également la pagination, la recherche, mais aussi un template responsive (qui s'affiche également parfaitement bien sur mobile ou tablette (Annexe 4).

CRUD:

Le CRUD désigne les quatre opérations Create Read Update et Delete.

La traduction pour mieux comprendre :

Create : Créer, Ajouter (SQL : INSERT ; http : POST)

Read : Lire, Voir (SQL : SELECT ; http: GET)

Update : Mettre à jour, Modifier (SQL : UPDATE ; http : PUT)

Delete : Effacer, Supprimer (SQL : DELETE ; http : DELETE)

Le CRUD avec EasyAdmin est géré grâce au getter et au setter de nos entités, mais aussi grâce à son AbstractCrudController. Le CRUD va faire en sorte que les visiteurs puissent voir les produits sur le site internet, et que l'administrateur du site puisse ajouter, modifier ou supprimer des produits dans le tableau de bord d'administration.

```
src > Controller > Admin > 🏠 DashboardController.php > 📁 DashboardController > ⚙️ configureMenuItems
1  <?php
2
3  namespace App\Controller\Admin;
4
5  use App\Entity\Product;
6  use App\Entity\Category;
7  use App\Entity\Subcategory;
8  use App\Entity\Order;
9  use App\Entity\OrderDetails;
10 use App\Entity\User;
11 use App\Entity\Supplier;
12 use App\Entity\Team;
13 use App\Entity\CommercialDepartment;
14 use EasyCorp\Bundle\EasyAdminBundle\Config\Dashboard;
15 use EasyCorp\Bundle\EasyAdminBundle\Config\MenuItem;
16 use EasyCorp\Bundle\EasyAdminBundle\Controller\AbstractDashboardController;
17 use Symfony\Component\HttpFoundation\Response;
18 use Symfony\Component\Routing\Annotation\Route;
19
20 class DashboardController extends AbstractDashboardController
21 {
22     /**
23      * @Route("/admin", name="admin")
24      */
25     public function index(): Response
26     {
27         return $this->render('admin/dashboard.html.twig');
28     }
29
30     public function configureDashboard(): Dashboard
31     {
32         return Dashboard::new()
33             ->setTitle('Villagegreen');
34     }
35
36     public function configureMenuItems(): iterable
37     {
38         return [
39             MenuItem::linkToDashboard('Dashboard', 'fa fa-home'),
40             MenuItem::section('Produits'),
41             MenuItem::linkToCrud('Produits', 'fa fa-guitar', Product::class),
42             MenuItem::linkToCrud('Sous-catégories', 'fa fa-folder-open', Subcategory::class),
43             MenuItem::linkToCrud('Catégories', 'fa fa-folder', Category::class),
44             MenuItem::section('Commandes'),
45             MenuItem::linkToCrud('Commandes', 'fa fa-shopping-cart', Order::class),
46             MenuItem::linkToCrud('Détails de commande', 'fa fa-list-alt', OrderDetails::class),
47             MenuItem::section('Comptes'),
48             MenuItem::linkToCrud('Utilisateurs', 'fa fa-user', User::class),
49             MenuItem::linkToCrud('Fournisseurs', 'fa fa-truck', Supplier::class),
50             MenuItem::linkToCrud('Equipe', 'fa fa-users', Team::class),
51             MenuItem::linkToCrud('Département Commercial', 'fa fa-user-tie', CommercialDepartment::class),
52
53         ];
54     }
55
56 }
57 }
```

30 à 34 : le nom du Dashboard sera Villagegreen.

36 à 54 : on crée une fonction qui va permettre de configurer le menu du Dashboard (tableau de bord d'administration).

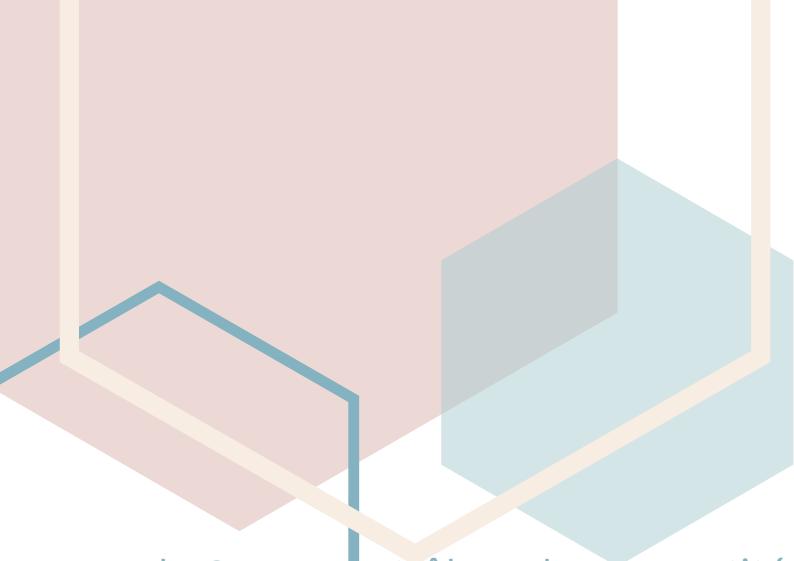
39 : on aura comme libellé « Dashboard » avec un logo en forme de maison 'fa fa-home' (Font Awesome). Lorsqu'on cliquera sur Dashboard, ça nous renverra directement à la page d'accueil du Dashboard.

40 : on crée une section « Produits » qui sera un bloc qui contiendra les produits, les sous-catégories et les catégories.

41 à 43 : linkToCrud permet de se connecter au CRUD de l'entité qui se trouve avant « ::class ». Grâce à ça, on va pouvoir ajouter, modifier ou supprimer les données de la table.

Villagegreen		Product								Actions	
		Search								Add Product	
		Name	Short Wording	Reference	Brand	Price	Stock	Stock Alert	Long Wording	Subcategory	Picture
<input type="checkbox"/>	SAS-75 Alto	Saxophone	188679	Startone	620	24	5	View content	Saxophone		Edit Delete
<input type="checkbox"/>	F310 Natural	guitare acoustique	14110	Yamaha	133.99	87	5	View content	Guitare acoustique		Edit Delete
<input type="checkbox"/>	C80	guitare classique	20966	Yamaha	204.99	76	5	View content	Guitare classique		Edit Delete
<input type="checkbox"/>	RS420 Fired Red	guitare électrique	305487	Yamaha	575.99	23	5	View content	Guitare électrique		Edit Delete
<input type="checkbox"/>	Go : Piano	clavier arrangeur	238197	Roland	279.99	65	5	View content	Clavier arrangeur		Edit Delete
<input type="checkbox"/>	A-88 MK2	clavier maître	312956	Roland	949.99	34	5	View content	Clavier maître		Edit Delete
<input type="checkbox"/>	Genos	clavier arrangeur	255494	Yamaha	3,798	12	5	View content	Clavier arrangeur		Edit Delete
<input type="checkbox"/>	Stage Custom Birch Natural Wood	batterie acoustique	184330	Yamaha	1,025.99	14	5	View content	Batterie acoustique		Edit Delete
<input type="checkbox"/>	DTX6K-X	batterie électronique	341155	Yamaha	988	25	5	View content	Batterie électronique		Edit Delete

```
src > Controller > Admin > 🐘 ProductCrudController.php > ...
1  <?php
2
3  namespace App\Controller\Admin;
4
5  use App\Entity\Product;
6  use EasyCorp\Bundle\EasyAdminBundle\Field\IdField;
7  use EasyCorp\Bundle\EasyAdminBundle\Field\TextField;
8  use EasyCorp\Bundle\EasyAdminBundle\Field\ImageField;
9  use EasyCorp\Bundle\EasyAdminBundle\Field\NumberField;
10 | use EasyCorp\Bundle\EasyAdminBundle\Field\TextEditorField;
11 | use EasyCorp\Bundle\EasyAdminBundle\Field\AssociationField;
12 | use EasyCorp\Bundle\EasyAdminBundle\Form\Type\FileUploadType;
13 | use EasyCorp\Bundle\EasyAdminBundle\Controller\AbstractCrudController;
14
15 class ProductCrudController extends AbstractCrudController
16 {
17     public static function getEntityFqcn(): string
18     {
19         return Product::class;
20     }
21
22
23     public function configureFields(string $pageName): iterable
24     {
25         return [
26             // IdField::new('id'),
27             TextField::new('name'),
28             TextField::new('short_wording'),
29             TextField::new('reference'),
30             TextField::new('brand'),
31             NumberField::new('price'),
32             NumberField::new('stock'),
33             NumberField::new('stock_alert'),
34             TextEditorField::new('long_wording'),
35             AssociationField::new('subcategory'),
36             ImageField::new('picture')
37                 ->setBasePath('/uploads/picture/')
38                 ->setUploadDir('public/uploads/picture')
39                 ->setFormType(FileUploadType::class)
40                 ->setUploadedFileNamePattern('[randomhash].[extension]')
41                 ->setRequired(false),
42
43         ];
44     }
45
46 }
```



Pour le CRUD contrôleur de mon entité Produit (ProductCrudController), on crée une fonction qui permet de configurer les propriétés (23 à 44) dans la classe ProductCrudController.

27 à 30 : les propriétés name, short_wording, reference et brand seront à remplir dans des champs de type text.

31 à 33 : à l'inverse, les propriétés price, stock et stock_alert seront à remplir dans des champs de type number (car ce sont des nombres).

34 : la propriété long_wording, qui est la description du produit, aura un champ de type éditeur de texte.

35 : l'AssociationField va permettre de récupérer la sous-catégorie (qui se trouve dans une autre Entité avec laquelle l'entité Product est associée), on va pouvoir associer une sous-catégorie à un produit.

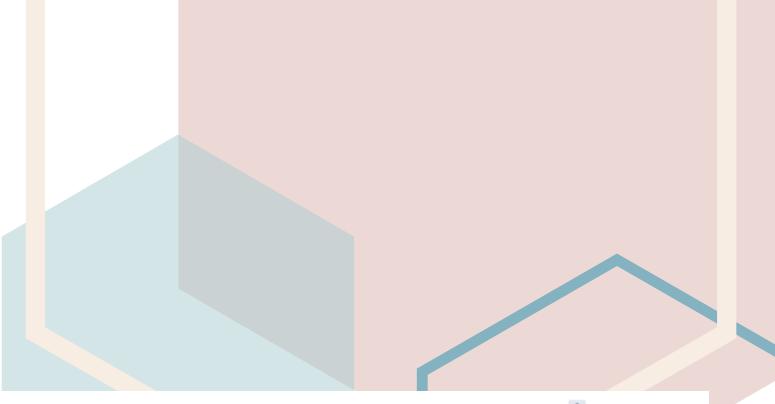
Pour que cela puisse fonctionner, il ne faut pas oublier de rajouter cette fonction dans la classe Subcategory :

```
public function __toString()
{
    return $this->name;
}
```

Afin que le contrôleur sache quoi retourner comme réponse au moment de la création ou de la modification d'un produit.

36 à 41: pour picture, j'ai décidé de créer un uploader d'images. Lorsque l'admin va cliquer sur Picture, il aura accès aux fichiers qui se trouvent sur son ordinateur. Il va donc pouvoir sélectionner l'image qu'il souhaite ajouter au produit et l'image sera directement stockée dans le chemin public/uploads/picture.

Avec setUploadedFileNamePattern on va renommer chacune des photos qui seront uploadées, les noms seront hashés. Pour finir, la photo ne sera pas obligatoire pour le produit.



Villagegreen

Dashboard

PRODUITS

- Produits
- Sous-catégories
- Catégories

COMMANDES

- Commandes
- Détails de commande

COMPTEES

- Utilisateurs
- Fournisseurs
- Équipe
- Département Commercial

Create Product

Name *

Short Wording *

Reference *

Brand *

Price *

Stock *

Stock Alert *

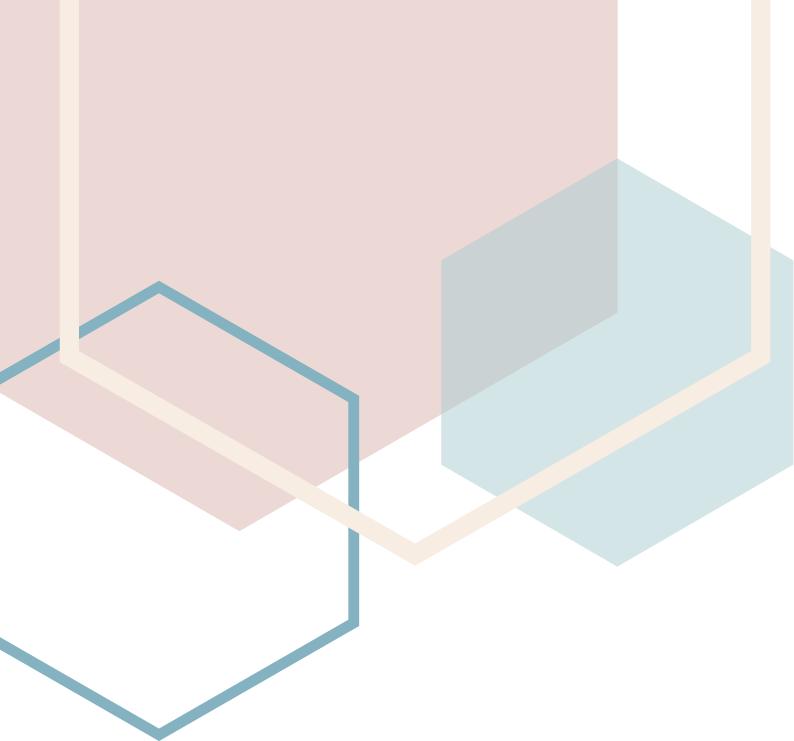
Long Wording

Subcategory

Picture

Les produits sont donc rentrés correctement dans la base de données avec les noms d'images hashés, et l'id de la sous-catégorie associée pour chaque produit :

<input type="checkbox"/>		Éditer		Copier		Supprimer	1	SAS-75 Alto	Saxophone	404544905849c8c8750479801403548cb5e65773.jpeg	188679	Startone	620.00	24
<input type="checkbox"/>		Éditer		Copier		Supprimer	2	F310 Natural	guitare acoustique	1d6a9f1a0a2ee7067be9b12526cdf641b95dbc85.jpeg	14110	Yamaha	133.99	87
<input type="checkbox"/>		Éditer		Copier		Supprimer	3	C80	guitare classique	762949c31249e1ad6f63d78679f0e96d145437c9.jpeg	20966	Yamaha	204.99	76
<input type="checkbox"/>		Éditer		Copier		Supprimer	4	RS420 Fired Red	guitare électrique	2534fd20d7f5c94b8fe3d587bec2c957483abc77.jpeg	305487	Yamaha	575.99	23
<input type="checkbox"/>		Éditer		Copier		Supprimer	5	Go : Piano	clavier arrangeur	ad080784a2ea2364ab193954e547de7b034b2c79.jpeg	238197	Roland	279.99	65
<input type="checkbox"/>		Éditer		Copier		Supprimer	6	A-88 MK2	clavier maître	99178696ff678a44359a8f1ed0f99d55028335eb.jpeg	312956	Roland	949.99	34
<input type="checkbox"/>		Éditer		Copier		Supprimer	7	Genos	clavier arrangeur	2ae4fc32f1bc5f30fd34acc34aa37091bbfe134c.jpeg	255494	Yamaha	3798.00	12
<input type="checkbox"/>		Éditer		Copier		Supprimer	8	Stage Custom Birch Natural Wood	batterie acoustique	9e155679f7294ff9576d17a90c0a9eb241277d3b.jpeg	184330	Yamaha	1025.99	14
<input type="checkbox"/>		Éditer		Copier		Supprimer	9	DTX6K-X	batterie électronique	96c43de0914c781046c5482f6d020596f9fc0f25.jpeg	341155	Yamaha	988.00	25
<input type="checkbox"/>		Éditer		Copier		Supprimer	10	X1832USB	console de mixage	33ee5f30ee684c04ed38c08a013ad11a1e8f842e.jpeg	80807	Behringer	221.00	36
<input type="checkbox"/>		Éditer		Copier		Supprimer	11	B115D	enceinte de façade active	82ef690d17b5f62961a9e97d1d4fbcd7ea626ac.jpeg	181760	Behringer	221.99	24
<input type="checkbox"/>		Éditer		Copier		Supprimer	12	CBR15	enceinte de façade passive	251d92da0b0574cb018daa9e751940364ca9a912.jpeg	189624	Yamaha	338.99	65
<input type="checkbox"/>		Éditer		Copier		Supprimer	13	FT Case T300	flight case	f4270ee30075a35531b404c86d012b361b19265c.jpeg	341488	Power Flightcase	299.99	45



Front

La partie front de mon projet fonctionne avec Twig, qui est le moteur de template PHP utilisé par défaut avec Symfony.

On peut compiler les templates grâce à un système d'héritage.

La syntaxe de Twig :

·{{ ... }} : fait appel à un template parent, ou à une variable ou à une fonction PHP.

·#{ ... #} : à l'intérieur se trouvent les commentaires.

·{% ... %} : représente en général les boucles, les conditions et les commandes.

J'utilise également le langage HTML pour la structure de nos codes templates. Pour personnaliser le site et le rendre responsive, je me sers de bootstrap et du CSS.

Architecture d'un document HTML :

```
<!DOCTYPE html>
  <html>
    <head>
      <meta charset="utf-8">
      <title>Titre de la page</title>
      <link rel="stylesheet" href="style.css">
      <script src="script.js"></script>
    </head>
    <body>
      <p>Le contenu de la page !</p>
    </body>
  </html>
```

C'est dans le fichier base.html.twig que l'on va mettre le header et le footer qui seront compilés dans les autres templates du site.

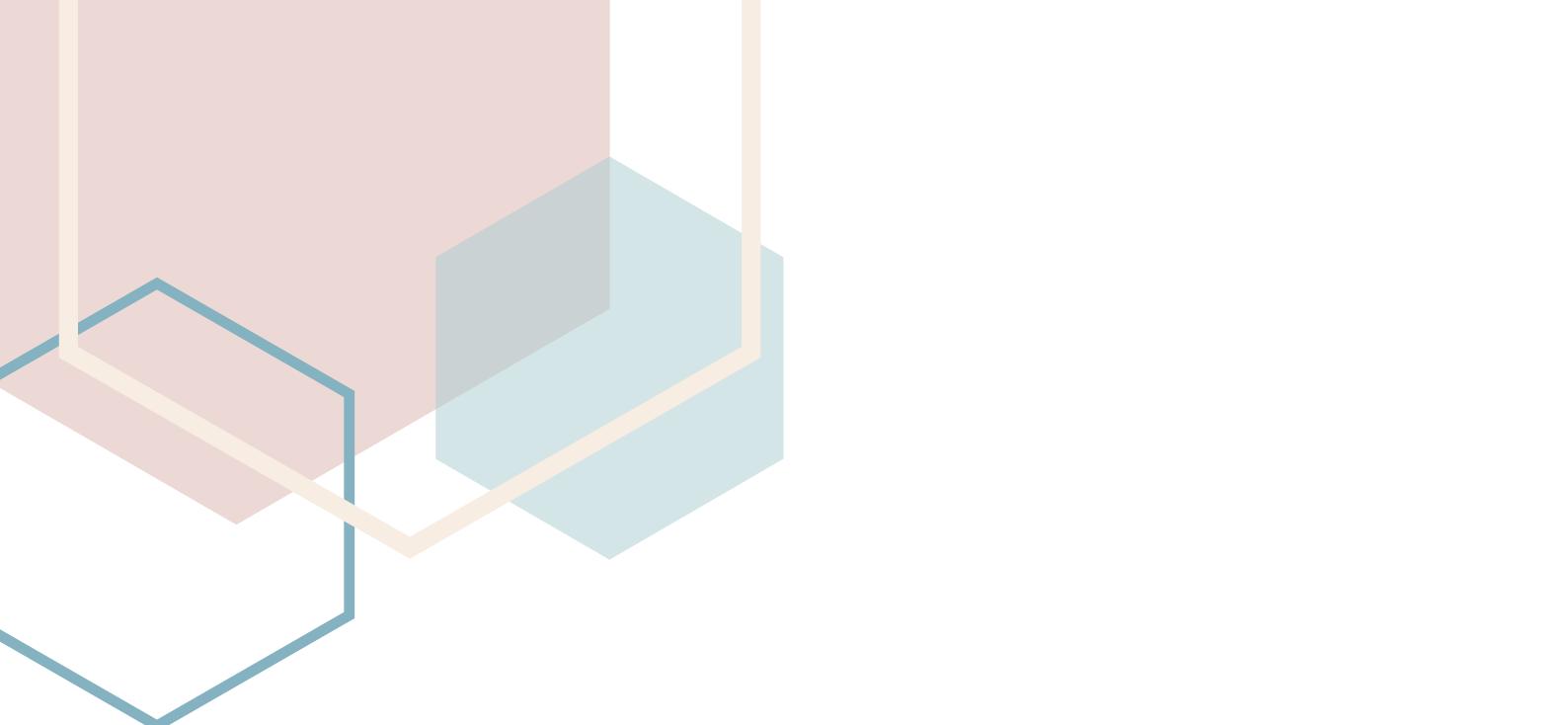
```
templates > / base.html.twig
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5           [# Run "composer require symfony/webpack-encore-bundle"
6              and uncomment the following Encore helpers to start using Symfony UX #]<head>
7          <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
8
9
10         <meta charset=" utf-8 ">
11         <meta http-equiv="X-UA-Compatible" content="IE=edge">
12         <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
13         <title>{% block title %}VillageGreen{% endblock %}</title>
14         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgJLIm9Nao0Yz1zcQtTwFspd3yD65VohhpuuCOMLASJC" crossorigin="anonymous">
15
16     {% block stylesheets %}<br/>
17
18     {% endblock %}<br/>
19     <link rel="stylesheet" href="{{ asset('build/app.css') }}>
20 </head>
21 <body>
22
23
24     <div class="container" name="background">
25         <header>
26             <nav class="navbar d-xl-none navbar-dark bg-secondary">
27                 <a href="/" class="navbar-left"></a>
28                 <a class="navbar-brand" href="#"></a>
29                 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
30                     <span class="navbar-toggler-icon"></span>
31                 </button>
32                 <div class="collapse navbar-collapse" id="navbarSupportedContent">
33                     <ul class="navbar-nav mr-auto">
34                         <li class="nav-item">
35                             
36                             </li>
37                         <li class="nav-item"><a class="nav-link" href="#">Infos</a></li>
38                         <li class="nav-item"><a class="nav-link" href="#">Espace client</a></li>
39                         <li class="nav-item dropdown">
40                             <a class="nav-link dropdown-toggle" href="#" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false" role="button">Menu</a>
41                             <div class="dropdown-menu">
42                                 <a class="dropdown-item" href="#">Produits</a>
43                                 <a class="dropdown-item" href="#">Service</a>
44                                 <a class="dropdown-item" href="#">Aide</a>
45                                 <a class="dropdown-item" href="#">À propos</a>
46                             </div>
47                         </li>

```

14 à 20 : dans la balise <head> on met le lien bootstrap, ainsi que le lien de notre feuille css grâce à asset qui est un composant Symfony pour joindre les feuilles de style css et javascript, ainsi que les images se trouvant également dans le dossier public.

26 à 63 : création d'une navbar qui sera visible sur tous les formats sauf les écrans XL (en gros c'est la navbar créée pour la partie responsive). La classe navbar-dark est ici pour dire que la navbar est foncée, et donc l'écriture sera automatiquement plus claire. La classe bg-secondary représente la couleur de background (de fond) de la navbar. Ici elle sera grise.

Pour la partie « responsive », j'ai décidé de mettre pour la navbar un menu hamburger avec des dropdown à l'intérieur. Le menu hamburger consiste à cliquer sur un bouton pour afficher le menu. Pour ne pas que le menu soit trop long, j'ai décidé que dans mon menu hamburger il y ait des dropdown, c'est-à-dire des sous-menus qui s'affichent seulement si on clique sur le libellé du menu. Par exemple, en cliquant sur le bouton du menu, on voit apparaître différents libellés tel que « Catégories ». En cliquant sur Catégories, on verra donc apparaître en dessous le nom des catégories comme Guitare, Batterie, Clavier, ou encore Studio.



```
templates > base.html.twig
48
49     <li class="nav-item dropdown">
50         <a class="nav-link dropdown-toggle" href="#" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false" role="button">Catégories</a>
51         <div class="dropdown-menu">
52             <a class="dropdown-item" href="/guitare">Guitare</a>
53             <a class="dropdown-item" href="/batterie">Batterie</a>
54             <a class="dropdown-item" href="/clavier">Clavier</a>
55             <a class="dropdown-item" href="/studio">Studio</a>
56             <a class="dropdown-item" href="/sono">Sono</a>
57             <a class="dropdown-item" href="/eclairage">Eclairage</a>
58             <a class="dropdown-item" href="/dj">DJ</a>
59             <a class="dropdown-item" href="/cases">Cases</a>
60             <a class="dropdown-item" href="/accessoires">Accessoires</a>
61         </div>
62     </li>
63 </ul>
</nav>
```

La balise `` veut dire Unordered List (liste désordonnée), et la balise `` veut dire List Item (un item de liste). Dans une liste de courses, la farine serait un item de liste, tout comme les œufs ou le beurre. Et la liste désordonnée serait la liste entière de courses comprenant la farine, le beurre et les œufs.

La balise `<a>` sert à créer un lien hypertexte, et est toujours accompagné de l'attribut `href` qui donne la direction du lien.

Voici le résultat pour la partie responsive :

The screenshot shows a mobile-optimized version of the Village Green website. At the top left is the logo 'village green'. To its right are language and currency options ('Fr' and '€') and a shopping cart icon. A menu icon (three horizontal lines) is in the top right corner. Below these are links for 'Infos', 'Espace client', and a dropdown menu labeled 'Menu ▾' containing 'Produits', 'Service', 'Aide', and 'A propos'. A 'Catégories ▾' section follows. At the bottom is a promotional banner for free delivery: 'LIVRAISON GRATUITE EN FRANCE METROPOLITaine & BELGIQUE' with '19 euros dès d'achat' and a button 'CLIQUEZ ICI POUR PLUS D'INFORMATION'.

```

templates > base.html.twig
64         <div class="d-none d-xl-block d-flex container-fluid" >
65
66             <div class="">
67                 <a href="/"></a>
68             </div>
69
70             <div class="">
71
72                 <nav class="navbar">
73                     <a class="navzero" href="/inscription">Inscription</a>
74                     <a class="navone" href="/">Infos</a>
75                     <a class="navtwo" href="/login">Espace client</a>
76                     <div class="">
77                         
78                         
79                     </div>
80                 </nav>
81
82                 <nav class="secondary_bar">
83                     <a class="secondary " href="/produits">Produits</a>
84                     <a class="secondary " href="/">Service</a>
85                     <a class="secondary " href="/">Aide</a>
86                     <a class="secondary " href="/">A propos</a>
87                 </nav>
88
89                 <nav class="dark-navbar bg-dark">
90                     <a class="dark" href="Guitare.html">Guitare</a>
91                     <a class="dark" href="Batterie.html">Batterie</a>
92                     <a class="dark" href="Clavier.html">Clavier</a>
93                     <a class="dark" href="Studio.html">Studio</a>
94                     <a class="dark" href="Sono.html">Sono</a>
95                     <a class="dark" href="Eclairage.html">Eclairage</a>
96                     <a class="dark" href="Dj.html">DJ</a>
97                     <a class="dark" href="Cases.html">Cases</a>
98                     <a class="dark" href="Accessoires.html">Accessoires</a>
99                 </nav>
100            </div>
101
102        </div>
103    </header>
104    {% block body %}
105
106    {% endblock %}

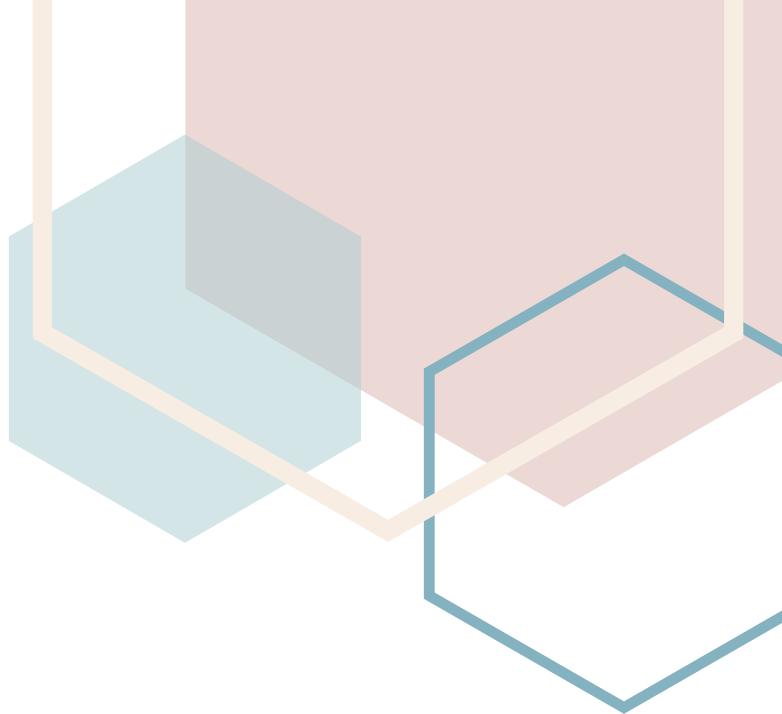
```

64 à 102 : Après la fermeture de la balise nav, on ouvre une nouvelle balise <div> où l'on demande à ce que ce bloc soit affiché seulement pour les écrans XL (d-none d-xl-block).

Dans ce bloc j'ai donc créé deux div : une première div avec le logo, et une deuxième div avec les trois navbars.

Le but est de positionner le logo sur les trois navbars

```
67 .img_logo{  
68     width:400px;  
69     margin-top: -250px;  
70     top: 290px;  
71     margin-right: -4px;  
72     position: relative;  
73     left: 0px;  
74     z-index: 1;  
75     height: 250px;  
76 }
```



Pour ça nous allons lui donner comme position la position relative. On va le positionner sur la hauteur grâce à margin-top (crée une marge vers le haut) et top (déplace vers le bas). On fait ensuite la même chose avec margin-right et left pour le positionner sur la longueur. Ensuite avec width et height, on règle sa largeur et sa grandeur. Pour finir, z-index va permettre d'empiler le logo par-dessus les trois navbars.

Pour la troisième navbar, on a réglé les padding (l'écart de remplissage intérieur), ainsi que la couleur. Pour les noms des catégories, l'écriture sera de couleur blanche sans effet, on définit également la taille, mais aussi la marge de gauche (pour laisser la place au logo).

```
138  
139 ↘ .dark_navbar{  
140     padding-left: 355px;  
141     padding-top: 13px;  
142     padding-bottom: 12px;  
143     color: #333333;  
144 }  
145  
146 ↘ .dark{  
147  
148     margin-left: 30px;  
149     font-size: 22px;  
150     color: white;  
151     text-decoration: none;  
152 }
```

Résultat:

Village Green

Inscription Infos Espace client Fr

Produits Service Aide A propos

Guitare Batterie Clavier Studio Sono Eclairage DJ Cases Accessoires

LIVRAISON GRATUITE
EN FRANCE MÉTROPOLITaine & BELGIQUE

19 euros dès d'achat

CLIQUEZ ICI POUR PLUS D'INFORMATION

Nos catégories

Nos meilleures ventes

Nos partenaires

Recevez nos offres exclusives

Votre adresse e-mail Valider

Suivez-nous par ici !

Chronopost
PayPal

Contactez-nous !

Conseil / Commande téléphone
du lundi au vendredi de 8h à 18h,
le samedi de 10h à 18h.
Depuis la France : **02 40 38 50 50**
Belgique, Suisse, International : **0033 2 40 38 50 50**

Service après vente :
02 51 80 68 76
Contactez-nous par téléphone
du lundi au samedi de 9h à 18h
ou depuis votre compte client.

Conseils pour choisir un instrument :
infog@villagegreen.com

Service Presse :
contact@villagegreen.com

Village Green Recrute !

Village Green Stores
Ouverts de 10h à 19h30 non-stop.

Guitares - Amplificateurs - Effets :
182 avenue Jean-Jaurès
75019 Paris

Clavier - Home Studio - Sonorisation
Équipement DJ - Eclairage
184 avenue Jean-Jaurès
75019 Paris

Librairie Musicale :
7 av. du Nouveau Conservatoire
75019 Paris

Instrument à vent :
9/11 av. du Nouveau Conservatoire
75019 Paris

Percussions :
13/15 av. du Nouveau Conservatoire
75019 Paris

Métro Ligne 5, station Porte de Pantin.

Plan d'accès / Visite virtuelle

Utilisés !

Qui sommes nous ?
F.A.Q.
Le blog Village Green

Vous avez un site Internet ?
Devenir partenaire Village Green

Conditions générales de vente
Mentions légales

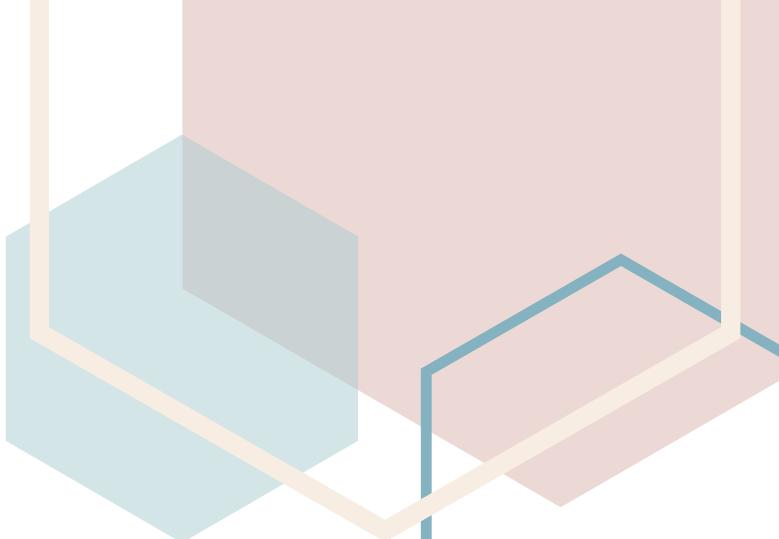
Plan du site
Partenage
Nouveautés
Assurance Woodbrass.com
Location d'instruments de musique

Village Green Preservation Society

Toute l'actualité musicale
Voir l'avis des musiciens
Concours : inscription et résultat... à vous de jouer !

p.49

Page home:

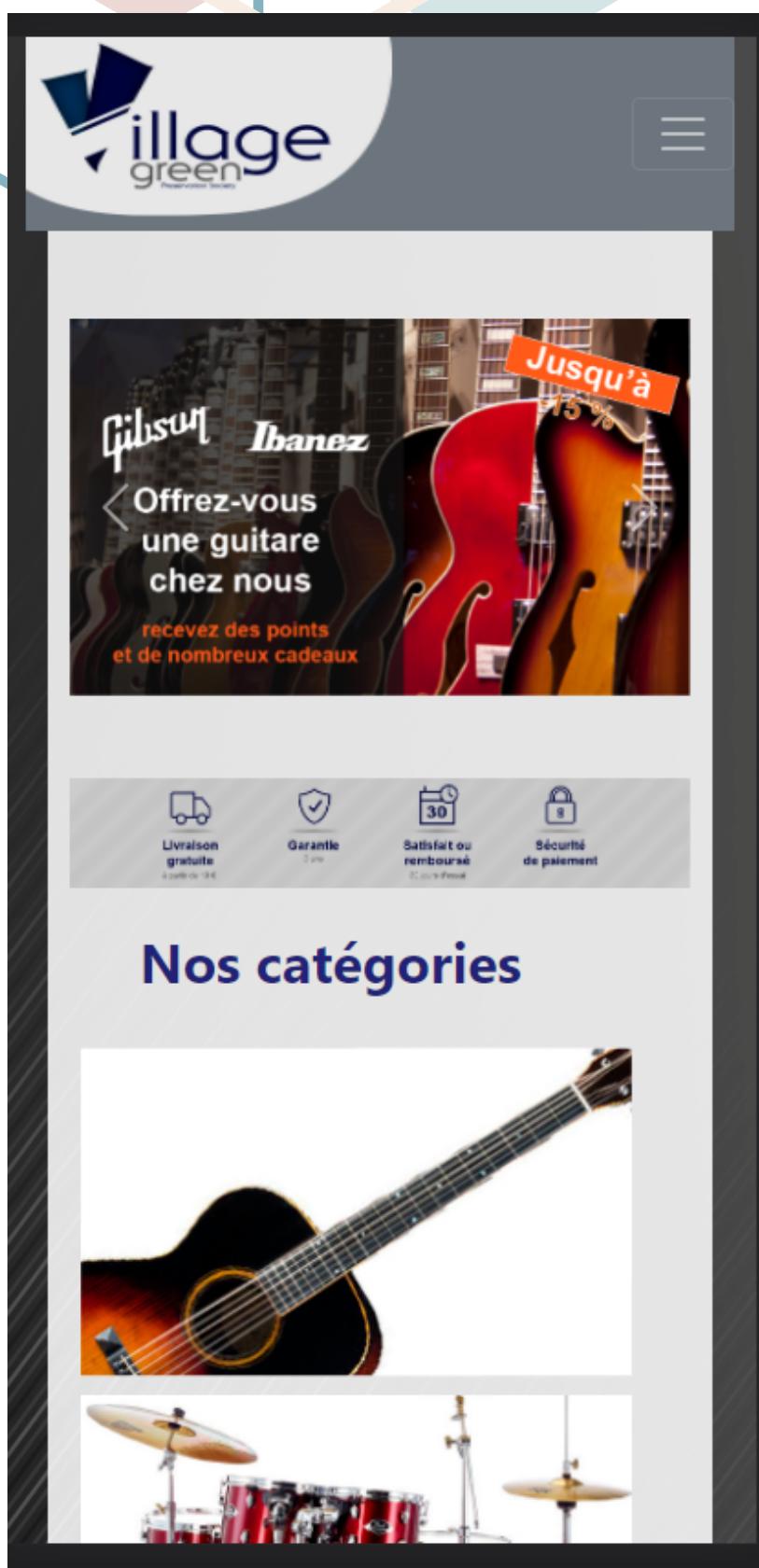


```
templates > villagegreen > home.html.twig
1  {% extends 'base.html.twig' %}
2
3  {% block body %}
4      <article class="pt-5">
5          <div class="d-none d-xl-block container-fluid">
6              <a href="#"></a>
7              <a href="#"></a>
8          </div>
9          <div class="d-xl-none container-fluid">
10             <div id="carouselExampleIndicators" class="carousel slide" data-ride="carousel">
11                 <ol class="carousel-indicators">
12                     <li data-target="#carouselExampleIndicators" data-slide-to="0" class="active"></li>
13                     <li data-target="#carouselExampleIndicators" data-slide-to="1"></li>
14                     <li data-target="#carouselExampleIndicators" data-slide-to="2"></li>
15                 </ol>
16                 <div class="carousel-inner">
17                     <div class="carousel-item active">
18                         
19                     </div>
20                     <div class="carousel-item">
21                         
22                     </div>
23                 </div>
24                 <a class="carousel-control-prev" href="#carouselExampleIndicators" role="button" data-slide="prev">
25                     <span class="carousel-control-prev-icon" aria-hidden="true"></span>
26                     <span class="sr-only"></span>
27                 </a>
28                 <a class="carousel-control-next" href="#carouselExampleIndicators" role="button" data-slide="next">
29                     <span class="carousel-control-next-icon" aria-hidden="true"></span>
30                     <span class="sr-only"></span>
31                 </a>
32             </div>
33         <br>
34     </div>
```

Pour la partie responsive de la page home, j'ai décidé de recréer l'image avec écrit « LIVRAISON GRATUITE » sur le canva, pour qu'elle soit de la même taille que l'image de promotion, afin de les faire défiler avec un carousel. Je trouve que le carousel fait plus présentable que de mettre les deux images à la suite. Le carousel permet de faire défiler les images de gauche à droite comme un diaporama.

Pour rendre le site responsive, j'utilise les class container-fluid et img-fluid qui font en sorte d'adapter le contenu à la taille de l'écran.

Résultat:



(Annexe 5)

Page Produits:

```
templates > villagegreen > produits.html.twig
1  {% extends 'base.html.twig' %}
2
3  {% block title %}Produits{% endblock %}
4
5  {% block body %}
6  <br>
7  <br>
8  <br>
9  <section class="articles row g-3">
10
11     {% for product in products %}
12
13
14
15     <br>
16     <div class=" card card-fluid col-5 rounded-3 border-5 article-produits">
17         
19                 <h2 class="card-title">{{ product.name }}</h2>
20
21                 <p class="card-text">{{ product.shortwording | raw }}</p>
22                 <h3 class="card-text prix">{{product.price}}€</h3>
23                 <a href="{{ path('show', {'id' : product.id}) }}" class="btn btn-submit">Lire la suite</a>
24             </div>
25         </div>
26
27
28     {% endfor %}
29     <br>
30     <br>
31     <br>
32     <br>
33     <br>
34 </section>
35 {% endblock %}
```

Dans la page où l'on affichera tous nos produits, on crée une boucle qui affichera un certain template pour chaque produit. Chaque produit se trouvera dans une carte aux bords arrondis avec des bordures grises autour. On récupère les informations de la base de données grâce à Twig. Et à la fin, on crée un bouton qui donne le chemin de l'article affiché (show), grâce à l'id associé qui a été récupéré dans la base de données avec le contrôleur.

Résultat:

Village green

Inscription Infos Espace client Fr

Produits Service Aide A propos

Guitare Batterie Clavier Studio Sono Eclairage DJ Cases Accessoires

SAS-75 Alto
Saxophone
620.00€
[Lire la suite](#)

F310 Natural
guitare acoustique
133.99€
[Lire la suite](#)

C80
guitare classique
204.99€
[Lire la suite](#)

RS420 Fired Red
guitare électrique
575.99€
[Lire la suite](#)

Go : Piano
clavier arrangeur
279.99€
[Lire la suite](#)

A-88 MK2
clavier maître
949.99€
[Lire la suite](#)

Genos
clavier arrangeur
2700.00€
[Lire la suite](#)

Stage Custom Birch Natural Wood
batterie acoustique
Livré sans caisse claire, sans cymbale et sans hardware Without snare, cymbal and hardware



Inscription Infos Espace client



Produits

Service

Aide

A propos

Guitare Batterie Clavier Studio Sono Eclairage DJ Cases Accessoires

C80 Yamaha



204.99€

Description:

Un concentré de qualités Le modèle C80 présente un ensemble de qualités rares dans cette catégorie de prix : fabrication irréprochable, confort de jeu assuré et sonorité agréable et bien équilibrée. L'utilisation de bois sélectionnés (épicéa, Nato, Palissandre) garantit une utilisation sereine durant de longues années. Le vernis brillant rehausse la beauté de l'instrument, tout comme les mécaniques dorées. Table: épicéa Dos et éclisses: Nato Manche: Nato Touche: Palissandre Chèvrelot: Palissandre Profondeur de caisse: 94-100mm Largeur sillet: 52mm Diapason: 650mm Mécaniques: Doré (YTM06) Coloris: Naturel Finition: Vernis Brillant

[Ajouter au panier](#)

Recevez nos offres exclusives

Votre adresse e-mail

Valider

Suivez-nous par ici !



1

EURO

.COM

Contactez-nous !

Conseil / Commande téléphone
du lundi au vendredi de 8h à 19h,
le samedi de 10h à 18h.

Depuis la France :
02 40 38 50 50

Belgique, Suisse, International :
0033 2 40 38 50 50

Service après vente :

02 51 80 68 76
Contactez-nous par téléphone
du lundi au samedi de 9h à 18h
ou depuis votre compte client.

**Conseils pour choisir
un instrument :**
infovg@villagegreen.com

Service Presse :
contact@villagegreen.com

Village Green Recrute !

Village Green Stores

Ouverts de 10h à 19h30 non-stop.

Guitares - Amplificateurs - Effets :
182 avenue Jean Jaurès
75019 Paris

Clavier - Home Studio - Sonorisation
Équipement DJ - Éclairage :
184 avenue Jean Jaurès
75019 Paris

Librairie Musicale :
7 av. du Nouveau Conservatoire
75019 Paris

Instruments à vent :
9/11 av. du Nouveau Conservatoire
75019 Paris

Percussions :
13/15 av. du Nouveau Conservatoire
75019 Paris

Métro Ligne 5, station Porte de Pantin.

Plan d'accès / Visite virtuelle

Utiles !

Qui sommes nous ?

F.A.Q.

Le blog Village Green

Vous avez un site Internet ?

Devenez partenaire Village Green

Conditions générales de vente

Mentions légales

Plan du site

Parrainage

Nouveautés

Assurance Woodbrass.com

Location d'instruments de musique

Village Green Preservation Society

Toute l'actualité musicale

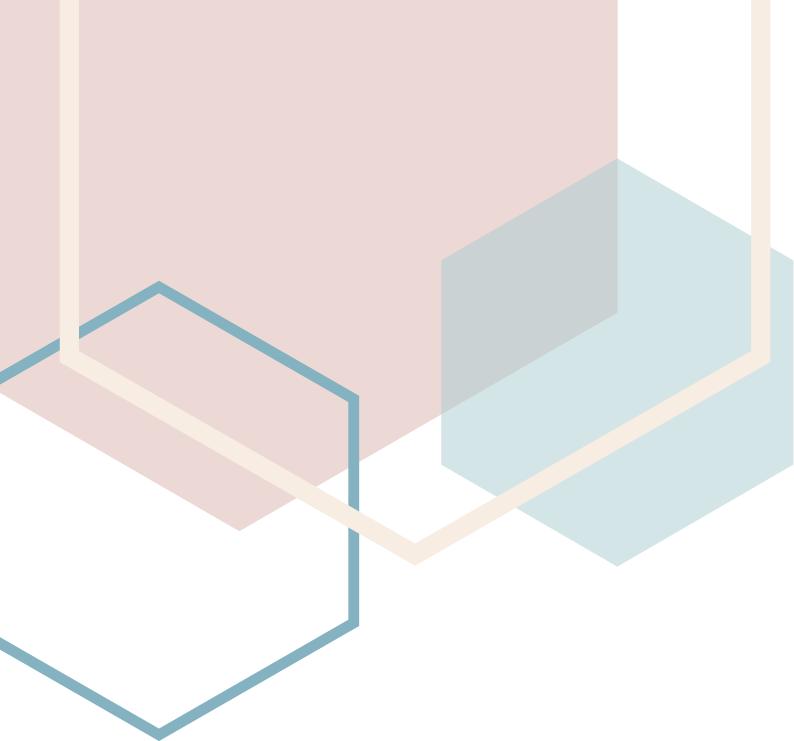
Voir l'avis des musiciens

Concours : inscription et résultat... à vous de jouer !

Village Green est une entreprise 100% Made in France !
Magasin de musique - Achat / Vente instruments de musique - Location instruments de musique - Atelier de réparation - Vente accessoires de musique au meilleur prix.
1969-2016 - RCS Paris B 222 333 444 Déclaration CNIL : 12345678.

Guitares - Amplis / Effets - Claviers / Pianos - Synths - Batteries / Percussions - Vents - Violons - Home Studio - Sonorisation - Deejay - Éclairage - Micros - Casques - Enregistreurs - Litestyle - Déstockage - Ecoles & Pros

Behringer - Boss - Conservatoire de musique de Paris - Fender - Focusrite Scarlett - Focusrite - Gibson 2016 - Gibson - Guitar Shop - Guitare Martin - Guitare Taylor - Guitare classique - Guitare folk - Guitare électrique
IK Multimedia - Luther Paris - Magasin DJ Paris - Magasin instruments de musique Paris - Magasin sonorisation Paris - Microphone USB - Musique - Namm - Partition alto - Partition clarinette - Partition flute - Partition guitare
Partition harmonica - Partition musique - Partition piano - Partition saxophone - Partition trombone - Partition trompette - Partition vibraphone - Partition violon - Partition xylophone - Piano pas cher - Pioneer DJ - Platine vinyle
Site internet DJ - Site internet Deejay - Site internet Vision - Sono Behringer - Tablature guitare - Tablatures - Ukulele - Vente sono - Yamaha RS - batterie débutant



Conclusion

Travailler sur ce projet a été une vraie opportunité pour moi. Il m'a permis d'acquérir plus de compétences et de connaissances dans le développement web.

Par la suite j'aimerais terminer ce site, en poffinant la partie front du site, et en créant le panier.

Annexe 1

DICTIONNAIRE DE DONNEES

Utilisateurs (User) :

Nom de la propriété	Description de la propriété	Type de propriété
use_email	L'adresse email de l'utilisateur	Texte (180)
use_password	Le mot de passe de l'utilisateur	Texte (255)
use_firstname	Le prénom de l'utilisateur	Texte (50)
use_lastname	Le nom de famille de l'utilisateur	Texte (50)
use_address	L'adresse postale de l'utilisateur	Texte (255)
use_city	La ville où réside l'utilisateur	Texte (50)
use_zipcode	Le code postal de la ville où réside l'utilisateur	Texte (10)
use_country	Le pays où réside l'utilisateur	Texte (50)
use_phone	Le numéro de téléphone de l'utilisateur	Texte (50)
use_type	Le type d'utilisateur (s'il est un client professionnel ou particulier)	Booléen (vrai ou faux)
use_society	Le nom de la société de l'utilisateur (s'il en a une)	Texte (50)
use_id	L'ID servira à identifier l'utilisateur dans notre base de données.	Nombre

Produits (Product):

Nom de la propriété	Description de la propriété	Type de propriété
pro_name	Le nom du produit	Texte (70)
pro_short_wording	Le libellé court du produit	Texte (70)
pro_picture	L'image du produit	Texte (70)
pro_reference	La référence du produit	Texte (50)
pro_brand	La marque du produit	Texte (50)
pro_price	Le prix du produit	Décimal (9,2)
pro_stock	Le stock du produit	Nombre
pro_stock_alert	Le stock d'alerte du produit	Nombre
pro_long_wording	Le libellé long du produit (sa description)	Long texte
pro_id	L'ID servira à identifier le produit dans notre base de données	Nombre

Commandes (Order):

Nom de la propriété	Description de la propriété	Type de propriété
ord_delivery_address	L'adresse de livraison de la commande	Texte (255)
ord_billing_address	L'adresse de facturation de la commande	Texte (255)
ord_order_date	La date de commande	Date et heure
ord_shipping_date	La date d'expédition de la commande	Date et heure
ord_delivery_date	La date de livraison de la commande	Date et heure
ord_billing_date	La date de facturation de la commande	Date et heure
ord_order_status	Le statut de la commande	Texte (50)
ord_id	L'ID servira à identifier le produit dans notre base de données	Nombre

Equipe (Team) :

Nom de la propriété	Description de la propriété	Type de propriété
tea_firstname	Le prénom du membre de l'équipe	Texte (50)
tea_lastname	Le nom de famille du membre de l'équipe	Texte (50)
tea_email	L'adresse email du membre de l'équipe	Texte (50)
tea_phone	Le numéro de téléphone du membre de l'équipe	Texte (50)
tea_address	L'adresse postale du membre de l'équipe	Texte (255)
tea_zipcode	Le code postal de la ville où réside le membre de l'équipe	Texte (10)
tea_city	La ville où réside le membre de l'équipe	Texte (50)
tea_id	L'ID servira à identifier le membre de l'équipe dans la base de données	Nombre

Fournisseurs (Supplier):

Nom de la propriété	Description de la propriété	Type de propriété
sup_society	Le nom de la société du fournisseur	Texte (50)
sup_zipcode	Le code postal de la ville où se trouve le fournisseur	Texte (10)
sup_email	L'adresse email du fournisseur	Texte (50)
sup_contact_name	Le nom de contact du fournisseur	Texte (50)
sup_address	L'adresse postale du fournisseur	Texte (255)
sup_city	La ville où se trouve le fournisseur	Texte (50)
sup_phone	Le numéro de téléphone du fournisseur	Texte (50)
sup_country	Le pays où se trouve le fournisseur	Texte (50)
sup_id	L'ID servira à identifier le fournisseur dans la base de données	Nombre

Département commercial (Commercial Department):

Nom de la propriété	Description de la propriété	Type de propriété
com_dep_type	Quel type d'utilisateur (particulier ou professionnel) le commercial va s'occuper	Booléen (vrai ou faux)
com_dep_firstname	Le prénom du commercial	Texte (50)
com_dep_lastname	Le nom de famille du commercial	Texte (50)
com_dep_email	L'adresse email du commercial	Texte (50)
com_dep_phone	Le numéro de téléphone du commercial	Texte (50)
com_dep_zipcode	Le code postal de la ville où réside le commercial	Texte (10)
com_dep_city	La ville où réside le commercial	Texte (50)
com_dep_address	L'adresse postale du commercial	Texte (255)
com_dep_id	L'ID servira à identifier le commercial dans la base de données	Nombre

Catégories (Category):

Nom de la propriété	Description de la propriété	Type de propriété
cat_name	Le nom de la catégorie	Texte (50)
cat_id	L'ID servira à identifier la catégorie dans la base de données	Nombre

Sous-catégories (Subcategory):

Nom de la propriété	Description de la propriété	Type de propriété
sub_name	Le nom de la sous-catégorie	Texte (50)
sub_id	L'ID servira à identifier la sous-catégorie dans la base de données	Nombre

Détail de commande (Order Details):

Nom de la propriété	Description de la propriété	Type de propriété
ord_det_tax	La taxe	Décimal (7,2)
ord_det_unit_price	Le prix unitaire	Décimal (7,2)
ord_det_payment_method	Le moyen de paiement	Booléen (vrai ou faux)
ord_det_quantity	La quantité pour chaque produit	Nombre
ord_det_total_price	Le prix total de la commande	Décimal (9,2)
ord_det_additional_discount	La réduction supplémentaire	Décimal (9,2)
ord_det_discount	La réduction	Décimal (9,2)
ord_det_id	L'ID servira à identifier le détail de commande dans la base de données	Nombre

Annexe 2:

```
src > Controller > SecurityController.php > SecurityController
1  <?php
2
3  namespace App\Controller;
4
5  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
6  use Symfony\Component\HttpFoundation\Response;
7  use Symfony\Component\Routing\Annotation\Route;
8  use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;
9
10 class SecurityController extends AbstractController
11 {
12     /**
13      * @Route("/Login", name="app_Login")
14      */
15     public function login(AuthenticationUtils $authenticationUtils): Response
16     {
17         // if ($this->getUser()) {
18         //     return $this->redirectToRoute('target_path');
19         // }
20
21         // get the login error if there is one
22         $error = $authenticationUtils->getLastAuthenticationError();
23         // last username entered by the user
24         $lastUsername = $authenticationUtils->getLastUsername();
25
26         return $this->render('security/login.html.twig', ['last_username' => $lastUsername, 'error' => $error]);
27     }
28
29     /**
30      * @Route("/Logout", name="app_Logout")
31      */
32     public function logout()
33     {
34         throw new \LogicException('This method can be blank');
35     }
36 }
37 }
```



Inscription Infos Espace client



Produits

Service

Aide

A propos

Guitare Batterie Clavier Studio Sono Eclairage DJ Cases Accessoires

Connexion

Email

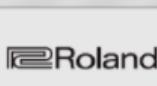
admin@villagegreen.com

Mot de passe

Valider

Vous êtes connecté en tant que admin@villagegreen.com, [Se déconnecter](#)

Nos partenaires



Livraison gratuite

à partir de 19 €



Garantie

3 ans



Satisfait ou remboursé

30 jours d'essai



Sécurité de paiement

Recevez nos offres exclusives

Votre adresse e-mail

Valider

Suivez-nous par ici !



Contactez-nous !

Conseil / Commande téléphone
du lundi au vendredi de 8h à 19h,
le samedi de 10h à 18h.

Depuis la France :
02 40 38 50 50

Belgique, Suisse, International :
0033 2 40 38 50 50

Service après vente :

02 51 80 68 76
Contactez-nous par téléphone
du lundi au samedi de 9h à 18h
ou depuis votre compte client.

Conseils pour choisir
un instrument :
infovg@villagegreen.com

Service Presse :
contact@villagegreen.com

Village Green Recrute !

Village Green Stores

Ouverts de 10h à 19h30 non-stop.

Guitares - Amplificateurs - Effets :
182 avenue Jean Jaurès
75019 Paris

Clavier - Home Studio - Sonorisation
Équipement DJ - Eclairage :
182 avenue Jean Jaurès
75019 Paris

Librairie Musicale :
7 av. du Nouveau Conservatoire
75019 Paris

Instruments à vent :
9/11 av. du Nouveau Conservatoire
75019 Paris

Percussions :
13/15 av. du Nouveau Conservatoire
75019 Paris

Métro Ligne 5, station Porte de Pantin.
Plan d'accès / Visite virtuelle

Utiles !

Qui sommes nous ?

F.A.Q.

Le blog Village Green

Vous avez un site Internet ?
Devenez partenaire Village Green

Conditions générales de vente

Mentions légales

Plan du site

Parrainage

Nouveautés

Assurance Woodbrass.com

Location d'instruments de musique

Village Green
Preservation Society

Toute l'actualité musicale

Voir l'avis des musiciens

Concours : inscription et résultat... à vous de jouer !

Village Green est une entreprise 100% Made in France !
Magasin de musique - Achat / Vente Instruments de musique - Location Instruments de musique - Atelier de réparation - Vente accessoires de musique au meilleur prix.
1989-2016 - RCS Paris B 222 333 444 Déclaration CNIL : 12345678.

Guitares - Amplis / Effets - Claviers / Pianos - Synths - Batteries / Percussions - Vents - Violons - Home Studio - Sonorisation - Deepay - Éclairage - Micros - Casques - Enregistreurs - Lifestyle - Déstockage - Ecoles & Pros

Behringer - Bose - Conservatoire de musique de Paris - Fender - Focusrite Scarlett - Focusrite - Gibson 2016 - Gibson - Guitar Shop - Guitare Martin - Guitare Taylor - Guitare classique - Guitare folk - Guitare électrique
IK Multimedia - Lutherie Parts - Magasin DJ Paris - Magasin instruments de musique Paris - Magasin sonorisation Paris - Microphone USB - Musique - Namm - Partition alto - Partition clarinette - Partition flûte - Partition guitare
Partition harmonica - Partition musique - Partition piano - Partition saxophone - Partition trombone - Partition trompette - Partition vibraphone - Partition violon - Partition xylophone - Piano pas cher - Pioneer DJ - Platine vinyle
Site internet DJ - Site internet Deepay - Site internet Violon - Sono Behringer - Tablature guitare - Tablatures - Ukulele - Vente sono - Yamaha K3 - batterie débutant

Annexe 3:

```
src > Controller > 🐘 RegistrationController.php > ...
1  <?php
2
3  namespace App\Controller;
4
5  use App\Entity\User;
6  use App\Form\RegistrationFormType;
7  use App\Security\LoginFormAuthenticator;
8  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
9  use Symfony\Component\HttpFoundation\Request;
10 use Symfony\Component\HttpFoundation\Response;
11 use Symfony\Component\Routing\Annotation\Route;
12 use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;
13 use Symfony\Component\Security\Guard\GuardAuthenticatorHandler;
14
15 class RegistrationController extends AbstractController
16 {
17     /**
18      * @Route("/inscription", name="app_register")
19      */
20     public function register(Request $request, UserPasswordEncoderInterface $passwordEncoder,
21                             GuardAuthenticatorHandler $guardHandler, LoginFormAuthenticator $authenticator): Response
22     {
23         $user = new User();
24         $form = $this->createForm(RegistrationFormType::class, $user);
25         $form->handleRequest($request);
26
27         if ($form->isSubmitted() && $form->isValid()) {
28             // encode the plain password
29             $user->setPassword(
30                 $passwordEncoder->encodePassword(
31                     $user,
32                     $form->get('plainPassword')->getData()
33                 )
34             );
35
36             $entityManager = $this->getDoctrine()->getManager();
37             $entityManager->persist($user);
38             $entityManager->flush();
39             // do anything else you need here, like send an email
40
41             return $guardHandler->authenticateUserAndHandleSuccess(
42                 $user,
43                 $request,
44                 $authenticator,
45                 'main' // firewall name in security.yaml
46             );
47         }
48
49         return $this->render('registration/register.html.twig', [
50             'registrationForm' => $form->createView(),
51         ]);
52     }
53 }
```

Annexe 4:

Villagegreen

Search

Product

Add Product

<input type="checkbox"/>	SAS-75 Alto
Short Wording	Saxophone
Reference	188679
Brand	Startone
Price	620
Stock	24
Stock Alert	5
Long Wording	View content
Subcategory	Saxophone
Picture	
...	
Edit	
Delete	
Name	F310 Natural
Short Wording	guitare acoustique

Annexe 5:

Nos catégories



Nos meilleures ventes





Nos partenaires

Roland

SENNHEISER

YAMAHA

behringer



Créer vos identifiants

Email

Mot de passe

Confirmation de mot de passe

Créer vos identifiants

Prenom

Prenom

Créer vos identifiants

Prenom

Prenom

Nom

Adresse

Ville

Code postal

Pays

Numero de telephone

J'accepte les conditions

Valider



SAS-75 Alto

Saxophone

620.00€

[Lire la suite](#)



 village
green

SAS-75 Alto Startone



620.00€

Description:

Info : Clé de Fa# aigu Bascule Repose-pouce réglable
Corps et clétage en laiton Finition: Vernis doré

Ajouter au panier