



Rapport de Stage

ATIWEB

Pierre Boulanger | Développeur Logiciel | 12/04/19

Table des matières

Présentation	5
Lieu de Stage	5
I. LES COMPETENCES COUVERTES.....	5
II. RESUME DU PROJET REALISE	6
III. CAHIER DES CHARGES ET EXPRESSIONS DES BESOINS	7
IV. SPECIFICATIONS TECHNIQUES DU PROJET	11
Méthode de travail de type Agile	17
V. EXTRAITS DE REALISATION	21
VI. JEU D'ESSAI	25
VII. VEILLE EFFECTUEE DURANT LE PROJET	31
VII. JOURNEE TYPE DU DEVELOPPEUR	31
VIII. EXTRAIT DU SITE ANGLOPHONE.....	32
AXES D'AMELIORATION	34
Back-End	35
Front-end.....	35

Présentation

Tout d'abord laissez-moi me présenter, je m'appelle Pierre Boulanger, j'ai 27 ans et après quelques années de travail au sein de la grande distribution, j'ai décidé de faire un virage professionnel et de me reconverter en tant que développeur web.

Ce virage n'est pas anodin puisque préparé depuis plusieurs mois après avoir été initié aux langages de base que sont le HTML et le CSS. Me passionnant pour le code informatique j'ai alors décidé de suivre cette formation « Développeur Logiciel » dans le but d'atteindre ce virage dans de bonnes conditions.

Lieu de Stage

J'ai effectué, lors de ma formation qualifiante « Développeur Logiciel » une période de stage d'une durée de 2 mois au sein d'ATIWEB, une agence web basée à Haguenau dont les principales clés métiers sont la création de site internet (Boutique en ligne, Sites vitrines), la mise en place de gestion commerciale, de solutions e-commerce, de supports de communication et de référencement.

ATIWEB est actuellement composée de 10 personnes : 5 développeurs orienté Back-End, 1 développeuse Front et 2 graphistes responsables de la création de maquettes et cartes de visite. 1 alternante est également présente tous les quinze jours, opérant sur le côté Front.

I. LES COMPETENCES COUVERTES

Lors de mon stage, j'ai pu mettre en œuvre des compétences liées à l'activité « Développer la partie back-end d'une application web d'une

application web ou web-mobile en intégrant les recommandations de sécurité ».

En effet, j'ai pu m'atteler sur le développement des composants d'accès aux données mais également d'élaborer et mettre en œuvre des composants dans une application de gestion de contenu et/ou de e-commerce.

Le projet m'ayant permis de développer ces compétences est l'intégration d'un système de paiement au sein d'une boutique en ligne.

II. RESUME DU PROJET REALISE

Un client a exprimé le besoin d'une boutique en ligne pour une marque de vêtements sportifs baptisée "Provoke" en souhaitant intégrer un module de paiement issu de la banque BNP Paribas qui s'intitule Mercanet.

Mercanet propose aux clients de payer leurs achats en ligne de manière sécurisée et offre la possibilité aux clients d'utiliser un large éventail de modes de règlement en ligne : cartes bancaires, cartes privatives, paiement en plusieurs fois, compte Paypal etc...

Cette méthode de paiement a donc été reliée à liste de sélection des moyens de paiement et relié au panier client.

Après validation des informations du mode de paiement sur la page Mercanet, deux réponses sont envoyées, une réponse client qui renvoie sur la boutique en ligne qui confirme le paiement ou non, et une réponse côté serveur dite réponse automatique pour s'assurer qu'au moins une réponse sera envoyée.



III. CAHIER DES CHARGES ET EXPRESSIONS DES BESOINS

Cahier des charges

1. Intégration front des pages principales d'une boutique en ligne

La première étape du projet a consisté à intégrer la plupart des pages que l'on peut retrouver au sein d'une boutique en ligne :

La création de la page d'accueil, les pages de contact et d'informations (Politique de confidentialité, Guide des Tailles, Politiques des retours). Un pied de page a été créé en y incluant les liens des pages de contact et d'informations.

Une barre de navigation a également été intégrée, où apparaissent le nom de la boutique ("Provoke") ainsi que les trois grandes catégories d'articles qui seront disponibles dans la boutique (Hommes, Femmes, Accessoires).

2. Création de la base de données

Installation de la base de données en utilisant l'ORM Doctrine inclus dans Symfony en créant les entités présentes pour une boutique : entités Articles, Sous-Catégorie, Main Catégorie pour le "bloc" Article ; entité Client et Commande pour le reste.

3. Mise en place de l'interface d'administration

Installation d'un bundle important une interface complète d'administration permettant d'être customisée à souhait et qui reflète ici les trois grands axes de la base de données (Articles, Clients, Commandes).

Optionnel, customisation d'ordre cosmétique afin d'embellir l'espace de travail des administrateurs.

4. Gérer le système d'authentification

Création d'un système d'authentification avec un espace utilisateur : formulaire d'inscription (avec hash du champ mot de passe ainsi qu'une confirmation), de connexion et d'édition. Permettre également la déconnexion.

Mise à jour de la barre de navigation avec liens de connexion/inscription si l'utilisateur n'est pas inscrit et, à l'inverse, si l'utilisateur est inscrit, liens "Mon Compte" et déconnexion.

5. La sécurité, l'autorisation et les erreurs

Mise en place des rôles utilisateurs dans un champs à part de Client, sécurisation des routes par les Controller avec les annotations `@isGranted()` selon le rôle de l'utilisateur, affichage de liens ou bouton en fonction du rôle de l'utilisateur.

Création de pages d'erreurs personnalisées pour les erreurs 404 (mauvaise adresse) et 403 (accès refusé pour les simples utilisateurs qui auraient trouvé le chemin de l'administration par exemple)

6. La dynamisation des vues produits

Dynamisation des vues renvoyant aux catégories et articles suivant la catégorie principale (Main Catégorie = Hommes, Femmes et Accessoires). Lorsqu'un visiteur cliquera sur l'une des Main Catégories visible dans la barre de navigation, il sera alors renvoyé sur une page où seront présentées les sous-catégories de celle-ci.

Même remarque pour les sous-catégories, lorsqu'un visiteur cliquera sur l'une d'entre elles, il sera renvoyé sur la page où seront présentés les articles de cette sous-catégorie appartenant à la Main Catégorie choisie.

7. La gestion du panier client

Dynamisation de la page panier en fonction de l'ajout au panier d'un ou plusieurs articles selon la quantité choisie.

Sur une page article, lors du clic sur le bouton " Ajouter au panier" on redirige l'utilisateur vers la page panier (un visiteur sera redirigé vers la page d'inscription) où l'article ajouté précédemment apparaît avec la quantité choisie et le prix qui en découle. Le client peut poursuivre ses achats ou confirmer sa commande.

Renvoi vers une page de confirmation du panier avec le résumé de la commande, l'adresse de livraison et l'option de paiement dans un formulaire POST.

8. La requête de paiement et l'appel au service Mercanet

Instanciation d'une classe Mercanet et construction des données à passer obligatoirement à Mercanet pour que la requête soit valide (formulaire de méthode POST avec en action l'URL de la plate-forme marchande et renseignement des paramètres obligatoires pour l'appel de la page de paiement)

9. Les réponses de paiement

Envoi de deux réponses, manuelle (Faire en sorte que le client revienne au site Web) et automatique (le moteur Mercanet envoie une réponse automatique vers le site Web) en confirmation que le paiement est bien passé et que la commande est confirmée.

Les réponses manuelles et automatiques renverront les mêmes choses.

1. Accueil

Le projet étant une boutique en ligne, elle aura cependant une composante que tous les sites ont en commun : une page d'accueil. La page d'accueil remplit 3 objectifs différents : présenter notre activité, mettre en avant les produits, orienter nos potentiels visiteurs.

Le visiteur doit être capable de comprendre en quelques secondes l'activité que nous exerçons, les produits que nous vendons et où peut-il s'orienter.

2. Fiches produits

Les pages produits doivent également être épurées de sorte que le client puisse directement voir les informations principales : le titre du produit, une image représentant le produit qui permette à l'internaute d'avoir un aperçu, le prix mis en valeur : l'internaute doit savoir immédiatement si le produit est dans son budget, ainsi qu'un bouton d'ajout au panier afin de guider le potentiel client vers l'achat.

3. Un accès rapide au panier

Le panier est mis en évidence dans la barre de navigation lorsque l'internaute est connecté. Il peut avoir un récapitulatif de son panier en cliquant sur l'onglet " Mon Panier". Il sait alors le nombre de produits qu'il a ajouté, leur quantité et le prix total qui en découle. Le bouton "Poursuivre ma commande" lui permet d'accéder au processus d'achat tandis qu'un bouton " Poursuivre mes achats" lui permet de continuer à naviguer sur le site.

4. Une page récapitulative de la commande

L'internaute peut supprimer un produit, modifier sa quantité et donc recalculer son panier. Il peut mettre à jour sa commande rapidement et simplement avant de commencer le processus d'achat.

5. Le processus de paiement

Notre site e-commerce redirige le client vers la page de paiement Mercanet. Le client renseigne les informations de son moyen de paiement pour que le serveur de paiement Mercanet prenne en charge la transaction, bien sûr avec le bon prix récupéré. A la fin du processus de paiement, le client est renvoyé à l'adresse URL précisé lors des réponses manuelle et automatique.

Il sera redirigé de préférence sur notre site e-commerce vers une page spéciale dont le contenu adressera au client un message de confirmation prenant en compte sa commande.

IV. SPECIFICATIONS TECHNIQUES DU PROJET

Le projet a été réalisé sur le framework **Symfony**:



- **SYMFONY** : Symfony est un framework du langage back-end PHP. Un framework est un ensemble de petits composants qui peuvent être utilisés un à un mais qui mis en relation forment un outil très puissant et facilitent le travail des développeurs.

Il possède différents avantages : C'est un des frameworks php les plus utilisés, il possède donc une communauté assez grande pour trouver des réponses à nos questions, avec une documentation simple d'accès.

Symfony fonctionne selon une architecture **MVC (Modèle, Vue, Contrôleur)** qui définit un cadre d'organisation de notre code :

Le **modèle** concerne le traitement des données, la **vue** est la réponse envoyée au client après que sa demande ait été traitée par l'intermédiaire du contrôleur chargé de solliciter les modèles correspondants à la demande.

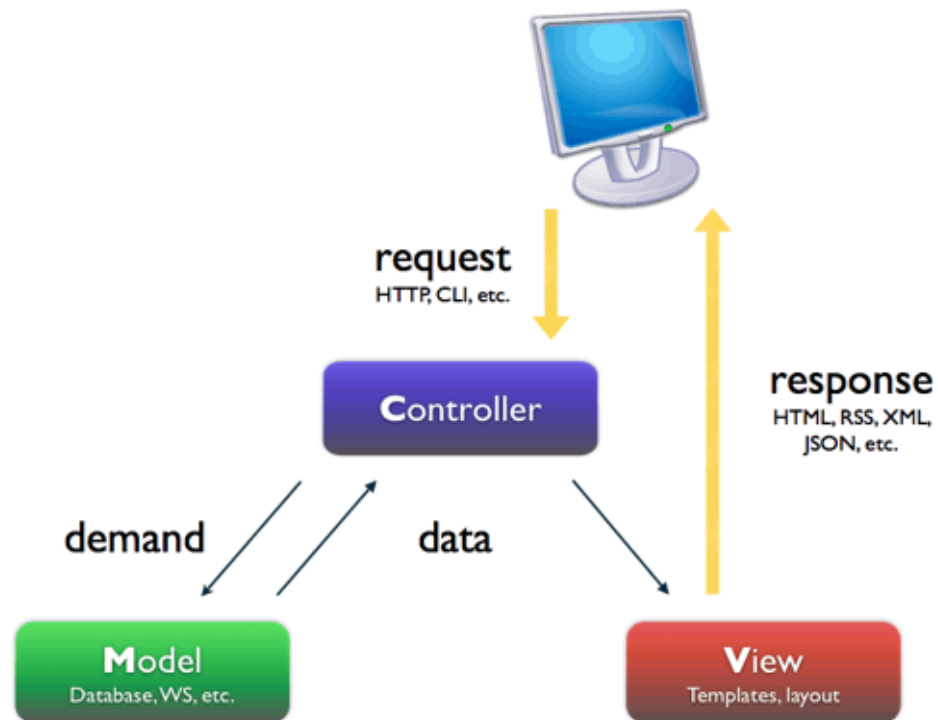


Schéma représentant une architecture MVC

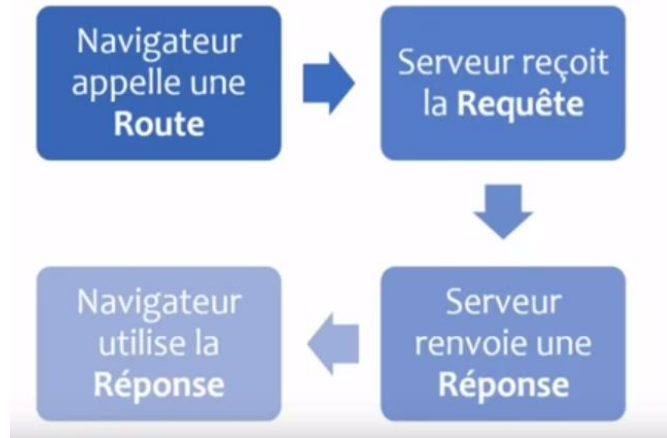
Les **3** piliers de **Symfony** :

Les **Contrôleurs** pour gérer les traitements.

Doctrine pour gérer l'accès aux données.

Twig pour gérer l'affichage.

La logique du Controller



Langage de rendu Twig

Simplicité

- Facilite l'écriture des affichages
- Apporte beaucoup de fonctionnalités

Absence de PHP

- Permet d'abstraire les affichages de balises PHP
- Plus simple pour un intégrateur



- **PHP** : Langage de programmation permettant de servir des pages web dynamiques à l'aide d'un serveur http permettant de développer en orienté-objet et permet de communiquer avec les systèmes de gestion de base de donnée.



- **TWIG** : Twig est un langage de rendu qui facilite l'écriture de rendu et apporte des fonctionnalités que l'HTML avait du mal à rendre (conditions, boucle). L'un des autres avantages est l'absence de PHP au sein du rendu, ce qui peut être plus simple pour de l'intégration ou lorsqu'une autre personne doit toucher au projet.

Il permet également d'isoler du code au sein de bloc (ex. `{% bloc body %}` `{% endblock %}`) qui facilite l'écriture de template propre pour une page en particulier par exemple ou l'intégration d'une barre de navigation présente pour toutes les pages du site. Nos pages deviennent alors plus allégées. Twig est le langage de rendu officiel de Symfony.



- **COMPOSER** : Gestionnaire de librairies/dépendances pour PHP. Lorsque nous avons besoin d'une librairie, il va nous la télécharger et bien la ranger.

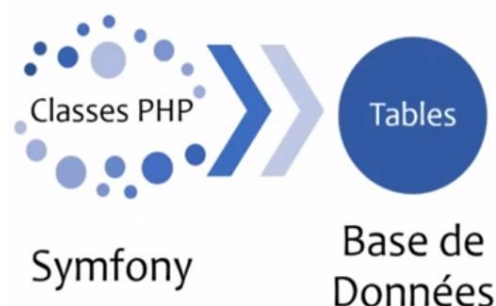
Afin de créer un projet Symfony, nous avons besoin de Composer, d'ouvrir une invite de commande et de demander à Composer de créer un projet avec la commande ci-dessous :

```
$ composer create-project symfony/website-skeleton my-project
```

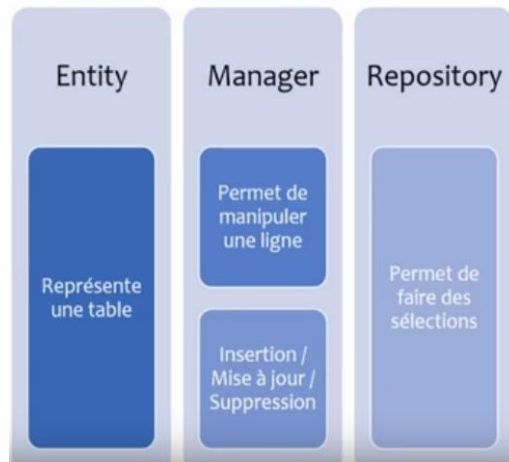
- **PHPMYADMIN** : Application de gestion pour les systèmes de gestion de base de données. Il fonctionne grâce au langage SQL. Pour ce projet, nos données sont injectées dans phpMyAdmin grâce aux commandes de **Doctrine**. **Doctrine** est un **ORM** (**surcouche** de base de données) qui est l'ORM par défaut de **Symfony**. C'est une brique logicielle qui fait le lien entre une application écrite dans n'importe quel langage informatique et une base de données.

Le but est de gérer au sein de notre site - par des classes et des objets (PHP dans notre projet) nos données - et que ce que l'on fait dans notre site se reflète dans la base de données grâce à l'ORM à travers des commandes doctrines qui envoie à la base nos instructions en langage SQL.

L'ORM de Symfony :



L'ORM de Symfony :



Les autres langages utilisés lors du projet :



- **HTML**: Le HTML est le langage de balisage de base servant à représenter une page web. Le code renvoie de l'hypertexte, et sert de langage de base à l'intégration.



- **CSS**: Le CSS est l'un des langages principaux du web. C'est un langage de feuilles de style permettant d'appliquer ses styles sur nos pages web.



- **BOOTSTRAP**: A l' occasion de ce projet, je n'ai pas voulu prendre trop de temps concernant la réalisation d'un design unique au projet afin de me consacrer aux fonctionnalités back-end de la boutique.

J'ai donc choisis de prendre un thème préfabriqué en utilisant le sélecteur de thème Bootswatch qui permet d'habiller votre contenu simplement en sélectionnant le thème qui vous paraît adapté à votre site en incluant dans votre code un lien link incluant l'url de votre thème (.min.css) dans le contenu de votre balise <head>. Ainsi votre site et les composants ajoutés apparaîtront selon le style choisis par vos soins.



- **SQL**: Le SQL est un langage permettant de communiquer avec une base de données. Le code renvoie de l'hypertexte, et sert de langage de base à l'intégration. Il a essentiellement été généré par les migrations de Doctrine (voir plus loin).

Méthode de travail de type Agile

Avantages

La méthode de travail employé lors de ce projet repose sur une méthodologie de planification de projet baptisé méthode **Agile** :

Cette méthode repose sur le principe que planifier totalement son projet de A à Z avant de le développer est contre-productif et est une perte de temps. En effet tout peut ne pas se passer comme prévu et des contretemps où des difficultés qui pourraient prendre plus de temps que prévu peuvent survenir.

La méthode Agile recommande de se fixer des objectifs à plus court terme et est divisé en plusieurs sous-projets qui, une fois tous atteints, forment le projet final. Elle laisse également plus de place aux imprévus.

Inconvénients

Les inconvénients d'une telle méthode pour une équipe restreinte nécessite une motivation pointue du fait qu'elle ne laisse peu de place aux sujets de fond puisque le but n'est pas de rester sur le même sujet plusieurs jours/semaines d'affilées.

Le cycle de vie du développement du projet possède un rythme difficile à suivre compte tenu du rythme imposé au préalable et une phase d'adaptation est indispensable puisque les changements sont continuels.

LES COMPOSANTS DU PROJET

1. L'espace Administrateur

La boutique en ligne "Provoke" comporte au mieux deux administrateurs, le gérant et le développeur chargé du projet. Afin d'avoir de leur donner un espace propre j'ai fais appel à un Bundle Symfony : Un bundle est une "brique" de votre application qui contient tout ce qui concerne une fonctionnalité donnée et qui s'adapte à votre projet. Pour l'ajout d'une partie administration, j'ai choisi le bundle EasyAdminBundle.

Il permet de mettre rapidement et simplement une interface d'administration afin d'y gérer nos contenus tout en mettant en place un **CRUD** déjà installé.

*Qu'est-ce qu'un **CRUD** ?*

De l'anglais Create, Read, Update, Delete (Créer, Lire, Mettre à jour, Supprimer) désigne les opérations permettant le stockage d'informations en base de données et leur **persistance** (dans notre cas nous faisons persister des objets en base de données).

Son utilisation se fait à travers la configuration YAML(format de représentation de données) dans le fichier `easy_admin.yaml` récupéré lors de l'importation du bundle grâce à la commande Composer : `require admin`.

L'interface propose **3** grandes catégories :

- Les articles avec la possibilité d'ajouter/supprimer des articles, des catégories d'articles ou des grandes catégories d'articles.
- Les clients avec la possibilité de checker les informations des utilisateurs du site.
- Les commandes passés au sein de la boutique "Provoke".

Cet espace administrateur est relié directement à la partie front de notre site mais n'est atteignable uniquement pour les utilisateurs dont le rôle est ADMIN. Cette condition a été mise en place grâce au composant Security de Symfony. En effet au sein du template Twig de base, j'ai mis au sein de la barre de navigation un lien menant vers la partie Back-Office seulement si l'utilisateur est un administrateur.

J'ai ensuite précisé au sein du Controller en question que la route menant au chemin de l'administration était seulement accessible aux administrateurs puis j'ai placé au sein de l'`access_control` du fichier `security.yaml` (le système de sécurité est configuré dans ce dossier) la route où seul le rôle administrateur est autorisé à passer (cf. photo ci-dessous)

2. L'espace visiteur

L'accès à la boutique en ligne est quant à lui disponible à tout utilisateur anonyme et chacun à accès à la consultation des grandes

catégories d'articles, des sous catégories correspondants ainsi que des articles qui en découlent. L'utilisateur anonyme peut consulter les informations associées tel que son nom, son prix, l'image d'illustration du produit. Il a également la possibilité d'ajouter un ou plusieurs articles et la quantité souhaitée dans un panier enregistré au sein de la session de l'utilisateur. Il a la possibilité de voir un résumé de son panier, d'ajouter ou de supprimer des articles, cependant s'il souhaite passer commande, il devra créer un compte client.

3. L'espace client

L'espace client de la boutique en ligne permet aux utilisateurs de pouvoir passer commande après avoir possédé un panier d'articles. Après confirmation de la part du client possédant un compte, celui-ci peut alors choisir son mode de paiement (ici Mercanet).

Concernant les autres fonctionnalités de l'espace client, celui-ci aura la possibilité à travers un tableau de bord de consulter et modifier ses informations personnelles, que ce soit son adresse physique, son mot de passe ou encore son adresse e-mail.

4. Mercanet

La mise en place de la passerelle de paiement vers les services BNP Paribas Mercanet permettant au client de régler son achat puis d'être redirigé vers la boutique, que ce soit lors du succès de l'opération avec un message de confirmation d'achat, ou un message d'erreur au cas où le paiement n'aurait pas marché.

Pour cela, utilisation de l'**API** BNP Paribas Mercanet afin de l'intégrer dans le site.

Qu'est-ce qu'une API ?

Une **API (Application Programming Interface)** est comme son nom l'indique une interface permettant de connecter plusieurs applications en une seule entité. Ici, concevoir le système de paiement en partant de zéro serait long, difficile et contre-productif. J'ai donc fais appel à cette

API grâce à la documentation fournie pour mettre en place un système de paiement rapide et sécurisé.

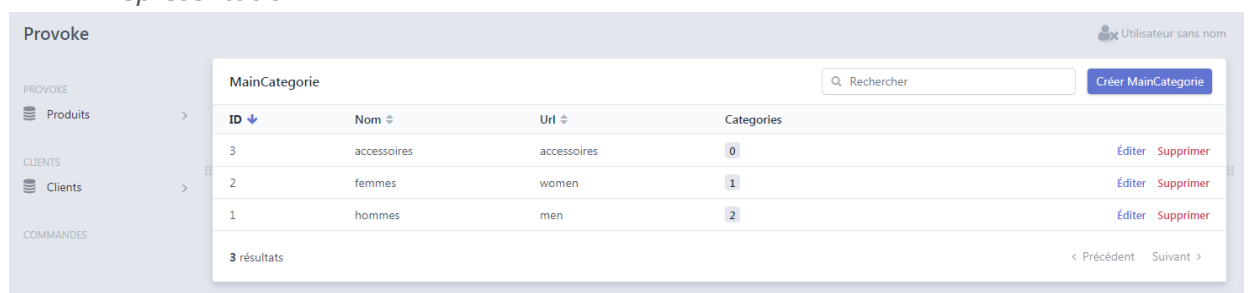
V. EXTRAITS DE REALISATION

Voici des extraits de réalisations prodigués par mes soins (code et interprétation du navigateur de celui-ci) concernant la boutique en ligne “Provoke” :

Configuration du menu de l'espace administrateur dans le fichier easy_admin.yaml:

```
easy_admin:
  site_name: 'Provoke'
  design:
    menu:
      - label: 'Provoke'
      - label: 'Produits'
        icon: 'database'
        children:
          - {entity: 'MainCategorie', icon: 'inbox', label: 'Categories Principales'}
          - {entity: 'Categorie', icon: 'inbox', label: 'Sous-Categories'}
          - {entity: 'Article', icon: 'inbox', label: 'Articles'}
      - label: 'Clients'
      - label: 'Clients'
        icon: 'database'
        children:
          - {entity: 'Client', icon: 'inbox', label: 'Clients'}
      - label: 'Commandes'
```

Représentation:



The screenshot shows the Provoke admin interface. On the left is a sidebar with a menu containing 'Produits', 'Clients', and 'Commandes'. The main area displays a table titled 'MainCategorie'. The table has columns for 'ID', 'Nom', 'Url', and 'Categories'. There are three rows of data. At the bottom of the table, it says '3 résultats'. On the right side of the table, there are links for 'Éditer' and 'Supprimer' for each row. At the top right of the table, there is a search bar labeled 'Rechercher' and a button labeled 'Créer MainCategorie'.

ID	Nom	Url	Categories
3	accessoires	accessoires	0
2	femmes	women	1
1	hommes	men	2

Initialisation des champs jugés utiles de chaque entité (ici l'entité article):

```

entities:
  MainCategorie:
    class: App\Entity>MainCategorie
    form:
      fields:
        - {property: 'nom'}
        - {property: 'url'}
  Categorie:
    class: App\Entity\Categorie
  Article:
    class: App\Entity\Article
    list:
      fields:
        - "nom"
        - { property: 'image', type: 'image', base_path: '%app.path.article_images%' }
        - "prix"
        - "actif"
        - "categorie"
        - "caracteristiques"
    form:
      fields:
        - "nom"
        - "url"
        - "description"
        - "prix"
        - { property: 'imageFile', type: 'vich_file' }
        - "actif"
        - "categorie"
        - "caracteristiques"
      actions:
        - {name: 'editer', icon: 'pencil', label: false, css_class: 'btn btn-secondary'}
        - {name: 'supprimer', icon: 'trash', label: false, css_class: 'btn btn-danger'}

```

Représentation:




Provoke Utilisateur sans nom

PROVOKE

- Produits
 - Categories Principales
 - Sous-Categories
 - Articles
- CLIENTS
 - Clients
- COMMANDES

Article

Créer Article

Nom	Image	Prix	Actif	Categorie	Caracteristiques	
haut homme noir		24.99	<input checked="" type="checkbox"/>	haut	Taille : S, M, L, XL	Éditer Supprimer
Pantalon de marque		34.99	<input checked="" type="checkbox"/>	pantalons		Éditer Supprimer
Pantalon de jogging		29.99	<input checked="" type="checkbox"/>	pantalons		Éditer Supprimer

3 résultats
< Précédent Suivant >

Barre de navigation lorsqu'un utilisateur est un administrateur :

[Back-Office](#)
[Mon compte](#)
[Panier](#)
[Deconnection](#)

Code représentant la barre d'utilisateur selon son rôle (la condition `is_granted('ROLE_ADMIN')` envoyant un lien vers le Back-Office pour les administrateurs) :

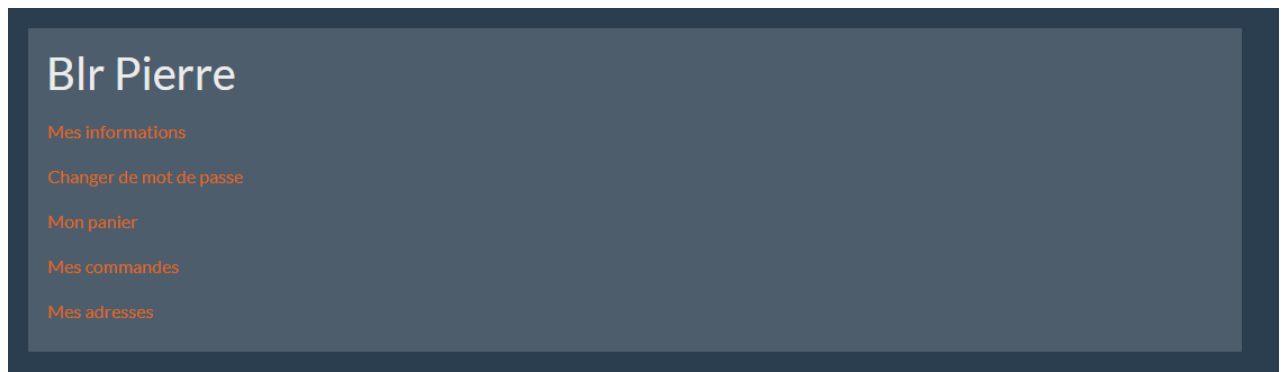
```

<ul class="navbar-nav ml-auto">
{% if app.user %}
{% if is_granted ('ROLE_ADMIN') %}
<a href="/admin" class="nav-link">Back-Office</a>
{% endif %}
    <a href="{{path('account_dashboard')}}" class="nav-link">Mon compte</a>
</li>
<li class="nav-item">
    <a href="{{path('shopping_cart')}}" class="nav-link">Panier</a>
</li>
<li class="nav-item">
    <a href="{{path('account_logout')}}" class="nav-link">Deconnection</a>
</li>
{% else %}

<li class="nav-item">
    <a href="{{path('account_login')}}" class="nav-link">Connexion</a>
</li>
<li class="nav-item">
    <a href="{{path('account_register')}}" class="nav-link">Inscription</a>
</li>
{% endif %}
</ul>

```

Espace client :



Espace résumé (profil et panier du client):

Modification du profil

Nom

Bir

Prenom

Pierre

Email

pierre@symfony.com

Telephone

0388562532

Enregistrer les modifications

Votre Panier

Livraison/Paiement

Confirmation

Articles	Prix Unitaire TTC	Quantité	Prix Total
 Pantalon de marque	34.99 €	- 6 +	209.94

Total de vos achats 209.94

Dont éco-participation 0.06 €

Continuer mes achats

Continuer ma commande

Extrait du code d'initialisation de la classe Mercanet permettant l'appel de la page de paiement ainsi que l'URL de la réponse envoyée :

```
<?
// Initialisation de la classe Mercanet avec passage en parametre de la cle secrete
$paymentRequest = new mercanet('mx03ETCpSvqxEFDihhdcYbhlL7BwhbsBip0mq3kvZzQ');

// Indiquer quelle page de paiement appeler : TEST ou PRODUCTION
$paymentRequest->setUrl(mercanet::PRODUCTION);

// Renseigner les parametres obligatoires pour l'appel de la page de paiement
$paymentRequest->setMerchantId('211000077770001');
$paymentRequest->setKeyVersion('1');
$paymentRequest->setTransactionReference(uniqid() . "n". $id_commande);
$prix_sanitize = filter_var($prix_total, FILTER_SANITIZE_NUMBER_INT);
$prix_int = intval($prix_sanitize);
$paymentRequest->setAmount($prix_int);
$paymentRequest->setCurrency('EUR');

// Reponse manuelle et automatique
if (empty($_SERVER["HTTPS"])) $http = "http://";
else $http = "https://";
$urlReturn = $http . $_SERVER["HTTP_HOST"] . dirname($_SERVER["REQUEST_URI"]) . "/commande-traitement-ok.html";
$paymentRequest->setNormalReturnUrl($urlReturn);
$urlAuto = $http . $_SERVER["HTTP_HOST"] . dirname($_SERVER["REQUEST_URI"]) . "/paiement/feedback.php";
$paymentRequest->setAutomaticResponseUrl($urlAuto);
$paymentRequest->setLanguage('fr');
$paymentRequest->validate();
```

Redirection au service de paiement après sélection de la solution Mercanet:

Informations sur la transaction

Trésor-bébé

Référence de la transaction :
5c6fc0e59028an3314

Identifiant du commerçant :
211000077770001

Montant de la transaction :
2 344,00 €

Modes de paiement

Veuillez choisir votre mode de paiement

Payer par carte

Autres moyens de paiement

Annuler

Page de saisie d'informations classique après sélection du mode de paiement (type de carte):

Informations sur la transaction

Trésor-bébé

Référence de la transaction :
5c6fc0e59028an3314

Identifiant du commerçant :
211000077770001

Montant de la transaction :
2 344,00 €

Informations de la carte

Veuillez saisir les informations de votre paiement

Numéro de carte :

Date d'expiration :

Mois : 01

Année : 2019

Cryptogramme visuel :

Annuler

Valider

Selon votre établissement bancaire, vous pourrez être redirigé vers la page d'authentification de votre banque avant la validation de votre paiement.

Copyright © 2019 - Tous droits réservés

VI. JEU D'ESSAI

Premier jeu d'essai concernant la création d'un compte client, avec composants de formulaire : création d'une fonction fabriquant le formulaire en lui faisant passer les caractéristiques de ces champs (ex. le mot de passe sera un champ de type Password). Nous ne voulons pas que l'utilisateur puisse renseigner une adresse mail existante dans notre

base de données et renseigner deux mots de passe différents. Pour ce deuxième point un champ de confirmation de mot de passe permettra d'éviter ce point.

```
<?php

namespace App\Form;

use App\Entity\Client;
use App\Form\ApplicationType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;
use Symfony\Component\Form\Extension\Core\Type\PasswordType;

class RegistrationType extends ApplicationType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('nom', TextType::class, $this->getConfiguration("Nom", "Nom"))
            ->add('prenom', TextType::class, $this->getConfiguration("Prénom", "Prénom"))
            ->add('email', EmailType::class, $this->getConfiguration("Email", "Email"))
            ->add('hash', PasswordType::class, $this->getConfiguration("Mot de Passe", "Mot de Passe"))
            ->add('passwordConfirm', PasswordType::class, $this->getConfiguration("Confirmation Mot de Passe", "Confirmation Mot de Passe"))
            ->add('adresse', TextType::class, $this->getConfiguration("Adresse", "Adresse"))
            ->add('codePostal', TextType::class, $this->getConfiguration("Code Postal", "Code Postal"))
            ->add('telephone', TextType::class, $this->getConfiguration("Telephone", "Telephone"))
            ->add('ville', TextType::class, $this->getConfiguration("Ville", "Ville"))
            ->add('pays', TextType::class, $this->getConfiguration("Pays", "Pays"));
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Client::class,
        ]);
    }
}
```

Définition de la fonction register ("s'inscrire") dans le contrôleur associé à la partie Compte : on passe un formulaire qui, s'il est soumis et bien valide va encoder le mot de passe à travers un algorithme de type bcrypt

```

/**
 * @Route("/register", name="account_register")
 *
 * @return Response
 */
public function register(Request $request, ObjectManager $manager, UserPasswordEncoderInterface $encoder)
{
    $client = new Client();
    $form = $this->createForm(RegistrationType::class, $client);
    $form->handleRequest($request);

    if($form->isSubmitted() && $form->isValid()){
        $hash = $encoder->encodePassword($client,$client->getHash());
        $client->setHash($hash);

        $manager->persist($client);
        $manager->flush();

        dump($client);

        $this->addFlash('success','Votre compte à bien été enregistré');

        return $this->redirectToRoute('home');
    }

    return $this->render('account/registration.html.twig', [
        'form' => $form->createView()
    ]);
}

```

Rendu Twig grâce à sa propre méthode de formulaire et essai réalisé :

```

1  [% extends 'base.html.twig' %]
2
3  {% block title %}Register | Provoke{% endblock %}
4
5  {% block body %}
6
7  <div class="container">
8      <h1>Créer un Compte</h1>
9
10     {{ form_start(form) }}
11     {{ form_widget(form) }}
12     <button type="submit" class="btn btn-success">Créer un Compte</button>
13     {{ form_end(form) }}
14 </div>
15
16 {% endblock %}

```

Les tests sont concluants puisque des messages d'erreurs indiquent à l'utilisateur que les champs renseignés ne sont pas valides, le compte ne peut donc pas être créé.

Créer un Compte

Nom

Monsieur

Prénom

Test

Email

ERREUR L'adresse e-mail est déjà utilisé

test@exemple.fr

×

Mot de Passe

Mot de Passe

Confirmation Mot de Passe

ERREUR Veuillez renseigner un mot de passe identique

Confirmation Mot de Passe

×

Deuxième jeu d'essai concernant la connexion d'un client si celui-ci renseigne mal son mot de passe ou son adresse mail :

```
class AccountController extends AbstractController
{
    /**
     * @Route("/login", name="account_login")
     *
     * @return Response
     */
    public function login(AuthenticationUtils $utils)
    {
        $error = $utils->getLastAuthenticationError();
        $username = $utils->getLastUsername();

        //dump($error);
        return $this->render('account/login.html.twig', [
            'hasError'=> $error !== null,
            'username'=> $username
        ]);
    }
}
```

Affichage dans Twig en incluant une condition si une erreur survient. Un message d'alerte apparaît alors et l'utilisateur a de nouveau l'opportunité de renseigner ses identifiants pour tenter de se connecter.

```

1  {% extends 'base.html.twig' %}
2
3  {% block title %}Sign In | Provoke{% endblock %}
4
5  {% block body %}
6      <div class="container">
7          <div class="row">
8              <div class="col-md-3">&nbsp;</div>
9              <div class="col-md-6">
10                 <div class="bg-dark py-3 px-3">
11                     <h1>Connexion</h1>
12                     {% if hasError %}
13                         <div class="alert alert-danger">Une erreur est survenue, veuillez réessayer</div>
14                     {% endif %}
15                     <form action="{{ path('account_login') }}" method="post">
16                         <div class="form-group">
17                             <label for="email">Email</label>
18                             <input type="text" name="_username" id="email" class="form-control" placeholder="Adresse email" required value="{{ (username) }}">
19                         </div>
20                         <div class="form-group">
21                             <label for="password">Mot de passe</label>
22                             <input type="password" name="_password" id="password" class="form-control" placeholder="Mot de passe" required>
23                         </div>
24                         <div class="form-group">
25                             <button type="submit" class="btn btn-success">Se connecter</button>
26                         </div>
27                     </form>
28                 </div>
29             </div>
30         </div>
31     </div>
32 {% endblock %}

```

Démonstration : le paramètre Email récupère la dernière valeur renseigné par l'utilisateur

The screenshot displays a web application interface for a login page. At the top, the title "Connexion" is visible. Below the title, a red alert box contains the message "Une erreur est survenue, veuillez réessayer". The form consists of two input fields: "Email" and "Mot de passe". The "Email" field contains the text "test@exemple.com". Below the "Mot de passe" field, there is a green button labeled "Se connecter".

Modifions les informations du profil et voyons ce qu'il en est :

Avant modification

Modification du profil

Nom

Blr

Prenom

Pierre

Email

pierre@symfony.com

Telephone

0388562532

Enregistrer les modifications

Après modification

Le profil à été modifié

Modification du profil

Nom

Boulangier

Prenom

Pierre

Email

pierre@elan-formation.com

Telephone

0391753527

Enregistrer les modifications

Le profil a bien été modifié, on informe l'utilisateur à travers un message de succès tout en lui envoyant les changements qu'il a effectué. Voici les infos de l'utilisateur enregistrés avant la modification du profil :

id	nom	prenom	email	hash
8	Blr	Pierre	pierre@symfony.com	\$2y\$13\$NW0KZnaNI2aZEulpCEHjneliB7aRpXkdhzZ/dYb6WIK...

Et voici après la modification du profil :

id	nom	prenom	email	telephone
8	Boulangier	Pierre	pierre@elan-formation.com	0391753527

VII. VEILLE EFFECTUEE DURANT LE PROJET

La veille effectuée à porté sur le composant sécurité de **Symfony** afin d'assurer l'accès à l'administration uniquement aux utilisateurs dont le rôle est Admin, l'accès au tableau de bord et aux pages d'informations client aux utilisateurs dont le rôle est au minimum utilisateur : un visiteur ne peut pas avoir accès à des pages comme "Modifier son mot de passe " ou "Mes informations" par exemple. Si un visiteur tape l'URL en dur, il est redirigé vers la page de connexion.

A défaut le visiteur a accès à la route permettant de s'inscrire, ainsi que les autres pages du site, celles d'informations (Les pages produits, le panier, les informations supplémentaires tel que la politique de confidentialité).

Afin de mener ce projet à bien, il m'a fallu également suivre les étapes d'installation de Mercanet, qui dispose de sa propre documentation disponible sur <https://documentation.mercanet.bnpparibas.net>.

Mercanet utilise une clé secrète pour sécuriser et authentifier les échanges entre le serveur client et le serveur Mercanet. A l'occasion de ce projet j'ai utilisé l'environnement de test fourni par les services Mercanet.

VII. JOURNEE TYPE DU DEVELOPPEUR

La journée type pour le développement du site e-commerce "Provoke" consiste à gérer le projet et chaque composant de la conception à la production. Le projet a été découpé en une liste de fonctionnalités et de tâches et chaque début de journée une petite mise

à plat afin de déterminer quelle partie du projet était prioritaire, quelles tâches étaient liées entre elles et à combien de jours je pouvais estimer que telle tâche/fonctionnalité serait réalisée et ce, tout en respectant la méthode de travail **Agile**.

Une veille est plébiscitée dès l'apparition d'une difficulté à aborder ou développer un des composants. Pour cela il faut constamment augmenter le nombre d'outils à disposition pour résoudre les problèmes donnés. Il a fallu développer sa curiosité pour être au fait des pratiques et façons de faire les choses. J'ai eu recours à bon nombre de moyens pour être au fait des pratiques selon le ou les difficultés rencontrées du jour, les principaux étant la plate-forme Youtube, Github et, bien sûr, les documentations officielles des langages de programmation ou encore des sites web tel que StackOverflow.

Ce dernier est un site proposant des questions et réponses sur un large choix de thèmes concernant la programmation informatique et est l'une des principales références du milieu avec une communauté très forte et réactive. Si vous avez une difficulté sur un langage en particulier ou sur la manière de procéder pour tel fonctionnement, StackOverflow est idéal pour trouver les réponses à nos questions.

VIII. EXTRAITS DE SITES ANGLOPHONES

Comme je l'ai expliqué précédemment, Internet propose de nombreuses ressources permettant d'aider les développeurs. Voici un extrait du site SymfonyCast qui m'a permis de rétablir ma base de données après une mauvaise manipulation de migration de base de données, que je commenterai par après :


```

30 lines | src/Migrations/Version20180418130337.php
... Lines 1 - 10
11 class Version20180418130337 extends AbstractMigration
12 {
13     public function up(Schema $schema)
14     {
15         // this up() migration is auto-generated, please modify it to your needs
16         $this->abortIf($this->connection->getDatabasePlatform()->getName() !== 'mysql', 'Migration can only be executed
17
18         $this->addSql('ALTER TABLE article ADD created_at DATETIME DEFAULT NULL, ADD updated_at DATETIME DEFAULT NULL');
19         $this->addSql('UPDATE article SET created_at = NOW(), updated_at = NOW()');
20     }
21 ... Lines 21 - 28
29 }

```

We *still* need another query to change things *back* to not null, but don't do it yet: we can be lazy. Instead, find your terminal: let's try the migration again. But, wait! You may or may *not* be able to re-run the migration immediately. In this case, the original migration had only *one* query, and that one query failed. This means that *no* part of the migration executed successfully.

But sometimes, a migration may contain *multiple* lines of SQL. And, if the second or third line fails, then, well, we're in a *really* weird state! In that situation, if we tried to *rerun* the migration, the first line would execute for the *second* time, and it would probably fail.

Basically, when a migration fails, it's possible that your migration system is now in an invalid state. *When* that happens, you should completely drop your database and start over. You can do that with:

```
$ php bin/console doctrine:database:drop --force
```

And then:

```
php bin/console doctrine:database:create
```

And *then* you can migrate. Anyways, we are *not* in an invalid state: so we can just re-try the migration:

```
$ php bin/console doctrine:migrations:migrate
```

Sur le premier extrait de cette page anglophone, il est écrit que parfois une migration peut contenir plusieurs lignes de SQL et que si l'une de ces lignes ne s'exécute pas, si nous réexécutons la migration, la première ligne peut s'exécuter une deuxième fois et cela peut poser problème.

Lorsqu'une migration échoue, il est possible que l'état du système de migration soit tronqué et lorsque cela arrive nous devrions supprimer la base de données avec la commande indiquée (`php bin/console doctrine:database:drop --force`) puis la recréer et effectuer à nouveau la migration.

Le deuxième extrait provient de la documentation officielle de Symfony traitant des bases de données et de l'ORM Doctrine. La documentation est celle de la version 3.3 de Symfony mais qui est toujours d'actualité au niveau de la manipulation :

Creating the Database Tables/Schema ¶

You now have a usable `Product` class with mapping information so that Doctrine knows exactly how to persist it. Of course, you don't yet have the corresponding `product` table in your database. Fortunately, Doctrine can automatically create all the database tables needed for every known entity in your application. To do this, run:

```
$ php bin/console doctrine:schema:update --force
```



Actually, this command is incredibly powerful. It compares what your database *should* look like (based on the mapping information of your entities) with how it *actually* looks, and executes the SQL statements needed to *update* the database schema to where it should be. In other words, if you add a new property with mapping metadata to `Product` and run this command, it will execute the "ALTER TABLE" statement needed to add that new column to the existing `product` table.

An even better way to take advantage of this functionality is via [migrations](#), which allow you to generate these SQL statements and store them in migration classes that can be run systematically on your production server in order to update and track changes to your database schema safely and reliably.

Whether or not you take advantage of migrations, the `doctrine:schema:update` command should only be used during development. It should not be used in a production environment.

Your database now has a fully-functional `product` table with columns that match the metadata you've specified.

Cette deuxième alternative à mon problème montre comment nous pouvons mettre à jour nos tables. Je cite " Heureusement, Doctrine peut créer automatiquement toutes les tables nécessaires pour chaque entité connue dans notre application."

Au moyen de cette seule ligne de commande " `php bin/console doctrine:schema:update --force`" Doctrine compare l'état de la base actuelle à celle qu'elle devrait ressembler (selon les informations des entités générées) et s'il repère un éventuel changement, il exécutera un traitement SQL ("ALTER TABLE") afin de mettre à jour notre base de données. Cependant, la meilleure façon de tirer parti de cette fonctionnalité est de procéder à des migrations qui nous permettent de générer ces instructions SQL et de les stocker dans des classes de migration qui peuvent être exécutées systématiquement sur votre serveur de production afin de mettre à jour et de suivre les modifications apportées à notre schéma de base de données en toute sécurité.

Cette ligne de commande devrait être utilisée uniquement durant la phase de développement et non en production (mise en ligne du site).

AXES D'AMELIORATION

Les axes d'amélioration du site Provoke restent nombreux, aussi bien côté Back-End que Front-End.

Back-End

Tout d'abord, concernant les fonctionnalités liées au Back-End par l'intermédiaire de Symfony et PHP :

Panier et confirmation d'achat -> Après la redirection sur notre boutique lorsque le client à valider son paiement sur la plate-forme de Mercanet, nous pourrions intégrer un système d'envoi de mails au client en lui indiquant le succès de l'opération avec en contenu le résumé de sa transaction. L'intégration d'une application PHP dans notre projet Symfony tel que Swift Mailer serait à étudier.

Front-end

Le projet n'ayant pas permis de personnaliser son design faute de temps, nous pourrions imaginer une refonte graphique sans thème bootstrap intégré en y injectant notre propre fichier CSS.

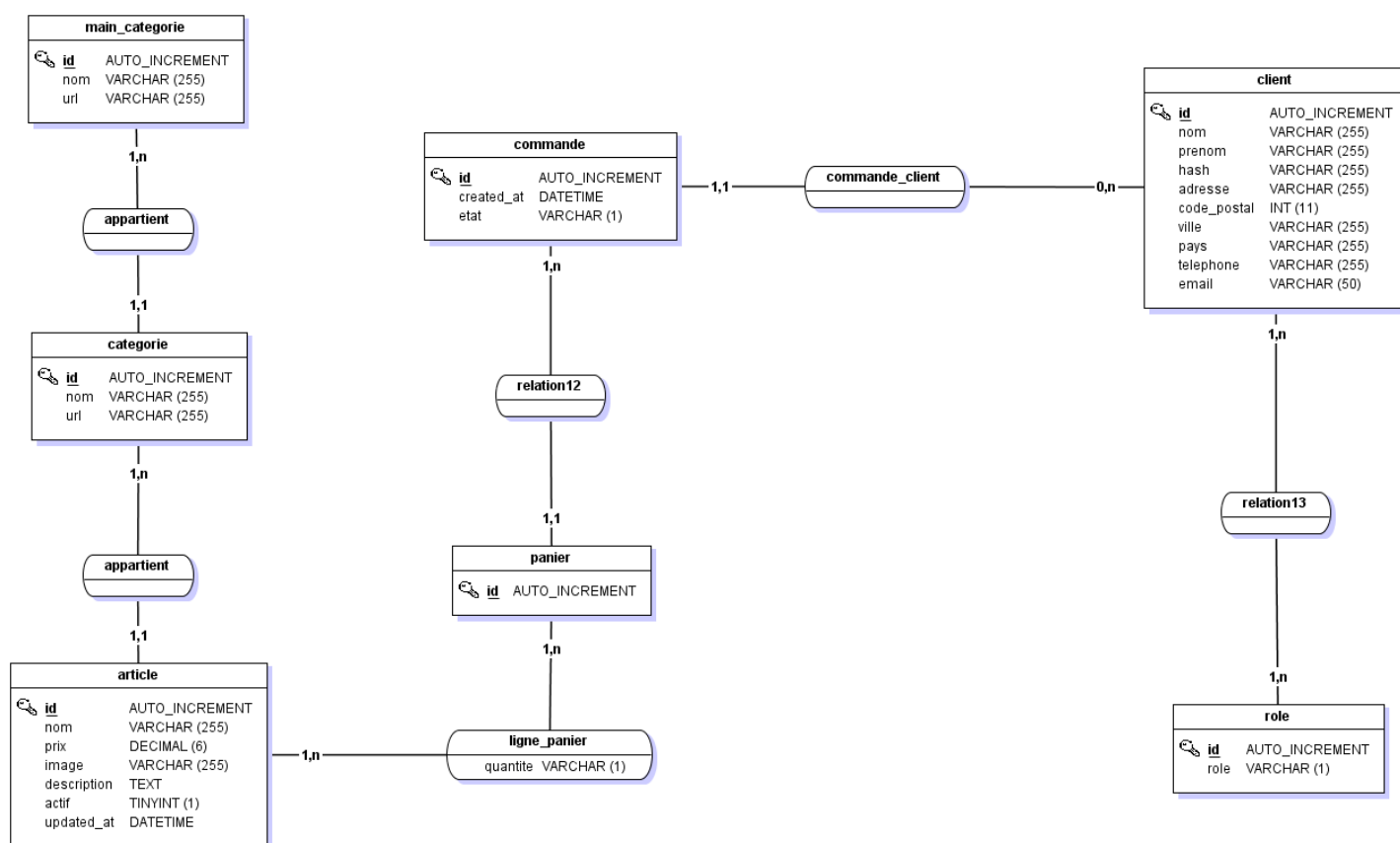
Pour une meilleure expérience client nous pourrions également harmoniser nos liens et boutons par l'intermédiaire de Javascript et l'un de ses frameworks tel que jQuery, Angular ou React JS afin de dynamiser nos pages côté client.

Grâce à Ajax, nous pouvons aussi travailler l'ajout au panier sans que le client ait à recharger la page, toujours dans le but d'améliorer l'expérience client.

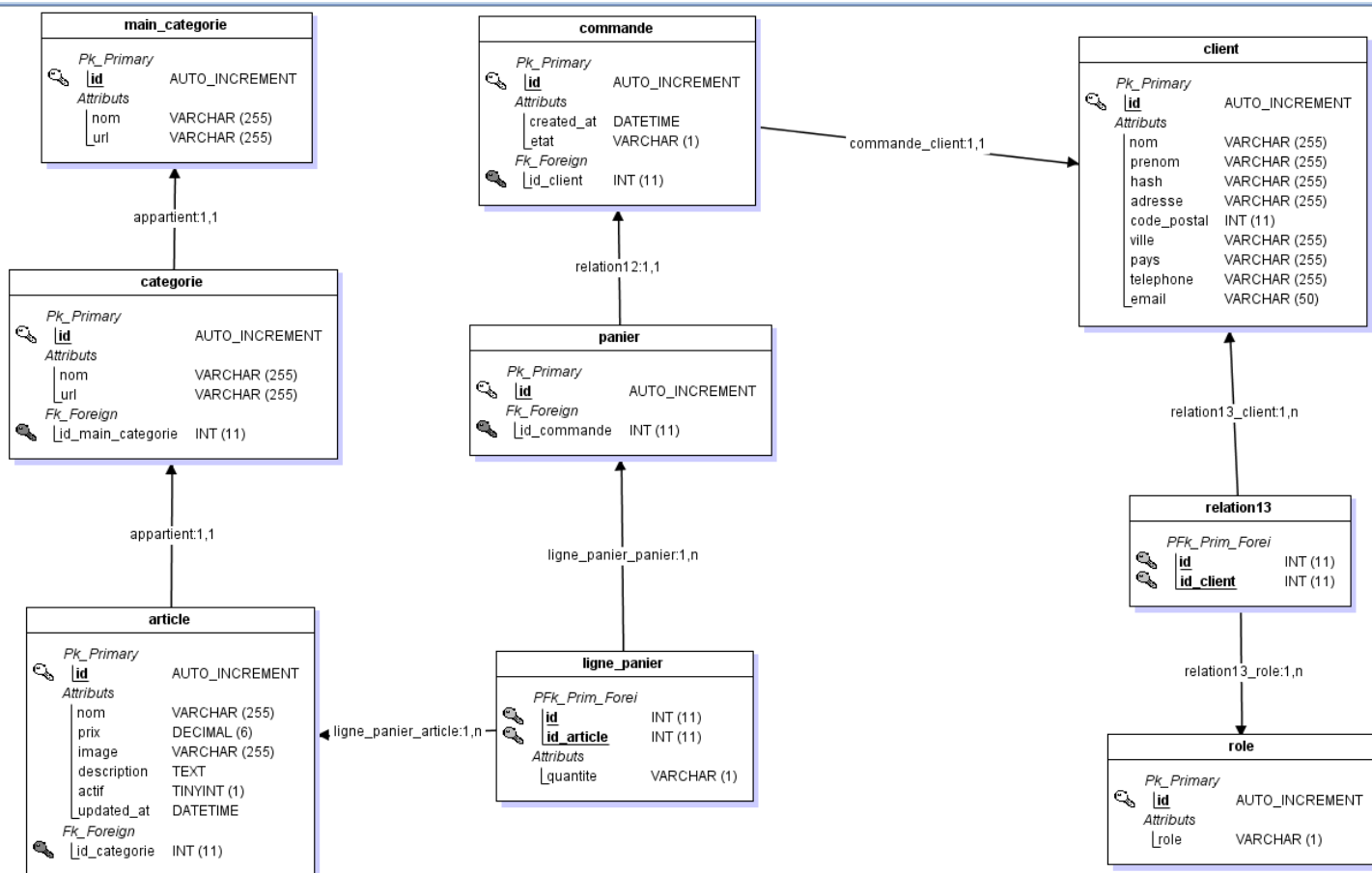
ANNEXES

Présentation du MCD et MLD

Schéma de représentation des entités à travers un modèle conceptuel de données MCD(entités, attributs et relation) défini par Merise :



Présentation des objets du MCD sous une forme compréhensible par un système de gestion de base de données, les objets formant des tables ainsi que les liens qui les unissent : C'est le MLD, Modèle Logique de Données qui est l'étape précédant la création de la base de données via Doctrine sous Symfony:



Schémas états-transitions

Schéma représentant les fonctionnalités de notre site depuis l'arrivée d'un utilisateur sur notre site jusqu'au processus d'achat :

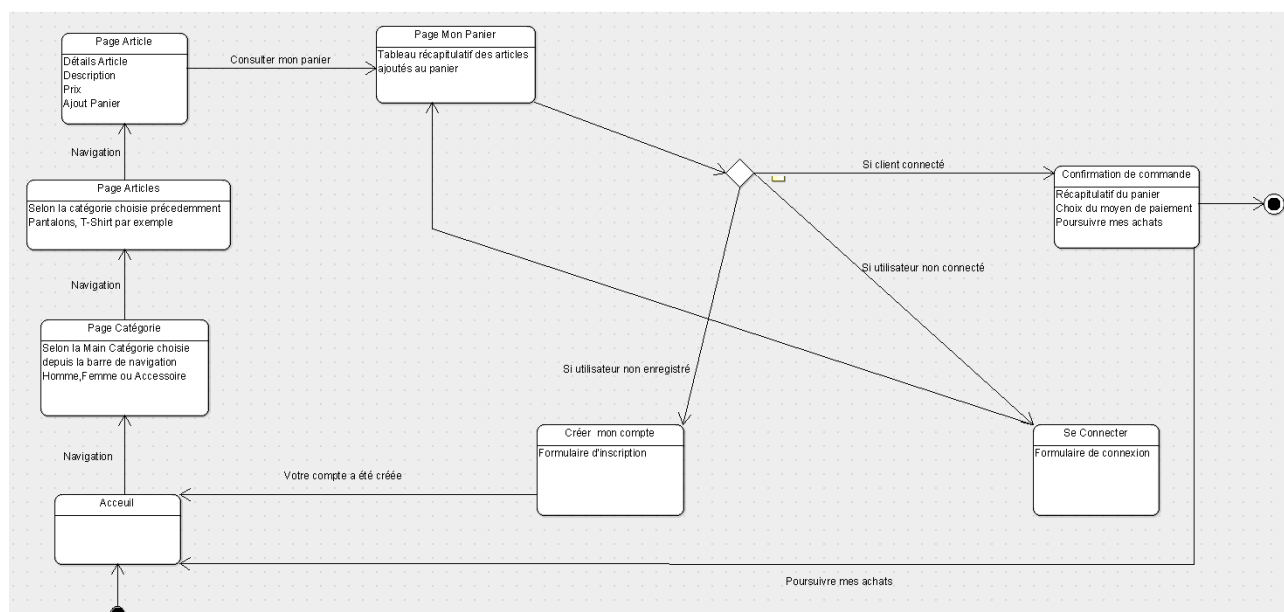


Schéma représentant les fonctionnalités de notre site depuis la page de confirmation de commande d'un utilisateur connecté à sa redirection vers les services de paiement sécurisée BNP Paribas Mercanet jusqu'à l'état du paiement :

