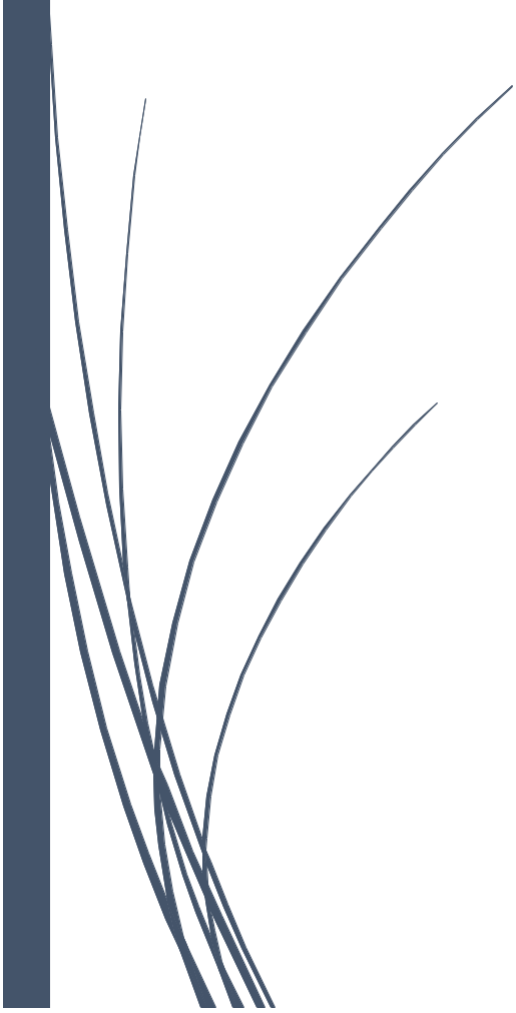
08/05/2021

PPE4

DOCUMENTATION API ET SITE WEB



Adrien FIGUERES
BTS SIO OPTION SLAM

SOMMAIRE

Introduction

Cahier des charges

L'Objectif est de créer un site WEB et une application mobile sur Android type API pour l'entreprise « Perpi&Co »

L'utilisateur pourras :

- S'inscrire sur le site
- S'authentifier avec son compte
- Ajouter au panier des produits
- Effectuer des commandes et payé via Paypal
- Se générer une facture .PDF
- Voir son profil et ses dernières commandes
- Evaluer des produits (Notation étoile et avis sous formes de commentaires)

L'administrateur pourras :

- Accéder à un back-office exclusif
- Gérer les commandes (Mais pas les supprimer)
- Gérer les produits
- Gérer les catégories de produits
- Gérer les données des clients
- Gérer les factures
- Hériter des droits inférieurs à lui (Hiérarchiquement l'administrateur et le plus haut rôle et visiteur / anonyme le plus bas il héritera de tous les droits de tous les rôles)

L'application mobile est gratuite et libre d'accès elle permettras de récupérer les informations en temps réel du site web tels que :

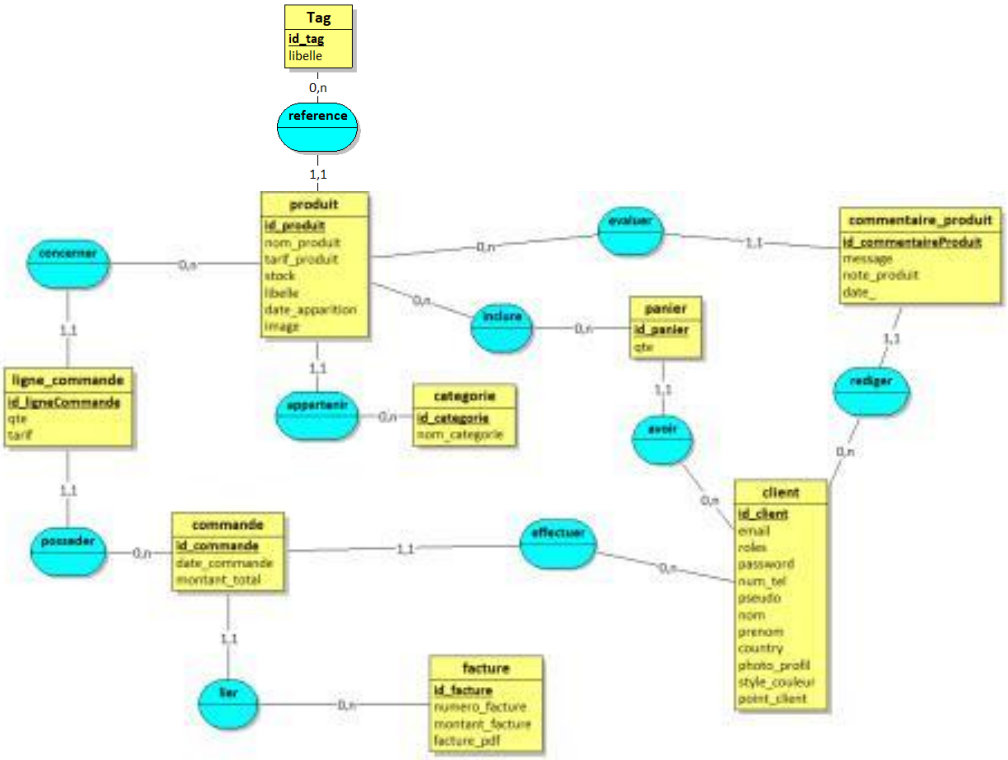
- Les produits
- Les clients (**ATTENTION** : les clients retournés auront certaines données pas afficher)

Tous cela hébergé sur un serveur docker sous Debian 9 (Linux) via machine virtuelle (VMWare).

Technique Base de Données

Modèle Conceptuel Des Données (MCD)

Pour réaliser un tel projet nous allons avoir besoin de réaliser une analyse réfléchie de la structure de notre base de données pour stocker les diverses informations.



Modèle Logique Des Données (MLD)

categorie = **(id_categorie**, nom_categorie**);**

client = **(id_client**, email, roles, password, num_tel, pseudo, nom, prenom, country, photo_profil,

style_couleur, point_client**);**

facture = **(id_facture**, numero_facture, montant_facture, facture_pdf**);**

commande = **(id_commande**, date_commande, montant_total, ***#id_client, #id_facture***);**

produit = **(id_produit**, nom_produit, tarif_produit, stock, libelle, date_apparition, image, ***#id_categorie, #id_tag***);**

commentaire_produit = **(id_commentaireProduit**, message, note_produit, date, ***#id_client, #id_produit***);**

panier = **(id_panier**, qte, ***#id_client***);**

ligne_commande = **(id_ligneCommande**, qte, tarif, ***#id_produit, #id_commande***);**

inclure = **(*#id_produit, #id_panier*);**

Script SQL

Voici le script SQL qui va permettre de conçevoir notre base de données si on le fait manuellement, cependant à savoir que le framework Symfony que nous allons utiliser gère la création des tables, Entités (Utilisation de Doctrine)

```
CREATE TABLE categorie(
  id_categorie VARCHAR(50),
  nom_categorie VARCHAR(50),
  PRIMARY KEY(id_categorie)
);

CREATE TABLE client(
  id_client VARCHAR(50),
  email VARCHAR(50),
  roles VARCHAR(50),
  password VARCHAR(50),
  num_tel VARCHAR(50),
  pseudo VARCHAR(50),
  nom VARCHAR(50),
  prenom VARCHAR(50),
  country VARCHAR(50),
  photo_profil VARCHAR(50),
  style_couleur VARCHAR(50),
  point_client VARCHAR(50),
  PRIMARY KEY(id_client)
);

CREATE TABLE facture(
  id_facture VARCHAR(50),
  numero_facture VARCHAR(50),
  montant_facture VARCHAR(50),
  facture_pdf VARCHAR(50),
  PRIMARY KEY(id_facture)
);

CREATE TABLE commande(
  id_commande VARCHAR(50),
  date_commande VARCHAR(50),
  montant_total VARCHAR(50),
  id_client VARCHAR(50) NOT NULL,
  id_facture VARCHAR(50) NOT NULL,
  PRIMARY KEY(id_commande),
  FOREIGN KEY(id_client) REFERENCES client(id_client),
  FOREIGN KEY(id_facture) REFERENCES facture(id_facture)
);

CREATE TABLE produit(
  id_produit VARCHAR(50),
  nom_produit VARCHAR(50),
  tarif_produit VARCHAR(50),
  stock VARCHAR(50),
  libelle VARCHAR(50),
  date_apparition VARCHAR(50),
  image VARCHAR(50),
  id_categorie VARCHAR(50) NOT NULL,
  PRIMARY KEY(id_produit),
  FOREIGN KEY(id_categorie) REFERENCES categorie(id_categorie)
  Id_tag INT NOT NULL,
  FOREIGN KEY(id_tag) REFERENCES tag(id),
  );
```

```
CREATE TABLE tag (
   Id INT (50) AUTO-INCREMENT NOT NULL,
   Libelle VARCHAR(50),
   PRIMARY KEY(id),
):

CREATE TABLE commentaire_produit(
  id_commentaireProduit VARCHAR(50),
  message VARCHAR(50),
  note_produit VARCHAR(50),
  date_ VARCHAR(50),
  id_client VARCHAR(50) NOT NULL,
  id_produit VARCHAR(50) NOT NULL,
  PRIMARY KEY(id_commentaireProduit),
  FOREIGN KEY(id_client) REFERENCES client(id_client),
  FOREIGN KEY(id_produit) REFERENCES produit(id_produit)
);

CREATE TABLE panier(
  id_panier VARCHAR(50),
  qte VARCHAR(50),
  id_client VARCHAR(50) NOT NULL,
  PRIMARY KEY(id_panier),
  FOREIGN KEY(id_client) REFERENCES client(id_client)
);

CREATE TABLE ligne_commande(
  id_ligneCommande VARCHAR(50),
  qte VARCHAR(50),
  tarif VARCHAR(50),
  id_produit VARCHAR(50) NOT NULL,
  id_commande VARCHAR(50) NOT NULL,
  PRIMARY KEY(id_ligneCommande),
  FOREIGN KEY(id_produit) REFERENCES produit(id_produit),
  FOREIGN KEY(id_commande) REFERENCES commande(id_commande)
);

CREATE TABLE inclure(
  id_produit VARCHAR(50),
  id_panier VARCHAR(50),
  PRIMARY KEY(id_produit, id_panier),
  FOREIGN KEY(id_produit) REFERENCES produit(id_produit),
  FOREIGN KEY(id_panier) REFERENCES panier(id_panier)
);
```

Technique côté serveur

Installation de Docker

Après avoir installé et configurer notre machine virtuelle sous VMWare nous pouvons désormais commencer notre procédure d'installation du serveur Docker.

Nous allons monter un serveur Docker qui comportera trois containers :

1) WEB (Port 8082)
2) PhpMyAdmin (Port 8083)
3) MySQL (Port 3306)

Pour être efficace il est possible à partir de Git de cloner un repo dans le répertoire de notre choix dans linux, pour lequel ce dernier contiens tous les fichiers de bases permettant de mettre en place tous les services précédemment cités voici le lien du Git :

https://github.com/cnadal/machine_docker

Lorsque l'installation est finie et que tous les fichiers ont bien été configurer pour notre projet (Nom du projet, adresse ip, mysql user et pass etc…) on peut exécuter avec la commande « docker-compose up -d » nos trois services.

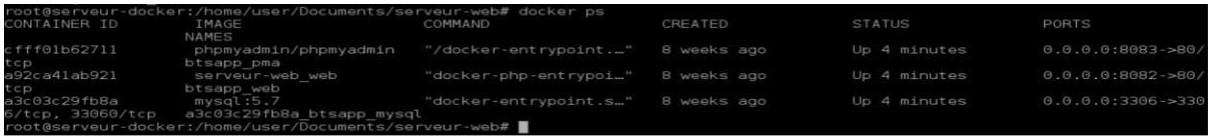
Ce qui nous donne le résultat suivant :

```
root@serveur-docker:/home/user/Documents/serveur-web# docker-compose up -d
Starting a3c03c29fb8a_btsapp_mysql ... done
Starting btsapp_pma                ... done
Starting btsapp_web                ... done
root@serveur-docker:/home/user/Documents/serveur-web#
```

Création du projet symfony dans notre serveur

Bien dès lors que nous avons lancé nos trois services via la commande : « docker-compose up -d »

Lorsque l'on effectue « docker ps » la commande nous renvoie les détails des trois services :



Ce qui va nous intéresser ici c'est le container web, nous allons de ce pas nous redirigez à la racine du container (var/www/html) pour ce faire il faut exécuter la commande suivante : « docker exec -it {NomDuContainerWeb} bash »

Puis lorsque l'on est à l'intérieur on va venir créer notre projet Symfony avec la commande suivante :

« symfony new {NomDuProjet} –full »

Puis comme la capture d'écran on se dirige dans le projet que l'on a créé :

Et on installe apache pack pour avoir le fichier .htaccess (Très important)

« Composer req symfony/apache-pack »

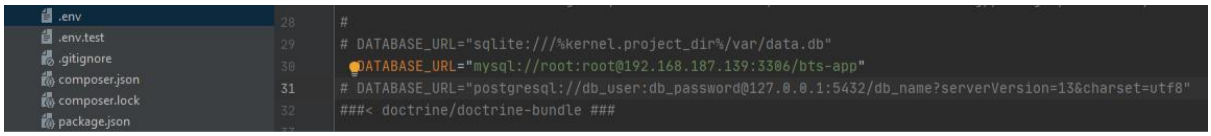
Et on peut désormais tester notre projet

http://ADRIP:8082/NOMPROJET/public/

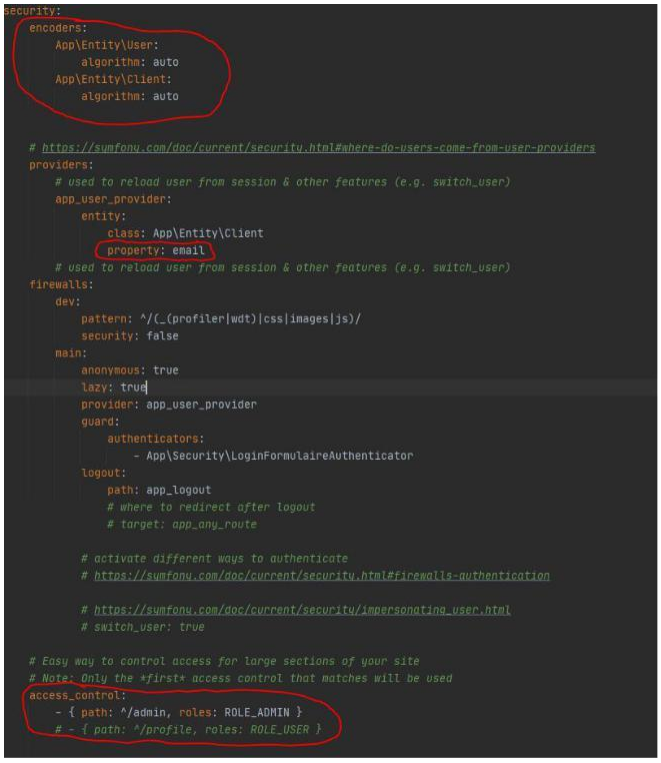
Technique programmation WEB

Configurations

Configuration sous phpstorm du fichier. env :



Configuration du fichir security.yaml de symfony :



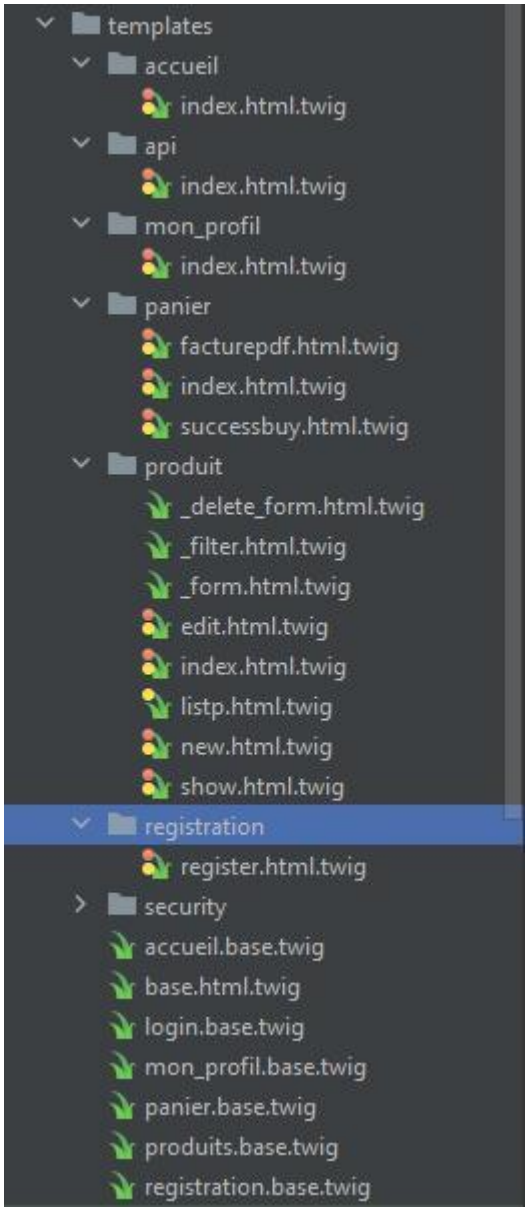
(Tous en haut on va venir définir l'algorithme d'encodage souvent utilisé pour les mots de passe lorsque l'on définit en auto sous Symfony 5 on est en Argon2id on peut définir MD5, SHA ou autre type d'encodage.

Tous en bas au niveau du « access_control » je viens bloquer et/ou autoriser des routes pour un certains type d'utilisateur par exemple on peut voir que sur mon site j'ai autorisé la route /admin uniquement pour le rôle administrateur, les autres rôles voulant y accéder auront un message d'erreur)

Les templates (Les vues)

Les Templates sous Symfony sont des pages HTML qui sont vue par l'utilisateur, cependant il y'a une subtilité qui se rajoute en effet ces Templates ont une particularité c'est qu'ils utilisent « TWIG » un langage ou plutôt un moteur qui rend le php plus performant est sécurisé la documentation détaillée de TWIG est visible sur leurs sites officiels : https://twig.symfony.com/

Voici les différents Templates existant sur mon site :

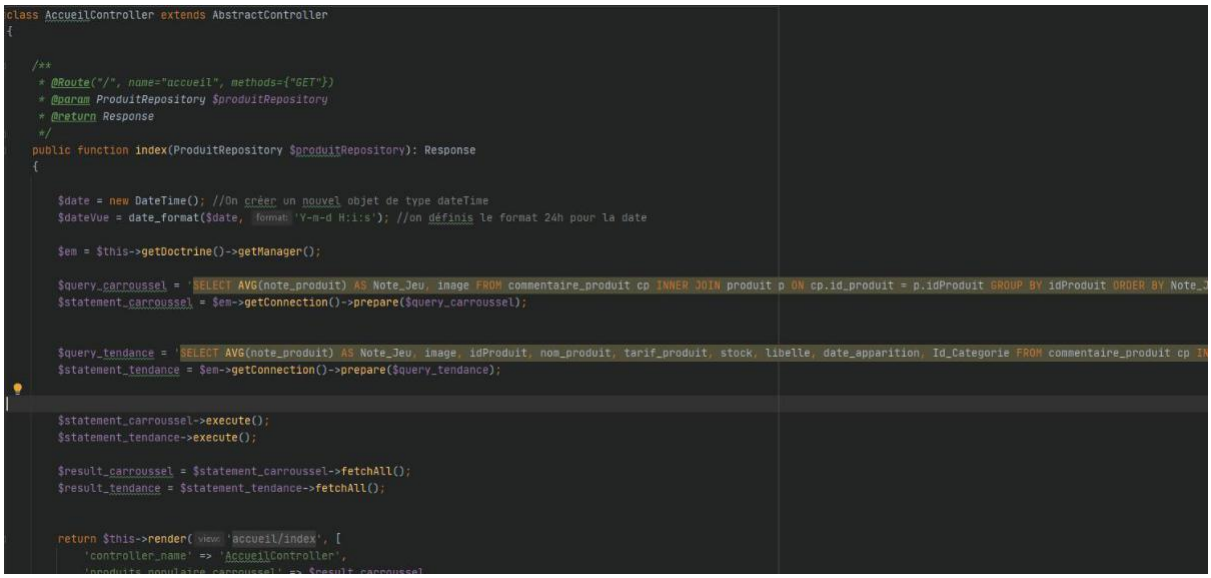


- **Accueil :** Concerne la page d'accueil du site
- **Api :** Concerne l'API mobile (*Cependant elle n'est pas accessible normalement par l'utilisateur détaillé plus tard dans la documentation*)
- **Mon_profil :** Affiche le profil de l'utilisateur connecté (*Customisable et possibilité de voir les récentes commandes passé sur le site*)
- **Panier :** Concerne le panier pour effectuer des achats sur le site
- **Produit :** Concerne toutes les pages pour les produits (*Liste des produits, détails d'un produit etc…*)
- **Registration :** Concerne le formulaire d'inscription
- **Security :** Concerne le formulaire de connexion

Puis, plus bas tous les Templates de bases pour chaque page.

Les contrôleurs

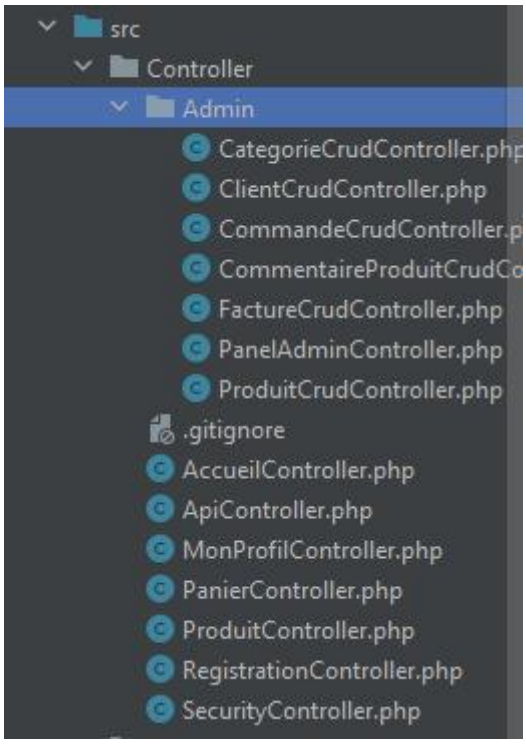
Les contrôleurs sont des classes PHP dans lequel on va définir plusieurs fonctions qui vont servir de routes (D'URL) par exemple pour la page d'accueil :



```php
class AccueilController extends AbstractController
{

    /**
     * @Route("/", name="accueil", methods={"GET"})
     * @param ProduitRepository $produitRepository
     * @return Response
     */
    public function index(ProduitRepository $produitRepository): Response
    {

        $date = new DateTime(); //On créer un nouvel objet de type dateTime
        $dateVue = date_format($date, format: 'Y-m-d H:i:s'); //on définis le format 24h pour la date

        $em = $this->getDoctrine()->getManager();

        $query_carroussel = 'SELECT AVG(note_produit) AS Note_Jeu, image FROM commentaire_produit cp INNER JOIN produit p ON cp.id_produit = p.idProduit GROUP BY idProduit ORDER BY Note_
        $statement_carroussel = $em->getConnection()->prepare($query_carroussel);


        $query_tendance = 'SELECT AVG(note_produit) AS Note_Jeu, image, idProduit, nom_produit, tarif_produit, stock, libelle, date_apparition, Id_Categorie FROM commentaire_produit cp IN
        $statement_tendance = $em->getConnection()->prepare($query_tendance);

        $statement_carroussel->execute();
        $statement_tendance->execute();

        $result_carroussel = $statement_carroussel->fetchAll();
        $result_tendance = $statement_tendance->fetchAll();


        return $this->render( view: 'accueil/index', [
            'controller_name' => 'AccueilController',
            'produits_populaire_carroussel' => $result_carroussel
```

L'annotation dans le « commentaire » *@Route('/' ……..)* signifie que si l'on tape l'adresse de notre site puis on rajoute après le « public/ » on sera rediriger vers la page d'accueil car le contrôleur retourne la vue (le Template) qui concerne l'accueil

Voici les différents contrôleurs du site web :



Tous le répertoire **Admin** concerne le bundle EasyAdmin pour le backoffice Administrateur

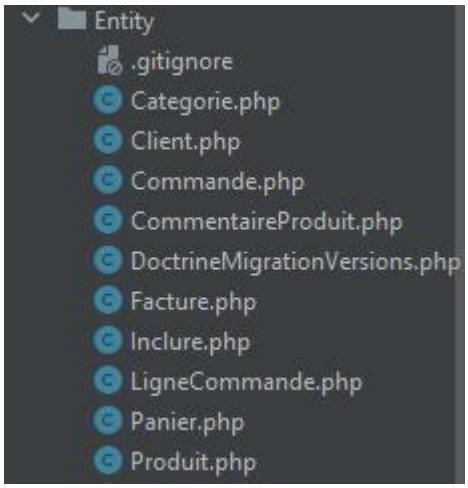
Puis on un contrôleur pour chaque vue Template.

Les entités

Les entités sont représente nos tables, en effet avec symfony il est possible de créer nos entités directement avec la commande « bin/console make :entity » puis de renseigner les champs, leurs type (varchar, int, float, double …) si ce champ est en relation avec un autre champ d'une autre entité (Clé étrangère) de quelle type (ManyToOne, ManyToMany ….).

Il est possible à partir d'une base de données existante de l'importer en entités Symfony on appelle ça du reverse engineering.

Voici les différentes entités :

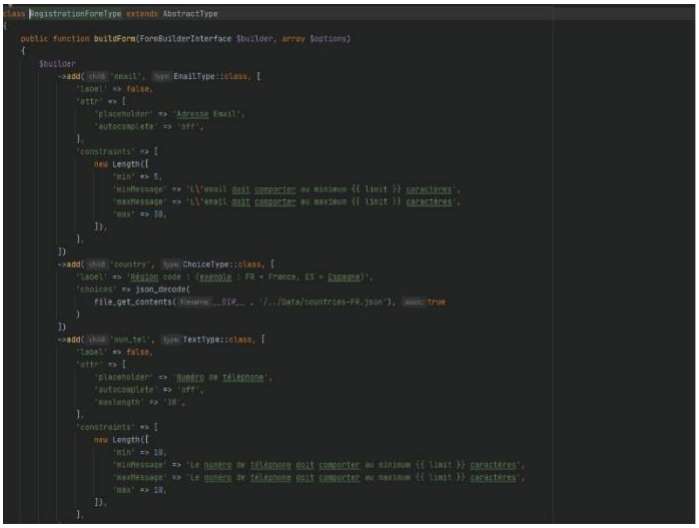


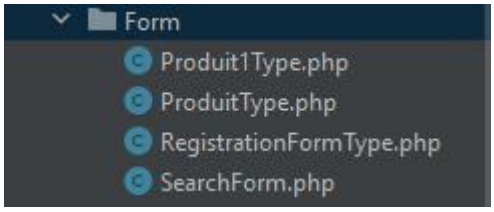
On retrouve les entités du MCD

Les formulaires (Symfony)

Avec Symfony il est possible de créer des formulaires grâce au « FormBuilder » :

Voici un exemple de formulaire qui concerne le formulaire d'inscription :





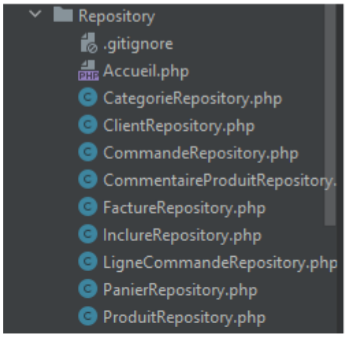
Voici les formulaires existants.

Les repository

Les repository sont des classes assez importantes car elles permettent d'effectuer des requêtes SQL mais attention qui dit Symfony dit plein de fonctionnalités et grâce à la magie des Query Builder et de son efficacité on peut effectuer des actions sur notre base de données comme on le souhaite.

Voici un exemple pour le repository listant les produits (Avec filtrage)

```php
/**
 * Récupère les produits en fonction d'une recherche
 * @param SearchData $search
 * @param $date
 * @return PaginationInterface
 */
public function findSearch(SearchData $search, $date): PaginationInterface
{
    $query = $this
        ->createQueryBuilder( alias: 'p')
        ->where( predicates: 'p.dateApparition < \'' . $date . '\'')
        ->andWhere('p.stock > 0')
        ->select( select: 'c', 'p')
        ->join( join: 'p.id_categorie', alias: 'c');

    if(!empty($search->keyword)){
        $query = $query
            ->andWhere('p.nomProduit LIKE :keyword')
            ->setParameter( key: 'keyword', value: "%{$search->keyword}%");
    }

    if(!empty($search->min_price)){
        $query = $query
            ->andWhere('p.tarifProduit >= :min_price')
            ->setParameter( key: 'min_price', $search->min_price);
    }

    if(!empty($search->max_price)){
        $query = $query
            ->andWhere('p.tarifProduit <= :max_price')
            ->setParameter( key: 'max_price', $search->max_price);
    }

    if(!empty($search->categories)){
        $query = $query
            ->andWhere('c.idCategorie IN (:categories)')
            ->setParameter( key: 'categories', $search->categories);
    }

    $query = $query->getQuery();

    return $this->paginator->paginate(
        $query,
```



Voici les différents repository présent dans le site

Technique programmation Android

Configuration des routes API

Avant de concevoir notre application mobile nous allons nous demander quelles sont les informations que nous voulons retournés :

- Tous les produits en tendances sur le site
- Tous les produits récemment ajoutés
- Tous les produits qui vont prochainement sortir et achetable
- Tous les produits existants
- Le montant total de tous les produits cumulés sur le site (Valeur stock)
- Le nombre de clients inscrits
- Le nombre de clients ayant effectué au moins une commande sur le site

Voici la liste des différentes fonctions créer en php pour chaque élément cité ci-dessus :

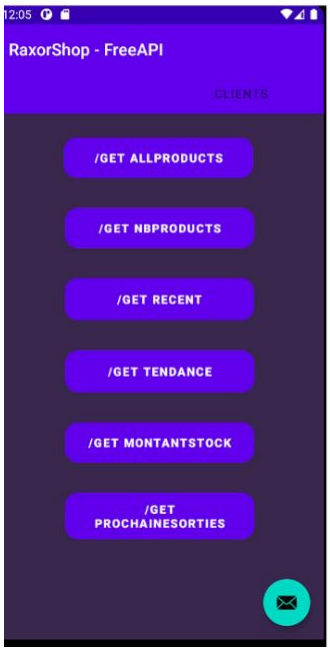
- webServiceAllProducts()
- webServiceAllRecentsProducts()
- webServiceAllTendanceProducts()
- webServiceComingSoonProducts()
- webServiceCountTotalProductss()
- webServiceValuesStockProducts()
- webServiceAllClients()
- webServiceNbClientsInscrits()
- webServiceNbClientsCommandes()

Programmation Android

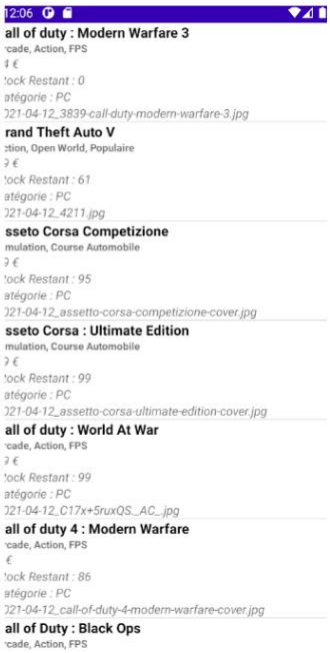
Au niveau de Android nous allons programmer en JAVA. L'objectif est d'appeler nos fonctions qui retourne des valeurs formats JSON et d'afficher ces résultats sous une formes compréhensibles par l'utilisateur au niveau de l'application.

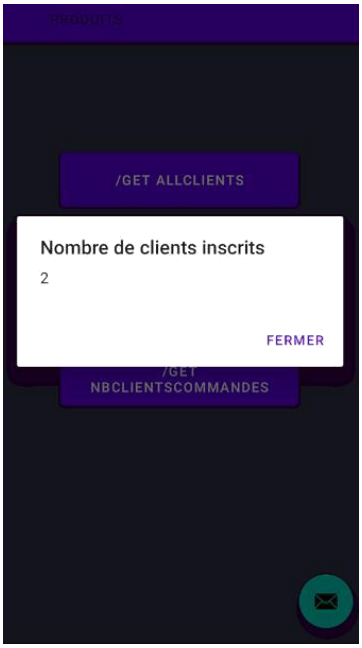
Voilà un exemple ou l'on récupère tous les produits :

```
protected void onStart() {
    super.onStart();

    //le contexte est l'instance (this) de l'activity APIActivity
    RequestQueue queue = Volley.newRequestQueue( context: GetAllProducts_Activity.this);

    //url du service à consommer
    String url = v.IPServeur + v.API_FetchApiAllProducts;

    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    //parcours des Clients retournés
                    //ArrayList<String> lesClients = new ArrayList<String>();
                    ArrayList <HashMap <String, String >> listAllProducts = new ArrayList<>();

                    SimpleAdapter adapter = new SimpleAdapter( context: GetAllProducts_Activity.this, listAllProducts, R.layout.custom_row_view_products, new String[] {"nom", "libelle", "tarifProduit", "stock
                    JSONArray jsonArray = null;
                    try {
                        jsonArray = new JSONArray(response);
                        for (int i = 0; i < jsonArray.length(); i++) {
                            JSONObject item = jsonArray.getJSONObject(i);

                            String nomPrd = item.getString( name: "nomProduit");
                            String libelle = item.getString( name: "libelle");
                            String tarifProduit = item.getString( name: "tarifProduit");
                            String stock = item.getString( name: "stock");
                            String nomCategorie = item.getString( name: "nomCategorie");
                            String image = item.getString( name: "image");

                            HashMap <String, String> temp = new HashMap <> ();
                            temp.put("nom", nomPrd);
                            temp.put("libelle", libelle);
                            temp.put("tarifProduit", tarifProduit + " €");
                            temp.put("stock", "Stock Restant : " + stock);
                            temp.put("nomCategorie", "Catégorie : " + nomCategorie);
                            temp.put("image", image);
                            listAllProducts.add(temp);
                        }
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                    //Remplissage de la listview
                    setListAdapter(adapter);
                }
            }, new Response.ErrorListener() {       // CAS d'ERREUR
        @Override
```



/GET ALLPRODUCTS





A gauche voici un exemple lorsque l'on récupère le nombre de clients inscrits sur le site.

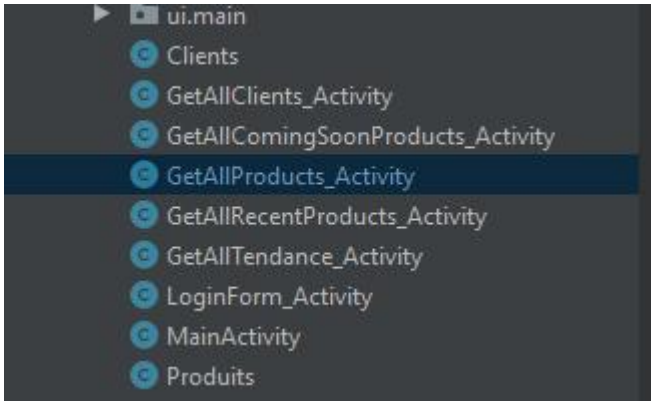
Information : Les variables globales concernant les url de l'API sont stockées dans cette classe

```
public class VariablesGlobales {

    String IPServeur = "http://192.168.187.139:8082/bts-app/public";
    String API_FetchAllClients = "/ApiAllClients";
    String API_FetchAllComingSoonPrd = "/ApiAllCommingSoonProducts";
    String API_FetchApiAllProducts = "/ApiAllProducts";
    String API_FetchApiAllRecentProducts = "/ApiAllRecentProducts";
    String API_FetchApiAllTendanceProducts = "/ApiAllTendanceProducts";
    String API_FetchApiCountTotalProducts = "/ApiCountTotalProducts";
    String API_FetchApiValueStockProducts = "/ApiValueStockProducts";
    String API_FetchApiNbClientsCommandes = "/ApiNbClientsCommandes";
    String API_ApiNbClientsInscrits = "/ApiNbClientsInscrits";
}
```

Les Activity

Une Activity est à la vue, la fenêtre que voit l'utilisateur.

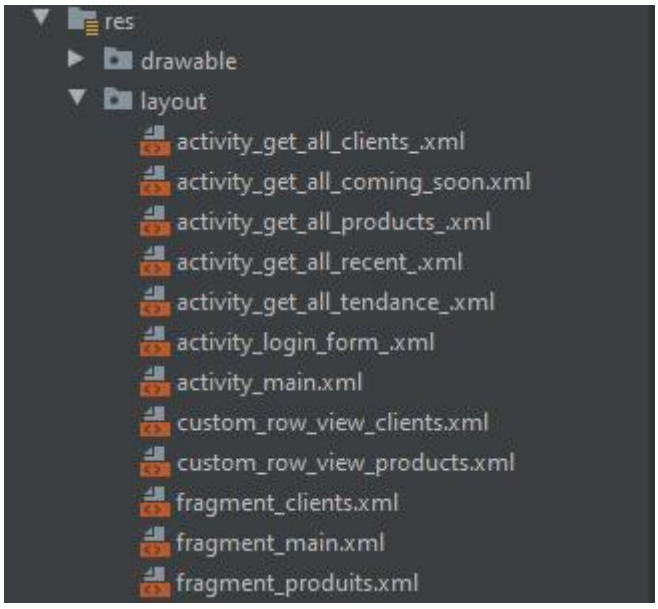
Voici les différentes Activity présentes :



Les Layouts et Fragments

Les Layouts concerne les designs, les contraintes des différents composants de chaque Activity.

Voici les différents Layouts et Fragments de l'application mobile :



Information créateur

Crédits

Créateur : Adrien FIGUERES

Age : 20 Ans

Statut : Etudiant

Formation : BTS SIO 2nd année en option SLAM

Stockage des projets : https://github.com/adriraxor