Rapport d’Architecture Web

**Bar Manager**



1 Table des matières

[2 Spécifications 4](#_Toc28625896)

[2.1 Objectif du site 4](#_Toc28625897)

[2.1.1 Acteurs 4](#_Toc28625898)

[2.1.2 User stories 4](#_Toc28625899)

[2.1.3 Contraintes non fonctionnelles 5](#_Toc28625900)

[2.1.4 Mockup 5](#_Toc28625901)

[3 Base de données 7](#_Toc28625902)

[3.1 Diagramme Entité relation 7](#_Toc28625903)

[3.2 Diagramme relationnel 8](#_Toc28625904)

[3.3 Description des tables et de leurs relations 8](#_Toc28625905)

[3.4 Description précise des champs 8](#_Toc28625906)

[3.4.1 Table Command 8](#_Toc28625907)

[3.4.2 Table Client 9](#_Toc28625908)

[3.4.3 Table Product 9](#_Toc28625909)

[4 MVC – Symfony 10](#_Toc28625910)

[4.1 En tant que barman je veux pouvoir supprimer un client 10](#_Toc28625911)

[4.1.1 Supprimer un client 10](#_Toc28625912)

[4.1.2 Gestion des erreurs et de la sécurité 10](#_Toc28625913)

[4.1.3 Diagramme de séquence -> enchainement des scripts 11](#_Toc28625914)

[4.1.4 Scripts concernés (lien vers github) 11](#_Toc28625915)

[4.2 En tant que barman je veux pouvoir ajouter un client 12](#_Toc28625916)

[4.2.1 Ajouter un client 12](#_Toc28625917)

[4.2.2 Gestion des erreurs et de la sécurité 12](#_Toc28625918)

[4.2.3 Diagramme de séquence -> enchainement des scripts 13](#_Toc28625919)

[4.2.4 Code significatif 13](#_Toc28625920)

[4.2.5 Scripts concernés (lien vers github) 14](#_Toc28625921)

[4.3 En tant que barman, je veux pouvoir débiter/créditer le compte d’un client 14](#_Toc28625922)

[4.3.1 Débiter/créditer le compte d’un client 14](#_Toc28625923)

[4.3.2 Gestion des erreurs et de la sécurité 15](#_Toc28625924)

[4.3.3 Diagramme de séquence -> enchainement des scripts 15](#_Toc28625925)

[4.3.4 Code significatif 15](#_Toc28625926)

[4.3.5 Scripts concernés (lien vers github) 16](#_Toc28625927)

[4.4 En tant que barman, je veux pouvoir consulter le solde d’un client 17](#_Toc28625928)

[4.4.1 Consulter le solde d’un client 17](#_Toc28625929)

[4.4.2 Gestion des erreurs et de la sécurité 17](#_Toc28625930)

[4.4.3 Diagramme de séquence -> enchainement des scripts 18](#_Toc28625931)

[4.4.4 Code significatif 18](#_Toc28625932)

[4.4.5 Scripts concernés (lien vers github) 18](#_Toc28625933)

[4.5 En tant que barman, je veux pouvoir encoder la commande d’un client 19](#_Toc28625934)

[4.5.1 Encoder la commande d’un client 19](#_Toc28625935)

[4.5.2 Gestion des erreurs et de la sécurité 19](#_Toc28625936)

[4.5.3 Diagramme de séquence -> enchainement des scripts 20](#_Toc28625937)

[4.5.4 Code significatif 21](#_Toc28625938)

[4.5.5 Scripts concernés (lien vers github) 23](#_Toc28625939)

[4.6 En tant que barman, je veux pouvoir indiquer si le client paye sa commande cash ou avec son compte 24](#_Toc28625940)

[4.6.1 Payer en cash/compte 24](#_Toc28625941)

[4.6.2 Gestion des erreurs et de la sécurité 25](#_Toc28625942)

[4.6.3 Diagramme de séquence -> enchainement des scripts 25](#_Toc28625943)

[4.6.4 Code significatif 25](#_Toc28625944)

[4.6.5 Scripts concernés (lien vers github) 26](#_Toc28625945)

[4.7 En tant que barman je veux pouvoir supprimer le panier du client 27](#_Toc28625946)

[4.7.1 Supprimer le panier du client 27](#_Toc28625947)

[4.7.2 Gestion des erreurs et de la sécurité 27](#_Toc28625948)

[4.7.3 Code significatif 27](#_Toc28625949)

[4.7.4 Scripts concernés (lien vers github) 28](#_Toc28625950)

[4.8 En tant que patron, je veux pouvoir consulter le chiffre d’affaire et les commandes 28](#_Toc28625951)

[4.8.1 Consulter le chiffre d’affaire & les commandes 28](#_Toc28625952)

[4.8.2 Gestion des erreurs et de la sécurité 29](#_Toc28625953)

[4.8.3 Code significatif 29](#_Toc28625954)

[4.8.4 Scripts concernés (lien vers github) 30](#_Toc28625955)

[4.9 En tant que barman et patron, je veux pouvoir ajouter un produit 30](#_Toc28625956)

[4.9.1 Ajouter un produit 30](#_Toc28625957)

[4.9.2 Gestion des erreurs et de la sécurité 31](#_Toc28625958)

[4.9.3 Diagramme de séquence -> enchainement des scripts 31](#_Toc28625959)

[4.9.4 Code significatif 31](#_Toc28625960)

[4.9.5 Scripts concernés 32](#_Toc28625961)

[4.10 En tant que barman et patron, je veux pouvoir supprimer un produit 32](#_Toc28625962)

[4.10.1 Supprimer un produit 32](#_Toc28625963)

[4.10.2 Gestion des erreurs et de la sécurité 33](#_Toc28625964)

[4.10.3 Code significatif 33](#_Toc28625965)

[4.10.4 Scripts concernés 33](#_Toc28625966)

[4.11 En tant que patron, je veux pouvoir consulter les ventes des différentes consommations 34](#_Toc28625967)

[4.11.1 Affichage des ventes des différentes consommations 34](#_Toc28625968)

[4.11.2 Gestion des erreurs et de la sécurité 34](#_Toc28625969)

[4.11.3 Code significatif 34](#_Toc28625970)

[4.11.4 Scripts utilisés 35](#_Toc28625971)

[5 Concusion 35](#_Toc28625972)

[5.1 Difficultés rencontrés 35](#_Toc28625973)

[5.2 Objectifs atteints/non atteints 36](#_Toc28625974)

[5.3 Eléments intéressants 36](#_Toc28625975)

[5.4 Pistes d’amélioration 37](#_Toc28625976)

# 2 Spécifications

## 2.1 Objectif du site

Le site doit permettre à un barman de gérer la consommation de ces clients facilement. Le client peut décider de peut déposer une somme d’argent sur son compte dans le bar. Le patron doit pouvoir consulter les ventes en inscrivant sont mot de passe

### 2.1.1 Acteurs

**Barman** : personne qui encode des nouveaux clients, ajoute leur solde et indique la consommation choisit par le client. Il est le principal utilisateur de l’application.

**Client** : personne qui commande une consommation chez le barman

**Patron** : personne à qui appartient le bar et peut consulter les commandes. Il est un utilisateur secondaire de l’application.

### 2.1.2 User stories

|  |  |
| --- | --- |
| **User stories** | **Priorité** |
| En tant que barman, je veux pouvoir ajouter/supprimer un client | 1 |
| En tant que barman, je veux pouvoir débiter/créditer le compte d’un client | 1 |
| En tant que barman, je veux pouvoir consulter le solde sur le compte d’un client | 1 |
| En tant que barman, je veux pouvoir encoder la commande d’un client | 1 |
| En tant que barman, je veux pouvoir indiquer si le client paye sa commande cash ou avec son compte | 1 |
| En tant que barman je veux pouvoir supprimer le panier du client | 1 |
| En tant que patron, je veux pouvoir consulter le chiffre d’affaire et les commandes | 1 |
| En tant que barman et patron, je veux pouvoir ajouter/supprimer un produit | 2 |
| En tant que patron, je veux pouvoir consulter les ventes des différentes consommations | 2 |

### 2.1.3 Contraintes non fonctionnelles

* L’application doit être ergonomique
* L’application doit augmenter la rapidité de gestion des commandes d’un bar en automatisant ceux-ci

### 2.1.4 Mockup

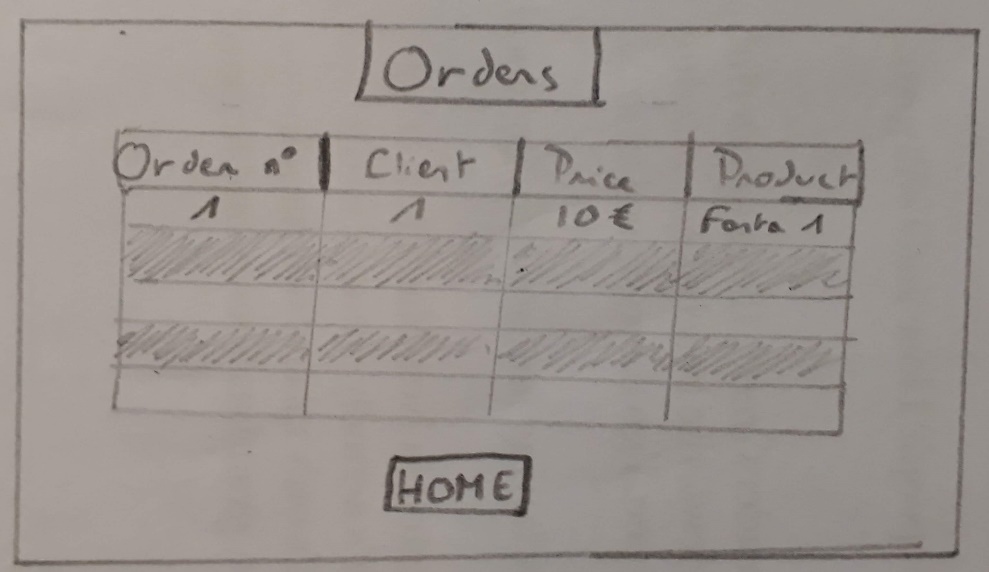
Voici les mockup initialement imaginé (réalisé à la main) :

#### 2.1.4.1 Page d’accueil

#### 2.1.4.2 Panier interactif

#### 2.1.4.3 Ajouter un client

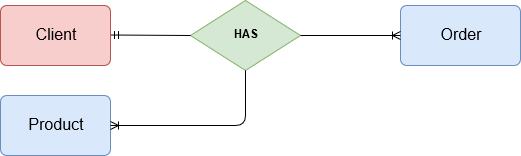
#### 2.1.4.3 Consulter les commandes



#### 2.1.4.4 Ajouter un produit

# 3 Base de données

## 3.1 Diagramme Entité relation



Command

## 3.2 Diagramme relationnel

## 3.3 Description des tables et de leurs relations

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Table** | **Type** | **Description** |
| Command | Client | N à 1 | Un client peut passer plusieurs commandes, plusieurs commandes peuvent avoir le même client |
| Command | Product | N à N | Plusieurs commandes contiennent plusieurs produits et plusieurs produits peuvent se retrouver dans différentes commandes |

## 3.4 Description précise des champs

### 3.4.1 Table Command

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nom** | **Key** | **Non null** | **Type** | **Référence** | **Intitulé** | **Commentaire** |
| id | PK | O | Int |  | Identifiant |  |
| client\_id | FK | N | Int | id from client | Identifiant client |  |
| prix |  | N | Float |  | Prix |  |
| products |  | O | array | id from product | Identifiant des produits | Contient un tableau ayant comme clé l’identifiant du produit et comme valeur sa quantité voulu dans la commande |

### 

### 3.4.2 Table Client

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nom** | **Key** | **Non null** | **Type** | **Référence** | **Intitulé** | **Commentaire** |
| id | PK | O | Int |  | Identifiant |  |
| nom |  | N | String (len 255) |  | Nom |  |
| prenom |  | N | String (len 255) |  | Prénom |  |
| solde |  | O | float |  | Solde du compte |  |

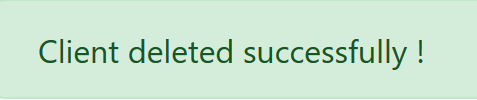
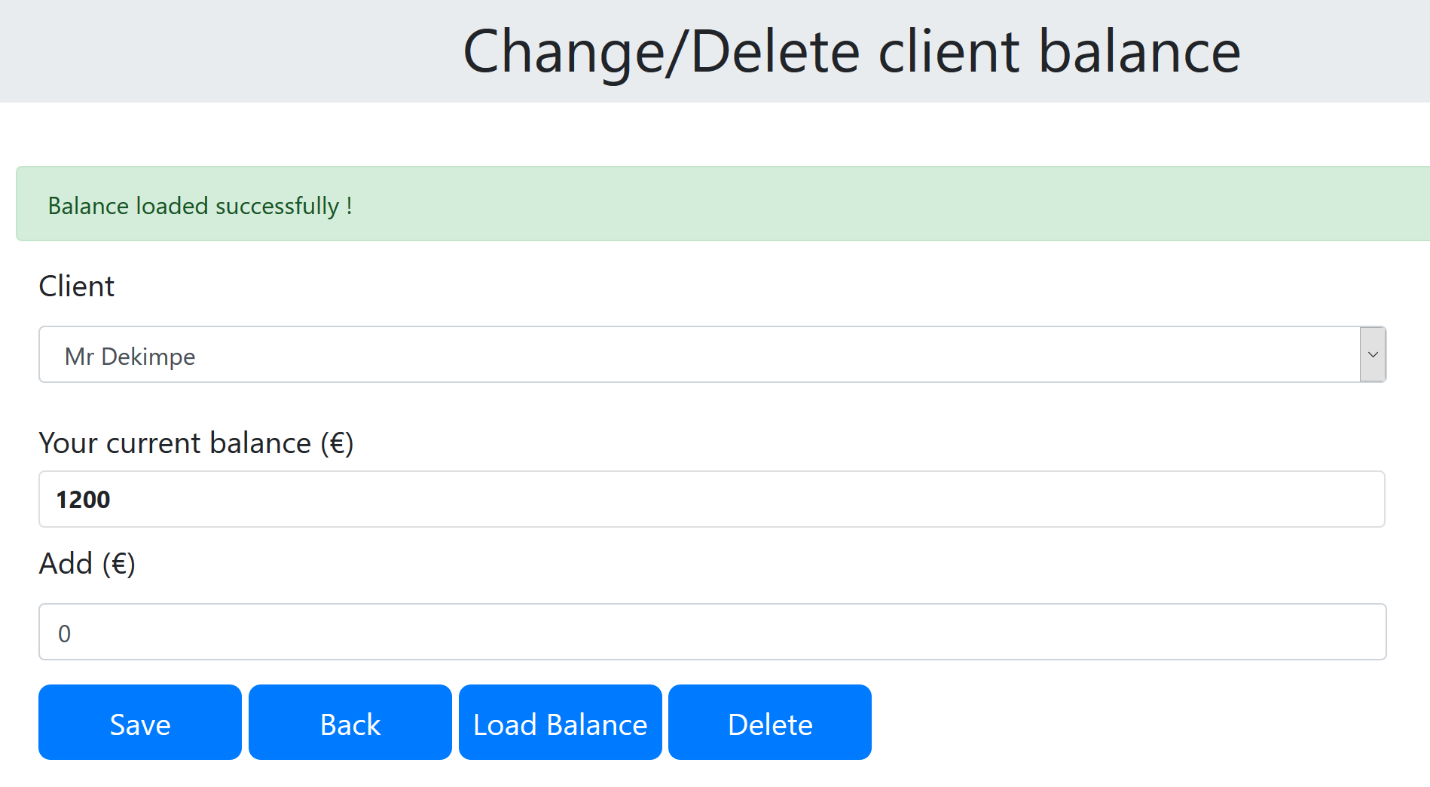
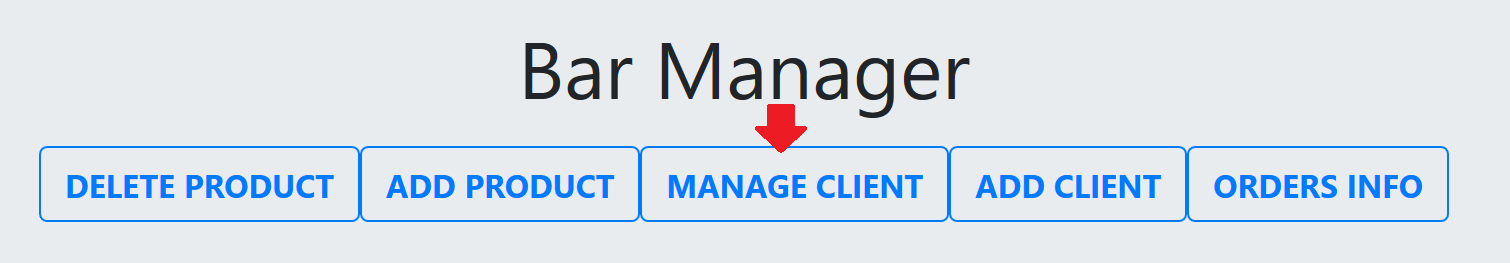
### 3.4.3 Table Product

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nom** | **Key** | **Non null** | **Type** | **Référence** | **Intitulé** | **Commentaire** |
| id | PK | O | Int |  | Identifiant |  |
| name |  | N | String (len 255) |  | Nom |  |
| prix |  | N | Float |  | Prix |  |

# 4 MVC – Symfony

## 4.1 En tant que barman je veux pouvoir supprimer un client

### 4.1.1 Supprimer un client

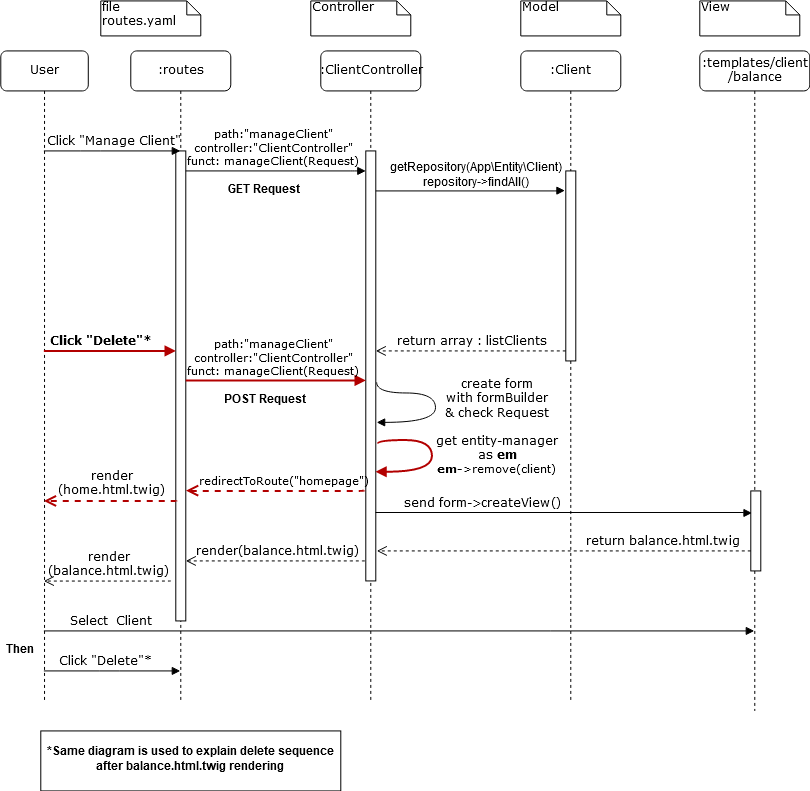


1. L’utilisateur clique sur « Manage client »
2. Il arrive sur la nouvelle page « Manage client »
3. Il sélectionne un client parmi la liste des clients existant
4. Il clique sur « Delete »
5. Un message confirme le succès de l’opération ou son échec

### 4.1.2 Gestion des erreurs et de la sécurité

Un client est sélectionné par défaut et les champs du formulaire sont remplies afin de le rendre valide. Le contrôleur vérifie la validité du formulaire, un message d’erreur apparait s’il ne l’est pas.

### 4.1.3 Diagramme de séquence -> enchainement des scripts



### 4.1.4 Scripts concernés (lien vers github)

**balance.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/client/balance.html.twig>

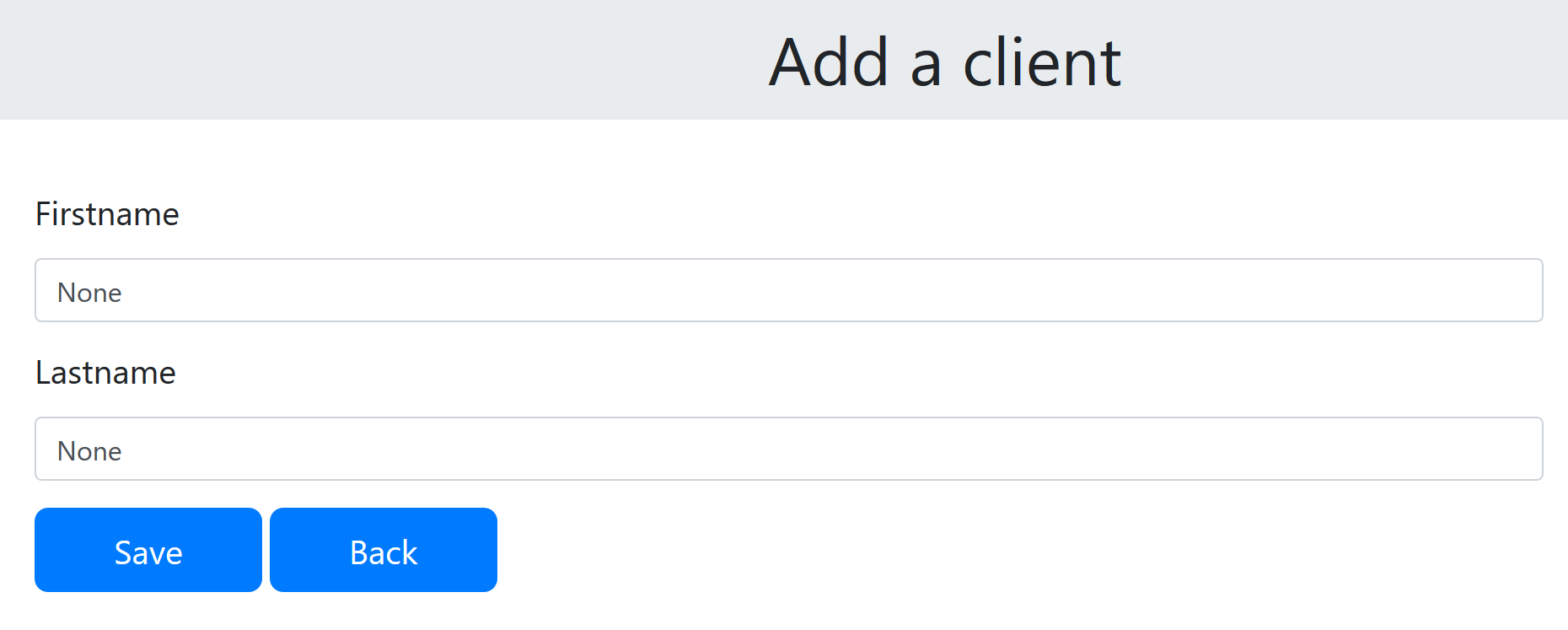
**home.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/home/home.html.twig>

**Client.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Client.php>

**ClientController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/ClientController.php>

## 4.2 En tant que barman je veux pouvoir ajouter un client

### 4.2.1 Ajouter un client





1. L’utilisateur clique sur « Add client »
2. Il arrive sur la nouvelle page « Add a client »
3. Il indique le prénom et le nom du nouveau client
4. Il clique sur « Save »
5. Un message flash confirme le succès de l’opération

### 4.2.2 Gestion des erreurs et de la sécurité

Le contrôleur vérifie la validité du formulaire, ceux-ci sont préremplis avec la valeur « None ». Grâce à l’utilisation d’un navigateur tel que Firefox, un message d’erreur s’affiche dans le cas où un champ du formulaire est vide.

### 4.2.3 Diagramme de séquence -> enchainement des scripts

La séquence d’obtention de la vue du formulaire « add.html.twig » lors d’une GET request n’est pas représenté dans ce diagramme de séquence car celle-ci se déroule de manière similaire à l’obtention de la vue du formulaire lors de la suppression d’un client (4.1.3).

### 4.2.4 Code significatif

Dans la fonction **addClient(**Requet $request**)** du contrôleur **ClientController.php** :

        if ($request->isMethod('POST')) {

          $form->handleRequest($request);

          if ($form->isValid() and $form->get('save')->isClicked()) {

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

            $em->persist($client);

            $em->flush();

            $this->addFlash('client', 'Client successfully added !');

          }

          return $this->redirectToRoute('homepage');

        }

* Dans le cas d’une requête POST, le formulaire s’occupe de la requête. Si le formulaire à été remplis correctement et que le bouton « save » est cliqué, on enregistre dans le nouveau client dans la base de données. Un message de succès est dès lors disponible dans la rubrique « client ». Dans le cas contraire, on est redirigé vers la page d’accueil.

### 4.2.5 Scripts concernés (lien vers github)

**add.html.twig :**

<https://github.com/yannis97/Bar_manager/blob/master/templates/client/add.html.twig>

**home.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/home/home.html.twig>

**Client.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Client.php>

**ClientController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/ClientController.php>

## 4.3 En tant que barman, je veux pouvoir débiter/créditer le compte d’un client

### 4.3.1 Débiter/créditer le compte d’un client

1. L’utilisateur clique sur « Manage client »
2. Il arrive sur la nouvelle page
3. Il sélectionne le client duquel il veut débiter ou créditer le compte
4. Il inscrit dans le champ « Add » un montant négatif (Ex : -10) ou un montant positif
5. Il appui sur « Save » pour enregistrer la modification
6. Un message flash confirme le succès de l’opération ou une erreur si le champ « Add » est mal remplie

### 4.3.2 Gestion des erreurs et de la sécurité

Un client est sélectionné par défaut et les champs du formulaire sont remplies afin de le rendre valide. Le contrôleur vérifie la validité du formulaire, un message d’erreur apparait s’il ne l’est pas.

### 4.3.3 Diagramme de séquence -> enchainement des scripts

Le diagramme de séquence est identique à la séquence de suppression d’un client (4.1.3), si ce n’est qu’on utilise le bouton « Save » à la place de « Delete» et qu’on reçoit le formulaire (donc la même vue) avec le client sélectionné précédemment ainsi que le nouveau solde enregistré.

### 4.3.4 Code significatif

Dans la fonction **manageClient()** de **ClientController.php** :

        $solde = 0;

        $repository = $this->getDoctrine()

        ->getManager()

        ->getRepository('App\Entity\Client')

      ;

        $listClients = $repository->findAll();

        $listClientsName = [];

        foreach ($listClients as $c)

        {

            array\_push($listClientsName, $c->getPrenom()." ".$c->getNom());

        }

* On récupère l’ensemble des clients dans la base de données afin de créer une liste comprenant les prénoms des clients.

$formBuilder = $this->get('form.factory')->createBuilder(FormType::class);

$formBuilder->add('client',   ChoiceType::class , ['choices'=>[array\_combine($listClientsName,$listClients)]])

            ->add('save',   SubmitType::class)

            ->add('back',   SubmitType::class)

            ->add('loadinfo',   SubmitType::class , ['label'=>'Load Balance'])

            ->add('delete',   SubmitType::class , ['label'=>'Delete'])

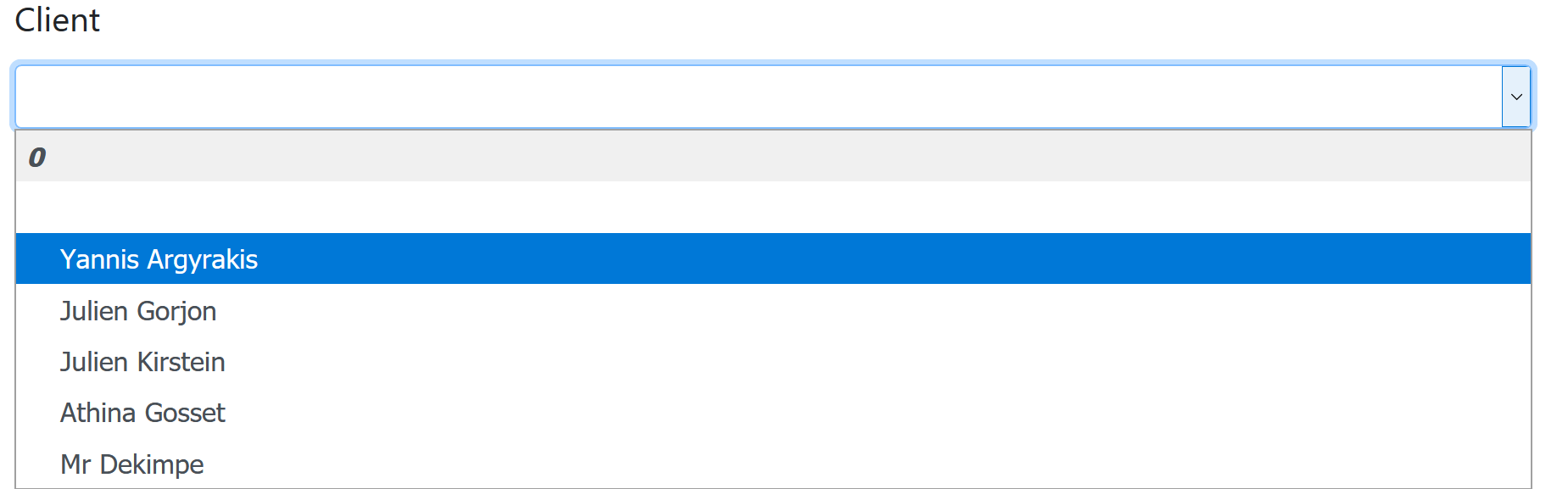
            ->add('add\_balance',   NumberType::class , ['data'=>'0'])

            ;

        $form = $formBuilder->getForm();

* On combine les 2 listes (listClientsName et listClients) ainsi que de nom de client afin de créer un tableau de clé-valeur de type :

**(‘’Nom Prénom‘‘ => Client $client)**

Ce qui permet de proposer une liste à choix sur base de la clé du tableau (Nom+Prénom du client)

if($form->isValid() and $form->get('save')->isClicked())

            {

                $client= $form->get('client')->getData();

                $add\_solde = $form->get('add\_balance')->getData();

                $current\_solde = $client->getSolde();

                $client->setSolde($current\_solde + $add\_solde);

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

                $em->flush();

                $this->addFlash('balance', 'Balance changed successfully !');

                return $this->render('client\balance.html.twig', array(

                    'form' => $form->createView(), 'solde' => $current\_solde + $add\_solde

                  ));

            }

* On peut dès lors récupérer le client grâce à la fonction $form->get('client')->getData(); afin de mettre à jour le nouveau solde dans la base de donnée.
* Une fois terminé, on charge le formulaire avec le nouveau solde.

### 4.3.5 Scripts concernés (lien vers github)

**balance.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/client/balance.html.twig>

**home.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/home/home.html.twig>

**Client.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Client.php>

**ClientController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/ClientController.php>

## 4.4 En tant que barman, je veux pouvoir consulter le solde d’un client

### 4.4.1 Consulter le solde d’un client

1. L’utilisateur clique sur « Manage client »
2. Il arrive sur la nouvelle page
3. Il sélectionne le client duquel il veut charger le compte
4. Il clique sur « Load Balance » pour charger le solde du compte
5. Un message flash confirme le succès de l’opération

### 4.4.2 Gestion des erreurs et de la sécurité

Un client est sélectionné par défaut et les champs du formulaire sont remplies afin de le rendre valide. Le contrôleur vérifie la validité du formulaire, un message d’erreur apparait s’il ne l’est pas.

Grâce à l’utilisation d’un navigateur tel que Firefox, un message d’erreur s’affiche dans le cas où un champ du formulaire est vide.

### 4.4.3 Diagramme de séquence -> enchainement des scripts

Le diagramme de séquence est identique à la séquence de débit/crédit d’un compte client (point 4.3.3), si ce n’est qu’on utilise le bouton « Load Balance » à la place de « Save».

### 4.4.4 Code significatif

Dans la fonction **manageClient()** de **ClientController.php** :

            if ($form->get('loadinfo')->isClicked())

            {

                $client= $form->get('client')->getData();

                $solde = $client->getSolde();

                $this->addFlash('balance', 'Balance loaded successfully !');

                return $this->render('client\balance.html.twig', array(

                    'form' => $form->createView(), 'solde' => $solde

                  ));

            }

* On vérifie que le bouton « loadinfo » soit cliqué
* Si c’est le cas, on récupère l’information du solde du client
* On charge un message flash de succès d’opération
* On recharge le même formulaire avec le solde obtenu

Dans **balance.html.twig** :

        {% for message in app.flashes('balance') %}

            <div class="alert alert-success">

                {{ message }}

            </div>

        {% endfor %}

* On affiche le message de succès de l’opération

      <div class="card";>

        <div><b> {{solde}}</b></div>

      </div>

### 4.4.5 Scripts concernés (lien vers github)

**balance.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/client/balance.html.twig>

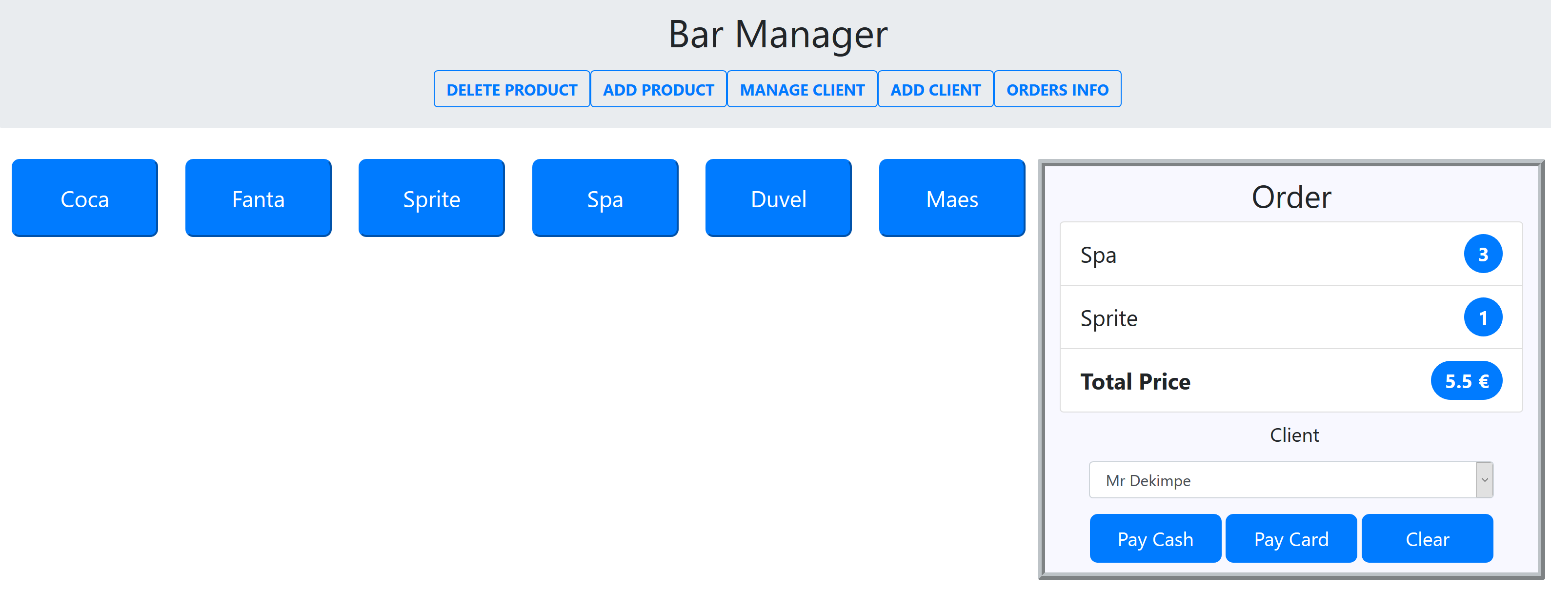
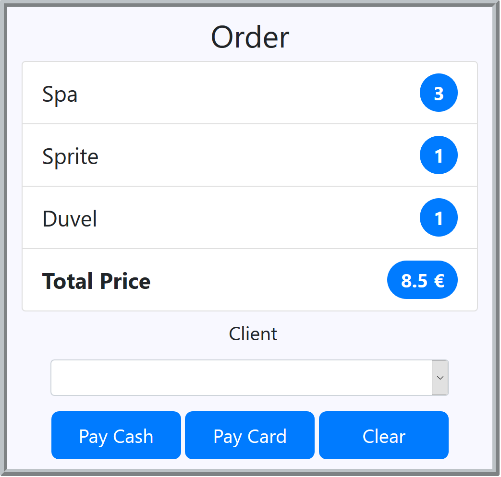
**home.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/home/home.html.twig>

**Client.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Client.php>

**ClientController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/ClientController.php>

## 4.5 En tant que barman, je veux pouvoir encoder la commande d’un client

### 4.5.1 Encoder la commande d’un client



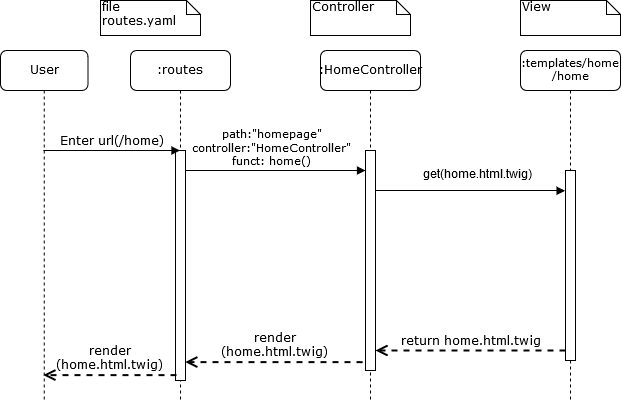
1. L’utilisateur charge la page d’accueil « /home »
2. Une page d’accueil contenant les produits et un panier « Order » est affiché
3. Il ajoute un produit au panier en cliquant dessus (Ex : Duvel)
4. La page d’accueil est rechargée
5. Le prix total du panier est ainsi mis à jour

### 4.5.2 Gestion des erreurs et de la sécurité

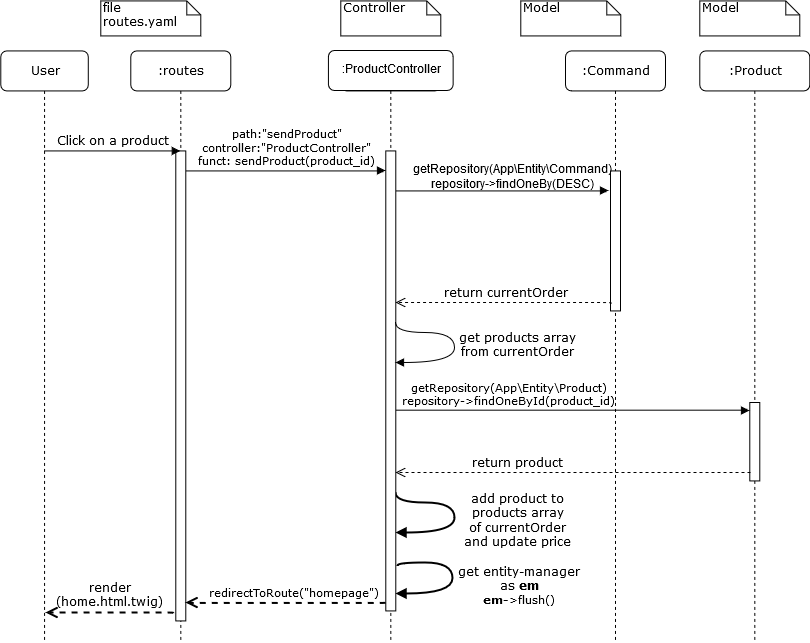
Un objet commande est préalablement créée. Cette commande est enregistrée dans la base de données, ajouter un produit en cliquant sur le bouton (Ex : Duvel) permet d’ajouter ce produit dans la commande et d’enregistrer la modification. Cela permet de s’assurer de retrouver le même panier dans le cas où l’on souhaite effectuer une autre action pendant la prise de la commande

### 4.5.3 Diagramme de séquence -> enchainement des scripts

#### 4.5.3.1 Séquence d’obtention de la page d’accueil



#### 4.5.3.2 Séquence d’ajout d’un produit au panier



### 4.5.4 Code significatif

Dans le template **home.html.twig** :

 {{ render(controller("App\\Controller\\ProductController::displayProduct")) }}

* On fait appel à la fonction **displayProduct()** du controller **ProductController.php**:

    public function displayProduct()

    {

      $repository = $this->getDoctrine()

      ->getManager()

      ->getRepository('App\Entity\Product')

      ;

      $listProducts = $repository->findAll();

      return $this->render('product\products.html.twig',

        array('listProducts' => $listProducts)

      );

    }

* Cette fonction récupère la liste des produits dans la base de données et l’envoi dans **products.html.twig**:

{% for product in listProducts %}

      <div class="col-sm-2">

        <h3>

         <form action="{{ path('sendProduct',{ 'product\_id' : product.id}) }}">

          <button type="input" class="btn-xlarge btn-primary btn-rounded">{{ product.name }}</button>

         </form>

        </h3>

      </div>

  {% endfor %}

* Pour chaque produit, on crée un formulaire contenant un bouton étant lié à la route « **sendProduct**» et prenant en paramètre l’identifiant du produit **« product\_id »**

Dans le template **home.html.twig** :

{{render(controller("App\Controller\CommandController::displayCurrentCommand"))}}

* On fait appel à la fonction **displayCurrentCommand()** du controller **CommandController.php**:

        $listProductsName = [];

        if ($listProducts !== null)

        {

            foreach ($listProducts as $key => $value)

            {

                $product = $repository->findOneById($key);

                if($product!==null)

                {

                    $product\_name = $product->getName();

                    $listProductsName[$product\_name]=$value;

                }

            }

        }

* On y récupère la liste des produits de la commande actuelle **listProducts**
* Un nouveau tableau **listProductsName** est créée de type :
  + (‘’product\_name’’ => ‘’quantity’’)
* Pour chaque produit présent dans la commande, on récupère son nom (grâce à son **id**) ainsi que la quantité de fois où il est présent dans celle-ci

        return $this->render('order\order.html.twig',

        array('listProducts' => $listProductsName , 'price' => $price)

        );

* On envoie **listProductsName** et le prix de la commande (**price**) à **order.html.twig** qui se charge d’afficher le panier

### 4.5.5 Scripts concernés (lien vers github)

**home.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/home/home.html.twig>

**btnOptions.html.twig:** <https://github.com/yannis97/Bar_manager/blob/master/templates/home/btnOptions.html.twig>

**products.html.twig :**

<https://github.com/yannis97/Bar_manager/blob/master/templates/product/products.html.twig>

**order.html.twig :**

<https://github.com/yannis97/Bar_manager/blob/master/templates/order/order.html.twig>

**Product.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Product.php>

**Command.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Command.php>

**ProductController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/ProductController.php>

**CommandController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/CommandController.php>

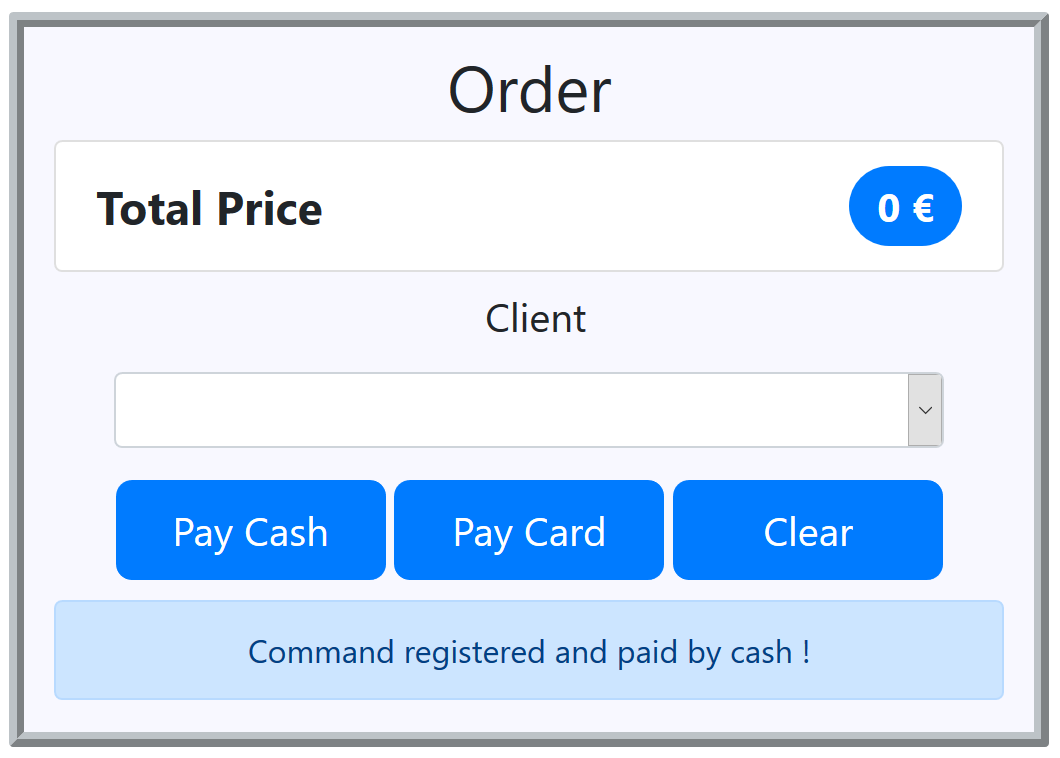
**HomeController.php** :

<https://github.com/yannis97/Bar_manager/blob/master/src/Controller/HomeController.php>

## 4.6 En tant que barman, je veux pouvoir indiquer si le client paye sa commande cash ou avec son compte

### 4.6.1 Payer en cash/compte

1. L’utilisateur sélectionne un client
2. Il clique sur « Pay Cash » ou « Pay Card » en fonction du souhait du client
3. La page d’accueil est rechargée avec une nouvelle commande et un panier vide
4. Un message indique le succès de l’opération.
5. Une nouvelle commande vide est créée

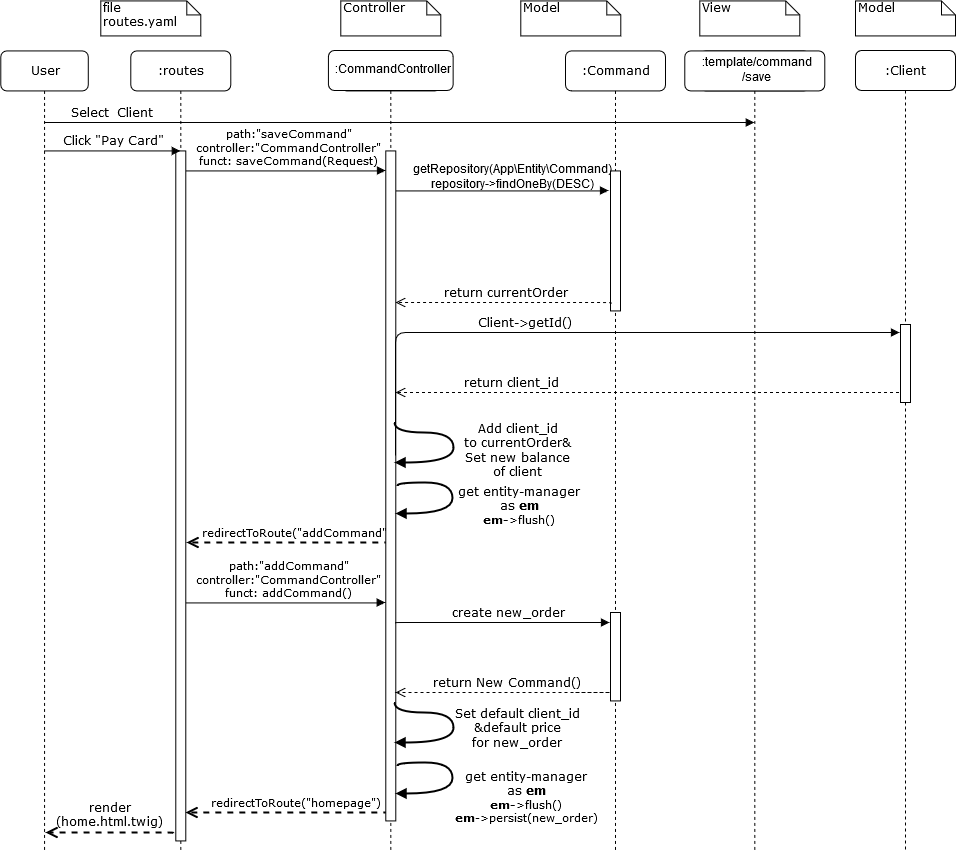


### 4.6.2 Gestion des erreurs et de la sécurité

Un client est sélectionné par défaut. Dans le cas où le client souhaite payer avec son compte, on vérifie que son solde soit suffisant, un message d’erreur apparait dans le cas contraire.

### 4.6.3 Diagramme de séquence -> enchainement des scripts

Ci-dessous, la séquence lors du paiement par compte. Lors d’un paiement par cash, la séquence est identique si ce n’est que le solde du compte du client ne change pas.



:template/command/save

### 4.6.4 Code significatif

Le formulaire servant également à vider le panier, il sera expliqué plus en détail dans la prochaine user storie. Comme dans le cas du débit/crédit du compte d’un client, une fonction est responsable de la création et de la gestion du formulaire. Il s’agit de **saveCommand()** dans **CommandController.php**

Il est tout de même à noter que la liste de choix de client pour le formulaire est créée de la même manière qu’au point **4.3.4**:

        return $this->render('order\save.html.twig', array(

            'form' => $form->createView(), ));

* Le formulaire est créér dans **save.html.twig**
* Celui-ci est affiché au niveau du panier se trouvant dans **home.html.twig** via :

{{ render(controller('App\\Controller\\CommandController::saveCommand')) }}

Dans **saveCommand()** de **CommandController.php**:

return $this->redirectToRoute('addCommand');

* Si la commande est payée en cash ou avec le compte, on fait appelle à **addCommand()** de **CommandController.php**

        $new\_order = new Command();

        $new\_order->setPrix(0);

        $new\_order->setClientId(0);

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

        $em->persist($new\_order);

        $em->flush();

        return $this->redirectToRoute('homepage');

* Cette fonction créé une nouvelle commande avec identifiant de client = 0

Et prix = 0

* Elle l’enregistre en base de donnée et renvoie la page **/home (**voir4.5.3.1)

### 4.6.5 Scripts concernés (lien vers github)

**home.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/home/home.html.twig>

**save.html.twig:**

<https://github.com/yannis97/Bar_manager/blob/master/templates/order/save.html.twig>

**Command.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Command.php>

**Client.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Client.php>

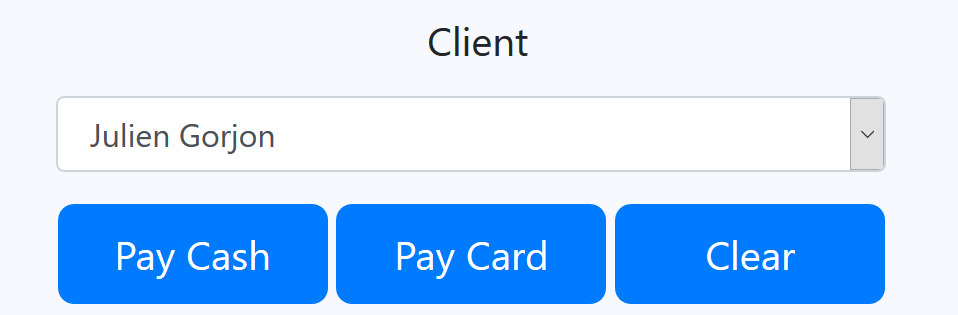
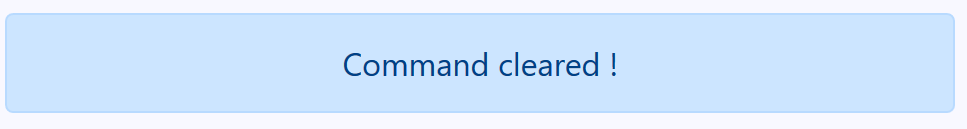
**CommandController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/CommandController.php>

**HomeController.php** :

<https://github.com/yannis97/Bar_manager/blob/master/src/Controller/HomeController.php>

## 4.7 En tant que barman je veux pouvoir supprimer le panier du client

### 4.7.1 Supprimer le panier du client



1. Dans le panier, appuyer sur « Clear »
2. Le panier se vide
3. Un message d’information apparait pour confirmer l’opération

### 4.7.2 Gestion des erreurs et de la sécurité

La validité du formulaire est vérifiée.

### 4.7.3 Code significatif

Le bouton pour clear fait partie du formulaire créée par la fonction **saveCommand()** dans **CommandController.php** :

$formBuilder = $this->get('form.factory')->createBuilder(FormType::class);

        $formBuilder

            ->add('client',   ChoiceType::class , ['choices'=>[array\_combine($listClientsName,$listClients)]])

            ->add('card',   SubmitType::class ,['label'=>'Pay Card'])

            ->add('cash',   SubmitType::class,['label'=>'Pay Cash'])

            ->add('clear',   SubmitType::class , ['label'=>'Clear'])

            ;

        $form = $formBuilder->getForm();

* Le formulaire est créé avec les 3 options « Pay Card » « Pay Cash » et « Pay Clear » ainsi que la liste de choix de client (il s’agit du même formulaire que celui permettant de valider le paiement d’un client)
* Celui-ci est envoyé dans **save.html.twig**

                else {

                    $currentOrder->setProducts([]);

                    $currentOrder->setPrix(0);

                    $currentOrder->setClientId(0);

                    $em->flush();

                    $this->addFlash('info', 'Command cleared !');

                    return $this->redirectToRoute('homepage');

                }

* Après avoir vérifié l’utilisation d’une requête **POST** et la validité du formulaire, on vérifie qu’elle bouton a été cliqué
* Le dernier cas géré par le **else** correspond à l’utilisation du bouton « Clear »
* Il a pour effet de vider la liste de produit de la commande, mettre son prix à 0 et l’id du client à 0 (pour dire qu’il s’agit d’un clear)
* On redirige le client vers la page d’accueil **/home** (voir4.5.3.1)

### 4.7.4 Scripts concernés (lien vers github)

Identique au point 4.6.5

## 4.8 En tant que patron, je veux pouvoir consulter le chiffre d’affaire et les commandes

### 4.8.1 Consulter le chiffre d’affaire & les commandes

1. L’utilisateur clique sur « Orders Info »
2. La page « Orders informations » s’affiche
3. Un récapitulatif des commandes est affiché avec le détail des produits et leurs quantités
4. L’information sur le chiffre d’affaire est disponible tout en bas de la page

### 4.8.2 Gestion des erreurs et de la sécurité

Pour chaque commande, on affiche la liste de produit. On vérifie préalablement si la liste n’est pas vide et que le nom du produit est trouvable donc que celui-ci existe encore.

### 4.8.3 Code significatif

Dans la fonction **displayAllOrders()** de **CommandController.php** :

        foreach ($listOrders as $Order)

           $Order\_data = [];

           $Order\_data['client'] =  $Order->getClientId();

           $Order\_data['id'] = $Order->getId();

           $Order\_data['listproducts'] = $listProductsName;

           $Order\_data['price'] =  $Order->getPrix();

            array\_push($newlistOrders, $Order\_data);

            $totalSold += $Order->getPrix();

* On injecte dans **newlistOrders** l’ensemble des commandes avec la liste de leurs noms de produits (**listProductsName**)
* Celle-ci est obtenu de manière identique au point 4.5.4

        return $this->render('order\displayAll.html.twig',

        array('listOrders' => $newlistOrders , 'total' => $totalSold)

        );

* Le **newlistOrders** ainsi que le solde des ventes **totalSold** est envoyé au template **displayAll.html.twig**

Dans **displayAll.html.twig**:

  {% for order in listOrders %}

    <tr>

        <td>{{order.id}}</td>

        <td>{{order.client}}</td>

        <td>{{order.price}} €</td>

        <td><ul class="list-group">

          {% for product,quantity in order.listproducts %}

          <li class="list-group-item d-flex justify-content-between align-items-right">{{product}} : {{quantity}}</li>

          {% endfor %}</ul>

        </td>

        </tr>

  {% endfor %}

* Pour chaque commande, ont créé un élément de tableau affichant les détailles de la commande à l’aide d’une boucle for sur **listOrders**
* Une boucle for est également nécessaire pour obtenir la clé et la valeur du tableau **listproducts** de chaque commande

### 4.8.4 Scripts concernés (lien vers github)

**displayAll.html.twig :**

<https://github.com/yannis97/Bar_manager/blob/master/templates/order/displayAll.html.twig>

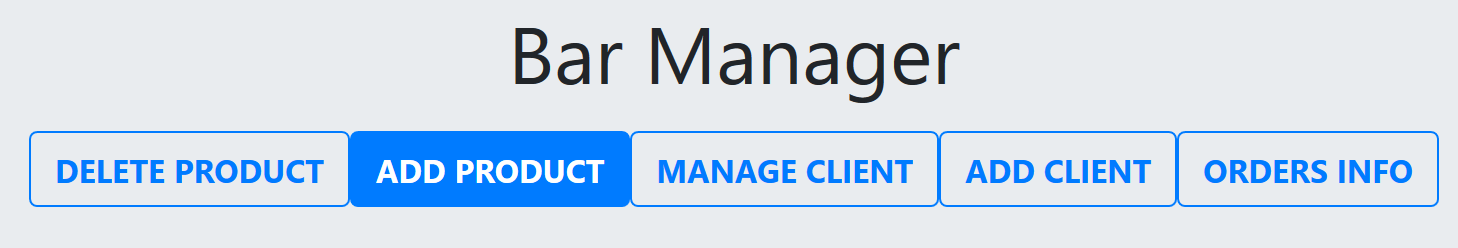
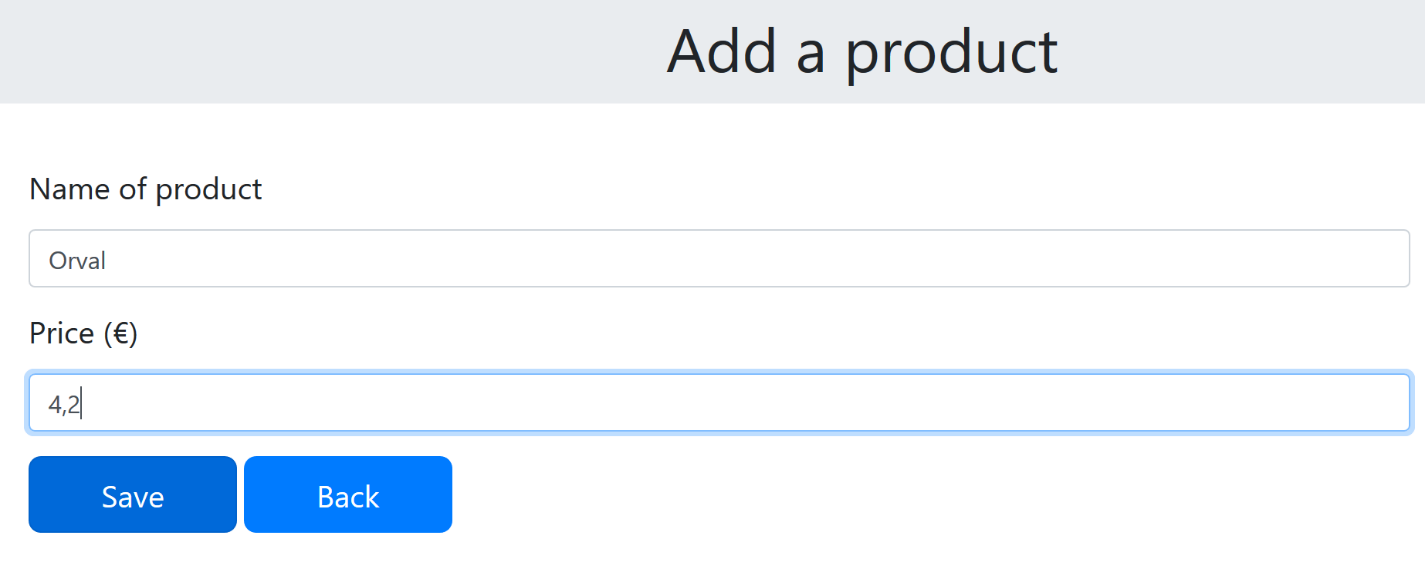
**Product.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Product.php>

**Command.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Command.php>

**CommandController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/CommandController.php>

## 4.9 En tant que barman et patron, je veux pouvoir ajouter un produit

### 4.9.1 Ajouter un produit

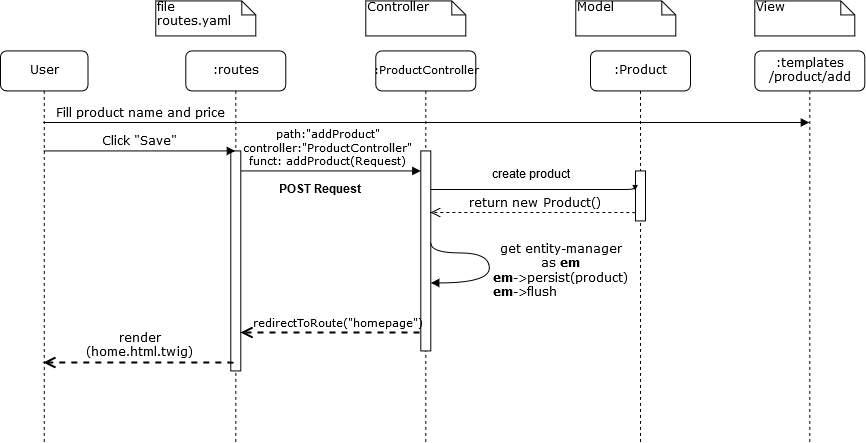


1. L’utilisateur clique sur « Add product»
2. Il arrive sur la nouvelle page « Add a product »
3. Il indique le nom et le prix du produit
4. Il clique sur « Save »
5. Un message flash confirme le succès de l’opération , le produit est ajouté

### 4.9.2 Gestion des erreurs et de la sécurité

La validité du formulaire est vérifiée. Un message d’erreur s’affiche dans le cas où le prix indiqué ne correspond pas à un nombre.

### 4.9.3 Diagramme de séquence -> enchainement des scripts



### 4.9.4 Code significatif

Dans la fonction **addProduct()** de P**roductController.php :**

        if ($request->isMethod('POST')) {

          $form->handleRequest($request);

          if ($form->isValid() and $form->get('save')->isClicked()) {

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

            $em->persist($product);

            $em->flush();

            $this->addFlash('product', 'Product successfully added !');

          }

          if (!$form->isValid())

          {$this->addFlash('error', 'Incorrect value for product price !');}

          return $this->redirectToRoute('homepage');

        }

* On vérifie si la requête est un **POST**
* On vérifie ensuite la validité du formulaire avec **form->isValid()** en même temps que le bouton « Save » est utilisé
* Si l’opération est un succès, ajouter un message flash de succès
* Dans le cas contraire où le formulaire n’est pas valide, un message d’erreur est ajouté aux messages flash
* On redirige le client vers la page d’accueil **/home** (voir 4.5.3.1)

        return $this->render('product\add.html.twig', array(

          'form' => $form->createView(),

        ));

* Si la requête est un **GET** le formulaire est créé dans **add.html.twig**

### 4.9.5 Scripts concernés

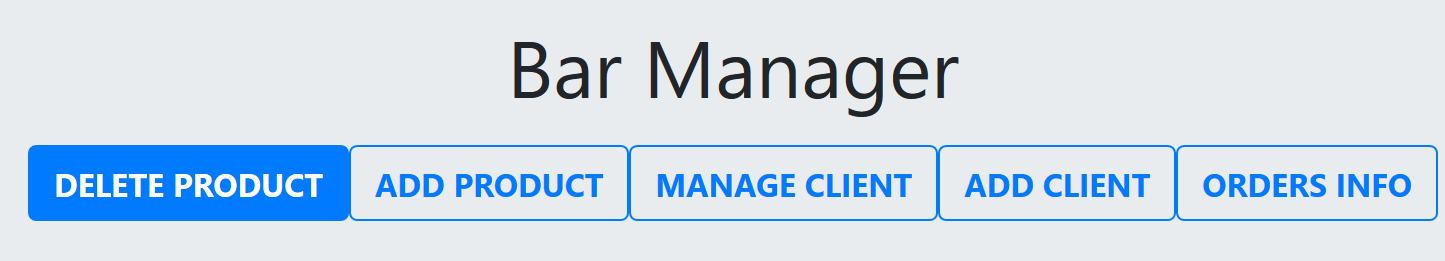
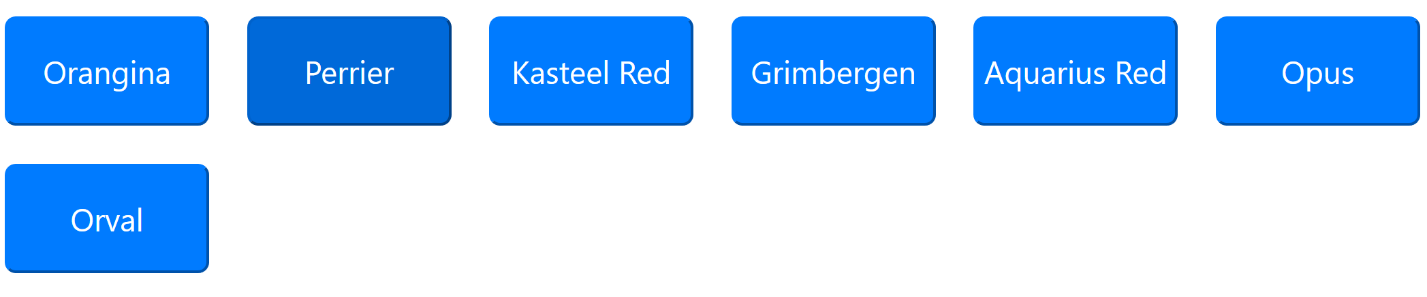
**add.html.twig :** <https://github.com/yannis97/Bar_manager/blob/master/templates/product/add.html.twig>

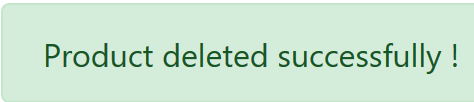
**Product.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Product.php>

**ProductController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/ProductController.php>

## 4.10 En tant que barman et patron, je veux pouvoir supprimer un produit

### 4.10.1 Supprimer un produit





1. L’utilisateur clique sur « Delete product »
2. Le dernier produit ajouté est supprimé

### 4.10.2 Gestion des erreurs et de la sécurité

Dans le cas où il n’y a pas de produits en base de données, un message informe qu’aucun produit n’a été trouvé.

### 4.10.3 Code significatif

Dans la fonction **deleteProduct()** de **ProductController.php** :

      $product = $repository->findOneBy(array(), array('id' => 'DESC'));

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

      if ($product !== null)

      {

        $em->remove($product);

        $this->addFlash('deleteProduct', 'Product deleted successfully !');

        $em->flush();

      }

      else

      {

        $this->addFlash('error', 'No product found !');

      }

* On cherche le dernier produit (**product**) ajouté avec **repository->findOneBy()**
* Dans le cas où le résultat de la recherche trouve un produit , on le supprime avec

**em->remove(product)**

* Un message flash est ajouté si l’opération est un succès (message d’erreur dans le cas où aucun produit n’est trouvé)
* On redirige le client vers la page d’accueil **/home** (voir 4.5.3.1)

### 4.10.4 Scripts concernés

**Product.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Entity/Product.php>

**ProductController.php** : <https://github.com/yannis97/Bar_manager/blob/master/src/Controller/ProductController.php>

## 4.11 En tant que patron, je veux pouvoir consulter les ventes des différentes consommations

### 4.11.1 Affichage des ventes des différentes consommations

1. L’utilisateur clique sur « Orders info »
2. La page « Orders informations » s’affiche
3. En dessous de l’affichage du chiffre d’affaire est affiché les produits vendus et leurs quantités

### 4.11.2 Gestion des erreurs et de la sécurité

On vérifie préalablement si la liste n’est pas vide et que le nom du produit est trouvable donc que celui-ci existe encore.

### 4.11.3 Code significatif

La fonction **displayAllOrders()** de **CommandController()** s’exécute lors de l’appuie sur le bouton « Orders info » , celle-ci a déjà été expliqué au point 4.8.1.

Cette fonction récupérant déjà l’ensemble de la liste des produits des différentes commandes avec leurs quantités :

                       if(array\_key\_exists($product\_name, $listtotalProducts))

                       {

                        $listtotalProducts[$product\_name]+=$value;

                       }

                       else{

                        $listtotalProducts[$product\_name]=1;

                       }

* Nous ajoutons la quantité totale du produit dans le tableau **listtotalProducts** de type :
  + **(‘’Nom du produit’’ => ‘’quantité’’)**

'listTotalProducts' => $listtotalProducts

* Il est passé en paramètre de **displayAll.html.twig**

    {% for product,quantity in listTotalProducts %}

    <li class="list-group-item d-flex justify-content-between align-items-center">

        <div id="mediumFont">{{product}}</div>

        <span class="badge badge-pill badge-secondary">{{quantity}}</span>

    </li>

    {% endfor %}

* Qui se charge d’afficher la **listTotalProducts** sous forme de liste contenant chaque produit et leurs quantités

### 4.11.4 Scripts utilisés

Identique au point 4.8.4

# 5 Concusion

## 5.1 Difficultés rencontrés

La difficulté principale du projet réside dans le codage de la page principale rendu dans **home.html.twig** celle-ci contient sans doute la fonctionnalité la plus importante qu’est le remplissage interactif du panier sur de l’utilisation des boutons créé pour chaque produit. Elle contient également toutes les options disponibles pour l’utilisateur.

La solution a été de structurer avec soin le fichier **home.html.twig** pour ainsi faire appel aux différents scripts, chacun ayant un rôle spécifique.

Vous retrouverez donc dans ce fichier les lignes suivantes :

{% extends "layout.html.twig" %}

* Permettant d’inclure le fichier **layout.html.twig**

{{ include('style/style.html.twig') }}

* Lui-même situé dans **layout.html.twig** permettant la mise en place de bootstrap en incluant le fichier définissant le CSS **style.html.twig**

{{ include('home/btnOptions.html.twig') }}

* Ensuite, l’inclusion de **btnOptions.html.twig** va permettre l’affichage du groupe de boutons proposant les différentes options :

{{ include('flash.html.twig') }}

* Incluant les messages flashs des différentes opérations

{{ render(controller("App\\Controller\\ProductController::displayProduct")) }}

* Renvoi l’affichage des différents boutons de chaque produit (**products.html.twig**)

{{render(controller("App\\Controller\\CommandController::displayCurrentCommand")}

* Fffiche la commande actuelle dans le panier (situé à droite)

{{ render(controller('App\\Controller\\CommandController::saveCommand')) }}

* Affiche le formulaire situé dans le panier

        {% for message in app.flashes('info') %}

            <div class="alert alert-primary">

                {{ message }}

            </div>

        {% endfor %}

* Affiche les messages flashs issus des opérations dans le panier

Renvoyer le formulaire de sauvegarde de commande dans [home.html.twig](http://home.html.twig) (utilisé un url puis l’inclure , on peut pas render le controller)

## 5.2 Objectifs atteints/non atteints

|  |  |  |  |
| --- | --- | --- | --- |
| **User stories** | **Priorité** | **Réalisation** | **Reste à faire** |
| En tant que barman, je veux pouvoir ajouter/supprimer un client | 1 | 100% |  |
| En tant que barman, je veux pouvoir débiter/créditer le compte d’un client | 1 | 100% |  |
| En tant que barman, je veux pouvoir consulter le solde sur le compte d’un client | 1 | 100% |  |
| En tant que barman, je veux pouvoir encoder la commande d’un client | 1 | 100% |  |
| En tant que barman, je veux pouvoir indiquer si le client paye sa commande cash ou avec son compte | 1 | 100% |  |
| En tant que barman je veux pouvoir supprimer le panier du client | 1 | 100% |  |
| En tant que patron, je veux pouvoir consulter le chiffre d’affaire et les commandes | 1 | 100% |  |
| En tant que barman et patron, je veux pouvoir ajouter/supprimer un produit | 2 | 80% | Proposer de choisir le produit à supprimer |
| En tant que patron, je veux pouvoir consulter les ventes des différentes consommations | 2 | 50% | Séparer cette fonctionnalité de l’affichage des commandes et du chiffre d’affaire en créant une fonction uniquement pour cela |

## 5.3 Eléments intéressants

L’ajout d’une unité d’un produit dans le panier se fait grâce à la route **/sendproduct/{product\_id}**

On a préalablement créé pour chaque produit un formulaire dans **products.html.twig**:

<form action="{{ path('sendProduct',{ 'product\_id' : product.id}) }}">

<button type="input" class="btn-xlarge btn-primary btn-rounded">{{product.name}}

</button>

</form>

* Ce formulaire comprend un button qui déclenche une requête **POST** vers la fonction **sendProducts()** de **ProductController.php**
* Cette fonction à pour effet d’ajouter l’unité de produit au panier et de recharger la page d’accueil **home.html.twig**

## 5.4 Pistes d’amélioration

* Ajouter une étape de confirmation pour supprimer un client ou modifier son solde
* Ajouter une étape de confirmation pour supprimer un produit
* Permettre l’encodage de produit dans le panier de plus qu’une unité à la fois
* Enregistrer dans la commande si celle-ci à été payé en cash ou avec le compte par l’intermédiaire d’une variable booléene
* Permettre de supprimer les produits un à un dans le panier
* Proposer un système de tri lors de la consultation des commandes sur base du paiement en cash/compte
* Pouvoir modifier le prix d’un produit