





**APPLICATION BARBIER**

**GUI ADMIN**

**Remerciements**

Je souhaite remercier l’ensemble de l’équipe de MomentTech pour son accueil et la qualité de son accompagnement tout au long de ce stage. Plus particulièrement Arnaud Fossard qui m’a permis d’aborder le projet dans les meilleures conditions, en abordant dans un premier temps le contexte particulier à l’entreprise puis le cheminement qui a mené à l’application de barbier, à travers l’appréhension de la partie fonctionnelle.

Je remercie David Duquenoy qui m’a accompagné tout au long de mon stage sur la partie technique, en me formant à de nouveaux outils, de nouvelles méthodes et de nouveaux langages (Gherkin, Angular4).

Je remercie également Bertrand Louargant pour son initiation au Deep Learning, ainsi que Frédéric Delacourt avec qui j’ai passé une journée autour de PostrGreSQL.

Enfin merci à Gérald Vannier, Jean-Marc Deshayes et Philippe Boursin pour m’avoir permis d’effectuer ce stage au sein de MomentTech ; l’aspect humain et l’organisation horizontale ont été des aspects particulièrement appréciables et enrichissants.

Sommaire

**Remerciements** ... 3

**Introduction – Contexte** ... 5

**Résumé du projet en anglais** … 6

**Compétences à mettre en œuvre** … 7

**Présentation de l’entreprise** … 7

* Organisation … 8
* Méthode … 8
* Environnement … 8

**Cahier des charges** … 9

**Analyse du besoin** … 12

* QQOQCCP … 12
* Fonctionnalités attendues … 12
* Diagramme de pieuvre … 13
* Tableau des fonctionnalités primaires et contraintes … 14

**Gestion de projet**

* WBS … 15

**Spécifications fonctionnelles**

* Diagramme de cas d’utilisation … 16
* Use Cases … 17
* Tableau des fonctionnalités demandées dans le cahier des charges … 18

**Maquettage** … 19

**Spécifications techniques**

Modélisation de la base de données … 25

* MCD … 25
* MLD … 26
* Script de la base de données … 27
* Dictionnaire des données … 30

**Architecture du projet** … 31

* Backend … 34
* Tests … 35

**Présentation de composants** … 38

**Rendu du site** … 47

**Conclusion** … 50

INTRODUCTION – Contexte et objectif de l’outil

Le pôle de développement application web a la mission de réaliser une application de fidélisation client. Initialement cette idée est venue du besoin d’un barbier qui a souhaité faire une étude. MomentTech n’a pas été retenue lors de l’appel d’offre, toutefois l’équipe a choisi de poursuivre le projet avec comme objectif de réaliser une application générique pouvant être déclinée, agrémentée et adaptée à tout type de contexte marchand (aussi bien PME que grand compte).

Compétences à mettre en oeuvre

Développer une application client-serveur

* Maquetter une application p20
* Modifier une base de données p40
* Développer une interface utilisateur p33
* Développer des composants d’accès aux données p39

Développer une application web

* Développer des pages web en lien avec une base de données p41
* Utiliser de l’anglais dans son activité professionnelle en informatique

Résumé du projet en anglais

The project who was suggested to me was the development of an admin graphical user interface. The initial case, aborted meanwhile, was about a barber shop ; consequently the project was named Barber app.

This app must to allow giving fidelity points to customers, who permits earning gifts and developing relationship.

This app is made up of three parts : a back end and two front end (rewarded points GUI and Gui Dashboard Service).

My task was to develop an interface of aministration for the managers of shops and business, as well as for the great administrator (the app developer).

Every user has to have defined accesses: the manager of shop must be able to reach to manage the customers of his shop as well as the obtained points and gifts

The manager of business will have to be able to manage all the shops of his business as well as the list of products. Besides, he can reach all the functions(offices) of the manager of shop.

Finally, the Super Admin can substitute itself for any user for the modification of objects.

The base of the project was implemented by two developers : the database was created just like the Web interface of management of points.

Concerning the technologies, it was decided to work in architecture symfony (v3) with PHP7 for the backend and Angular 4 for the frontend.

Présentation de l’entreprise

MomentTech est une start-up normande au parcours singulier.

En effet, l’entreprise a vu le jour dans un contexte particulier : l’ensemble de son effectif est issu de NetCentrex, entreprise fleuron de la région caennaise. Née en 1998 d’un essaimage de France Telecom et spécialisée dans la transmission de la voix par internet(VoIP), elle a compté jusqu’à 350 salariés dans 25 pays et des clients tels que les opérateurs AOL, Fastweb, Telefonica, Tiscali ou France Telecom.

Après huit années de très forte croissance, NetCentrex est cédée par ses deux dirigeants à la société israélienne Comverse, spécialisée dans la sécurité des logiciels de télécommunication. NetCentrex devient alors Xura et est confrontée à de nouvelles problématiques en terme d’enjeux et de gestion. Pour défaut de rentabilité, Comverse décide de fermer progressivement Xura en 2013.

A la fermeture de l’entreprise défintive en 2016, trois managers ont initié le projet d’une nouvelle entité reprenant 15 personnes de NetCentrex, permettant de capitaliser sur l’expertise acquise et l’expérience humaine, avec pour objectif de se développer tant dans la continuité des activités historiques (téléphonie) que dans l’innovation (développement web, application logiciel, deep-learning).

Un des objectifs est de recentrer l’activité localement, notamment à travers des prestations liées à la téléphonie.

J’ai ainsi été accueilli chez MomentTech au sein du service de développement web (5 personnes) et intégré au projet de création d’une application de fidélisation client. Il s’agit d’une version pilote qui sera amenée à évoluer selon les besoins de prospects.

Organisation interne - Organigramme

MomentTech est organisée en trois pôles :

* Téléphonie - VoIP (poursuite des activités historiques)
* Deep Learning (recherche et développement)
* Application web (recherche et développement)

Les aspects d’autonomie, d’initiative, de collaboration participative et horizontale sont des valeurs intrinsèques à MomentTech. L’expertise de chacun favorise également la modulabilité des équipes et des missions.

Il y a actuellement 3 dirigeants. La volonté est d’ouvrir le capital à part égale pour chacun des quinze salariés.

Méthodes de travail

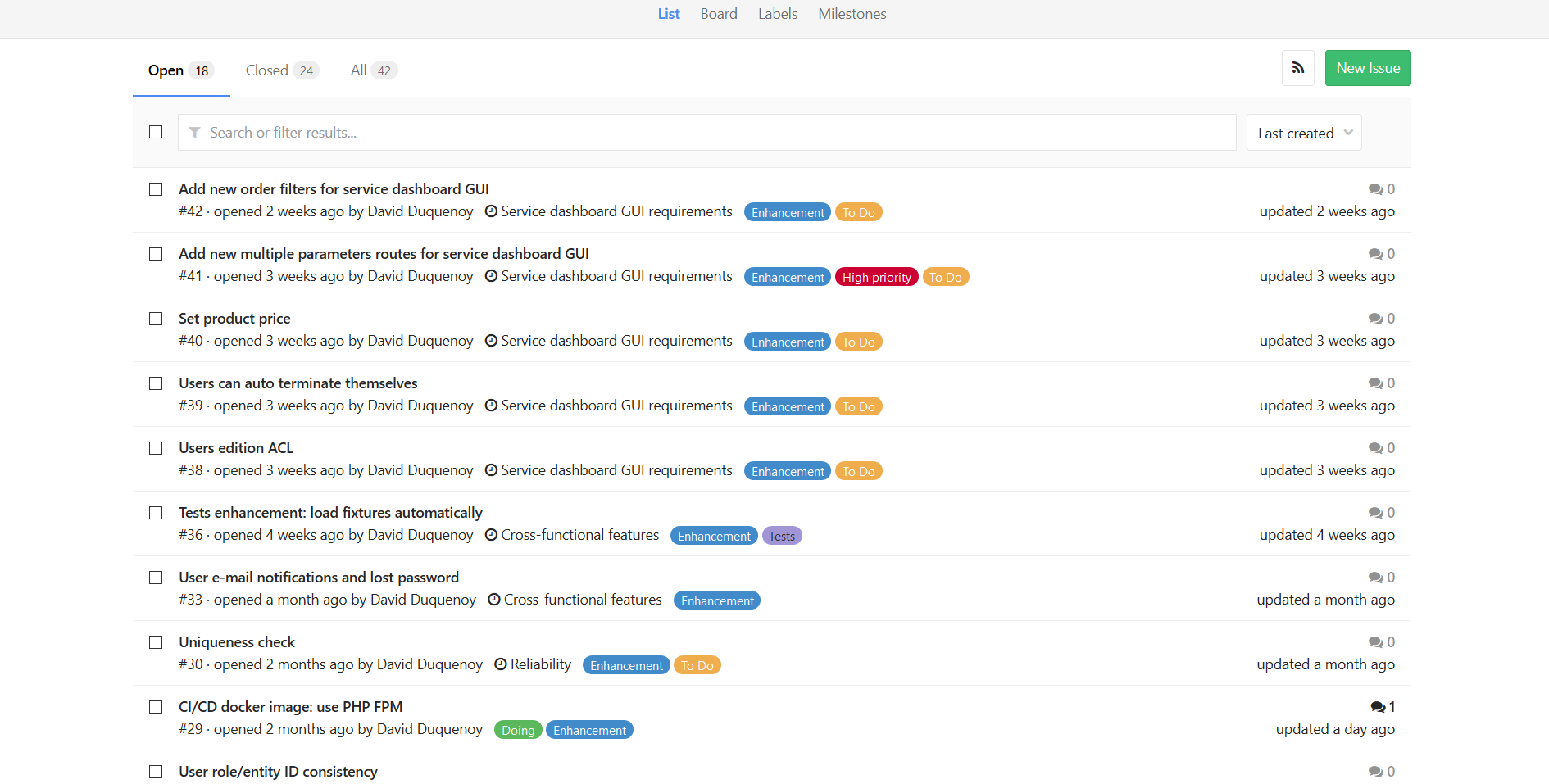
Les équipes fonctionnent en agilité, une réunion hebdomadaire regroupant le product owner, le scrum master et l’équipe de développement a lieu pour chaque mission (une personne peut faire partie de plusieurs équipes à la fois et donc collaborer à plusieurs missions en même temps).

Environnement de travail

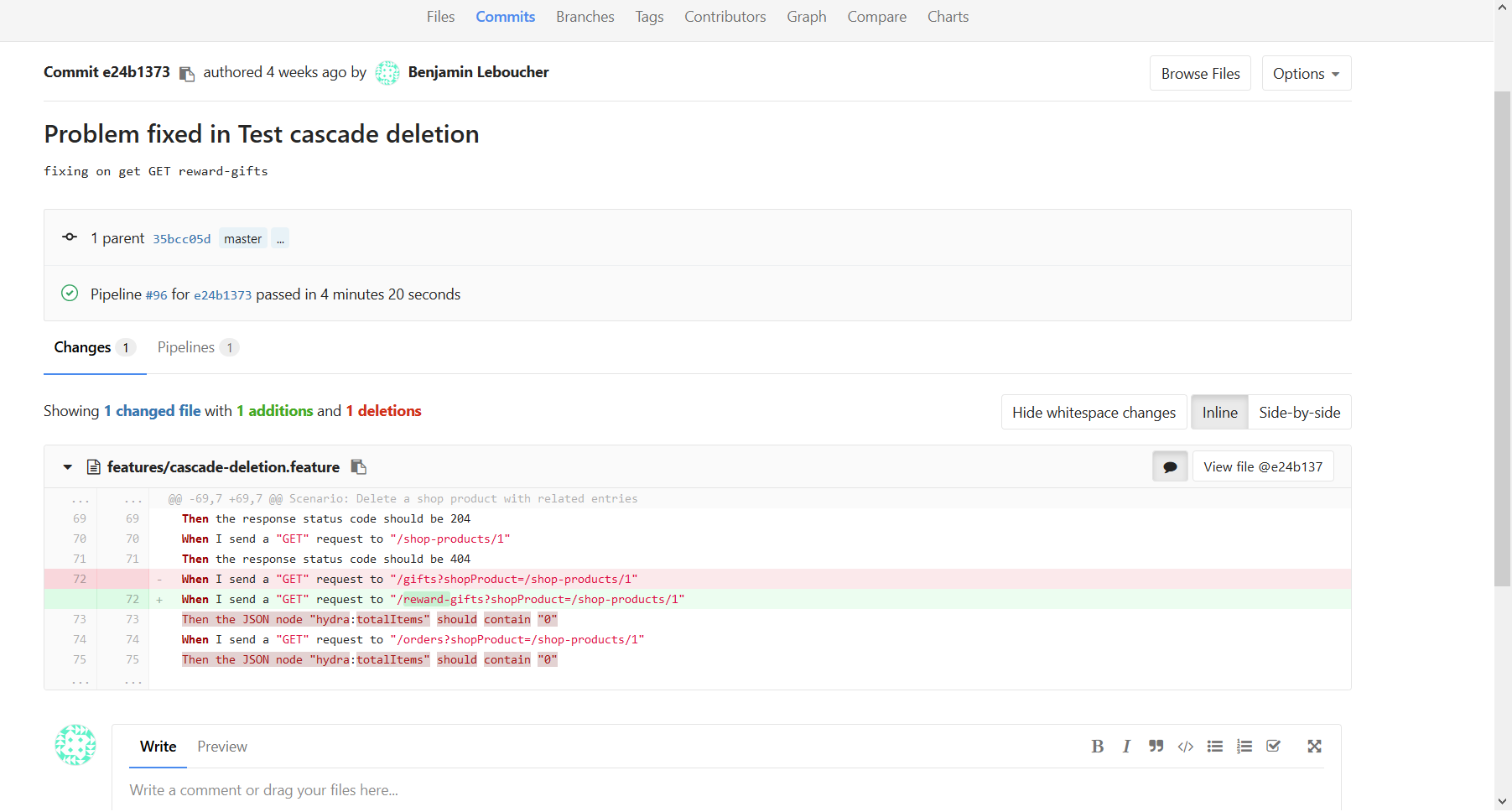
Deux développeurs sont à temps complet sur la partie développement web ; ils peuvent être renforcés le cas échéant par 3 autres personnes.

Pour le projet Application Barbier, nous avons travaillé à 3 (les deux personnes dédiées et moi-même).

Nous avons utilisé Gitlab pour le suivi de projet, les tests, et le déploiement. Chaque partie était découpée en ticket (ou tâche).



*Capture de la liste de tickets du backend #39*



*Interface de ticket – Détail du dernier commit – Tâche effectuée*

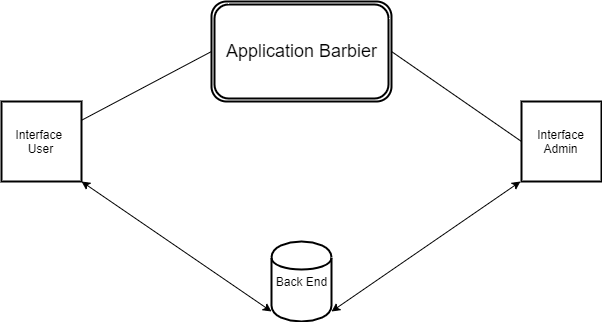
Projet – Cahier des charges

**Conception d’une application de gestion et fidélisation client | Partie gestion par l’administrateur**

**Temps estimé pour la réalisation : fin du stage**

Dans une application ayant comme fonction la gestion de points fidélités et se présentant comme une application de caisse de boutique, l’interface de gestion administrateur devra permettre trois niveaux de login et permettre des fonctions de lecture et édition sur des données client et produit. Elle comportera plusieurs écrans.

La partie backend est déjà implémentée, ainsi qu’une interface utilisateur. La partie dont j’ai eu la charge est l’interface d’administration.



**Détail de l’application Barbier, partie Admin**

L'écran de connexion

Il permet de se connecter avec un couple login/mot de passe.

Un gestionnaire de boutique sera connecté à sa boutique.

Un gestionnaire de business sera connecté à son business.

Le super administrateur aura accès à toutes les données.

Chaque univers devra comporter les éléments suivants :

**Univers gestionnaire de boutique**

Ecran d’accueil / dashboard

Header avec bouton de déconnexion, informations du compte

Liste des dernières commandes

Widgets affichant les informations en temps réel

Menu à gauche

Ecran de gestion des clients

Liste des clients

Formulaires de création / édition de client

Ecran de gestion des produits

Liste des produits du shop

Fonctionnalité pour ajouter un produit du business au shop

Edition du prix

Ecran de gestion des plans de fidélité

Affichage des plans de fidélité

Tableau des produits gagnants (points éditables)

Tableau des cadeaux (points éditables)

**Univers gestionnaire de business**

Ecran d’accueil / dashboard

Header avec bouton de déconnexion, informations du compte

Widgets affichant les informations en temps réel

Ecran de gestion des boutiques

Affichage de la liste des boutiques

Formulaire de création / édition de boutique

Ecran de gestion des produits

Affichage de la liste des produits du business

Edition du prix

Affichage et activation / désactivation des plans de fidélité

Ecran de gestion des gérants de boutique

Affichage de la liste des gérants de boutique

Création de gérant avec login/mdp

**Univers super administrateur**

Ecran d’accueil / dashboard

Header avec bouton de déconnexion, informations du compte

Widgets affichant les informations en temps réel

Ecran de gestion des business

Affichage de la liste des business

Affichage de la liste des gérants de business au clic sur la liste des business dans un formulaire

Lien pour envoyer un mail au gérant de business

Ecran de gestion des boutiques

Affichage de la liste de toutes les boutiques triées par business

Au clic, les données de la boutique s’affichent

Activation / désactivation d’une boutique

Lien pour envoyer un mail au gérant de boutique

Ecran de gestion des gérants de business

Affichage de la liste de tous les gérants de business

Formulaire de création / édition de gérants de business, affichage des données au clic

Analyse du besoin

QQOQCCP

Le QQOQCCP nous a permis d’identifier le contexte en synthétisant les données principales qui nous ont été transmises à travers le cahier des charges.

**Qui** : Client (fictif)

**Pour qui** : Gérant de boutique / gérant de business

Par qui : Stagiaire en développement et développeur titulaire

**Quoi** : Création d’une interface administrateur pour une application de fidélisation – Le backend et l’interface user sont existants

**Où** : Internet

**Quand** : Pour le 13 juillet

**Comment** : Sur la base d’existant ( backend et interface user), en PHP / Symfony / API REST / Angular

**Combien** : 2 personnes

**Pourquoi** : Commande transformée en étude suite à une demande non concrétisée

Fonctionnalités attendues

L’application Barbier doit permettre d’enregistrer des points de fidélité à partir d’un compte client.

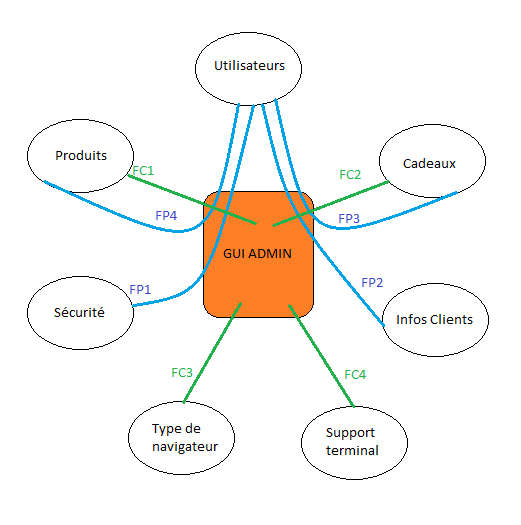
Le site devra être responsive.

Le client pourra souscrire à une ou plusieurs cartes, qui lui permettront d’obtenir des cadeaux.

Les points sont obtenus à l’achat d’un produit (ou prestation).

L’application devra pouvoir être utilisée par un gérant de boutique, et par un gérant de business (qui regroupe plusieurs boutiques), avec des accès différents permettant d’accéder à des fonctionnalités dédiées.

Diagramme de pieuvre



Le diagramme de pieuvre met en évidence les types de fonctionnalités attendues.

On distingue deux types de fonctions : les fonctions principales et les fonctions contraintes.

Les premières traduisent des actions réalisées par le produit. Les secondes concernent l’adaptation du produit à son environnement.

Tableau des fonctionnalités primaires et contraintes

Les fonctions précédemment identifiées sont mises en forme dans le tableau ci-dessous.

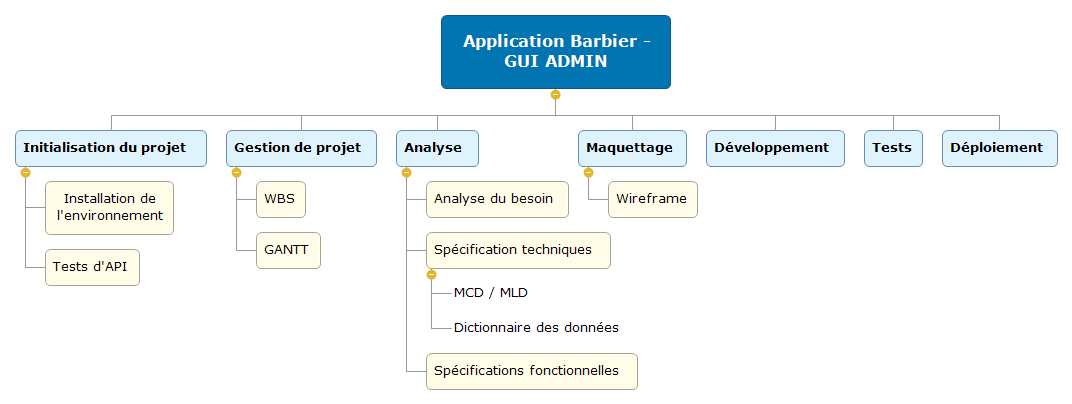
|  |  |  |
| --- | --- | --- |
| **Fonctions** | | **Critères d'appréciation** |
| **FP1** | Définir un accès sécurisé et relatif à l’interface d’administration | Mise en place d'un login |
| **FP2** | Permettre à l’utilisateur d’accéder aux informations client | Niveau d’autorisation |
| **FP3** | Permettre à l’utilisateur la consultation et/ou l’édition de produits | Niveau d’autorisation |
| **FP3** | Permettre à l’utilisateur la consultation et/ou l’édition de cadeaux | Niveau d’autorisation |
| **FC1** | Faire en sorte que le site soit compatible avec tous les terminaux | Nécessité d'être responsive |
| **FC2** | Faire en sorte que le site soit compatible avec tous les browsers | Tests correspondants |
| **FC3** | Permettre un accès à la base de données produit | Lecture et édition des données liées aux produits |
| **FC4** | Permettre un accès à la base de données cadeaux | Lecture et édition des données liées aux cadeaux |

Gestion de projet

WBS (*Work Breakdown Structure)*

Compte tenu du contexte d’agilité, le WBS a été défini de façon synthétique et haut niveau, de façon à englober les grandes étapes tout en permettant une flexibilité.

Etant donné le fonctionnement en agilité (et donc en itérations ), la conception d’un GANTT et d’un PERT n’était pas pertinente.



Spécifications fonctionnelles

Diagramme de cas d’utilisation

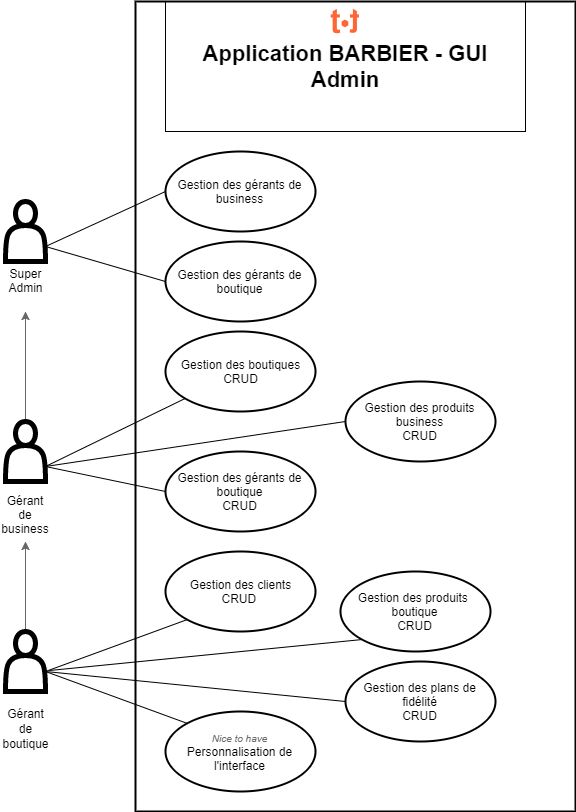
L’application devra être conçue pour 3 types d’utilisateurs, déterminés par leur type d’accès à celle-ci :

Gérant de magasin ou Shop Admin

Gérant de business ou Business Admin

Prestataire concepteur ou Super Admin

Nous avons ainsi 3 cas d’utilisation :



Use Cases

**Fonctionnalité : Définir un accès sécurisé et relatif à l’interface d’administration**

Use Case

Acteurs : Gérant de boutique (Shop Admin)

Description : L’utilisateur doit pouvoir se logger et accéder à l’espace de gestion de sa boutique

Pré-conditions : L’utilisateur doit disposer d’un accès avec login et mot de passe

Démarrage : L’utilisateur accède à la page de connexion de l’interface administrateur (GUI Admin)

Scénario nominal Gérant de boutique (Shop Admin)

1. Le système affiche une page contenant de login
2. Le Shop Admin saisit ses identifiants et mot de passe et clique sur le bouton Valider
3. Le système affiche la page de Dashboard qui fait office de page d’accueil de sa boutique. Un menu y apparait avec les liens vers les clients, produits et plans de fidélité, ainsi que des widgets pour ces trois entités faisant apparaitre leurs soldes respectifs.

**Fonctionnalité : Editer l’adresse mail d’un client existant**

Use Case

Acteurs : Gérant de boutique (Shop Admin)

Description : Trouver un client existant et modifier son adresse email

Pré-conditions : L’utilisateur doit disposer d’un accès avec login et mot de passe

Démarrage : L’utilisateur a accès à la page de Dashboard après s’être loggué

Scénario nominal Gérant de boutique (Shop Admin)

1. Le système affiche la page de Dashboard
2. Le Shop Admin clique sur le bouton Clients du menu latéral
3. Le système affiche la liste client de la boutique et un formulaire vide
4. Le Shop Admin choisit le client dans la liste et clique dessus
5. Le système remplit le formulaire avec les données du client
6. Le Shop Admin saisit une nouvelle adresse email dans le champ correspondant
7. Le Shop Admin valide l’opération en cliquant sur le bouton Valider du formulaire

Tableau des fonctionnalités demandées dans le cahier des charges

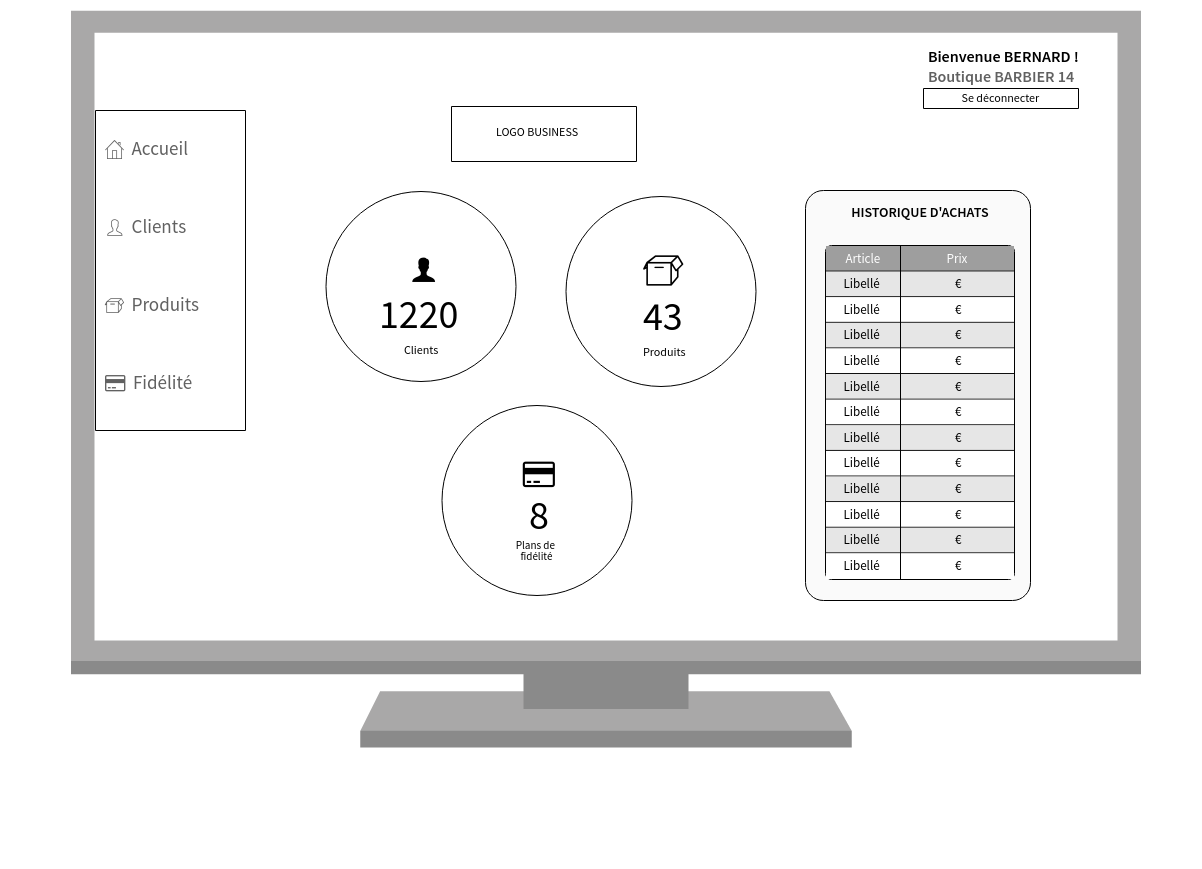
|  |  |  |
| --- | --- | --- |
| **Exigence** | **Type d'utilisateur** | **Réponse** |
| Accéder à un espace sécurisé | Shop Admin  Business Admin  Super Admin | Requête vers la BDD via doctrine, utilisation du CRUD dans le back end (REST + SQL) |
| Consulter une liste clients | Shop Admin  Business Admin  Super Admin | Requête vers la BDD via doctrine, utilisation du CRUD dans le back end (REST + SQL) |
| Consulter une fiche client | Shop Admin  Business Admin  Super Admin | Requête vers la BDD via doctrine, utilisation du CRUD dans le back end (REST + SQL) |
| CRUD client | Shop Admin  Business Admin  Super Admin | Requête vers la BDD via doctrine, utilisation du CRUD dans le back end (REST + SQL) |
| CRUD produit | Shop Admin  Business Admin  Super Admin | Requête vers la BDD via doctrine, utilisation du CRUD dans le back end (REST + SQL) |
| CRUD cadeau | Shop Admin  Business Admin  Super Admin | Requête vers la BDD via doctrine, utilisation du CRUD dans le back end (REST + SQL) |
| Ajouter une boutique | Business Admin  Super Admin | Requête vers la BDD via doctrine, utilisation du CRUD dans le back end (REST + SQL) |
| Ajouter un shopAdmin | Business Admin  Super Admin | Requête vers la BDD via doctrine, utilisation du CRUD dans le back end (REST + SQL) |
| Ajouter un business | Super Admin | Requête vers la BDD via doctrine, utilisation du CRUD dans le back end (REST + SQL) |

MAQUETTAGE

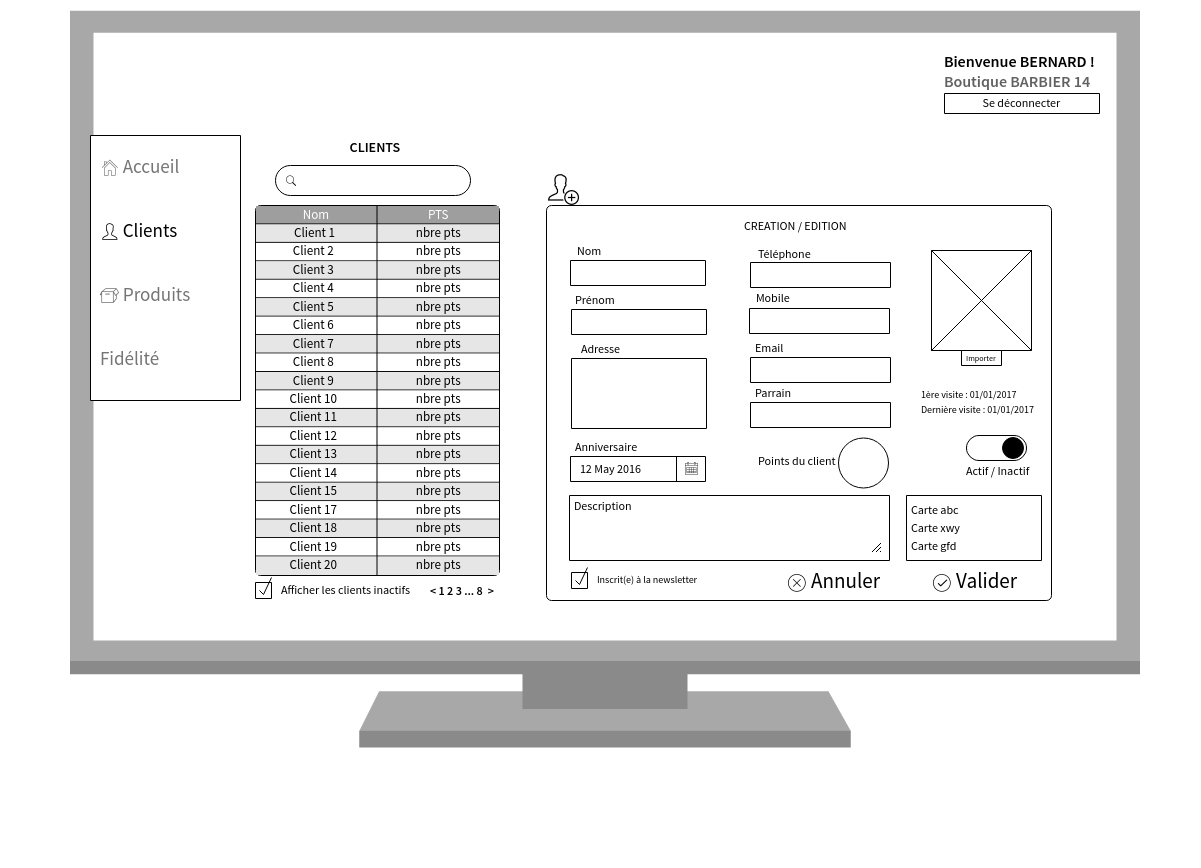
Wireframes

**Accueil**

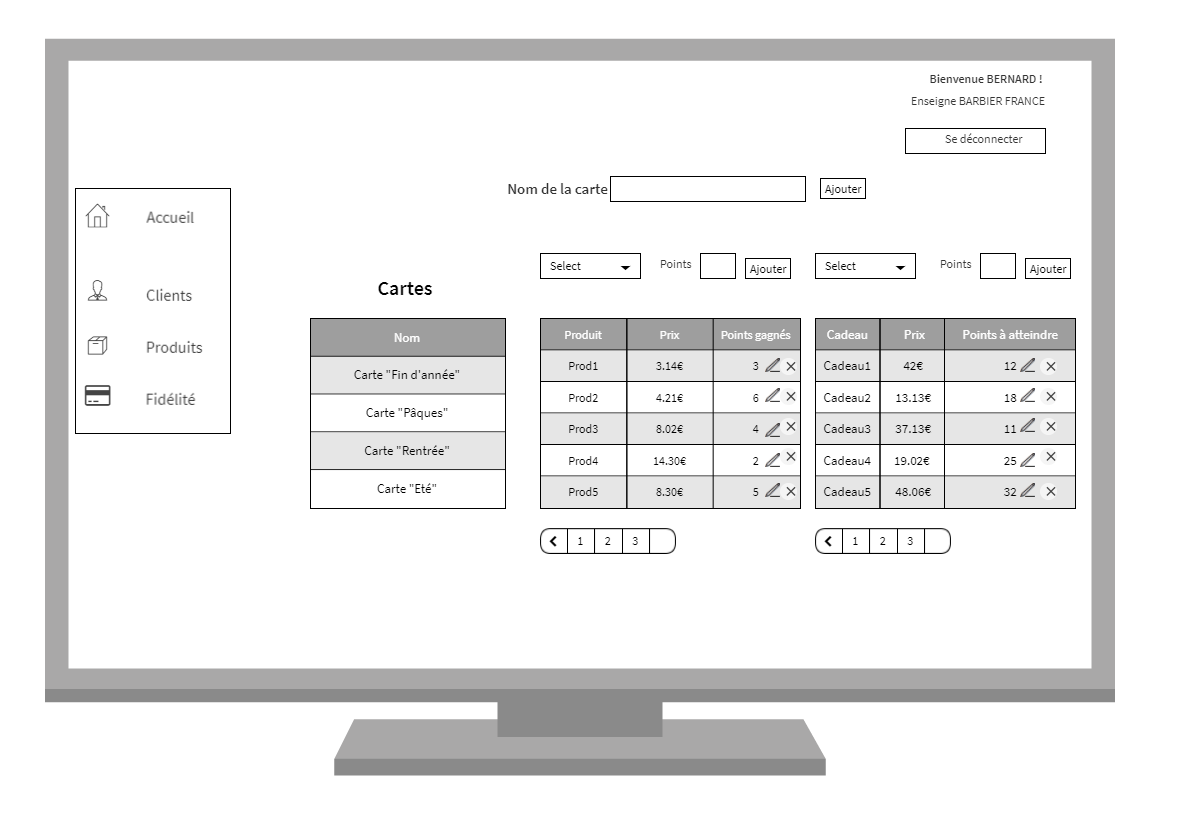
**Dashboard – Shop Admin**



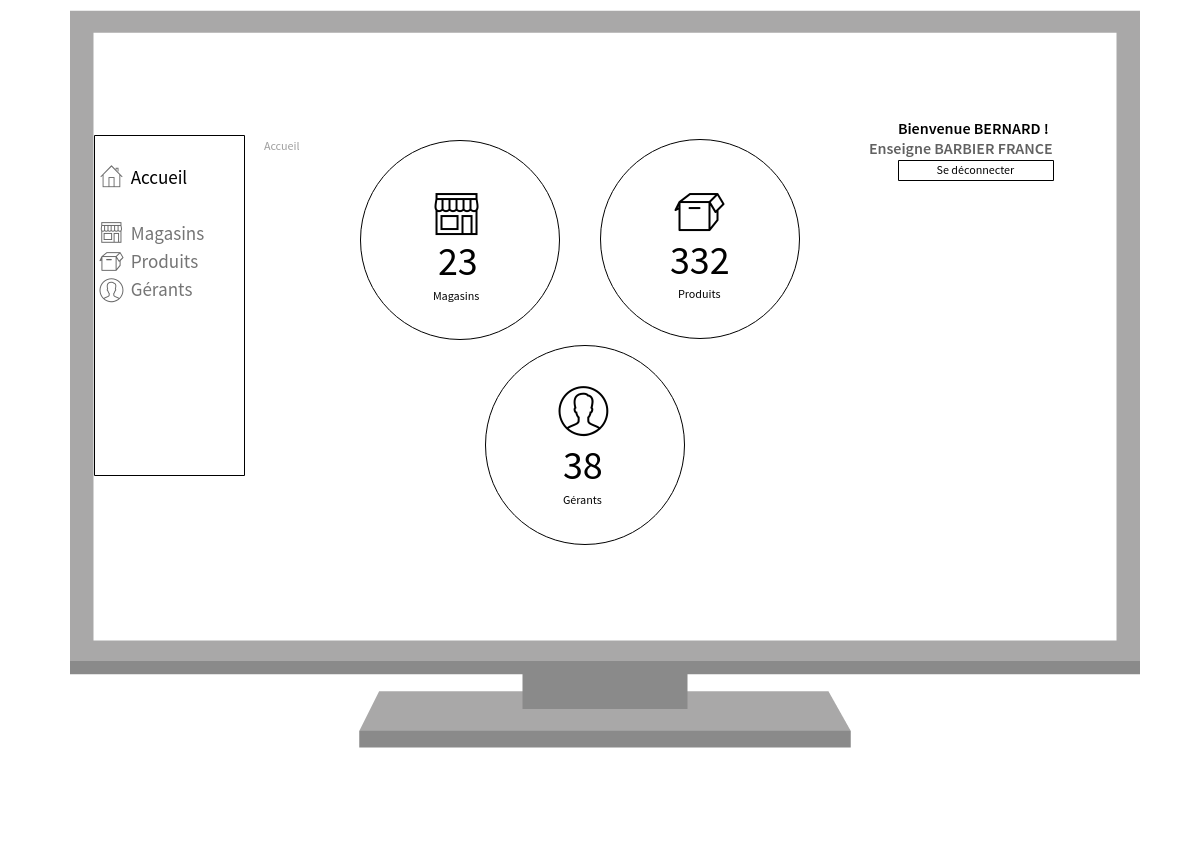
**Page Clients – Shop Admin**



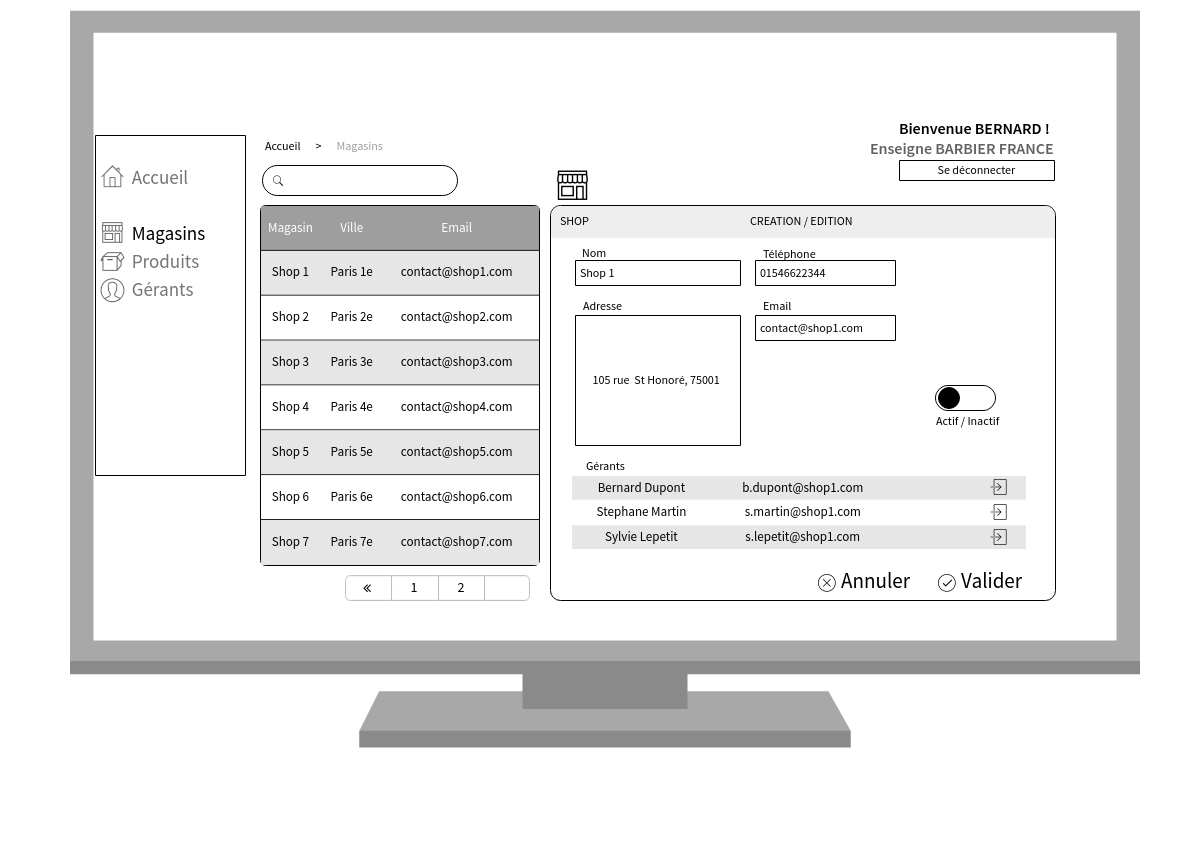
**Page Fidélité – Shop Admin**



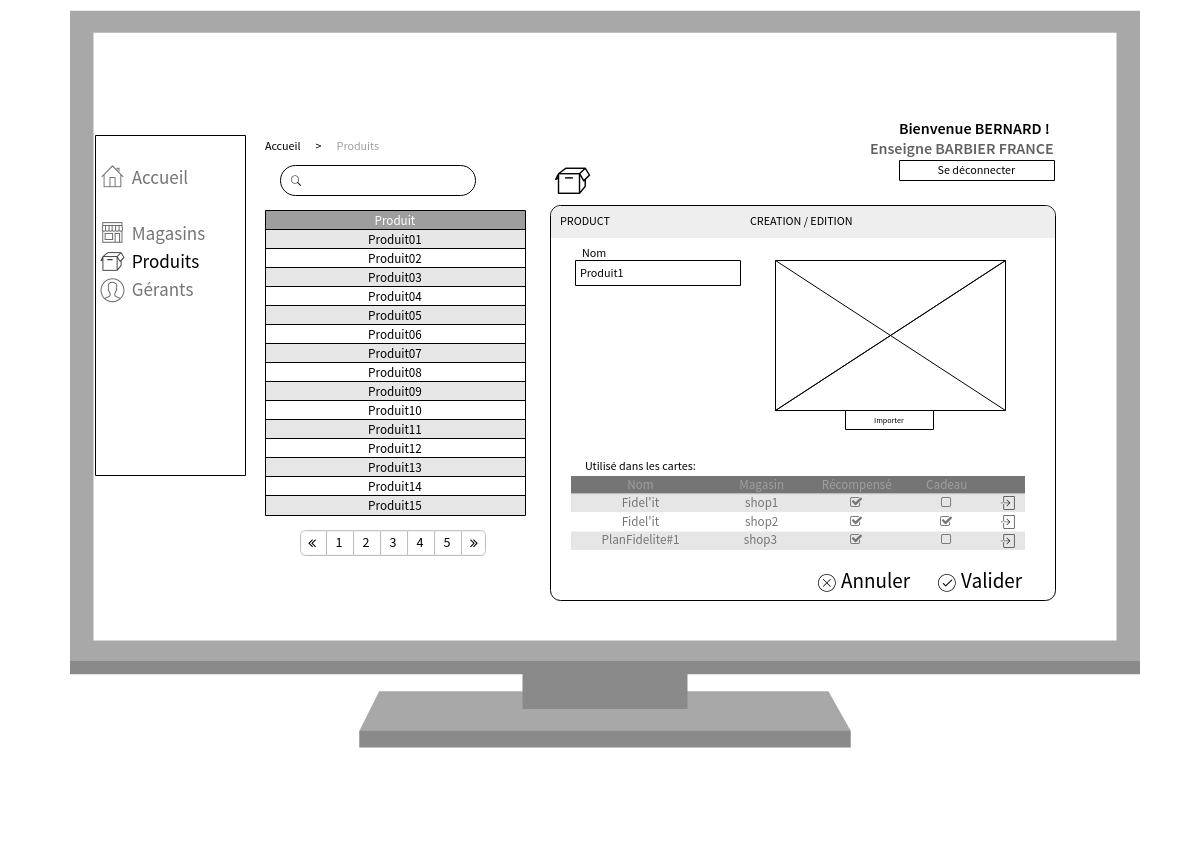
**Dashboard – Business Admin**



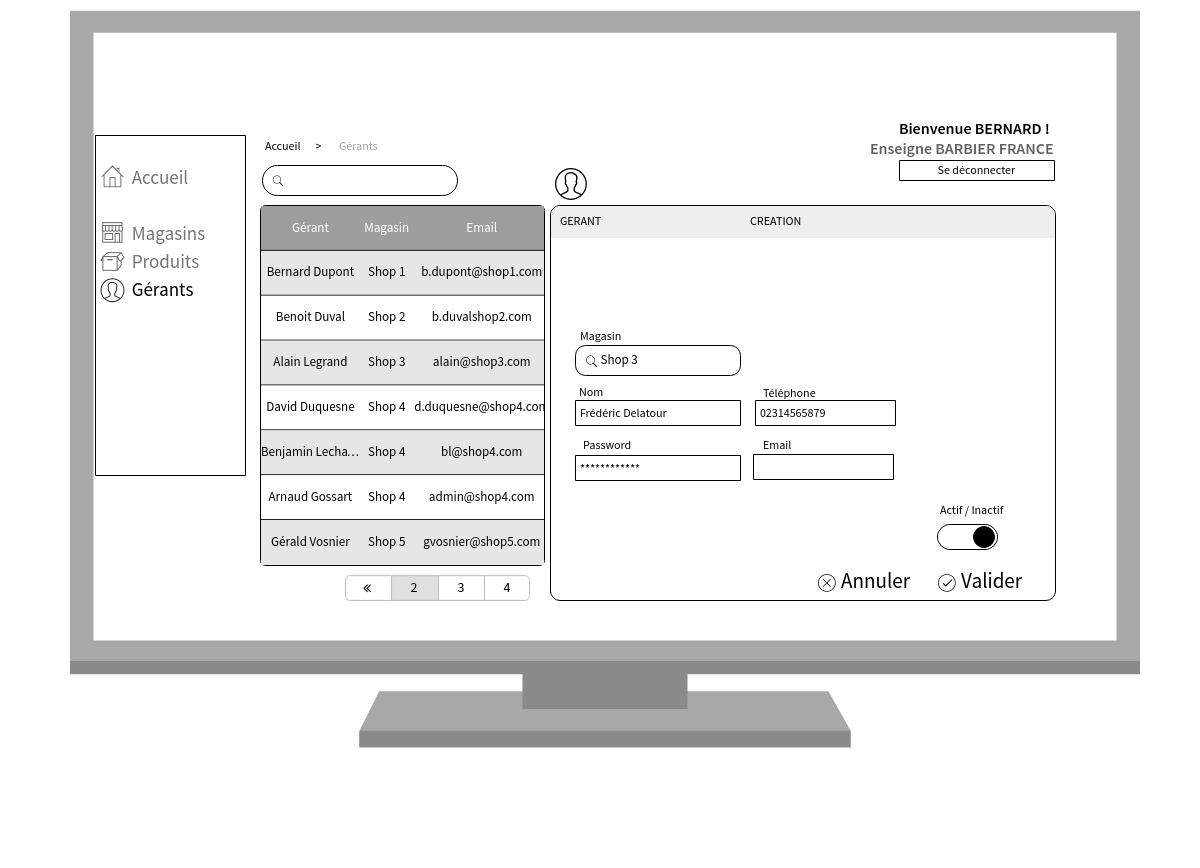
**Page Magasins – Business Admin**



**Page Produits – Business Admin**



**Page Gérants – Business Admin**



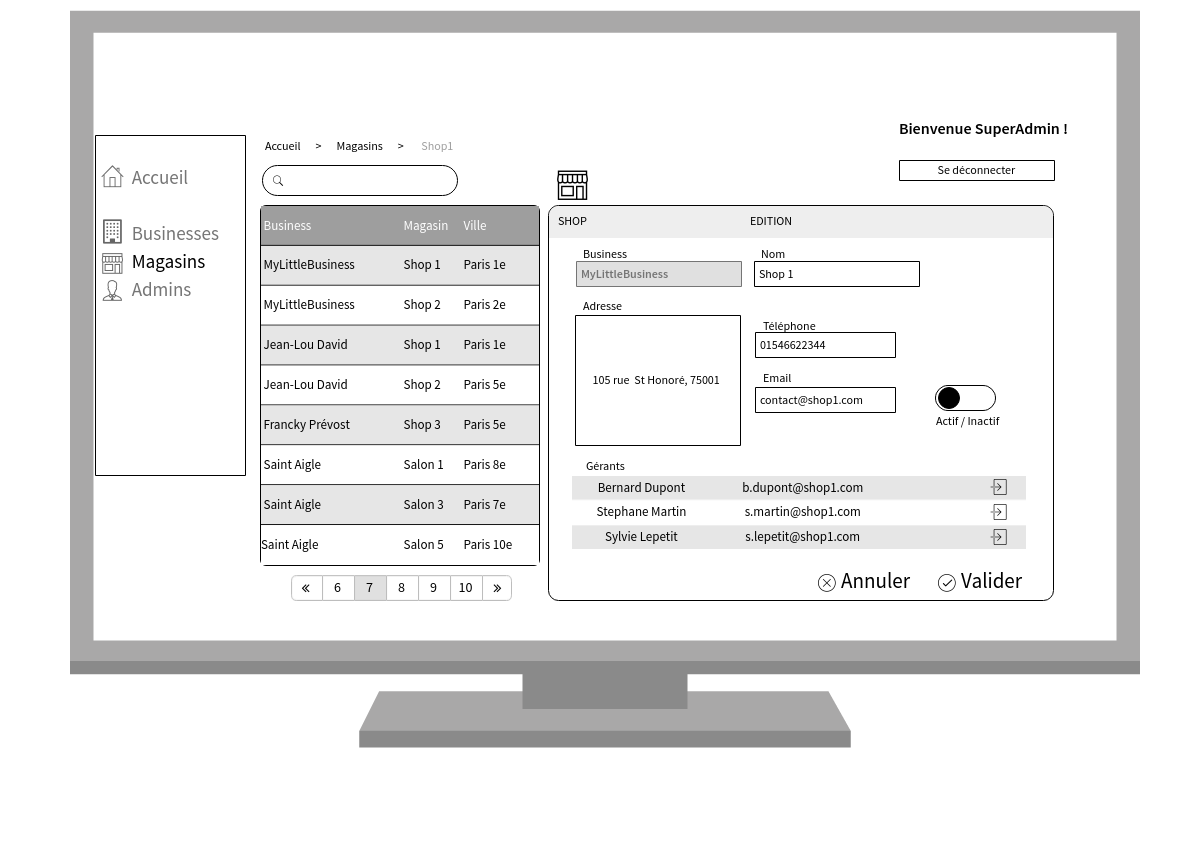
**Page Accueil – Super Admin**



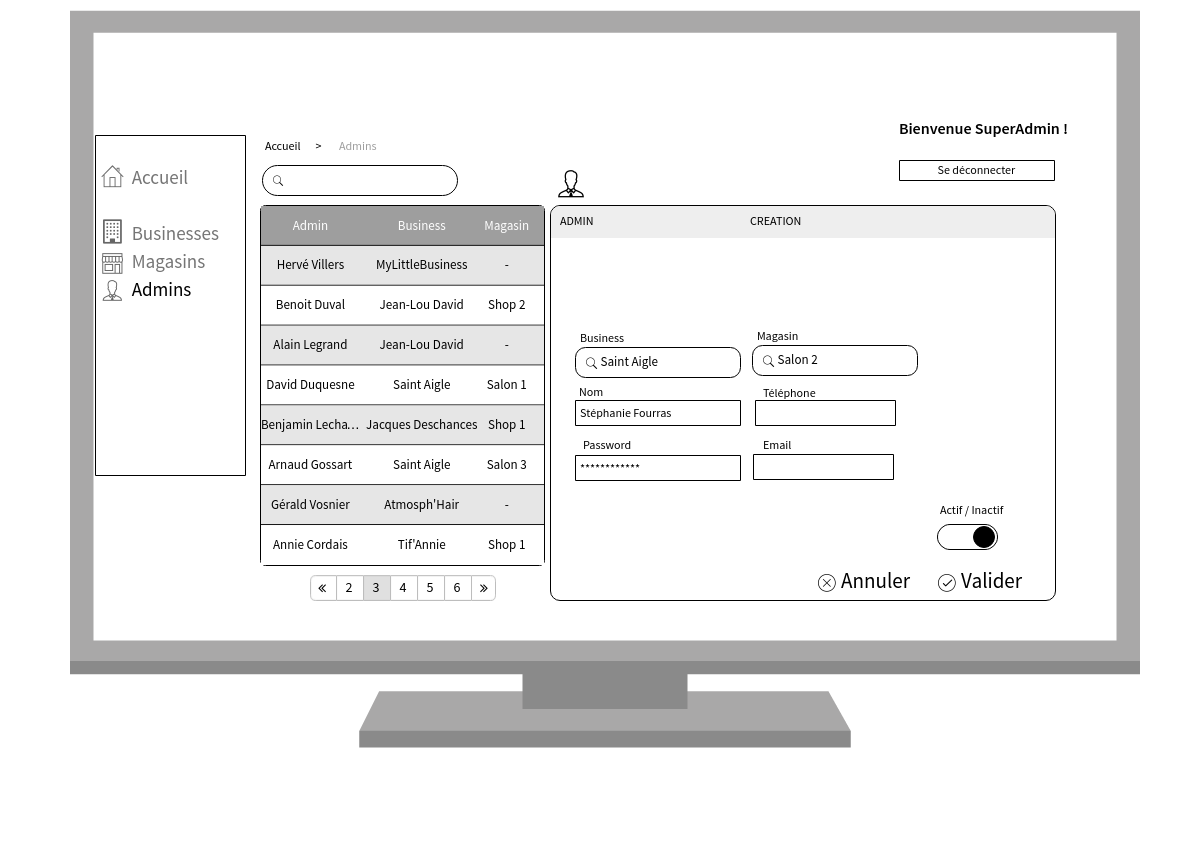
**Page Business – Super Admin**



**Page Magasins – Super Admin**

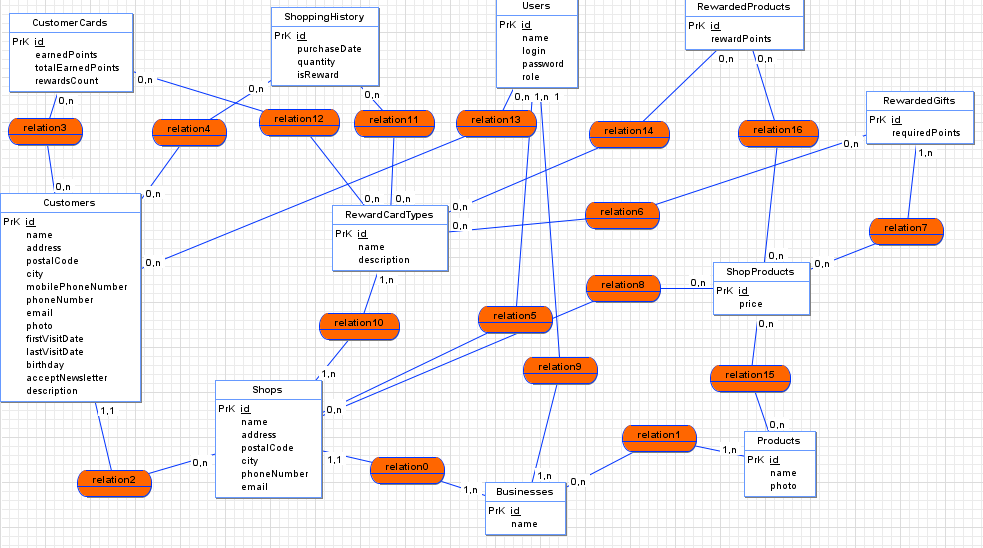


**Page Gérants de boutique – Super Admin**



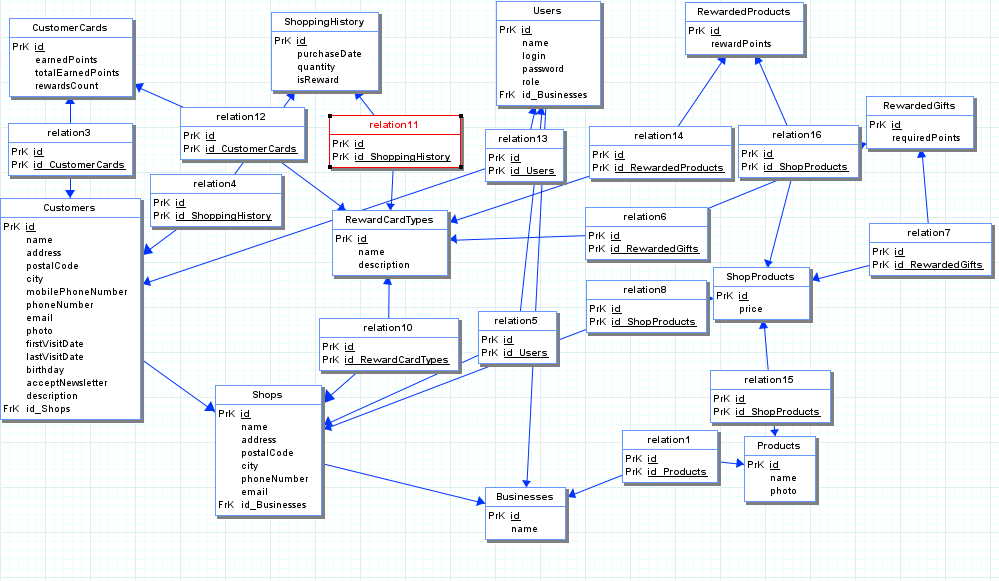
Spécifications techniques générales

MCD



*\* La relation 13 entre Customers et Users a été créée en vue de permettre à terme à un client d’accéder à un compte utilisateur pour gérer lui-même son compte*

MLD



Conception de la base de données

**Script SQL**

#------------------------------------------------------------

# Table: Products

#------------------------------------------------------------

CREATE TABLE Products(

id int (11) Auto\_increment NOT NULL ,

name Varchar (50) ,

photo Varchar (120) ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: Businesses

#------------------------------------------------------------

CREATE TABLE Businesses(

id int (11) Auto\_increment NOT NULL ,

name Varchar (25) NOT NULL ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: ShopProducts

#------------------------------------------------------------

CREATE TABLE ShopProducts(

id int (11) Auto\_increment NOT NULL ,

price Integer ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: ShoppingHistory

#------------------------------------------------------------

CREATE TABLE ShoppingHistory(

id int (11) Auto\_increment NOT NULL ,

purchaseDate Datetime NOT NULL ,

quantity Integer NOT NULL ,

isReward Varchar (25) ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: Users

#------------------------------------------------------------

CREATE TABLE Users(

id int (11) Auto\_increment NOT NULL ,

name Varchar (50) NOT NULL ,

login Varchar (25) NOT NULL ,

password Varchar (25) NOT NULL ,

role Varchar (25) ,

id\_Businesses Int NOT NULL ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: Customers

#------------------------------------------------------------

CREATE TABLE Customers(

id int (11) Auto\_increment NOT NULL ,

name Varchar (25) NOT NULL ,

address Varchar (25) ,

postalCode Integer ,

city Varchar (25) ,

mobilePhoneNumber Varchar (25) ,

phoneNumber Varchar (25) ,

email Varchar (25) NOT NULL ,

photo Varchar (120) ,

firstVisitDate Datetime ,

lastVisitDate Datetime ,

birthday Datetime ,

acceptNewsletter Bool ,

description Varchar (500) ,

id\_Shops Int NOT NULL ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: RewardCardTypes

#------------------------------------------------------------

CREATE TABLE RewardCardTypes(

id int (11) Auto\_increment NOT NULL ,

name Varchar (25) NOT NULL ,

description Varchar (250) ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: RewardedProducts

#------------------------------------------------------------

CREATE TABLE RewardedProducts(

id int (11) Auto\_increment NOT NULL ,

rewardPoints Integer NOT NULL ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: CustomerCards

#------------------------------------------------------------

CREATE TABLE CustomerCards(

id int (11) Auto\_increment NOT NULL ,

earnedPoints Integer ,

totalEarnedPoints Integer ,

rewardsCount Integer ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: Shops

#------------------------------------------------------------

CREATE TABLE Shops(

id int (11) Auto\_increment NOT NULL ,

name Varchar (50) ,

address Varchar (75) NOT NULL ,

postalCode Int ,

city Varchar (25) ,

phoneNumber Varchar (25) ,

email Varchar (40) ,

id\_Businesses Int NOT NULL ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

#------------------------------------------------------------

# Table: RewardedGifts

#------------------------------------------------------------

CREATE TABLE RewardedGifts(

id int (11) Auto\_increment NOT NULL ,

requiredPoints Integer NOT NULL ,

PRIMARY KEY (id )

)ENGINE=InnoDB;

**Dictionnaire de données**

|  |  |  |  |
| --- | --- | --- | --- |
| **Entités** | **Attributs** | **Type** | **Description** |
| **Business** | id | Integer /Auto Increment | Id |
|  | name | Varchar | Nom du business |
| **Customer** | id | Integer /Auto Increment | Id |
|  | name | Varchar | Nom du client |
|  | adress | Varchar | Adresse du licencié |
|  | postalCode | Integer | Code postal |
|  | city | Varchar | Ville |
|  | mobilePhoneNumber | Varchar | Numéro de portable |
|  | phoneNumber | Varchar | Numéro de téléphone fixe |
|  | email | Varchar | Email |
|  | photo | Varchar | photo |
|  | firstVisitDate | Datetime | Date de première visite |
|  | lastVisitDate | Datetime | Date de dernière visite |
|  | birthday | Datetime | Date d’anniversaire |
|  | acceptNewsletter | Boolean | Inscription à la newsletter |
|  | description | Varchar | Description du client |
| **CustomerCards** | id | Integer /Auto Increment | Id |
|  | earnedPoints | Integer | Points gagnés à l’achat |
|  | totalEarnedPoints | Integer | Total des points gagnés |
|  | rewardsCount | Integer | Points restants après achat |
| **Products** | id | Integer /Auto Increment | Id |
|  | name | Varchar | Nom du produit |
|  | photo | Varchar | Photo du produit |
| **ShoppingHistory** | id | Integer /Auto Increment | Id |
|  | purchaseDate | Datetime | Date d’achat |
|  | quantity | Integer | Nombre de produits |
|  | isReward | Varchar | Produit rapportant des points |
| **ShopProducts** | id | Integer /Auto Increment | Id |
|  | price | integer | Prix |
| **Shops** | id | Integer /Auto Increment | Id |
|  | name | Varchar | Nom de la boutique |
|  | address | Varchar | Adresse de la boutique |
|  | psotalCode | Integer | Code Postal |
|  | city | Varchar | Ville |
|  | phoneNumber | Varchar | Numéro de téléphone |
|  | email | Varchar | email |
| **RewardCardTypes** | id | Integer /Auto Increment | Id |
|  | name | Varchar | Nom de la carte de fidélité |
|  | description | Varchar | Description des conditions |
| **RewardedGifts** | id | Integer /Auto Increment | Id |
|  | requiredPoints | Integer | Points requis |
| **RewardedProducts** | id | Integer /Auto Increment | Id |
|  | rewardPoints | Integer | Points requis |
|  |  |  |  |
| **Users** | id | Integer /Auto Increment | Id |
|  | name | Varchar | Nom de l’utilsateur (ShopAdmin, BusinessAdmin ou SuperAdmin) |
|  | login | Varchar | Identifiant de connexion |
|  | Password | Varchar | Mot de passe de connexion |
|  | role | Varchar | Fonction / Niveau d’accès |

Architecture du projet

Backend

Le backend est implémenté en PHP avec les frameworks suivants :

* Le framework Symfony3 a été utilisé pour l’architecture. Il s’agit d’un framework PHP de type MVC (Model View Controller). Les routes et relations sont gérées en annotations.

Il est utilisé pour la gestion de la persistance (avec l'ORM doctrine) et la gestion globale du projet (tests, fixtures, injection de dépendances, ...)

* API Platform, qui est un framework au-dessus de Symfony, pour gérer l'API REST pour pouvoir être redondé simplement, le backend ne stocke pas le contexte de ses clients, grâce à JWT (JSON Web Token) : chaque requête vers le backend transporte ses propres informations (notamment de connexion).

Les réponses du backend sont formatées par défaut en JSON-LD, qui est une extension de JSON avec des informations supplémentaires, par exemple les informations de pagination dans le cas d'une collection.

Une page spéciale nommée entrypoint, générée automatiquement par API Platform, fournit la documentation de l'API, au format HTML pour être lue dans le navigateur, ou au format JSON-LD pour être interprétée par une autre application.

**Frontend**

Le frontend est implémenté avec Angular2/4.

Pour faciliter le développement, un outil NodeJS a été conçu pour générer automatiquement les classes TypeScript depuis le backend. La commande ng generate génère les services et les composants. L'outil dont il est question dans cette partie pour générer les classes de ressources, validateurs et services de   
communication est lysis :

*https://www.npmjs.com/package/api-lysis*

* génération des classes de ressources (Customer, Product, ...)
* génération des validateurs de formulaire
* génération des services de communication avec le backend : mise à disposition extensible des classes de gestion du CRUD (get, create, ...) adaptées au typage strict de TypeScript, pour améliorer la qualité logicielle.

**Présentation d’Angular 4**

Angular est un framework Javascript développé par Google. Il est basé sur la structure de la première version d’Angular : AngularJS.

Angular utilise notamment ces trois éléments particuliers :

### Les composants *(cf. le détail d’un composant dans la partie code)*

En Angular 2/4, tout est composant. C’est le seul élément que l’on peut présenter sur une page.

Il contient :

* Le « template » de présentation (HTML + CSS), qui permet de rappeler des contextes
* Les « services » injectés directement sur le composant
* Le code du composant et la logique de gestion des données.

### Les services

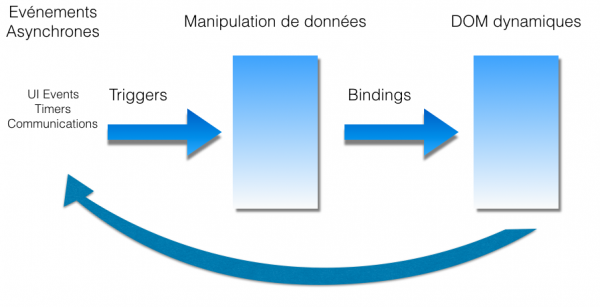
Un service est le mécanisme utilisé pour partager des fonctionnalités sur des composants : c’est la mécanique de base préconisée sur Angular 2 pour mutualiser des comportements applicatifs

### Le typage avec Typescript

C’est la grande nouveauté sur le Framework Angular 2, il est désormais possible de typer les données manipulées par les composants et les services. À mesure qu’une application croit en taille, il devient important qu’elles soient bien organisées avec une structure typée en place.

Les services et composants doivent être déclarés dans le fichier app.module.ts, qui regroupe les classes principales (ou génériques) d’Angular, les services et les composants.

Le Data Binding est un élément essentiel. Dans un système d’évènements asynchrones, il permet de synchroniser la vue au modèle. Voici un schéma général du fonctionnement du Data Binding :



Il utilise un système de traitement des requêtes asynchrone en utilisant le design pattern des observables, qui fonctionne sur principe d’abonnement *(subscribe).* Les objets ayant souscrit à l’observable (qui est déclaré dans le service) récupèrent ainsi sa classe lors d’un évènement.

Angular utilise le langage Typescript. Typescript représente un sur-ensemble de Javascript qui le sécurise et le simplifie. Il amène plusieurs fonctionnalités de Java au langage, notamment le typage de variables et des fonctions, l’import de modules et la création de classes et interfaces.

**Typescript**

Angular 2 est entièrement disponible en TypeScript et répond à la spécification ECMAScript 6 (TypeScript est une extension de la norme ES6).

 Il a pour but d'améliorer et de sécuriser la production de code [JavaScript](https://fr.wikipedia.org/wiki/JavaScript). C'est un sur-ensemble de JavaScript (c'est-à-dire que tout code JavaScript correct peut être utilisé avec TypeScript). Le code TypeScript est [transcompilé](https://fr.wikipedia.org/wiki/Compilateur_source_%C3%A0_source) en [JavaScript](https://fr.wikipedia.org/wiki/JavaScript), pouvant ainsi être interprété par n'importe quel navigateur web ou [moteur JavaScript](https://fr.wikipedia.org/wiki/Moteur_JavaScript).

Intégration continue

Le backend et le frontend intègrent le nécessaire pour construire les images docker (le fichier Dockerfile).

Une partie du travail était de trouver des solutions innovantes pour construire, tester et déployer le plus automatiquement possible et avec fiabilité.

Cette automatisation est permise par GitLab et la présence dans chaque projet (le backend et le frontend) d'un descriptif des actions à exécuter automatiquement au moment de modifications dans les sources (au commit/push) :

* construction de l'image
* test automatique
* déploiement sur la plateforme de recette (staging)
* après validation manuelle : déploiement sur la plateforme de production

L'intégration continue est en place, il reste le déploiement continu.

**Intégration**

Nous avons utilisé un système d’intégration continue avec gitlab, en se basant sur des outils de test automatiques (à partir de scénario écrits en Gherkin - Behat)

Chacun des éléments doit fonctionner de façon autonome (tests unitaires), l’application globale également.

**Déploiement**

Quand on aura atteint un niveau fonctionnel déterminé, on pourra procéder au déploiement du travail de la partie DEV à la partie PROD

**NoSQL**

La base de données pourra être changée en base NoSQL (Cassandra, MongoDB, ...)

API REST

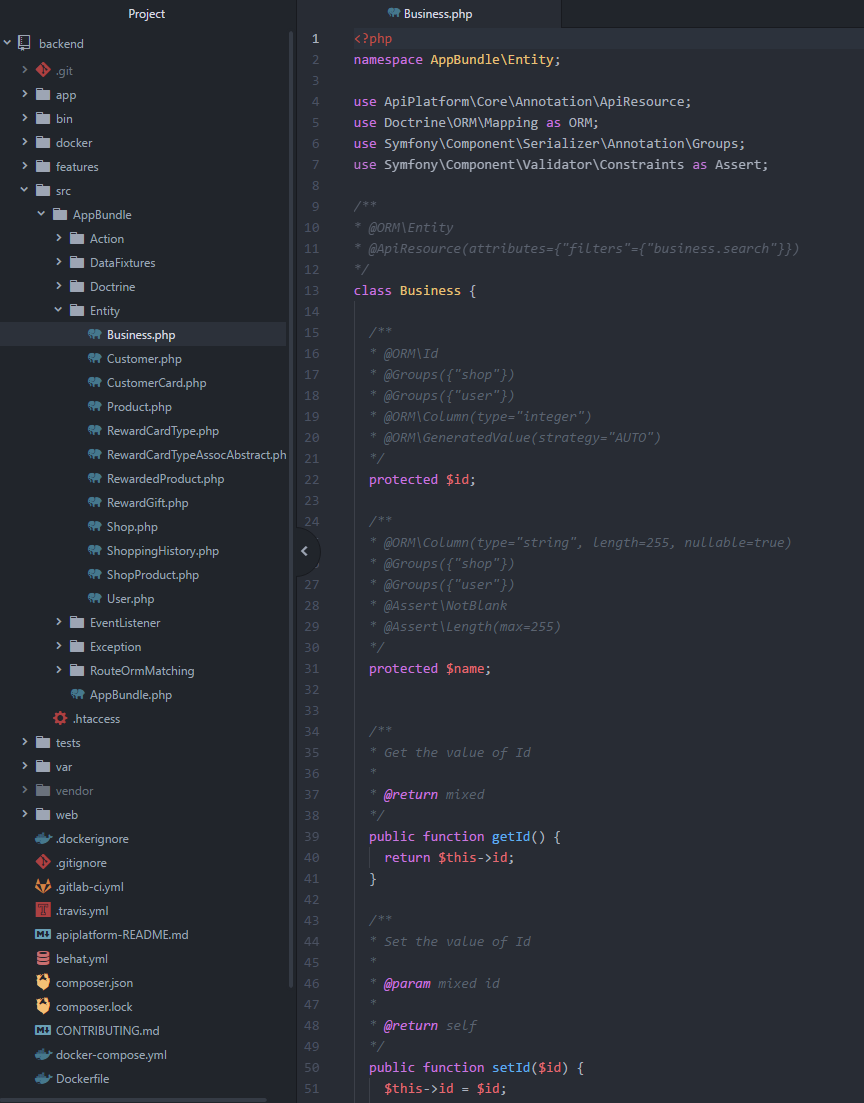
Pour la gestion de l’application, il a été choisi d’utiliser une API REST (système de réprésentation des transferts)

API Platform repose sur trois principes :

* créer une API facilement et rapidement : n’importe quel développeur Web doit pouvoir créer une API REST et obtenir le support du CRUD, une documentation auto‐générée, l’authentification via JSON Web Token ou OAuth, la gestion des en‐têtes CORS, la validation des données, les tris et les filtres, le cache, etc. ;
* les formats ouverts modernes sont gérés nativement, sans demander de travail supplémentaire pour le développeur : Swagger/OpenAPI, JSON-LD, Hydra, HAL, API Problem (RFC 7807), Schema.org sont pris en charge par défaut et la couche d’abstraction fournie par le cadriciel permet d’ajouter facilement le prise en charge d’autres nouveaux formats d’API (JSON API et GraphQL sont actuellement en cours de développement) ;
* chaque fonctionnalité du cadriciel doit être extensible, modulaire et débrayable.

Back End

L’architecture du Backend est implémentée en Symfony. On retrouve la liste des entités



Commande pour lancer le serveur du backend : php bin\console s :r

Fichier de test – Behat – Cascade Deletion \_

*Compétence mise en œuvre : Modifier une base de données*

A la prise en main du projet, j’ai effectué des tests d’API avec Gherkin sur le backend qui avait été initialisé préalablement.

Cas d’utilisation : Suppression d’un client

Ici on verifie qu’à la suppression d’un client, ses attributs sont également supprimés.



Backend- Présentation d’une entité

L’entité est déclarée avec ses attributs sous forme d’annotations par le biais de Doctrine

<?php

namespace AppBundle\Entity;

use ApiPlatform\Core\Annotation\ApiResource;

use Doctrine\ORM\Mapping as ORM;

use Symfony\Component\Serializer\Annotation\Groups;

use Symfony\Component\Validator\Constraints as Assert;

/\*\*

\* @ORM\Entity

\* @ApiResource(attributes={"filters"={"customer.search","customer.date"}})

\*/

*class* Customer {

/\*\*

\* @ORM\Id

\* @ORM\Column(type="integer")

\* @ORM\GeneratedValue(strategy="AUTO")

\* @Groups({"user"})

\*/

protected $id;

/\*\*

\* @ORM\Column(type="string", length=255, nullable=true)

\* @Groups({"user"})

\* @Assert\NotBlank

\* @Assert\Length(max = 255)

\*/

protected $name;

/\*\*

\* @ORM\Column(type="string", length=255, nullable=true)

\* @Groups({"user"})

\* @Assert\Length(max = 255)

\*/

protected $address ;

/\*\*

\* @ORM\Column(type="string", length=32, nullable=true)

\* @Groups({"user"})

\* @Assert\Length(max = 32)

\*/

protected $postalCode;

Exemple de jointure pour récupérer le nom des cartes d’un client lors de l’affichage des données clients :

SELECT CUS.name, RCT.name FROM RewardCardTypes RCT  
INNER JOIN CustomerCards CC ON RCT.id = CC.RewardCardType\_id  
INNER JOIN Customers CUS ON CC.Customer\_id = CUS.id  
WHERE CUS.id = 1

Présentation de composants du frontend

**Composant *Customer***

Ce composant correspond à la page de gestion des clients.

La route du composant a été déclarée préalablement dans le fichier src/app/app-routing.module.ts

import { Component, OnInit } from '@angular/core';

import { ActivatedRoute } from '@angular/router';

import { FormGroup, FormBuilder } from '@angular/forms';

Importation de composants génériques d’Angular

import { CustomersService, CustomerClass } from '../app-services/customers.service';

import { AdminContextIdService } from '../app-services/admin-context-id.service';

import { CustomerCardsService, CustomerCardClass } from '../app-services/customerCards.service';

import { CustomerValidator } from '../form-validators/customer-validator';

Importation des services et des classes auxquels on veut pouvoir accéder

@Component({

selector: 'app-customers',

templateUrl: './customers.component.html',

styleUrls: ['./customers.component.less']

})

Déclaration des chemins du template, du fichier de style

export *class* CustomersComponent implements *OnInit* {

private shopId: *number*;

private customers: CustomerClass[];

private currentCustomer: CustomerClass;

private currentCustomerCards: CustomerCardClass[];

private form: FormGroup;

private tableConfig: Object;

Déclaration de la classe du composant avec ses attributs propres et leurs typages

*constructor*(

private *activatedRoute*: ActivatedRoute,

private *adminContextId*: AdminContextIdService,

private *customersService*: CustomersService,

private *formBuilder*: FormBuilder,

private *customerCardsService*: CustomerCardsService

) { }

Injection des services pour le constructeur de la classe

ngOnInit() {

this.adminContextId.get('shop').subscribe(*shopId* *=>* {

this.shopId = shopId;

this.refresh();

});

//this.selectCustomer(new CustomerClass());

this.setNewCustomer();

this.tableConfig = {

columns: {

name: 'Client',

'firstVisitDate | date': 'Date de 1ere visite'

}

};

}

A l’affichage du composant la fonction ngOnInit appelle un ensemble de méthodes définies

Le tableau doit afficher les données *Client* et *Date de première visite*

refresh(*pageNumber*: *number* = 1) {

this.customersService.getCustomersByShopId(this.shopId, pageNumber).subscribe(

*customers* *=>* this.customers = customers

);

}

La méthode refresh récupère les clients depuis le backend, par défaut la première page.

selectCustomer (*customer* : CustomerClass) : *void* {

this.currentCustomer = customer;

this.form = this.formBuilder.group(newCustomerValidator().getGroup(this.currentCustomer));

if (customer.id) {

this.customerCardsService.getCardsByCustomerId(customer.id).subscribe(

*cards* *=>* this.currentCustomerCards = cards

);

}

}

Méthode qui, au clic sur un client dans le tableau, l'affiche dans le formulaire

setNewCustomer(){

this.selectCustomer(new CustomerClass);

}

Méthode pour créer un nouvel objet customer

addCustomer() {

if (this.form && this.form.valid) {

*Object*.assign(this.currentCustomer, this.form.value);

if (!this.currentCustomer.id) {

this.currentCustomer.setFavoriteShop(this.shopId);

}

if (!this.currentCustomer.birthday) this.currentCustomer.birthday = null;

if (!this.currentCustomer.id) {

this.customersService.addCustomer(this.currentCustomer).subscribe(*customer* *=>* *console*.log(customer));

} else {

this.customersService.updateCustomer(this.currentCustomer).subscribe(*customer* *=>* *console*.log(customer));

}

} else {

// force touched marker for errors, to display which fields are not valid

for (*let* controlName in this.form.controls) {

*let* control = this.form.controls[controlName];

if (control.invalid && !control.touched) control.markAsTouched();

}

}

}

}

Méthode pour créer ou modifier un client côté backend une fois le formulaire renseigné et valide

Morceau du template du composant customer : Le tableau des cartes de fidélité d’un client

<div class="form-group row">

<div class="col-lg-10 orders-container">

<table class = "table">

<thead>

<tr>

<th>Cartes </th>

<th>Points</th>

<th>Cartes épuisées</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let card of currentCustomerCards">

<td>{{card.rewardCardType.name}}</td>

<td>{{card.earnedPoints}}</td>

<td>{{card.rewardsCount}}</td>

</tr>

</tbody>

</table>

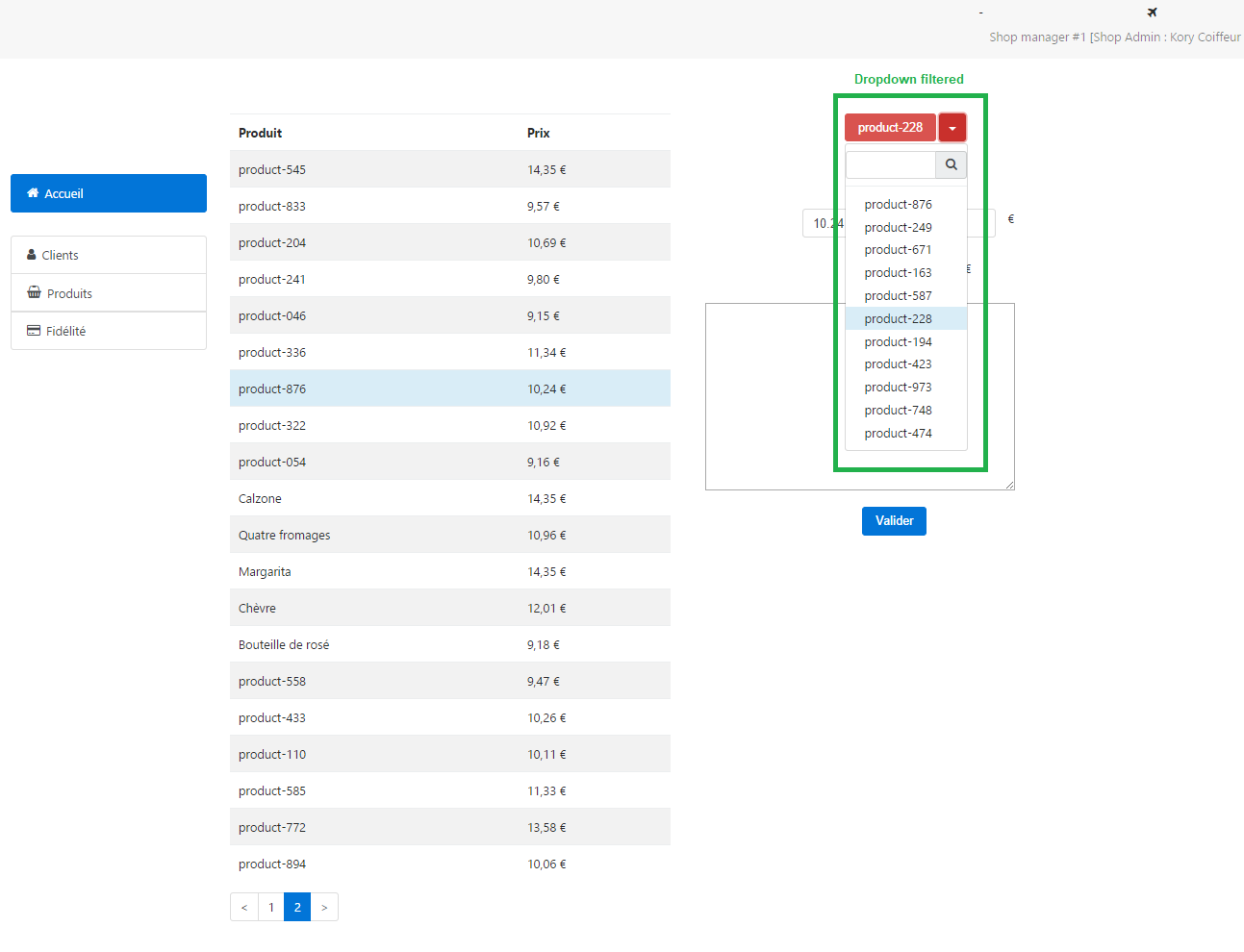
</div>

</div>

Ce tableau permet de récupérer les cartes de fidélité d’un client en utilisant les propriétés de la classe CustomerCard via son service. Ce dernier a été injecté dans le modèle customers.component.ts .

Les doubles accolades {{…}} sont utilisées pour binder les propriétés du contrôleur avec leur pendant dans le template.

Sous-composant *Filtered-Dropdown –* Ecran Produits de la boutique



Fonction : cet élément (champ de recherche auto-complété avec liste déroulante) se retrouve dans l’écran de gestion des produits et l’écran de gestion des plans de fidélité. Il doit permettre de rechercher une donnée (ici les produits liés à un business ainsi que leurs prix).

Il a été décidé de créer un composant à part entière qui sera rappelé dans chacun des templates, appelant des méthodes différentes qui ont été déclarée dans le modèle correspondant de la page appelée.

Code du composant :

import { Component, OnInit, Input, Output, forwardRef } from '@angular/core';

import { ControlValueAccessor, NG\_VALUE\_ACCESSOR } from '@angular/forms';

import { EventEmitter } from '@angular/core';

import { Observable } from 'rxjs/Observable';

@Component({

selector: 'filtered-dropdown',

templateUrl: './filtered-dropdown.component.html',

styleUrls: ['./filtered-dropdown.component.less'],

providers: [{

provide: NG\_VALUE\_ACCESSOR,

useExisting: forwardRef(() *=>* FilteredDropdownComponent),

multi: true,

}]

})

export *class* FilteredDropdownComponent implements *OnInit* {

@Input() selected: Object;

@Input() config: Object = {};

@Input() labelProp: *string*;

@Input() getData: (*filter*: *string*) *=>* Observable<*any*>;

@Output() onSelect: EventEmitter<Object> = new EventEmitter();

Propriétés du composant pour communiquer pour communiquer avec le composant appelant, en entrée (@Input) ou en sortie (@Output)

private items: Array<Object> = [];

private loading: *boolean* = false;

private listTemplate: *string* = '';

private firstTouched: *boolean* = false;

private selectedTemplate: *string* = '';

private defaultConfig = {

nothingSelectedLabel: 'Aucun élément sélectionné'

};

*constructor*() { }

ngOnInit() {

this.config = *Object*.assign(this.defaultConfig, this.config);

this.listTemplate = '{{item.' + this.labelProp + '}}';

this.selectedTemplate = '{{selected.' + this.labelProp + '}}';

}

Initatilisation du composant traditionnelle avec les attributs de la classe et le ngOnInit

selectItem(*item*: Object): *void* {

*console*.log('selected item:', item);

this.selected = item;

this.onSelect.emit(item);

this.propagateChange(item);

}

Au clic sur un produit on prévient le composant appelant au changement

refreshItems(*filter*:*any*):*void* {

this.loading = true;

this.getData(filter).subscribe(*items* *=>* {

this.loading = false;

this.items = items.slice(0, 10);

this.addSelectedToList();

this.firstTouched = true;

});

}

Chargement de la liste de recherche (limitée à 10 produits) à partir des données du backend

private addSelectedToList():*void* {

if (this.items[0] && this.items[0]['\_artificial']) {

this.items.shift();

}

if (!this.selected) return;

*var* selectedItemFound = false;

for (*let* i=0 ; i < this.items.length ; i++){

if (this.selected['@id'] === this.items[i]['@id']) {

*console*.log('found:', this.items[i]);

selectedItemFound = true;

}

}

if (!selectedItemFound) {

*var* copiedObject = *Object*.assign({\_artificial: true}, this.selected); // Object.assign copie les propriétés d'un objet à un autre.

// copiedObject['artificial'] = true;

this.items.unshift(copiedObject);

}

}

Méthode pour ajouter le produit (du business) sélectionné à la liste (du shop), s’il n’est pas dans le jeu de données récupérées du backend

// Methods below are here to make formControlName working

private propagateChange = (*\_*: *any*) *=>* { };

public writeValue(*obj*: *any*) {

*console*.log('writeValue', obj);

if (obj) {

this.selected = obj;

}

}

public registerOnChange(*fn*: *any*) {

*console*.log('registerOnChange');

this.addSelectedToList();

this.propagateChange = fn;

}

public registerOnTouched() { }

}

Quand on sélectionne un produit du filtered dropdown, le composant shop products doit être tenu au courant de ce changement. Les méthodes présentées ici dont on parle ont ce rôle.

Template

<button type="button" class="btn btn-danger">

<span \*ngIf="selected">

<ng-container \*compile="selectedTemplate; context: {selected: selected}"></ng-container>

</span>

<span \*ngIf="!selected">

{{ config.nothingSelectedLabel }}

</span>

Affiche « aucune donnée selectionnée » si aucune sélection n’est effectuée

</button>

<button type="button" class="btn btn-danger dropdown-toggle dropdown-toggle-split clickable" (click)="!firstTouched && !loading && refreshItems(filter.value)" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

<span class="sr-only">Toggle Dropdown</span>

</button>

<div class="dropdown-menu">

<div class="input-group">

<input type="text" class="form-control" aria-label="Chercher" #filter (keyup)="refreshItems(filter.value)">

<span class="input-group-addon " (click)="refreshItems(filter.value) ; $event.stopPropagation()" ><i class="fa fa-search" aria-hidden="true"></i></span>

</div>

<div class="dropdown-divider"></div>

<a [ngClass]="{'dropdown-item': true, 'clickable': true, 'selected': (selected && (selected['@id'] === item['@id'])) }" (click)="selectItem(item)" \*ngFor="let item of items">

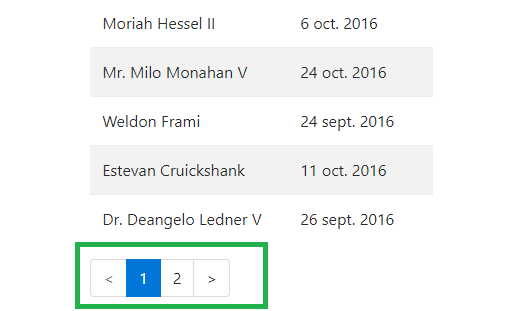
<ng-container \*compile="listTemplate; context: {item: item}"></ng-container>

</a>

<a class="dropdown-item primary-text" \*ngIf="items.length === 0 && loading === false "> aucun élément </a>

<span class ="fa fa-circle-o-notch fa-spin fa-2x fa-fw" \*ngIf="loading === true"></span>

</div>

Sous-composant Pagination

Le sous-composant Pagination est présent sur chacun des écrans qui gèrent l’affichage de tableaux.

Afin de factoriser le code, j’ai créé un composant générique qui sera rappelé dans chacun des écrans, en implémentant les méthodes souhaitées.

import { Component, OnInit, Input, Output } from '@angular/core';

import { EventEmitter } from '@angular/core';

@Component({

selector: 'pagination',

templateUrl: './pagination.component.html',

styleUrls: ['./pagination.component.less']

})

export *class* PaginationComponent implements *OnInit* {

@Input() config: *any*;

@Output() onChange: EventEmitter<*number*> = new EventEmitter();

// private id: number;

private currentPage: *number*;

private pagesCount: *number*;

private pages: Array<*number*>;

ngOnInit() {

this.currentPage = this.config.current;;

//this.currentPage = 2;

this.pagesCount = this.config.last;

this.populatePages();

}

private populatePages(): *void* {

this.pages = [];

for (*let* i = 1 ; i <= this.pagesCount ; i++) {

this.pages.push(i);

}

}

La méthode populatePages() permet de récupérer le nombre de pages

private goto(*pageNumber*: *number*): *void* {

this.currentPage = pageNumber;

this.onChange.emit(pageNumber);

}

}

La méthode goto permet de se déplacer d’une page à une autre dans le tableau

Template du sous-composant

<nav aria-label="Page navigation example">

<ul class='pagination'>

<!-- <span \*ngIf="currentPage === 1"</span> -->

<li [ngClass]="{'page-item': true, 'disabled':(currentPage === 1)}">

<a class="page-link" (click)="goto(currentPage - 1)">

<span class="sr-only">Précédent</span>

&lt;

</a>

</li>

<li [ngClass]="{'page-item': true, 'active':(page === currentPage)}" \*ngFor="let page of pages">

<a class="page-link" (click)="goto(page)">

{{page}}

<span class="sr-only" \*ngIf="page === currentPage"> (page courante)</span>

</a>

</li>

<li [ngClass]="{'page-item': true, 'disabled':(currentPage === pagesCount)}">

<a class="page-link" (click)="goto(currentPage + 1)">

<span class="sr-only">Suivant</span>

&gt;

</a>

</li>

</ul>

</nav>

Utilisation du composant pagination dans le template du composant (ici customer.component)

<div class="row">

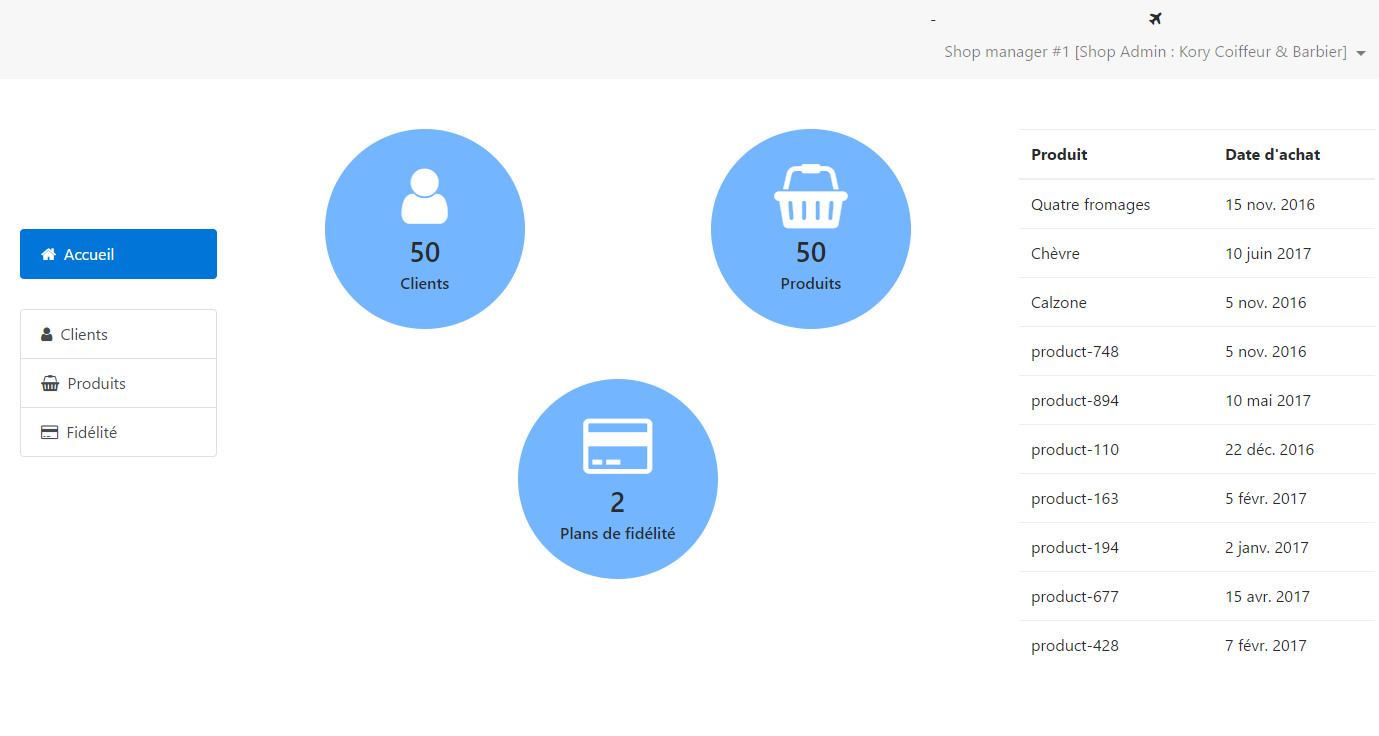
<pagination [config]="customers.pagination" (onChange)="refresh($event)" \*ngIf="customers"></pagination>

</div>

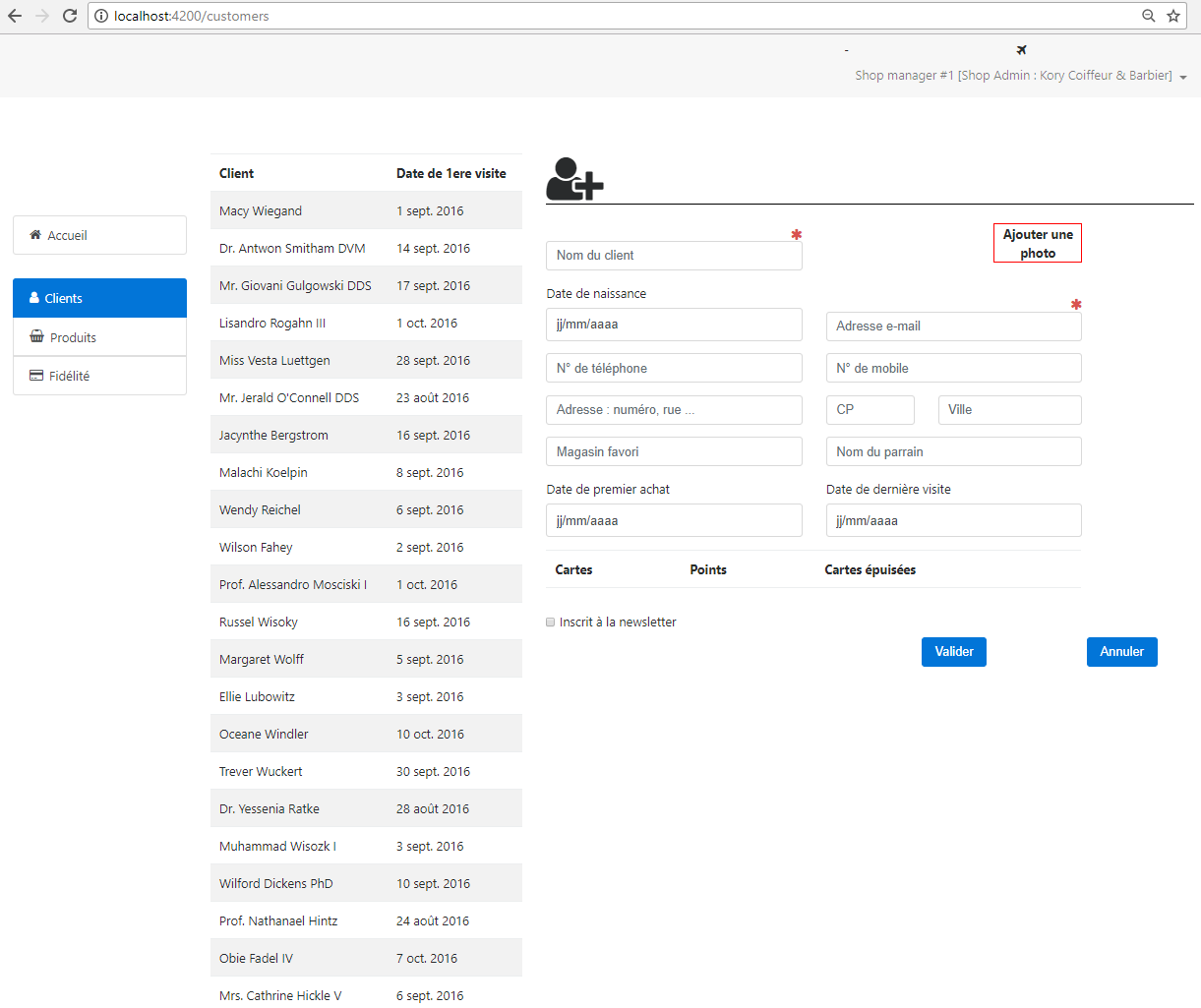
Rendu du site



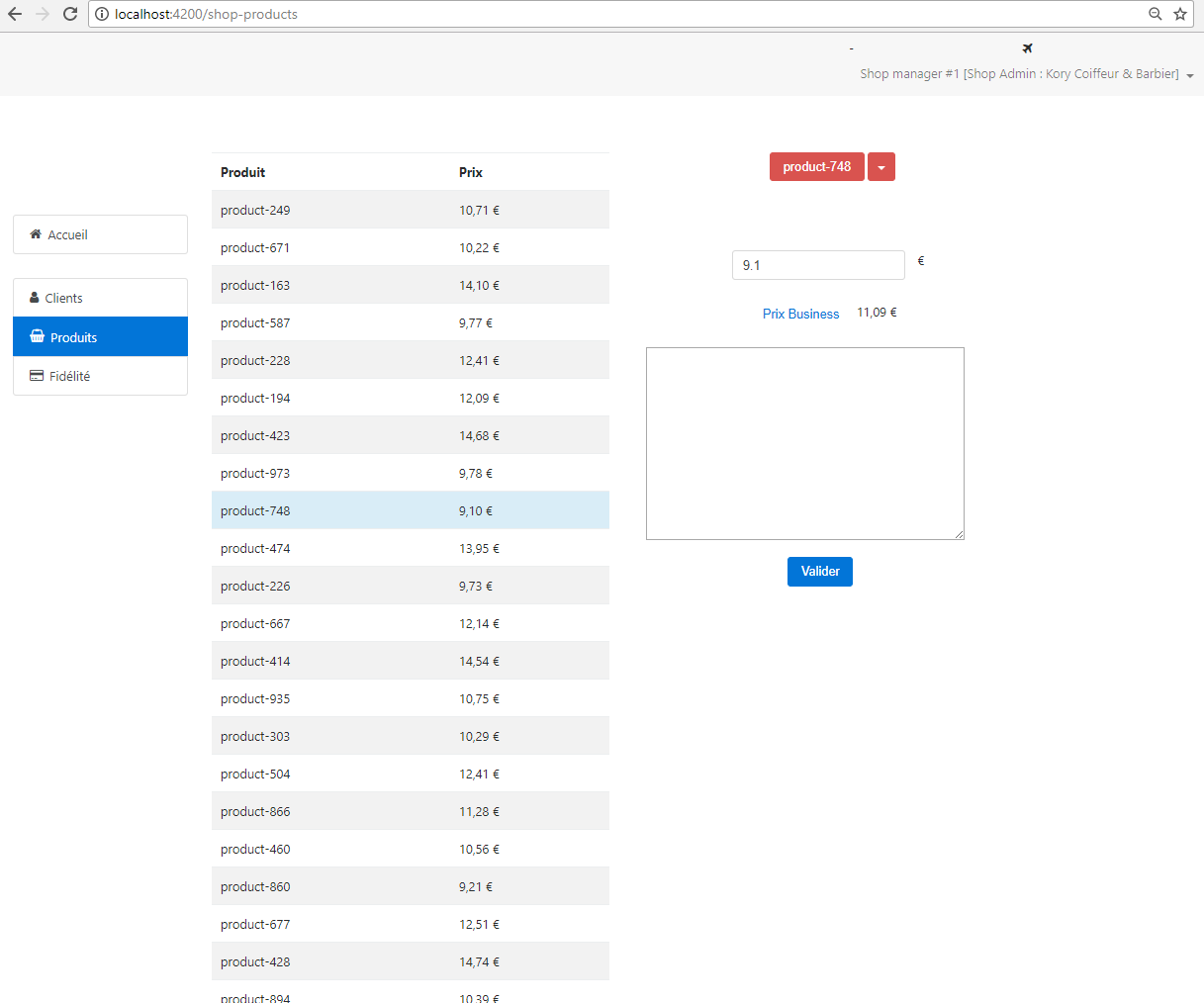
Page de login



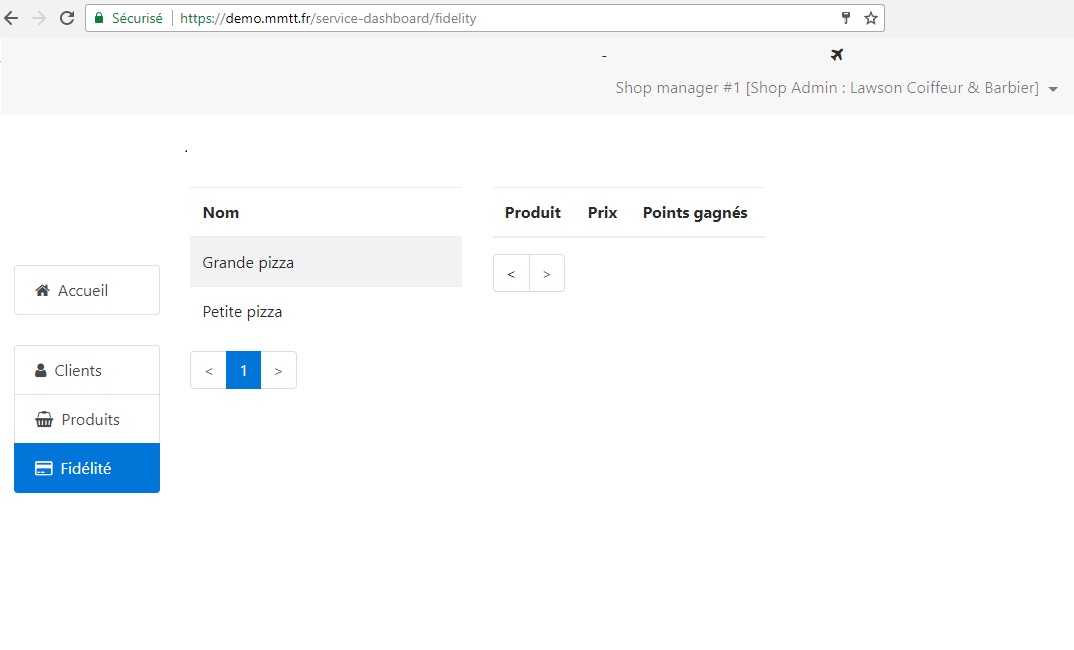
Page d’accueil shop admin



Page de gestion des clients shop admin



Page de gestion des produits shop admin



Page de gestion des plans de fidélité shop admin

Conclusion

Ce stage aura été déterminant pour moi tant dans l’approche du métier de développeur que dans l’apprentissage de nouvelles technologies.

En effet j’ai pu mesurer les problématiques d’organisation de travail en situation réelle, apprécier le travail d’équipe réellement collaboratif et perçu la nécessité de recherche, de veille et du goût pour l’apprentissage permanent.

Mon intégration au projet a été assez rapide grâce aux premières tâches de tests qui m’ont permis de me familiariser avec le backend. J’étais sensibilisé à la plupart des outils de base de l’environnement de travail.

Pour la partie gestion de projet que j’ai reprise, la lecture a été plutôt rapide (modèle de données, cahier des charges).

J’ai rencontré davantage de difficultés à l’apprentissage d’Angular4 et j’ai pu réellement apprécier le support de David Duquenoy. Le nombre important de composants et des fonctionnalités a ajouté de la complexité dans mon travail de programmation ; si la partie de conceptualisation m’a paru accessible, les subtilités et fonctions propres à Angular4 ont nécessités beaucoup de travail.

Néanmoins, même si le projet n’a pas été terminé en temps, j’ai beaucoup appris dans la conception, les principes, l’architecture et les fonctions propres à Angular ; c’est le premier projet à moyen terme sur lequel je suis amené à travaillé, et les automatismes sont venus au fur et à mesure.