Rapport Projet Java EE Médiathèques

2017/2018

- Ghiles FEGHOUL
- Juba TIDAF
- Islam GUETTOUCHE

Introduction :

Le but de ce projet est de réaliser une application web visant à aider à la mise en place d'un service Médiathèque / ludothèque, permettant à la fois au personnel de gérer les stocks, mais aussi aux utilisateurs d'accéder a des services mis en place par la médiathèque.

Objectifs :

- La médiathèque doit offrir une plateforme utilisateur avec un ensemble de services.
- La médiathèque doit pouvoir gérer des ressources des types suivants :
  - DVD.
  - CDs audio.
  - Jeux vidéo.
  - Jeux de sociétés.
  - Livres.

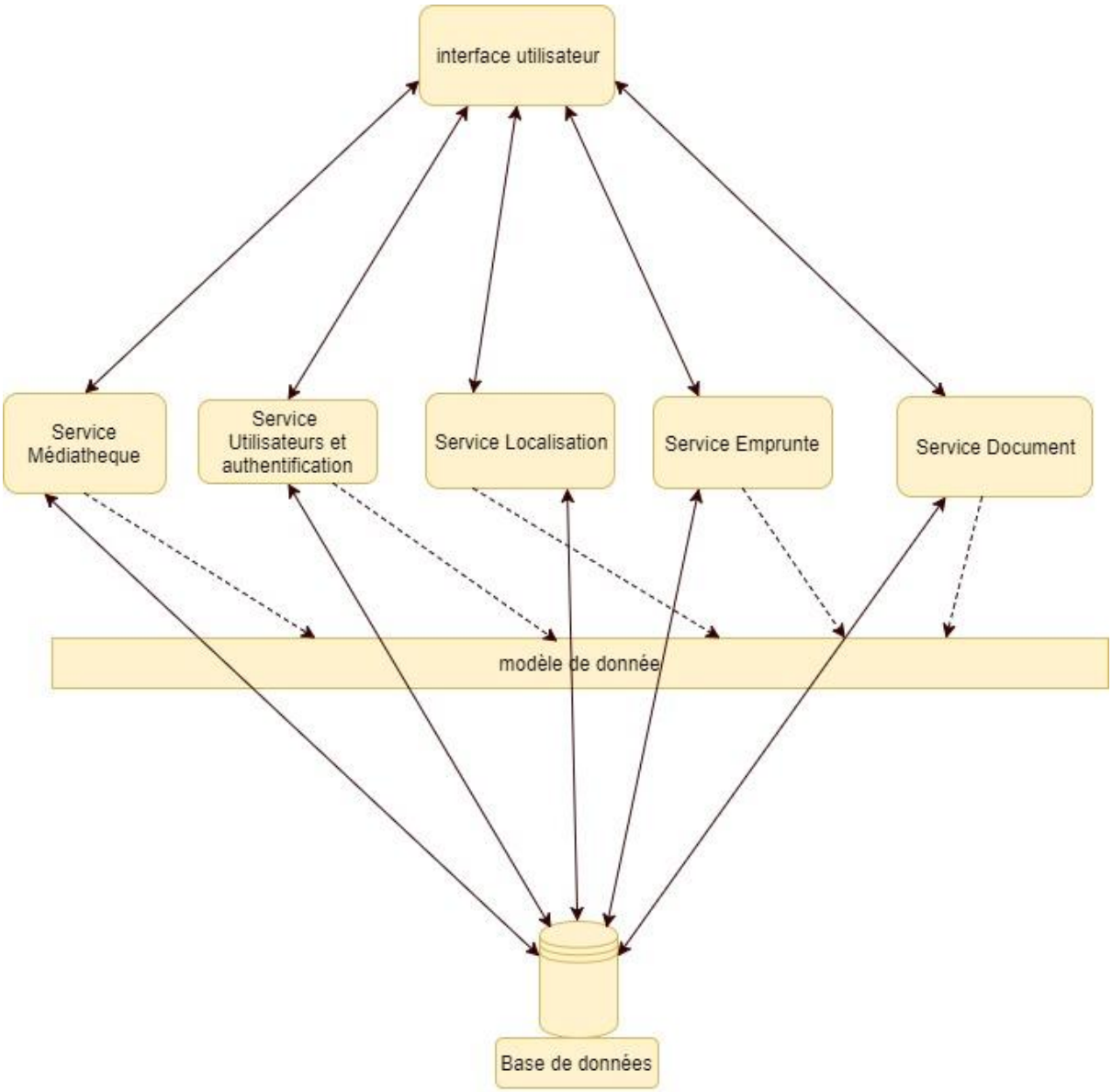
Structure du projet :

Initialement, notre but était de concevoir une application autour d'une architecture évolutive, extensible et maintenable dans le temps, et pour ce faire, nous avons opté pour le développement à l'aide de l'architecture de **Spring Boot** qui nous a été imposée, nous l'avons mis en œuvre avec le minimum de dépendances possibles entre les parties prenantes du projet (les différentes entités).

Pour la partie sécurité, **Spring** nous fournit une couche **Security** que nous avons utilisé pour gérer l'authentification et les différents rôles des utilisateurs du système, ceci étant donc dans l'objectif de sécuriser notre application. On a donc utilisé les **JWT** (JSON WEB TOKEN) pour générer des tokens temporaires qu'on attribue à l'utilisateur et ainsi assurer l'authentification.

Pour la partie données, on a utilisé la couche **Spring Data** pour persister nos données et ainsi pouvoir étendre nos interfaces (DAO) à partir de la classe JpaRepository et qui nous permet d'avoir déjà un panel de méthodes pour manipuler nos données aisément.

L'image suivante nous montre l'ensemble des services en communication avec le modèle et la base de données.



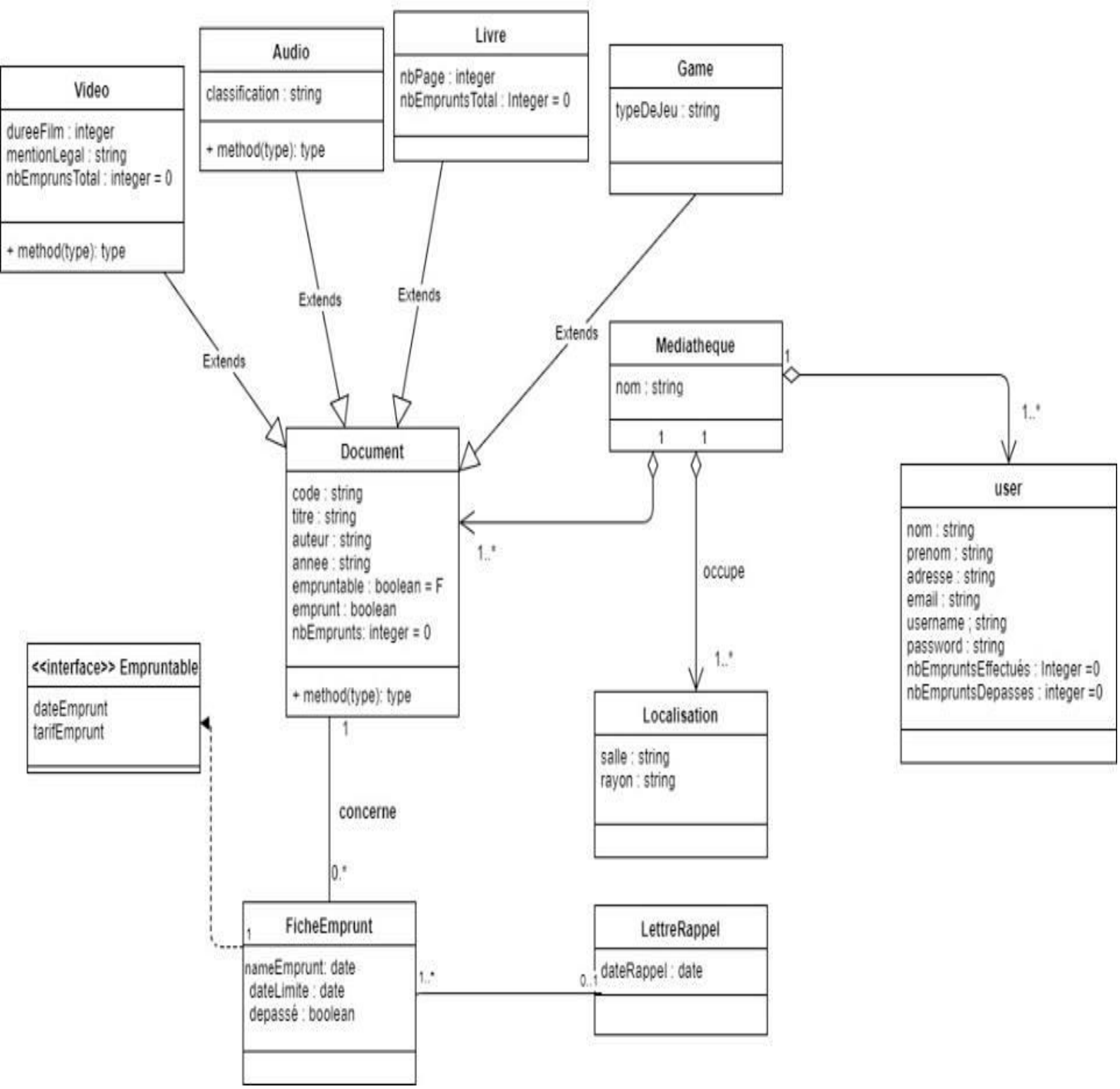
<u>Figure 1 :</u> Architecture de notre application.

Les interfaces des services :

Méthode		Description
Service Authentification		
❖ **GET**	**/api/refresh**	✓ Effectuée à chaque actualisation de la page. ✓ Elle nous retourne un nouveau token si l'ancien n'est plus valide sinon on conserve le token courant (paramétré à 10 min dans notre application.yml du serveur).
❖ **PUT**	**/api/changePassword**	✓ Permet de changer le mot de passe d'un compte utilisateur
Service Document		
❖ **GET**	**/api/document/all**	✓ Retourne tous les documents.
❖ **POST**	**/api/document**	✓ Permet d'ajouter un document (Cette méthode est propre à l'administrateur).
❖ **GET**	**/api/document/{id}**	✓ Permet de rechercher un document à partir de son identifiant.
❖ **DELETE**	**/api/document/{id}**	✓ Supprimer un document par son identifiant, gérée uniquement par l'administrateur
❖ **PUT**	**/api/document/{id}**	✓ Modifier un document à partir de son identifiant, (Cette méthode est propre à l'administrateur).
Service Emprunte		
❖ **GET**	**/api/emprunt/{idUser}/{idDoc}/{idMedia}**	✓ Retourne une fiche d'emprunt qu'un utilisateur (inscrit) doit remplir pour emprunter un document dans une médiathèque spécifique.
❖ **GET**	**/api/emprunt/restitution/{id}**	✓ Permet d'enregistrer la bonne restitution d'un document par un utilisateur, (Cette méthode est propre à l'administrateur).

❖ **GET**	**/api/emprunt/rappel/{id}**	✓ Permet à l'administrateur d'envoyer des rappels aux utilisateurs si la date limite d'emprunt d'un document est expirée.
Service Médiathèque		
❖ **POST**	**/api/media/{id}**	✓ Permet d'ajouter une médiathèque, (Cette méthode est propre à l'administrateur).
❖ **PUT**	**/api/media/{id}**	✓ Modifier une médiathèque par son identifiant, (Cette méthode est propre à l'administrateur).
❖ **DELETE**	**/api/media/{id}**	✓ Supprimer une médiathèque à partir de son identifiant, (Cette méthode est propre à l'administrateur).
Service Localisation		
❖ **GET**	**/api/localization**	✓ Retourne tous les localisations, dans la médiathèque (utile pour le client).
❖ **POST**	**/api/localization/**	✓ Permet d'ajouter une localisation étage, rayon,…etc. (Cette méthode est propre à l'administrateur).
❖ **DELETE**	**/api/localization/{id}**	✓ Supprimer une localisation à partir de son identifiant, (Cette méthode est propre à l'administrateur).
❖ **PUT**	**/api/localization/{id}**	✓ Modifier une localisation à partir de son identifiant, (Cette méthode est propre à l'administrateur).

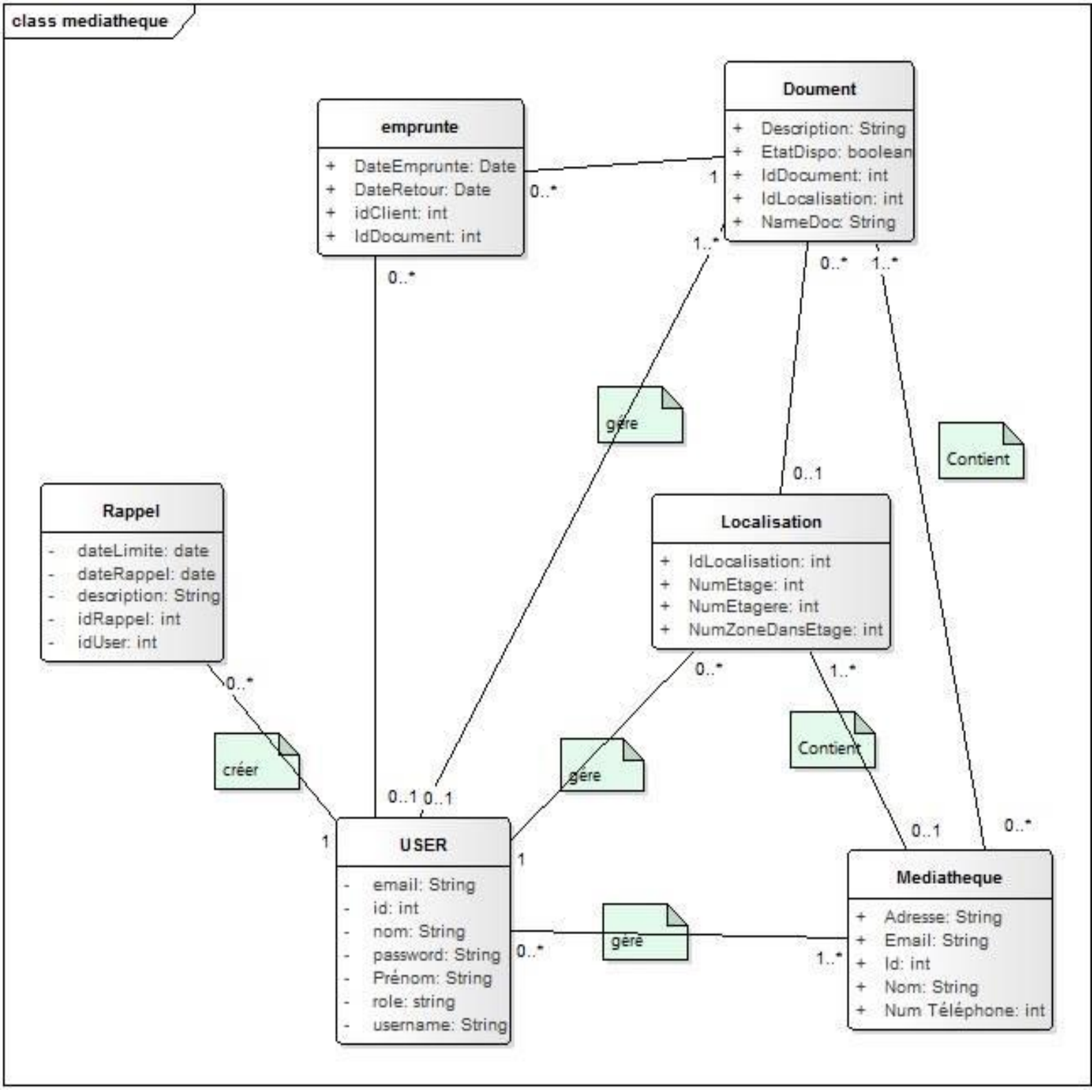
Diagramme UML :



<u>Figure 2 :</u> Diagramme de classe de notre modèle.

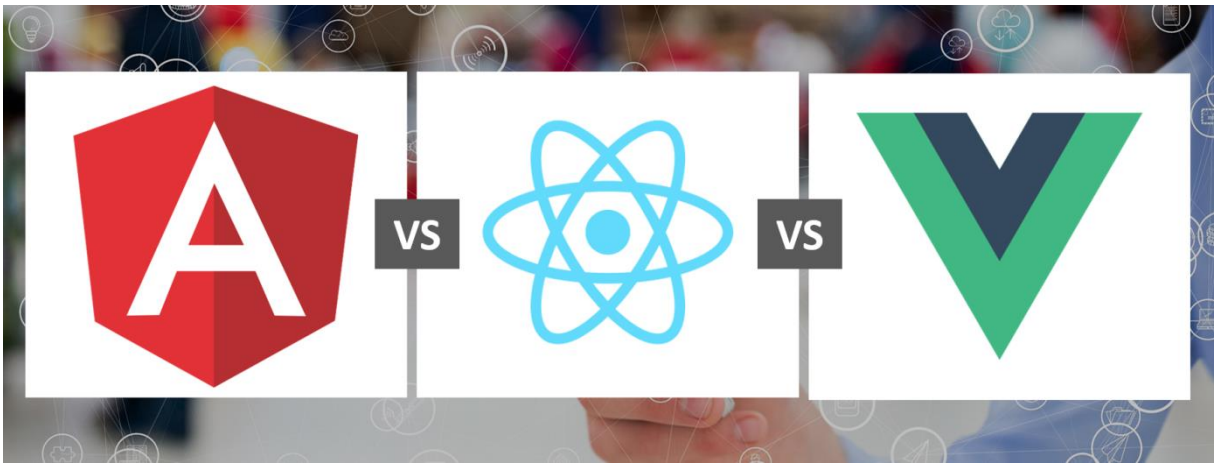
Schéma relationnel :

- **Audios (`tarif`, `id`)**
- **authority (`id`, `name`)**
- **documents (`document_type`, `id`, `author`, `code`, `empruntable`, `emprunte`, `gender`, `nb_emprunt`, `title`, `year`, `tarif`, `duree`, `nb_page`, `duree_film`, `localization_id`*, `mediatheque_id`*)**
- **emprunts (`id`, `date_emprunt`, `date_limite`, `depasse`, `client_id`*, `document_id`*, `mediatheque_id`*, `rappel_id`*)**
- **games (`tarif`, `id`)**
- **livres (`duree`, `nb_page`, `tarif`, `id`)**
- **localizations (`id`, `rayon`, `salle`)**
- **mediatheques (`id`, `name`)**
- **rappels (`id`, `corps`, `date_rappel`, `entete`, `fin`, `nom_media`, `en_retard_id`*)**
- **user (`id`, `address`, `current_loans`, `email`*, `firstname`, `inscription_date`, `lastname`, `loans`, `max_loans`, `outdated_loans`, `password`, `username`)**
- **user_authority (`user_id`*, `authority_id`*)**
- **videos (`duree_film`, `id`)**



<u>Figure 3 :</u> La représentation de notre base de données relationnelle avec MERISE.

La justification du choix des technologies pour la partie interface utilisateur (UI/UX) :



<u>Figure 4 :</u> Angulaire vs. React contre Vue: Comparaison 2018.

Vue du développeur

Vue, étant le cheval noir de **JavaScript Frameworks** a été développé par une équipe de douzaines de développeurs tandis qu'**Angular** et **React** ont été populairement utilisés par des géants comme **Facebook, Reddit, Airbnb, Netflix** et **Google**.

Codage et performance

Name	angular-v4.1.2-keyed	react-v15.5.4-redux-v3.6.0	vue-v2.3.3-keyed
create rows Duration for creating 1000 rows after the page loaded.	193.1 ± 7.9 (1.2)	212.2 ± 14.2 (1.3)	166.7 ± 8.6 (1.0)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	197.4 ± 5.3 (1.2)	206.7 ± 7.3 (1.2)	168.5 ± 5.0 (1.0)
partial update Time to update the text of every 10th row (with 5 warmup iterations).	13.0 ± 4.5 (1.0)	18.0 ± 1.6 (1.1)	17.3 ± 2.9 (1.1)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	3.4 ± 2.3 (1.0)	8.7 ± 2.9 (1.0)	9.3 ± 1.7 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	13.4 ± 1.0 (1.0)	17.1 ± 1.3 (1.1)	18.3 ± 1.5 (1.1)
remove row Duration to remove a row. (with 5 warmup iterations).	46.1 ± 3.2 (1.0)	52.4 ± 1.7 (1.1)	52.6 ± 2.7 (1.1)
create many rows Duration to create 10,000 rows	1946.0 ± 41.8 (1.2)	1931.7 ± 35.6 (1.2)	1587.5 ± 33.9 (1.0)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	324.6 ± 10.1 (1.0)	366.4 ± 10.9 (1.1)	399.5 ± 11.0 (1.2)
clear rows Duration to clear the table filled with 10.000 rows.	379.9 ± 11.3 (1.5)	410.9 ± 9.9 (1.6)	254.5 ± 5.0 (1.0)
startup time Time for loading, parsing and starting up	84.3 ± 2.6 (1.5)	93.8 ± 6.9 (1.7)	56.6 ± 2.5 (1.0)
slowdown geometric mean	**1.14**	**1.23**	**1.06**

Figure 5 : Comparaison des performances Angulaire vs. React contre Vue.

Compatibilité descendante

En ce qui concerne **l'Agilité**, **React** gagne la course avec des mises à niveau flexibles et fournissant aux développeurs un écosystème abondant de moteurs de rendu. **Angular JS** est un framework complet qui repose sur des mises à jour de versions et de composants antérieurs, tandis que **React** offre une excellente rétrocompatibilité, permet d'associer ses bibliothèques à d'autres packages et promet même des migrations héritées. **Vue** gagne en matière de mobilité et de modularité. En ce qui concerne **Angular**, les versions de support à long terme sont uniquement disponibles chez **Angular 4**.

Courbe d'apprentissage

Pour les développeurs juniors et la collaboration entre les membres de l'équipe de projets d'entreprise, **Vue** a été classé le plus simple et le plus facile à comprendre pour les programmeurs en herbe. La courbe d'apprentissage est raide pour **Angular** tout en réagissant, et **Vue** fournit des temps de développement plus rapides et des solutions de débogage plus rapides. Pour les développeurs JavaScript inexpérimentés qui ont déjà travaillé avec jQuery auparavant, l'utilisation de **Vue** est une bouée de sauvetage car elle ressemble à du JavaScript simple avec l'introduction d'idées existantes.

Le verdict

- ✓ Pour ceux qui aiment les écosystèmes massifs et plus de flexibilité, **React** est la voie à suivre.
- ✓ **Angular** utilise TypeScript, il est idéal pour les programmeurs ayant un environnement OOP (Object-Oriented Programming) solide qui a besoin de conseils et d'une structure détaillés.
- ✓ **Vue** est relativement simple à intégrer et à intégrer pour une petite équipe de développeurs principaux.

Le choix du **Framework** s'est fait en fonction de l'affinité envers la solution proposée. Dans notre cas on a un projet plus complexe avec plusieurs services à mettre en œuvre, qui implique une certaine complexité dans les données à gérer, et afin de répondre à toutes ces attentes on a opté pour le Framework **Angular**.

Ce dernier est fondé sur l'idée que la programmation déclarative doit être utilisée pour construire les interfaces utilisateurs et les composants logiciels de câblage, tandis que la programmation impérative excelle pour exprimer la logique métier.

La conception **d'Angular** est guidée par plusieurs objectifs :

- Découpler les manipulations du DOM de la logique métier. Cela améliore la testabilité du code.
- Considérer le test d'une application aussi important que l'écriture de l'application elle-même. La difficulté de la phase de test est considérablement impactée par la façon dont le code est structuré.

- Découpler les côtés client et serveur d'une application. Cela permet au développement logiciel des côtés client et serveur de progresser en parallèle, et permet la réutilisabilité de chacun des côtés.
- Guider les développeurs pendant toute la durée de la construction d'une application : de la conception de l'interface utilisateur, en passant par l'écriture de la logique métier, jusqu'au test de l'application.
- Rendre les tâches faciles évidentes et les tâches difficiles possibles.

On a choisi ce Framework pour sa popularité et la présence des formations spécifique pour le mettre en œuvre.

La multitude de ce Framework permet de combler les besoins suivant :

✓ **Une navigation plus fluide pour le visiteur :**

Développement selon la structure **MVVM** (Modèle-Vue-Vue-Modèle). Ce principe offre :

- Un avantage de taille.
- Une réduction considérable de la vitesse de chargement des pages.

En effet, le nombre d'accès au serveur est fortement diminué car la communication se fait majoritairement en mode asynchrone. Autrement dit, l'interface visuelle est portée côté client. En conséquence, une importante partie des requêtes supportées en arrière-plan est ainsi supprimée, ce qui permet de concevoir des applications web plus légères. Ceci explique sa parfaite adaptation pour les applications web mono-page (SPA) qui ne comportent qu'une seule et unique interface ainsi que pour les applications destinées aux dispositifs mobiles.

✓ **Une meilleure gestion de contenu dynamique :**

Dès qu'une vue est modifiée, la donnée est envoyée au model associé qui rafraîchit à son tour la vue. Concrètement, si un utilisateur remplit un champ texte, la valeur saisie peut s'afficher à un autre endroit de la page et ce sans rechargement ni soumission au préalable de l'information. Il s'agit donc d'une synchronisation entre le modèle et la vue qui permet de créer des applications plus responsives.

Le détail implémentation de l'authentification:

Dans cette partie on a utilisé **Spring Security** pour toutes les parties de la sécurité, il nous a fourni une couche de sécurité dans le framework **Spring Boot**, qui nous a permet de vérifier les authentifications des clients et de l'administrateur, à l'aide de la définition des rôles ('ADMIN', 'USER'). Pour la gestion des sessions on a opté pour l'utilisation des (jwt) **JSON Web Token** qui est un standard ouvert pour échanger de l'information de manière sécurisée via un jeton signé. Notre serveur envoie un jeton possédant l'affirmation "utilisateur identifié en tant qu'administrateur" et le fournir au client. Le client pourrait alors vérifier le jeton pour prouver que l'utilisateur est identifié en tant qu'administrateur.

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

- Un header.
- Un payload (les "claims").
- La signature.

Le **header** et le **payload** sont structurés en **JSON**. Ces trois parties sont chacunes **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 (souvent JWT, mais le champ a été prévu dans le cas où l'application traite d'autres types d'objet qu'un JWT).

Exemple de header :

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Ici, le header indique que la signature a été générée en utilisant **HMAC-SHA256**.

Le **payload** est la partie du token qui contient les informations que l'on souhaite transmettre. Ces informations sont appelées "**claims**". Il est possible d'ajouter au token les claims que l'on souhaite, mais un certain nombre de claims sont déjà prévus dans les spécifications de JWT.

Par exemple, 'sub' qui identifie le sujet du token (qui est le token identifie), issue va permettre d'identifier l'émetteur du token ou encore 'exp' qui indique la date d'expiration du token. Il est fortement conseillé d'assigner une valeur à ce dernier champ afin de limiter la durée de vie du token. Si la date d'expiration est dépassée, le token sera rejeté.

Exemple de payload :

```
{
   "sub":"JubaTIDAF",
   "exp": "1485968105",
   "admin": “true”
}
```

S’authentifier avec JWT et Spring Boot :

La manière la plus courante d’utiliser un JWT est de s’en servir pour s’authentifier. Un token est envoyé avec chaque requête que le client fera auprès de l’application, qui autorisera, ou non, le client à accéder à ses services, suivant la validité du token. Ce type d’authentification, dit **stateless**, ne stocke pas les sessions utilisateurs dans le contexte de l’application.

Dans notre cas pour l’utilisation de notre token, il faut tout d’abord le créer. Pour cela, il est nécessaire de s’authentifier avec son login et son mot de passe auprès de l’application afin que celle-ci nous renvoie le token. Une fois le token obtenu, on peut faire appel à nos URL sécurisées en envoyant le token avec notre requête. La méthode la plus courante pour envoyer le token est de l’envoyer à travers l’en-tête HTTP Authorization en tant que Bearer token :

```
Authorization: Bearer 'token'
```

Diagramme de séquence détaillé cas “authentification” :

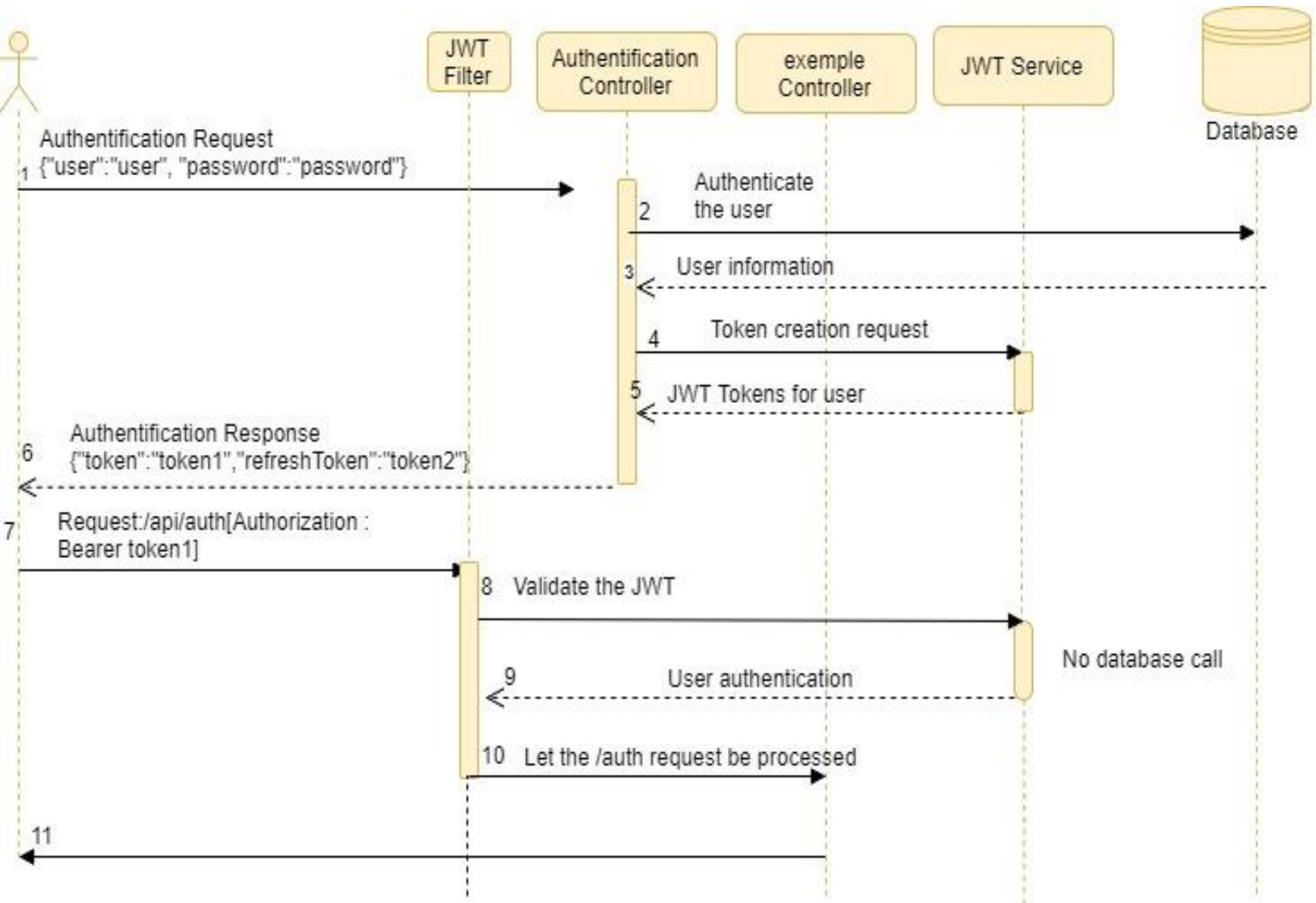


Figure 6 : Diagramme de séquence détaillé pour l’authentification.

Maquette IHM

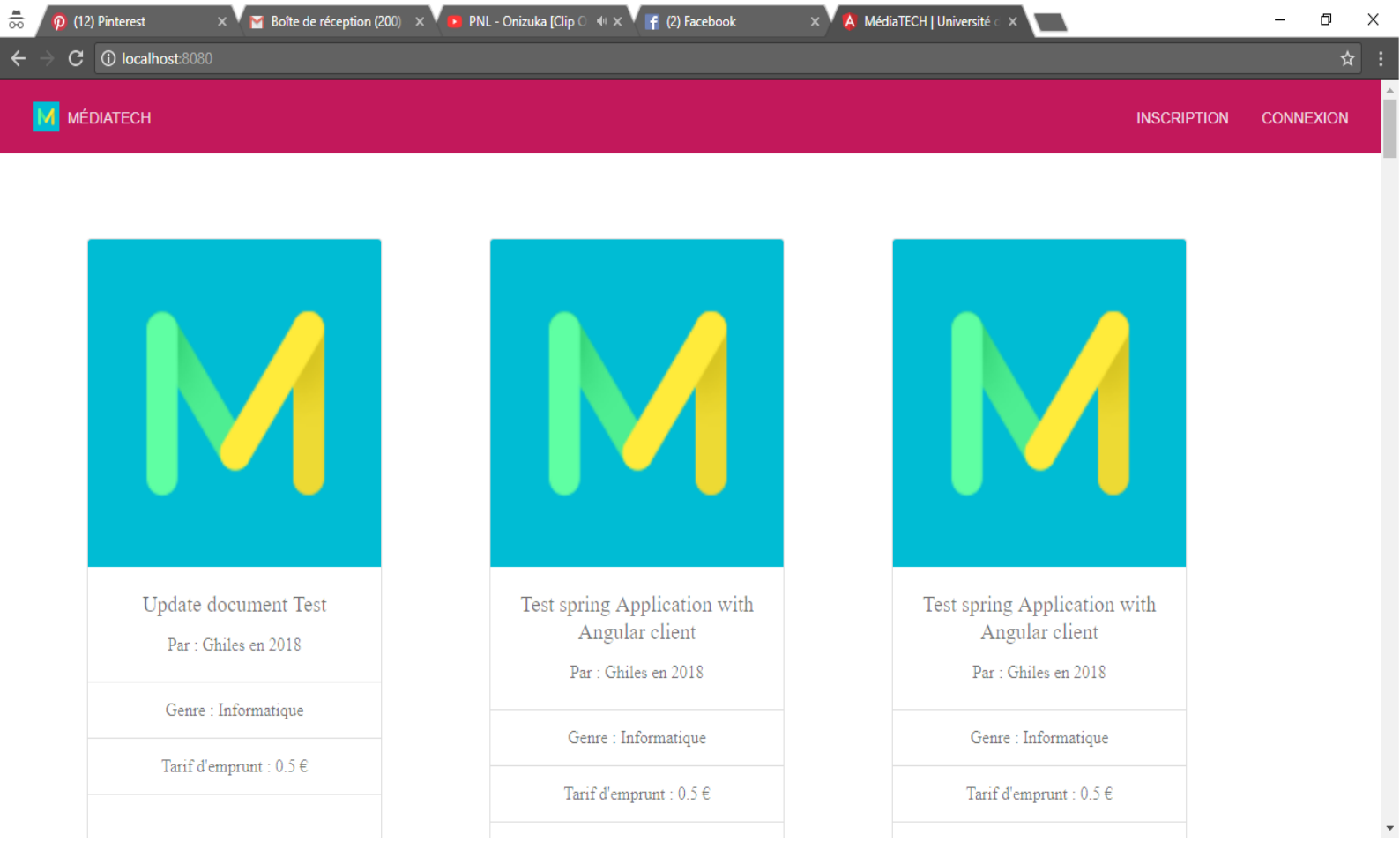


Figure 7 : Accueil.

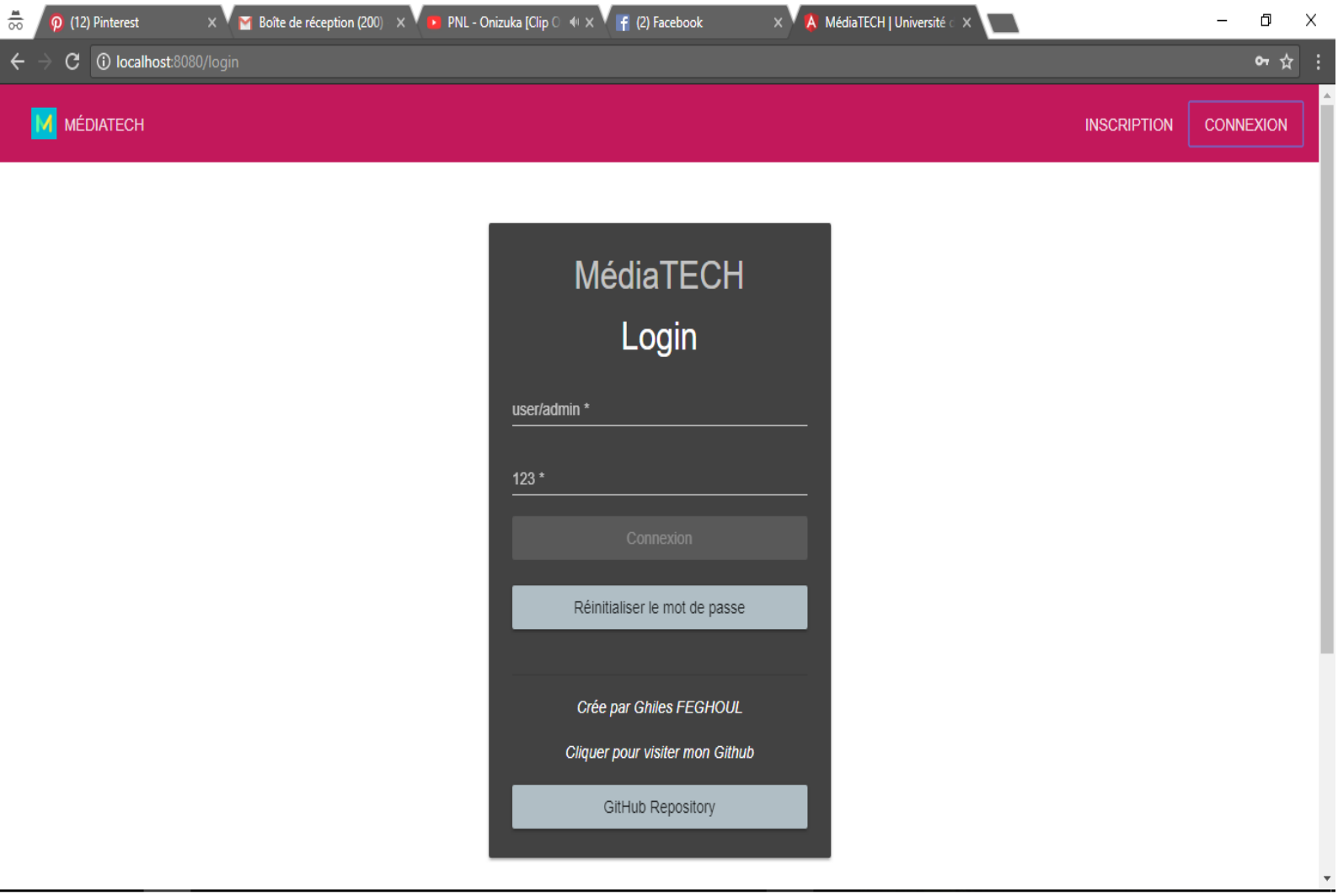


Figure 8 : Login.

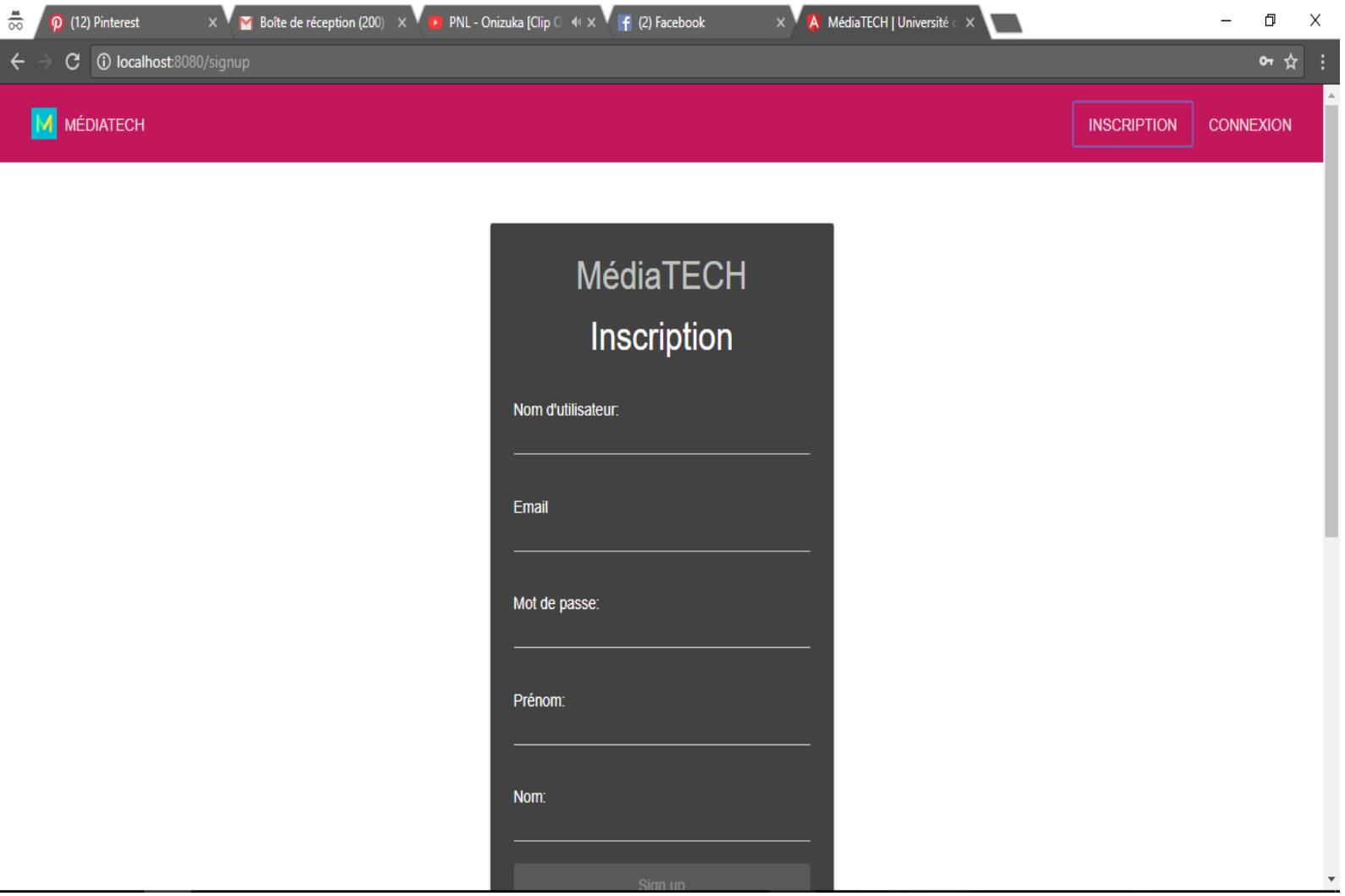


Figure 9 : Inscription.

Manuel partie Administration

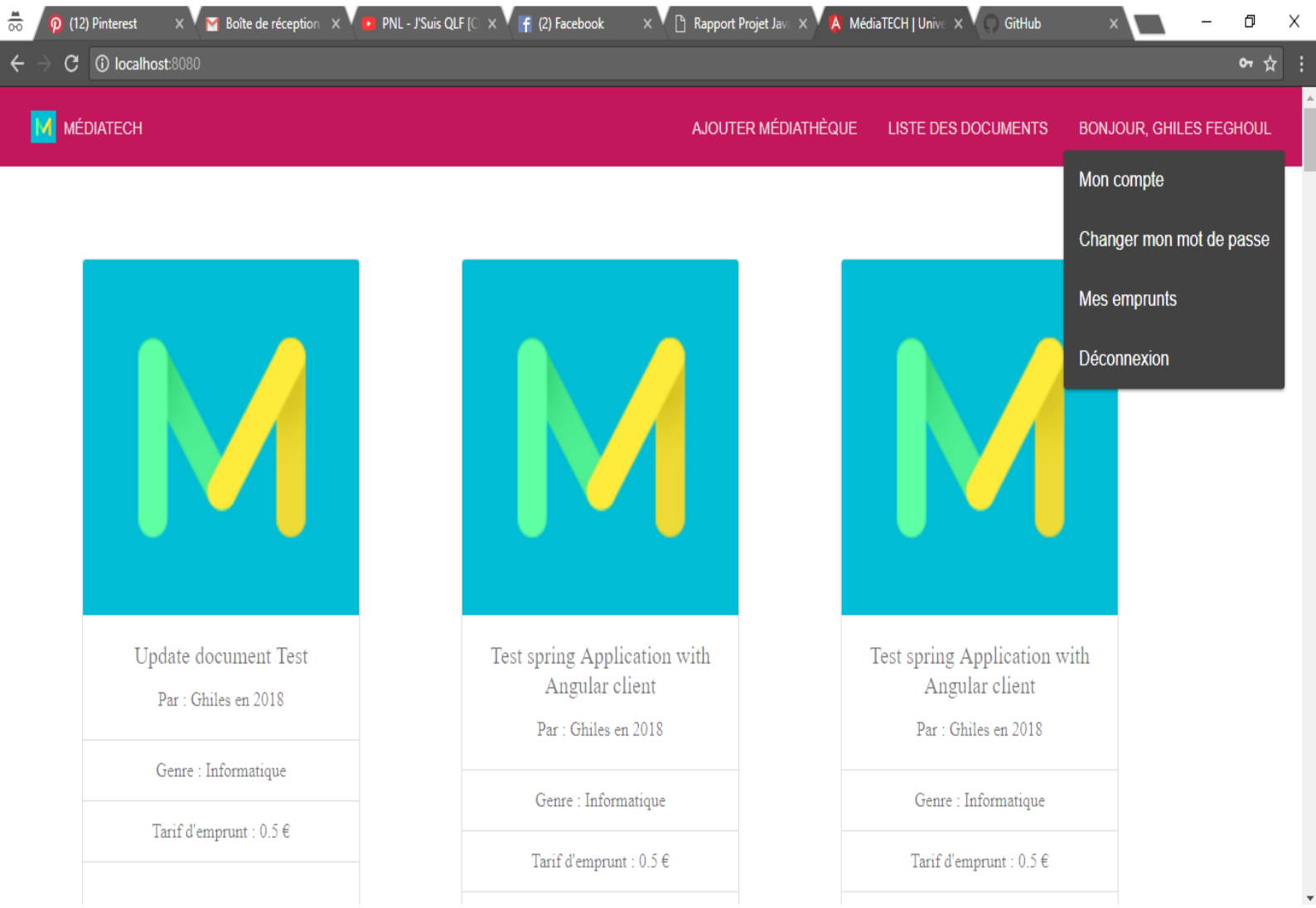
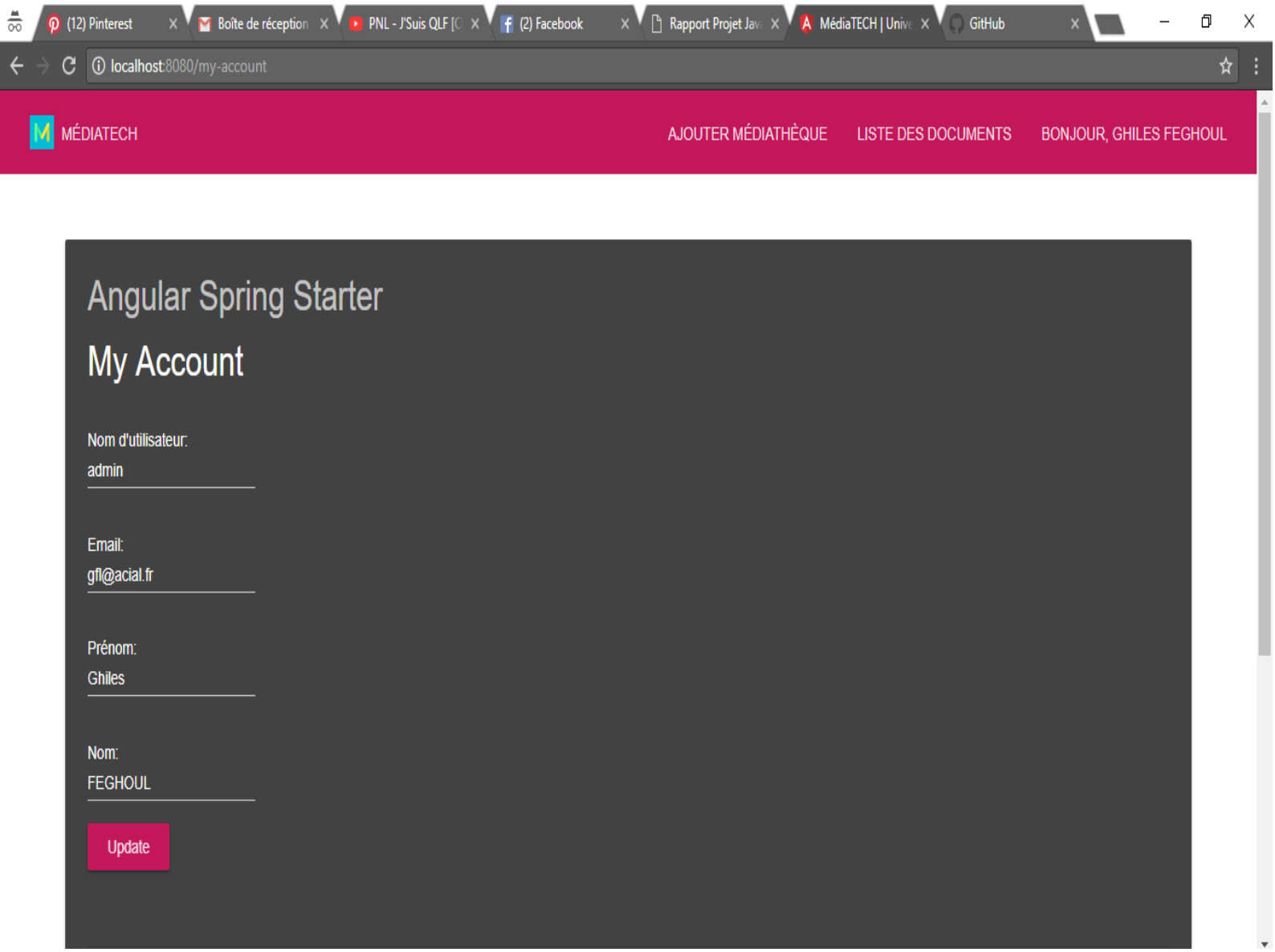
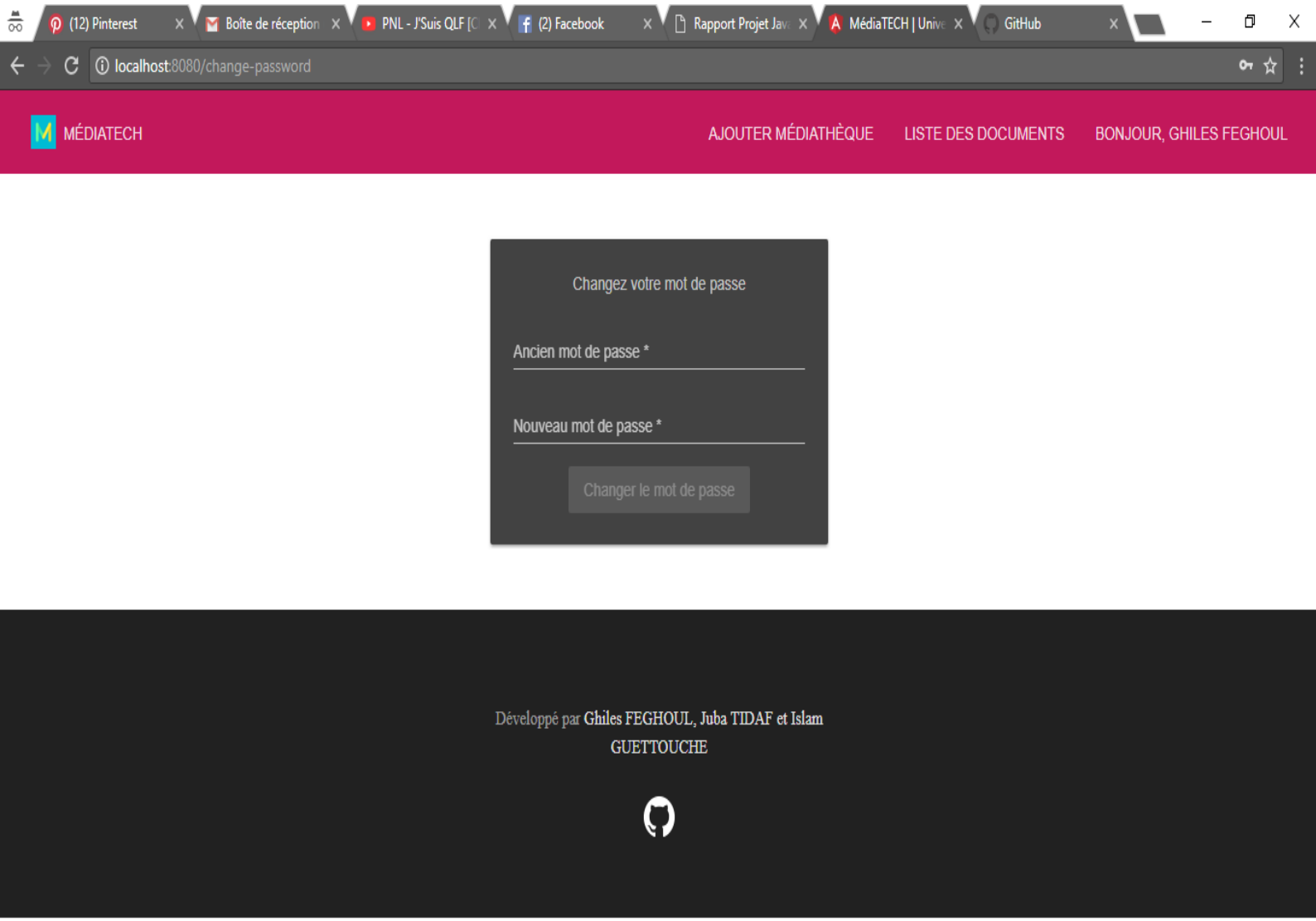


Figure 10 : Accueil coté Administrateur.



<u>Figure 11 :</u> Paramètres du profil coté Administrateur.



<u>Figure 12 :</u> La modification d'un mot de passe coté Administrateur.

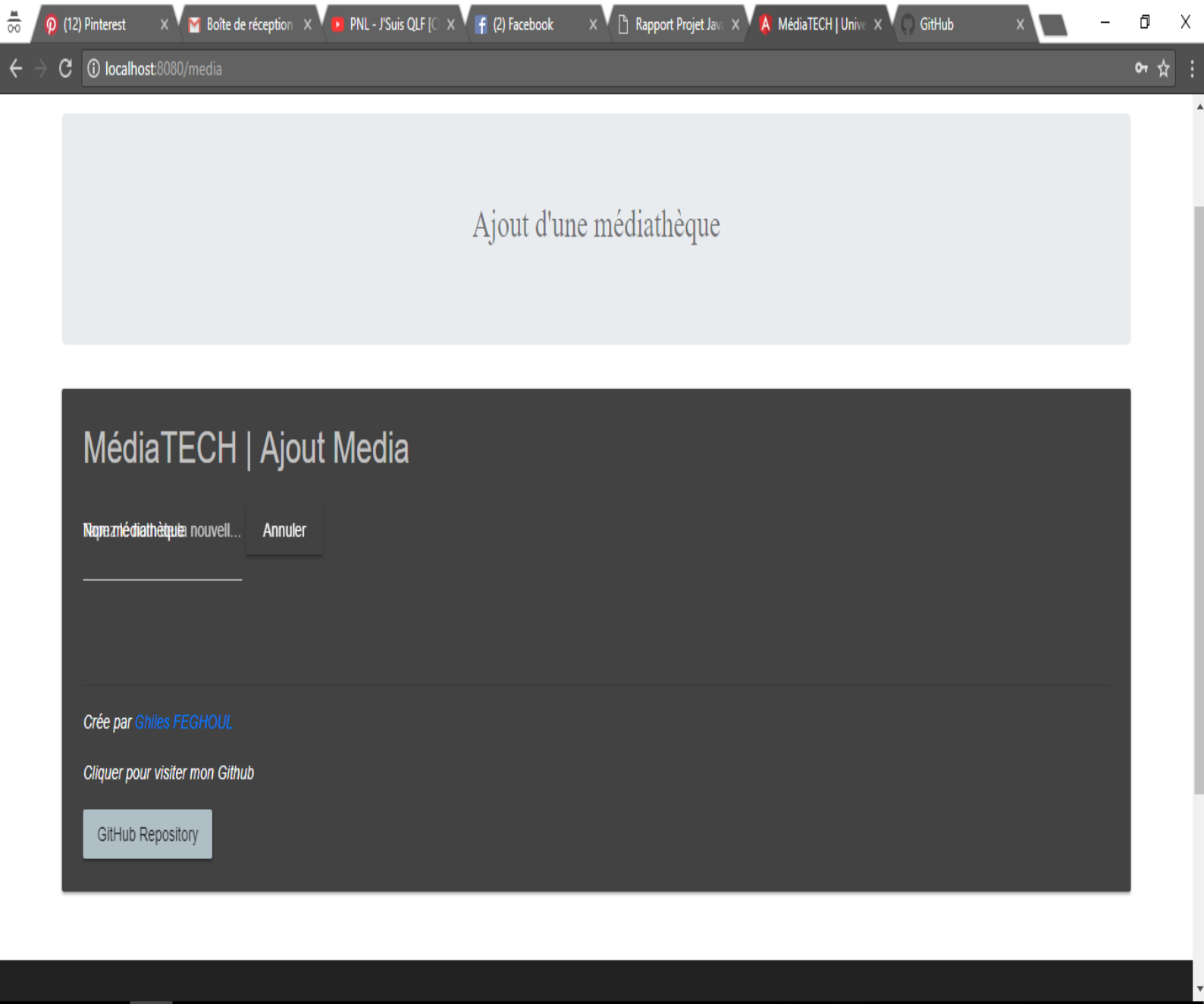
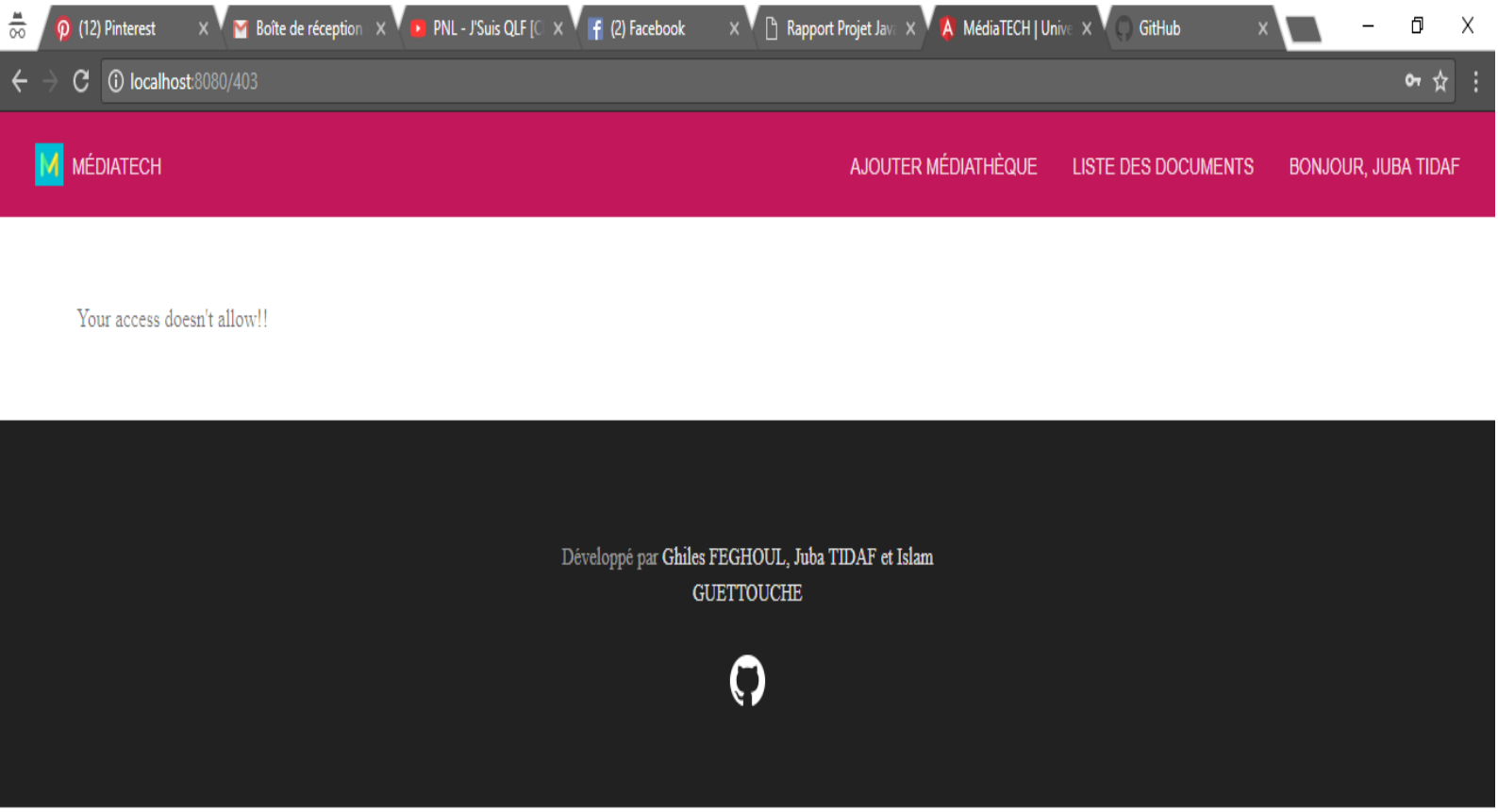


Figure 13 : L'Ajout d'une médiathèque.

Manuel partie Utilisateur



<u>Figure 14 :</u> Accès refusé apres une tentative d'ajout d'une médiathèque (coté Utilisateur).

Une notice d'exécution / installation :

Les installations requises :

- **Maven et Java 1.7 au plus.**
- **NPM 3.1 et NODE 6.9.5.**
- **angular-cli@1.0.0.**

Pour l'exécution il suffit de suivre les étapes suivant :

```
# clone our repo
# --depth 1 removes all but one .git commit history
git clone --depth 1 https://github.com/ghilesfeghoul/Springular.git
# change directory to the repo's frontend folder
cd angular-spring-starter/frontend
# install the frontend dependencies with npm
# npm install @angular/cli@1.0.0 -g
npm install
# start the frontend app
npm start
# change directory to the repo's backend folder
cd ../server
# install the server dependencies with mvn
mvn install
# start the backend server
mvn spring-boot:run
# the fronend angular app will be running on port 4200
# the spring-boot server will be running on port 8080
```

Bonus : SOAP VS REST

Simple Object Access Protocol (SOAP) et Representational State Transfer (REST) sont deux solutions pour un même problème: comment accéder à des services Web. Le choix d'abord peut sembler facile, mais parfois il peut être étonnamment difficile.

SOAP repose exclusivement sur XML pour fournir des services de messagerie. Microsoft a initialement développé SOAP pour remplacer des technologies plus anciennes qui ne fonctionnent pas parfaitement sur le web tels que le Distributed Component Object Model (DCOM) et le Common Object Request Broker Architecture (CORBA). Ces technologies échouent parce qu'elles comptent sur la messagerie binaire; la messagerie XML employée par SOAP fonctionne mieux sur le web.

REST offre une alternative plus simple. Au lieu d'utiliser XML pour faire une demande, REST repose sur une simple URL. Dans certaines situations, vous devez fournir des informations supplémentaires de façon particulière, mais la plupart des services Web qui utilisent REST compter exclusivement sur l'obtention de l'information nécessaire en utilisant l'approche de l'URL. REST peut utiliser quatre différents méthodes HTTP(GET, POST, PUT et DELETE) pour effectuer des tâches.

SOAP est certainement un choix de taille pour l'accès au service Web. Il offre les avantages suivants par rapport à REST:

- Non dépendance par rapport à la langue, la plate-forme et le transfert (REST nécessite l'utilisation de HTTP)
- Fonctionne bien dans des environnements distribués (REST nécessite une communication directe point à point)
- Standardisé
- Fournit une extensibilité de pré-compilation significative sous la forme de normes WS standards
- Intégre la gestion des erreurs
- Automatisé lorsqu'il est utilisé avec des produits utilisant certains langages de programmation

En majorité, REST est plus facile à utiliser et est plus souple. Il a les avantages suivants par rapport à SOAP:

- Aucun besoin d'outils coûteux pour interagir avec le service Web
- Une courbe d'apprentissage plus petite pour les développeurs
- Efficace (SOAP utilise XML pour tous les messages, REST peut utiliser des formats de message plus petits)
- Rapide (pas de traitement étendu requis)
- Proche d'autres technologies Web dans sa philosophie de conception

Certaines personnes estiment que l'un des protocoles est meilleur que l'autre, Cette affirmation reste incohérente. En effet, chaque protocole présente certains avantages des inconvénients tout aussi contraignants. Vous devez choisir entre SOAP et REST en vous basant sur le langage de programmation que vous utilisez, l'environnement dans lequel vous l'utilisez ainsi que les exigences de l'application. Parfois, SOAP est un meilleur choix et d'autres fois c'est REST qui l'est. Afin d'éviter d'avoir des difficultés, vous aurez vraiment besoin de souligner les avantages et les inconvénients d'une solution en particulier dans votre situation.