Programmation web

Gestion de produits utilisateurs

avec Authentification jeton (token).

Master informatique 2019-2020

Réalisé par :

- Linda Benboudiaf n° Étudiant : 21911019

Plan

- Introduction.
- Diagrammes de classes et de séquences.
- Algorithme.
- Conclusion.

Introduction

Le but de ce projet et de se familiariser avec le framework Spring Boot, AngularJS et le gestionnaire de base de données orientée document MongoDB.

L'idée du projet est de créer une application web avec une connexion sécurisée en utilisant un jeton (JSON Web Token), ce projet a été inspiré du projet que j'ai réalisé lors de mon stage l'an dernier au sein de l'entreprise Airship et des TPs et cours vu cette année.

Diagramme de classes

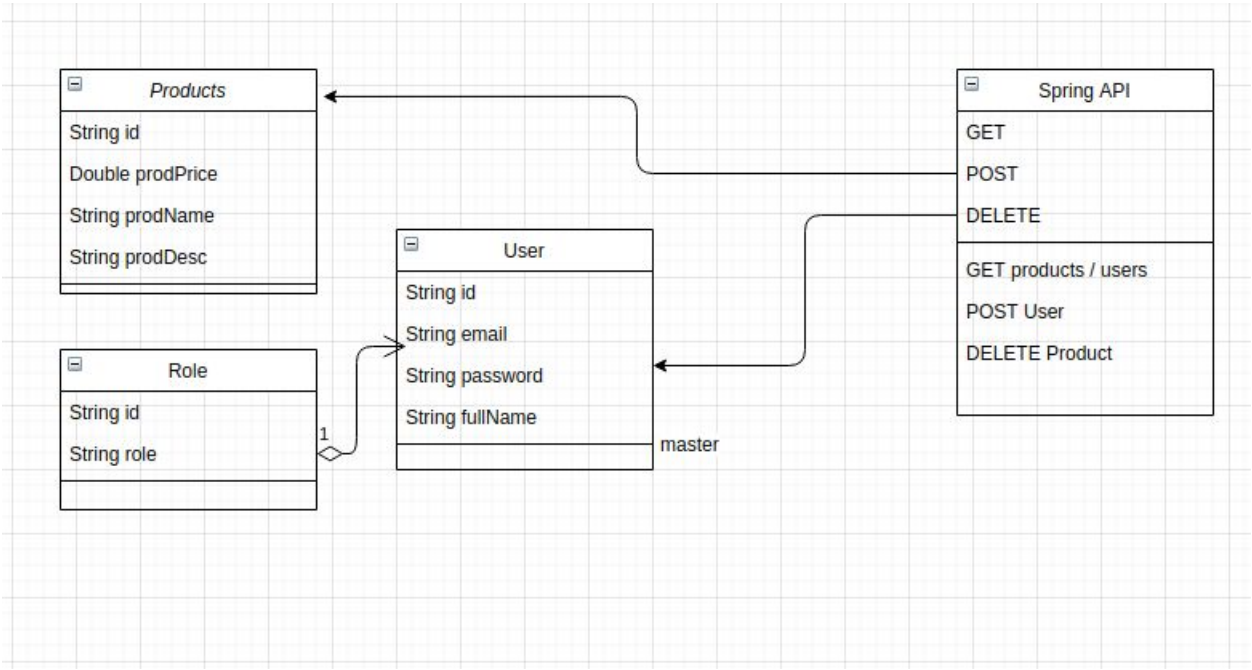
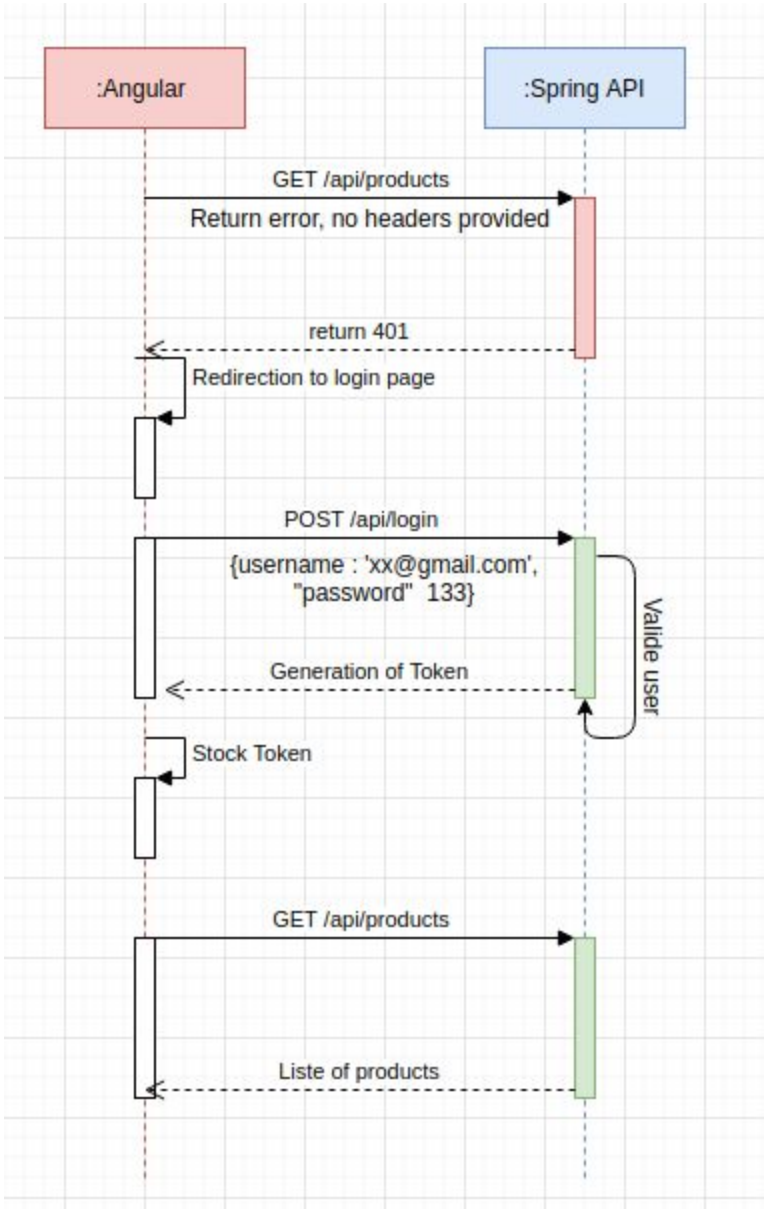


Diagramme de séquences



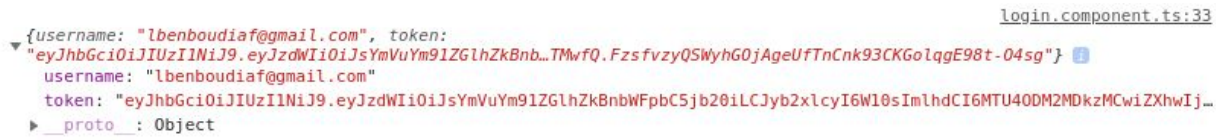
Technologies utilisées

- Back-End : Spring Boot
- Front-End : AngularJS
- Gestionnaire de base de données : MongoDB
- Gestionnaire de projet Git.

Algorithme

Ce projet à été développé en utilisant le model MVC (Model Vue Controller), la partie client est géré par l'application AngularJS.

- **Models** : Représente la structure de base de trois classes (Utilisateur, Produits, Roles)
    - L'utilisateur peut avoir deux rôle administrateur ou normal.
- **Token** : Dans ce package on a ajouter l'authentification avec jeton afin de sécuriser la connexion (Token Bearer) exemple :



Inspiré du document : [Token Spring Security]

On doit utiliser ce Token généré pour faire des appel API via Postman

- **Controlleurs** : Permet de relier les Models avec Vue, il contient principalement les APIs de Login, Register (ajouter nouvel utilisateur) et la gestion des produits (Ajouter, Afficher et Supprimer produit) exemple d'API:

Ajouter un utilisateur :

```
/** Insert new User into collection "users" **/
@PostMapping("/register")
public ResponseEntity register(@RequestBody User user) {
    User userExists = userService.findUserByEmail(user.getEmail());
    if (userExists != null) {
        throw new BadCredentialsException("User with username: " + user.getEmail() + " already exists");
    }
    userService.saveUser(user);
    Map<Object, Object> model = new HashMap<>();
    model.put( k: "message",  v: "User registered successfully");
    return ok(model);
}
```

Ajouter un produit

```
@PostMapping(value = "product")
@ResponseStatus(HttpStatus.ACCEPTED)
public String addProduct(@RequestBody Products p){
    this.products.add(p);
    productsInterface.save(p);
    return p.toString();
}
```

Supprimer un produit

```
@DeleteMapping("/api/{id}")
public void deleteProduct(@PathVariable("id") String id ) { productsInterface.deleteById(id); }
```

Récupérer la liste produit

```
private List<Products>products = new ArrayList<>();
@Autowired
ProductsInterface productsInterface;
@RequestMapping(method = RequestMethod.GET, value = "/products")
public Iterable<Products> products(){
    //return all products.
    return productsInterface.findAll();
}
```

- **Interfaces** : Dans ce package on regroupe toutes les Repositories des classes **Models**

Pour la partie base de données toutes à été généré automatiquement du Back end et injecter dans le gestionnaire **MongoDB,** On a créé trois collections (user, products, rôle) pour stocker les informations dans chacune.

```
> use ProjectWebMaster
switched to db ProjectWebMaster
> show collections
products
roles
users
>
```

Exemple d'un utilisateur et produit enregistré dans la base de données

```
db.users.find() {
    "_id": ObjectId("5eac29f1e3c4646b43e21778"),
    "email": "lbenboudiaf@gmail.com",
    "password": "$2a$10$K2njP/gP3N07VXorvQ59ZerYMeo4B9h2xVSHmHcEWwAgBeHbPecmG",
    "enabled": true,
    "roles": [],
    "_class": "com.ProductManager.ProjectWebMaster.models.User"
}
```

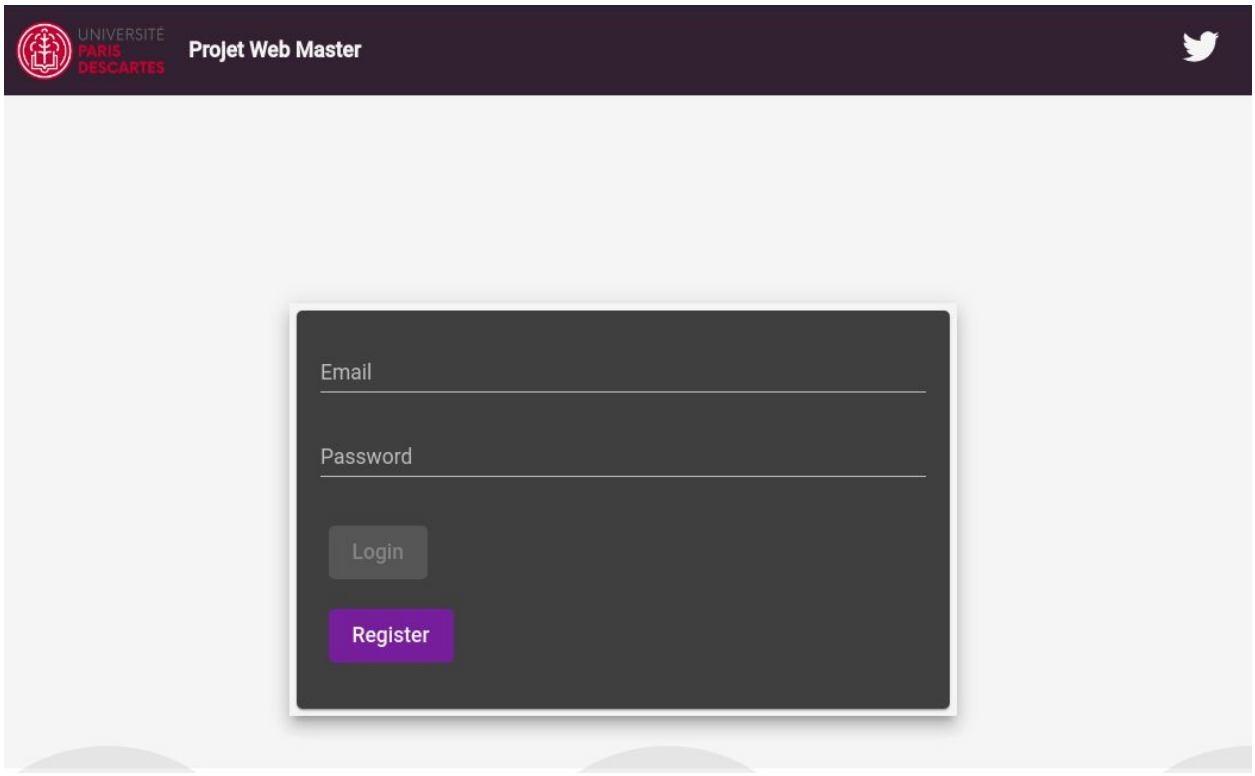
```
{
    "_id": ObjectId("5eac61215c48f8cfdf8465ec"),
    "id": "12",
    "prodName": "Logo Design",
    "prodDesc": "Book for logo pics",
    "price": 100
}
```


Une la partie Back end marche bien, on peut commencer à faire la partie UI (User Interface) à l'aide d'Angular.

- On crée une **nouvelle application**
- Pour chaque **use case** (login,register et liste de produit) on crée une **Component**.
- On ajoute **Http Interceptors** pour gérer le login Token.
- On inject API de Register, Login et Produits

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, of } from 'rxjs';
import { catchError, tap } from 'rxjs/operators';

const apiUrl = 'http://localhost:8080/api/auth/';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  isLoggedIn = false;
  redirectUrl: string;

  constructor(private http : HttpClient) {

  }
```

- On ajoute le **module @angular/material** puis on choisit un thème souhaité.



Conclusion :

La partie User Interface (UI) n'est pas encore terminé, néanmoin le côté BackEnd (Spring + MongoDB) est fini et marche bien. J'ai rencontré une réel difficulté dans le développement de l'authentification token et les appel Http via Angular.

Temps de développement : 2 semaines

Références

- Angular tutorial : https://angular.io/tutorial
- Front Design et couleurs : https://material.angular.io/
- HttpInterceptors : https://angular.io/api/common/http/HttpInterceptor#description
- Protection de l'API et création de Token : https://medium.com/@hantsy/protect-rest-apis-with-spring-security-and-jwt-5fbc90305cc5

- Inspiration app (Spring + Angular + MongoDB) : https://github.com/charroux/car-rental