



Mini projet JEE

La fabrique du père-noël

Réalisée par :

GOUMIDI Yeser,

HOUR MINA (Alternante),

HSAINA Mohamed (Alternant),

Zekri Fatma.

Enseignant :

Mr. ROUSSEAU Christopher

2020-2021

1. Présentation du projet :

Le projet consiste à développer une mini plateforme web : La fabrique du père-noël qui permet de gérer les lutins, les jouets, et les commandes. Cette application permet à un utilisateur de créer un compte ou s'identifier en tant que lutin pour consulter son profil et ses commandes ou en tant que père/mère-noël pour gérer les commandes, la liste de jouets et la liste des lutins.

2. Analyse du projet :

- User Stories:

La liste suivante représente les différents user stories de l'application :

User story 1: un utilisateur anonyme peut créer un compte lutin avec les informations suivantes : Nom, Prénom, Adresse mail, Mot de passe.

User story 2: un utilisateur anonyme peut s'authentifier avec son adresse mail et son mot de passe pour accéder à son compte lutin.

User story 3: un lutin connecté peut choisir ses compétences à partir d'une liste de compétences.

User story 4: un lutin connecté peut consulter son profil : consulter ses informations personnelles / Consulter la liste de ses compétences / Consulter le jouet qui lui a été accordé.

User story 5: un lutin connecté peut modifier ses informations personnelles et ses compétences.

User story 6: un utilisateur peut s'authentifier en tant qu'administrateur (père-noël)

User story 7: un administrateur connecté peut ajouter/modifier/supprimer des jouets.

User story 8: un administrateur connecté peut ajouter/modifier/supprimer des compétences.

User story 9: un administrateur connecté peut ajouter/modifier/supprimer des catalogues de jouets.

User story 10: un administrateur connecté peut affecter des compétences à un jouet

User story 11: un administrateur connecté peut affecter un jouet à un catalogue.

User story 12: un administrateur connecté peut affecter un jouet à une catégorie.

User story 13: un administrateur connecté peut ajouter/modifier/supprimer des lutins

User story 14: un administrateur connecté peut créer une commande avec une date de début de la commande, une date de fin, une liste des lutins et une liste des jouets

User story 15: un administrateur connecté peut affecter un lutin à un jouet dans une commande.

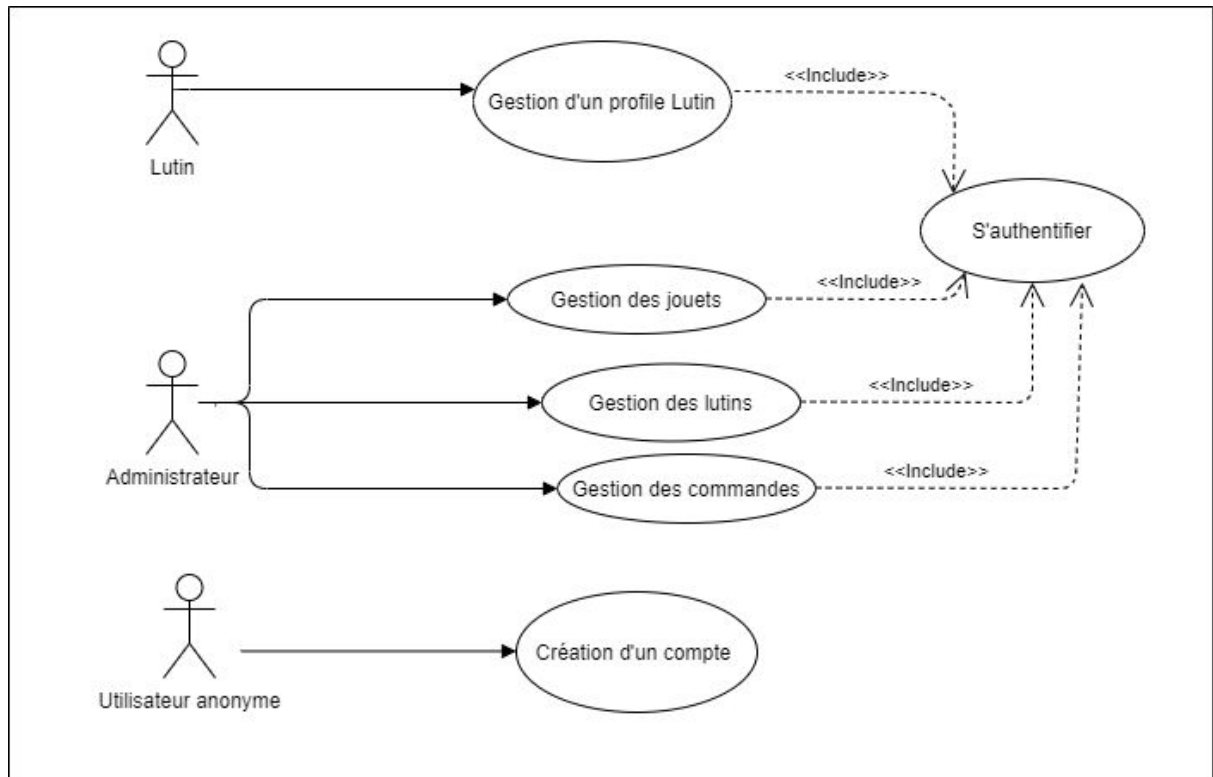
User story 16: un administrateur connecté peut consulter la liste des commandes.

User story 17: un administrateur connecté peut modifier/supprimer une commande

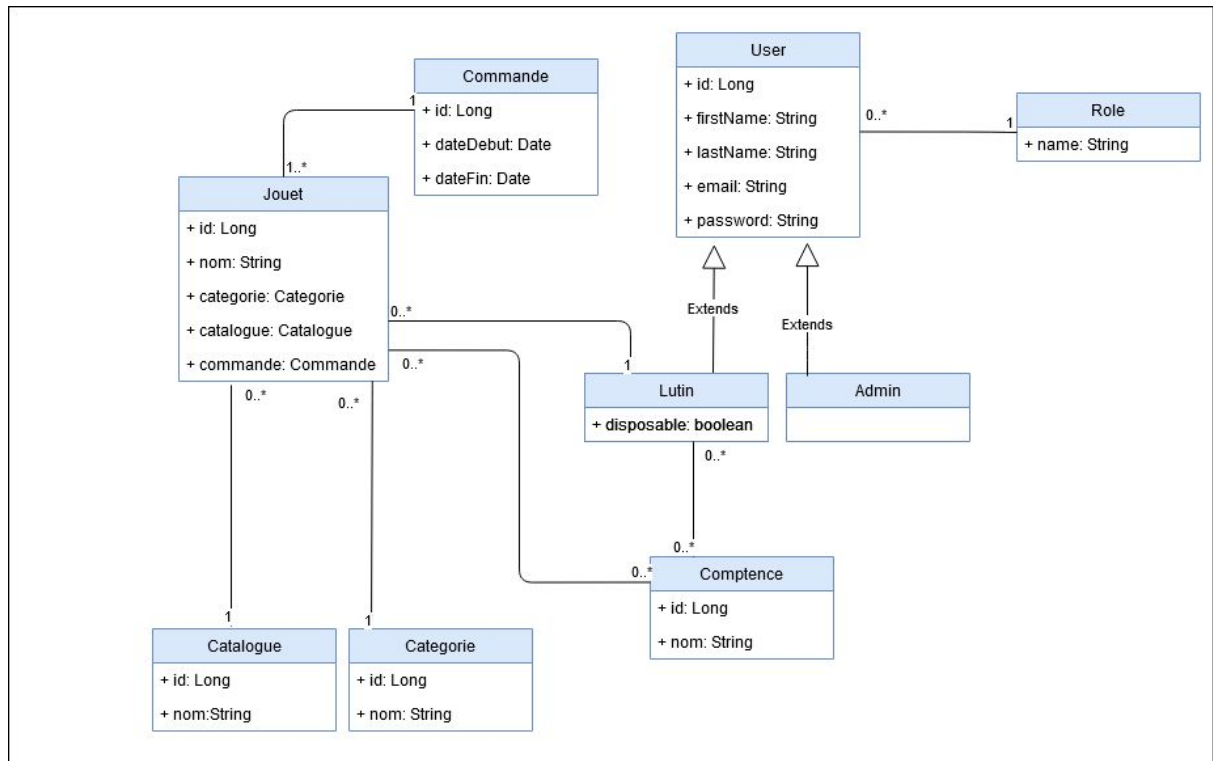
- Modèle de données:

La liste de user stories décrites au dessus, nous a permis de déterminer les entités: Jouet, Commande, Catégorie, Compétence et Catalogue pour modéliser les données de l'application et les entités : Lutin, Administrateur, Utilisateur pour indiquer les 3 types d'utilisateur de l'application.

- Diagramme des cas d'utilisations:



- Diagramme de classes :



- **classe Jouet**: cette classe présente l'objet Jouet, chaque objet Jouet contient un nom, catégorie, catalogue et commande dans laquelle il appartient et une liste de compétences qui est demandée pour fabriquer ce jouet.
- **classe Commande**: cette classe présente l'objet Commande faite par un admin, chaque commande contient un nom, date de début ,date de fin et une liste de jouets qui appartient à cette commande.
- **classe User**: cette classe présente les utilisateurs de l'application, chaque utilisateur a un nom , prénom, email, password et un rôle.
- **classe Catalogue**:cette classe présente les catalogues de jouets que l'application fabrique, chaque catalogue a son nom spécifique, chaque catalogue a une liste de jouets.
- **classe Categorie**: cette classe présente les catégories de jouets que l'application fabrique, chaque catégorie a son nom spécifique, chaque catégorie à une liste de jouets
- **classe Lutin**: cette classe présente un utilisateur avec le rôle Lutin, cet utilisateur a l'accès qu'à se connecter et gérer son profil et voir les jouets qui lui sont effectués, ainsi une liste de compétences avec son niveau pour chacune.

- **classe Admin**: cette classe présente un utilisateur avec le rôle Admin, cet utilisateur gère les jouets, catalogues, categories, lutins..
- **classe Competence**: cette classe présente les compétences de lutins et de jouets, chaque compétence a un nom spécifique.

3. Architecture de l'application :

L'application est constituée de deux parties:

- **Partie front-end**: correspond au côté client est développée avec le framework Angular (Décrire les outils utilisées pour le développement du front-end)
- **Partie back-end**: correspond au côté serveur est développée avec le framework spring qui permet de générer un projet java pré construit de type war.

Spring Boot : est un framework qui offre la possibilité de construire une application Java intégrant par défaut un écosystème des frameworks Spring. De plus, il donne la possibilité à l'application une fois construite d'être déployée et de fonctionner de façon autonome . Un de ses atouts majeurs est qu'il s'occupe de l'inclusion et de la gestion des versions de multiples dépendances aux frameworks dont peut nécessiter l'application lors de son développement.

➤ Architecture de l'application :

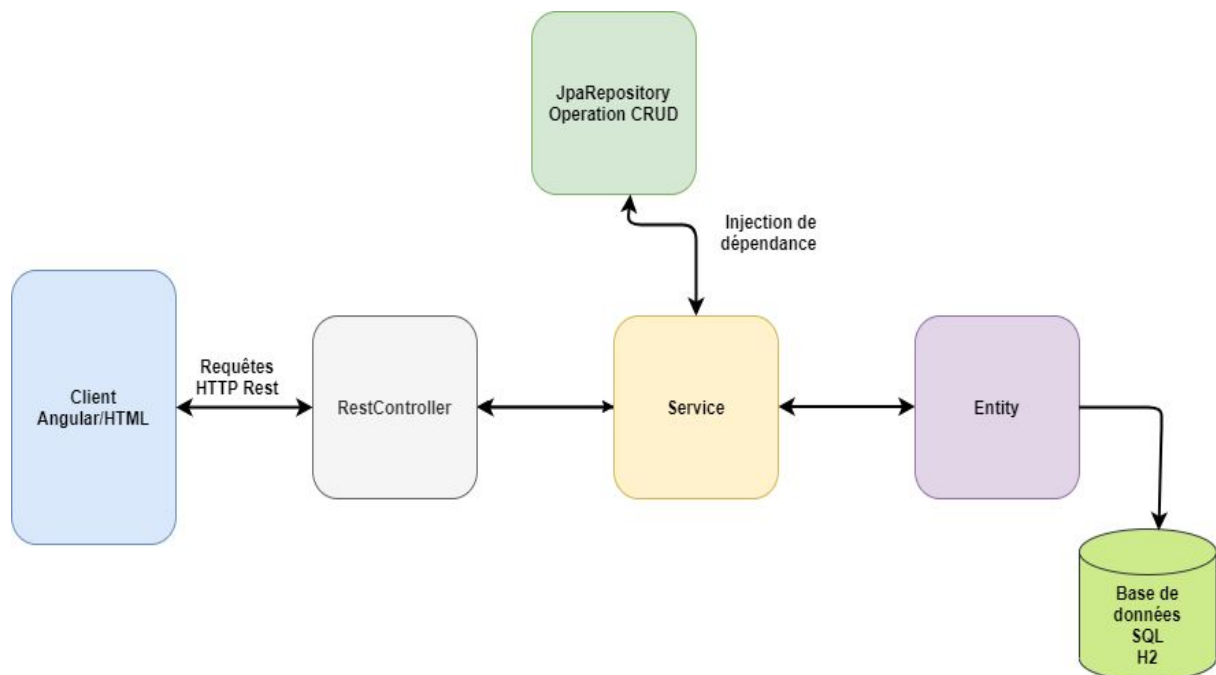


Schéma d'architecture de l'application

➤ Les outils et les framework utilisés :

-Angular : framework pour le développement de la partie front-end

- Spring Boot: framework pour la construction du projet .war.
- Spring Data JPA/Hibernate: framework pour la persistance des données
- Spring Boot Starter Web: pour la création d'applications Web, y compris RESTful, à l'aide de Spring MVC.
- Spring Security: pour la gestion de l'authentification et le contrôle d'accès.
- H2: une base de données embarquée afin d'avoir une application complètement portable et sans configuration à faire une fois développée. Une base embarquée est généralement utilisée uniquement pour les tests d'intégration, mais sa configuration reste identique à toutes autres bases de données telles que MySQL, MariaDB, Oracle.
- Java 11: pour le développement back-end
- Eclipse: IDE pour le développement de la partie Back-end.
- Visual Studio: IDE pour le développement de la partie Front-end.
- Maven: pour la gestion des dépendances de l'application.
- Git: pour la gestion des versions.

4. Comparaison entre les frameworks front-end (Angular, React , Vue.js):

Un framework est en général une collection de plusieurs librairies et un ensemble de scripts dans le but d'avoir une meilleure architecture de projets et de simplifier le développement.

Dans le côté front-end, il existe à ce jour de multiples frameworks, plus ou moins importants, plus ou moins complexes, plus ou moins fiables et plus ou moins performants, mais seuls trois qui sont adoptés par des millions de développeurs : Angular, React et Vue.js. Une comparaison rapide de ces frameworks :

	Angular	React	Vue.js
Type	Framework	Bibliothèque	Bibliothèque
Fondateurs	Google	Facebook	Ancien employé de Google
Écrit en	TypeScript	JavaScript	JavaScript
Première version	2016	2013	2014
Difficulté d'Apprentissage	+++	++	+
Modèle	MVC	Virtual DOM	Virtual DOM
	Utiliser un	Avoir de la	Avoir la séparation

Si vous voulez	framework basé sur la structure	flexibilité dans l'environnement de développement	des préoccupations
Community Support	Une large communauté de développeurs et de supporters	Communauté de développeurs Facebook	Projet open-source sponsorisé par le crowd-sourcing
Popularité	++	+++	+
Grandes entreprises utilisant	Google, Forbes, Wix...	Facebook, Uber, Netflix, Twitter, Paypal...	Alibaba, Baidu, GitLab...
Application mobile	Ionic	React Native	Vue Native

Notre choix de Angular est fait suite à plusieurs critères où ce dernier a plus d'avantages que les autres :

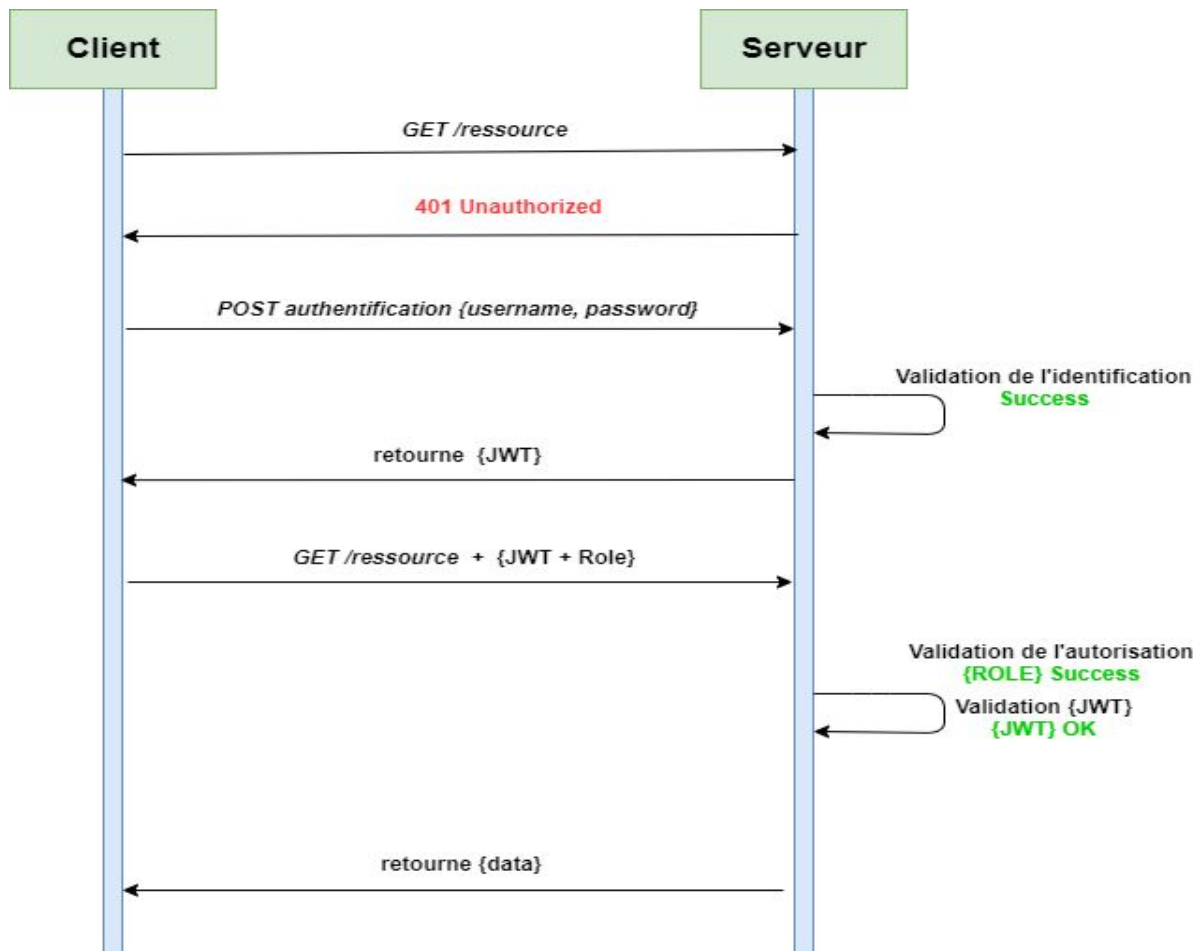
- L'utilisation de Typescript = Javascript + vérification de type + Classe + Héritage + Génériques etc.
- Le modèle MVC.
- La structure.

5. Réalisation technique

5.1 Authentification

La méthode d'authentification utilisée dans ce projet est l'authentification par jeton avec le spring security qui convient bien à Angular. Jwt (JSON Web Token) fonctionne comme suit :

1. Le client envoie une demande pour obtenir un token (Login).
2. Le serveur vérifie si le client est autorisé et lui remet un jeton avec date d'expiration et privilèges (selon son rôle dans l'application).
3. Le jeton sera stocké en session du client, et l'ajoute à l'en-tête de toutes ses requêtes au serveur (l'en-tête de token est préfixé par le mot : Bearer).
4. Pour chaque demande de ressource, le serveur vérifie si le jeton est valide et si le client dispose des privilèges pour accéder à la ressource demandée et fournit une réponse au client.



Systeme d'authentification avec JWT

La vérification des e-mails et des mots de passe est gérée par le gestionnaire d'authentification (AuthenticationManager). Si les identifiants sont bons, un token sera généré via la méthode `generateToken` (Le `JwtTokenUtil` est responsable de l'exécution des opérations JWT telles que la création et la validation. Il utilise le `io.jsonwebtoken.Jwts` et renvoie le token dans la réponse.

```

public String generateToken(UserEntity user) {
    return doGenerateToken(user.getEmail(), user.getRole().getName());
}

```

La classe `JwtRequestFilter` est responsable du filtrage et de l'interception du jeton et de l'extraction de l'utilisateur correspondant avec son rôle. chaque utilisateur (Lutin, Admin) a l'autorisation d'accéder à un certain nombre de méthodes privées (selon son rôle) grâce aux annotations `@PreAuthorize` `hasRole` («...»), et `isAuthenticated()`.

La classe `WebSecurityConfig` étend `WebSecurityConfigurerAdapter` est une classe pratique qui permet la personnalisation de `WebSecurity` et `HttpSecurity`.

grâce à cette classe nous avons retiré le lien “/ h2-console” (pour la base de données) et “/auth/login” (pour l’authentification) de l’autorisation de Spring Security, nous n’avons donc pas besoin de vous connecter à l’application avant de pouvoir nous connecter à la base de données et d’accéder au formulaire de connexion.

```
httpSecurity
    .authorizeRequests()
        // autoriser le console de bdd
        .antMatchers(
            "/h2-console/**"
        ).permitAll()
        .and().csrf().ignoringAntMatchers("/h2-console/**")
        .and().headers().frameOptions().sameOrigin()
        .and().csrf().disable()
        // ne pas authentifier cette demande particulière
        .authorizeRequests()
        .antMatchers(
            "/auth/login"
        )
        .permitAll()
        // toutes les autres demandes doivent être authentifiées
        .anyRequest().authenticated().and()
        // assurer que nous utilisons une session sans état; la session ne sera pas utilisée pour
        // stocker l'état de l'utilisateur.
        .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

// Ajouter un filtre pour valider les jetons à chaque demande
httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
```

Après avoir obtenu le jeton, la partie frontesauvgarde ce jeton en session (cookie). Chaque requête HTTP exécutée sera interceptée par la classe d'intercepteur qui vous permet d'ajouter le champ Autorisation contenant le jeton préfixé par “Bearer” dans l'en-tête de la requête (cette partie n'est pas développée manque de temps).

Exemples :

POST http://localhost:8080/auth/login Params Send Save

Authorization Headers (3) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary JSON (application/json)

```

1 {
2   "username": "admin@email.com",
3   "password": "P@sser1234"
4 }

```

Body Cookies Headers (13) Test Results Status: 200 OK Time: 539 ms

Pretty Raw Preview JSON

```

1 {
2   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbkBlbWVpbC5jb20iLCJmaXZzZG5hbWUiOiJST0xFX0FETU0iIiwiaXNzIjoiaRk5PRUxfQVBQIiwiaWF0IjoxNjExOTQ1ODI3LCJleHAiOjE2MTE5NjM4MjQ5LjYyF08Hy5pNDBhvu8xtZPezZsDC9-f3BpZuV6XTOr180"
3 }

```

exemple d'obtention d'un token avec un utilisateur admin.

GET http://localhost:8080/lutin

Authorization Headers (3) Body Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> Accept	application/json
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbkBlbWVpbC5jb20iLCJmaXZzZG5hbWUiOiJST0xFX0FETU0iIiwiaXNzIjoiaRk5PRUxfQVBQIiwiaWF0IjoxNjExOTQ1ODI3LCJleHAiOjE2MTE5NjM4MjQ5LjYyF08Hy5pNDBhvu8xtZPezZsDC9-f3BpZuV6XTOr180
<input checked="" type="checkbox"/> Content-Type	
New key	

Body Cookies Headers (12) Test Results

Pretty Raw Preview JSON

```

2 {
3   "id": 2,
4   "firstName": "houn",
5   "lastName": "mina",
6   "email": "lutin1@email.com",
7   "password": "$2a$10$jaXQuXf7DYPAs.9AE5vtDukFigkJaE408JYjIUtI7fyK8qoKrikt.",
8   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbkBlbWVpbC5jb20iLCJmaXZzZG5hbWUiOiJST0xFX0FETU0iIiwiaXNzIjoiaRk5PRUxfQVBQIiwiaWF0IjoxNjExOTQ1ODI3LCJleHAiOjE2MTE5NjM4MjQ5LjYyF08Hy5pNDBhvu8xtZPezZsDC9-f3BpZuV6XTOr180"
9 }

```


exemple d'ajout du jeton dans le champ Autorisation. (autorisation pour admin)

GET ▼ http://localhost:8080/lutin

Authorization Headers (3) Body Pre-request Script Tests

Type No Auth ▼

Body Cookies Headers (12) Test Results

Pretty Raw Preview JSON ▼ 

```

1 {
2   "timestamp": "2021-01-29T19:14:07.582+00:00",
3   "status": 403,
4   "error": "Forbidden",
5   "trace": "org.springframework.security.access.AccessDeniedException: Access is denied\r\n\tat
             .AffirmativeBased.decide(AffirmativeBased.java:73)\r\n\tat org.springframework.security.a
             .attemptAuthorization(AbstractSecurityInterceptor.java:238)\r\n\tat org.springframework.s
             .AbstractSecurityInterceptor.beforeInvocation(AbstractSecurityInterceptor.java:208)\r\n\t
             .annalliance.MethodSecurityInterceptor.invoke(MethodSecurityInterceptor.java:58)\r\n\tat

```

exemple d'échec d'autorisation pour un lutin (lutin n'est pas autorisé).

Nous avons défini la classe Util dans le package (com.univ.rouen.util) qui permet de définir le format et la vérification du email et mot de passe.

```

public static boolean isPasswordValid(String password) {
    String pattern = "(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=])(?=\\S+$).{8,}";

    public static boolean isEmailValid(String email) {
        String regex = "^\\w!#$%&'*/=?`{|}~^-]+(?:\\w!#$%&'*/=?`{|}~^-]+)*@(?:(?![a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,6})$";

```

Nous avons également utilisé la fonction `bCryptPasswordEncoder.encode` (fonction de hachage fort) pour le stockage des mots de passe.

SELECT * FROM USER;

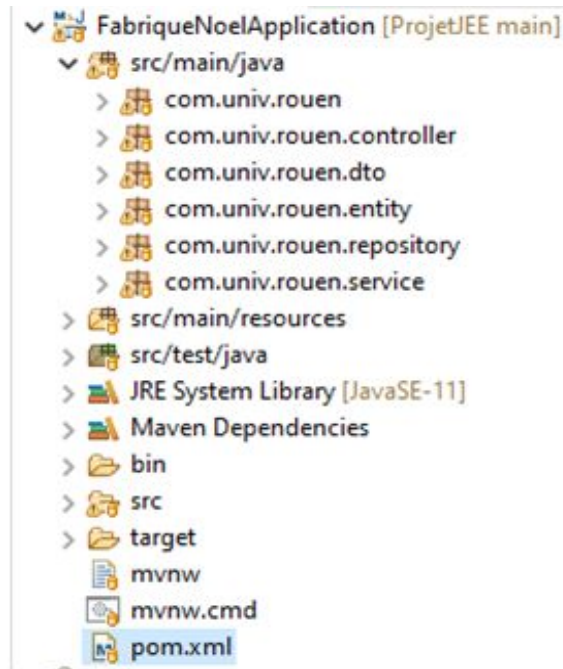
USER_TYPE	ID	EMAIL	FIRST_NAME	LAST_NAME	PASSWORD
admin	1	admin@email.com	admin	admin	\$2a\$10\$jaXQuXf7DYPA\$.9AE5vtDukFigJaE4O8JYjIUtI7fyK8qoKriKt.
lutin	2	lutin1@email.com	hour	mina	\$2a\$10\$jaXQuXf7DYPA\$.9AE5vtDukFigJaE4O8JYjIUtI7fyK8qoKriKt.
lutin	3	lutin2@email.com	hesina	moh	\$2a\$10\$jaXQuXf7DYPA\$.9AE5vtDukFigJaE4O8JYjIUtI7fyK8qoKriKt.
lutin	4	lutin3@email.com	zekri	fatma	\$2a\$10\$jaXQuXf7DYPA\$.9AE5vtDukFigJaE4O8JYjIUtI7fyK8qoKriKt.
lutin	5	lutin4@email.com	goumidli	yesser	\$2a\$10\$jaXQuXf7DYPA\$.9AE5vtDukFigJaE4O8JYjIUtI7fyK8qoKriKt.

(5 rows, 8 ms)

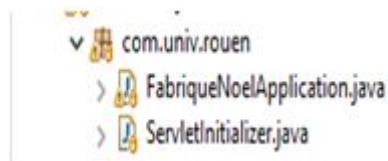
exemple de stockage des mots de passe dans la base de données

5.2 Back-end

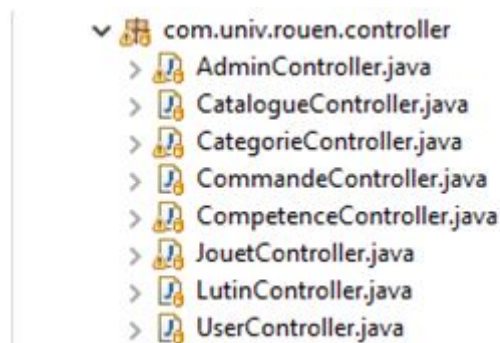
Pour bien organiser le code source de l'application et respecter l'architecture que nous avons défini au dessus les différents parties de l'application sont regroupés de la manière suivante:



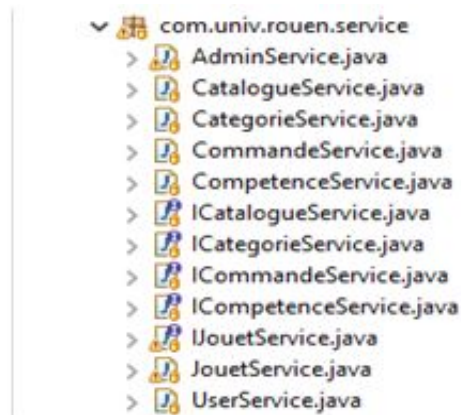
- **Package com.univ.rouen** : contient la classe main de l'application. C'est le point d'entrée à l'application.



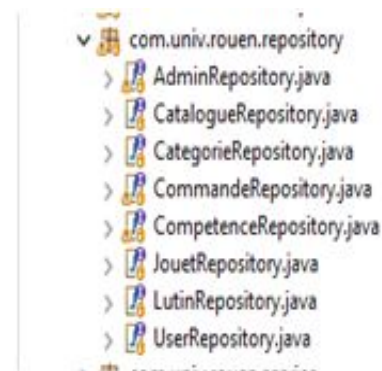
- **Package com.univ.rouen.controller** : contient les classes Rest Controller correspondant aux classes d'exploitation des services Rest (web service) de l'application en direction du front-end.



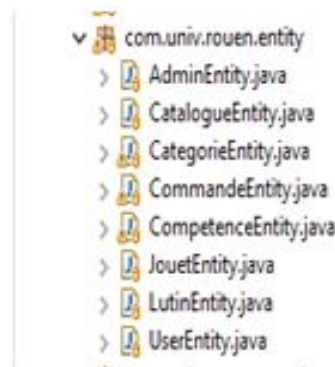
- **Package com.univ.rouen.service**: contient les services qui permettent le traitement des règles métiers de l'application.



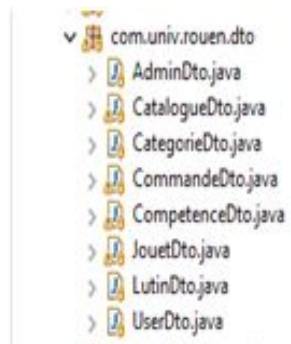
- **Package com.univ.rouen.repository:** contient les interfaces qui permettent l'accès à la base de données et de traitement des requêtes sur les entités.(les opérations CRUD)



- **Package com.univ.rouen.entity:** contient les classes qui correspondent aux tables de la base de données(ORM Hibernate/JPA).

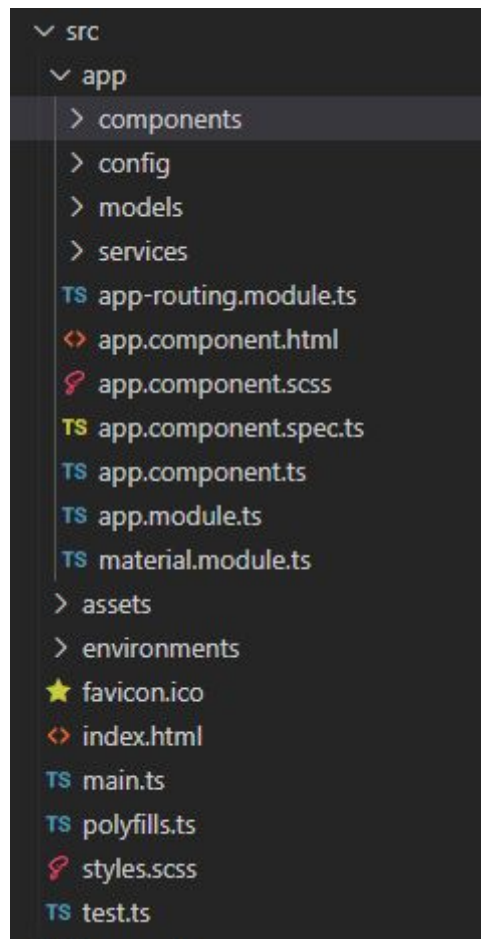


- **Package com.univ.rouen.dto** (Data Transfer Object): contient des classes qui ne possèdent que des accesseurs et des mutateurs. Elles sont utilisées pour transporter les données entre les différentes couches.



5.3 Front-end

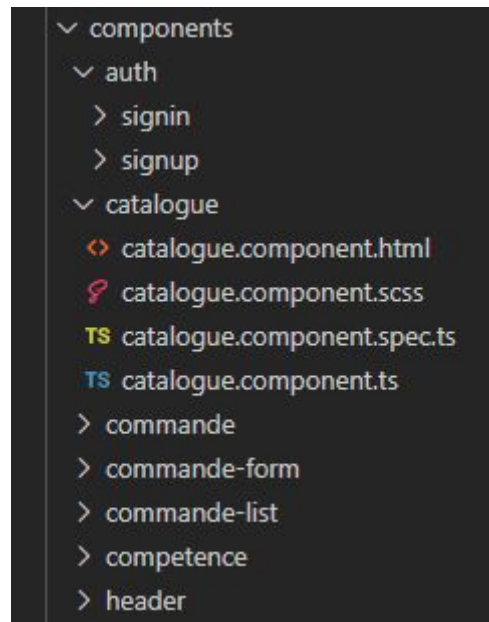
Pour un projet bien structuré nous avons utilisé l'organisation orientée composants qui permet d'avoir l'application comme plusieurs petites applications MVC. Nous ne classons pas les fichiers par rapport à leurs catégories mais par rapport au sujet auquel ils appartiennent, cela nous permet une meilleure modularité et lisibilité. Dans cette structure on peut vite et plus facilement s'y retrouver afin de modifier ou d'ajouter une fonctionnalité dans un fichier ou dans plusieurs qui sont liés entre eux.



- **Components :**

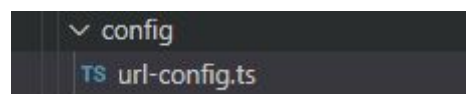
Il correspond aux différents composants et parties de l'application, chaque composant contient un controller, une vue et un scss, chaque composant est individuel, il ne dépend pas des autres et fonctionne comme une petite application MVC.

Dans le cas où il y a plusieurs sous-sections il est préférable de faire des sous-dossiers, comme dans le composant "auth" qui contient deux sous-dossiers "signin" et "signup", et cela toujours pour une question de lisibilité.



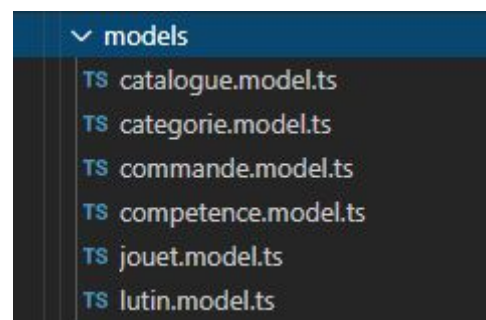
- **Config :**

Contient tous les fichiers de configuration partagés par les différentes parties de l'application. Dans le fichier "url-config" on a défini l'URL (http://localhost:8080) de la partie backend qui sera utilisée pour la communication des deux parties dans les services.



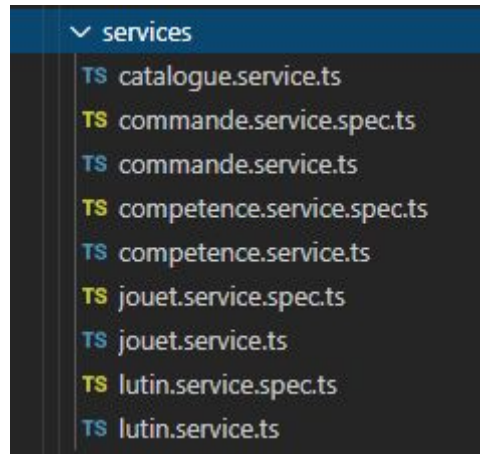
- **Models :**

Se compose des classes et interfaces qui permettent de définir la signature d'une classe où même une fonction.



- **Service :**

Contient les services qui permettent de communiquer avec la partie backend et de récupérer les données pour les partager avec toutes les parties de l'application .



- **Assets :**

L'emplacement de tous les documents, images, mocks ... qui seront utilisés dans l'application.

- **app-routing.module.ts :**

Contient toutes les routes définies, permettant d'accéder aux composants.

6. Manuel d'installation/exécution

l'application ne nécessite pas d'installation pour fonctionner, et lors du lancement de l'application, une base de données en mémoire (h2: mem) sera créée et pré-renseignée avec un compte administrateur (username : admin@email.com et password : P@sser1234) et quatre comptes utilisateurs (lutin) grâce au fichier data/data.sql.

Pour accéder à la base de données, il suffit de saisir les informations nécessaires (capture d'écran suivante) ces informations sont définies dans le fichier de propriétés de l'application.

English ▼ Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:fbNoel-db

User Name: sa

Password:

Connect Test Connection

partie back-end : l'url et le port de la plateforme (localhost et 8080 par défaut).

7. Manuel d'utilisation

7.1 Partie administration

une fois connecté, l'administrateur peut accéder,

- Gestion des Lutins (Listing + CRUD)

```
GET ▼ http://localhost:8080/lutin
```

Pretty Raw Preview JSON ▼

```

11  },
12  "disposable": true,
13  "lutinCompetences": [
14    {
15      "id": 1,
16      "nom": "Concentration"
17    },
18    {
19      "id": 2,
20      "nom": "Technique"
21    }
22  ],
23  },
24  {
25    "id": 3,
26    "firstName": "hesina",
27    "lastName": "moh",
28    "email": "lutin2@email.com",
29    "password": "$2a$10$jaXQuXf7DYPAs.9AE5vtDukFigkJaE408JYjIUtI7fyK8qoKriKt.",

```

exemple de listing avec la méthode GET

PUT http://localhost:8080/lutin/updateLutin Params Send

Authorization Headers (3) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {
2   "id": 3,
3   "firstName": "hesina",
4   "lastName": "moh",
5   "email": "lutin2@email.com",
6   "password": "$2a$10$jaXQuXf7DYPAs.9AE5vtDukFigkJaE408JYjIUtI7fyK8qoKriKt.",
7   "disposable": false
8 }
```

Body Cookies Headers (13) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
1 {
2   "firstName": "hesina",
3   "lastName": "moh",
4   "email": "lutin2@email.com",
5   "password": "$2a$10$U71IHpvYCYXKB6r41HvsZex1CV4k1DmZCr1U.b0LB6oF0wFAxQxvi",
6   "id": 3,
7   "disposable": false,
8   "lutinCompetences": null
9 }
```

Activer Windows

exemple de mettre à jour un lutin avec la méthode PUT

- Gestion des Jouets du catalogue

POST http://localhost:8080/catalogues/addCatalogue Params Send

Authorization Headers (3) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {
2   "id": 1,
3   "nom": "catalogue1"
4 }
```

Body Cookies Headers (13) Test Results Status: 201 Created

Pretty Raw Preview JSON

```
1 {
2   "id": 1,
3   "nom": "catalogue1",
4   "jouets": null
5 }
```


exemple d'ajouter un catalogue avec la méthode POST

GET ▼ http://localhost:8080/catalogues Params Send ▼

Authorization Headers (3) Body Pre-request Script Tests

Type No Auth ▼

Body Cookies Headers (12) Test Results Status: 200 OK

Pretty Raw Preview JSON ▼ 

```
1 [
2   {
3     "id": 1,
4     "nom": "catalogue1",
5     "jouets": []
6   },
7   {
8     "id": 2,
9     "nom": "catalogue2",
10    "jouets": []
11  }
12 ]
```

exemple de listing des catalogues avec la méthode GET

- Gestion des catégories de jouets (Listing + CRU, pas de delete)


POST ▼ http://localhost:8080/categories/addCategorie Params Send

Authorization Headers (3) Body ● Pre-request Script Tests

☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json) ▼

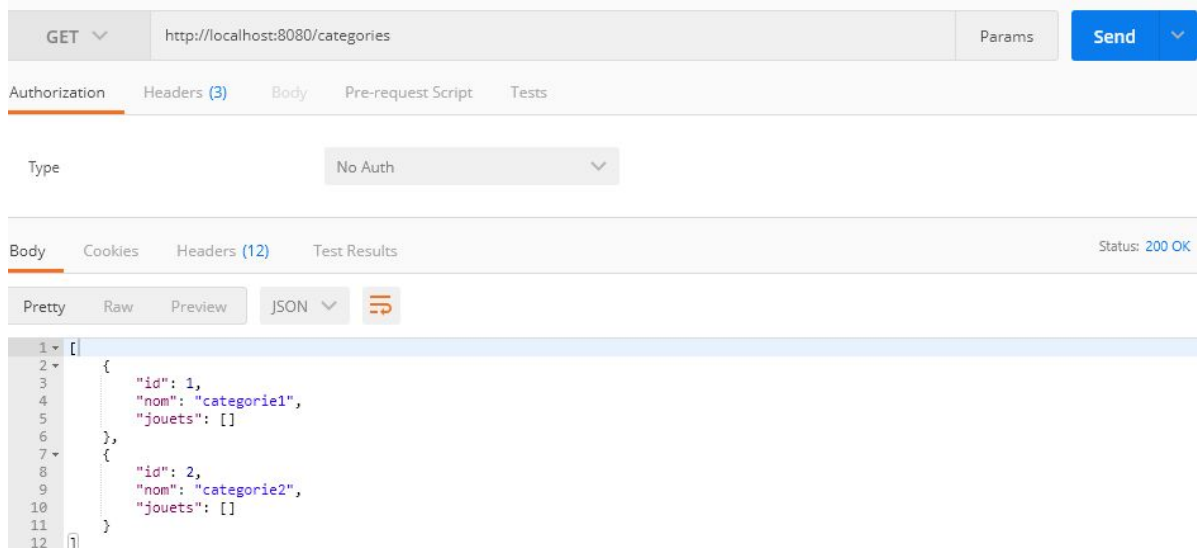
```
1 {
2   "id": 1,
3   "nom": "categorie1"
4 }
```

Body Cookies Headers (13) Test Results Status: 200 OK

Pretty Raw Preview JSON ▼ 

```
1 {
2   "id": 1,
3   "nom": "categorie1",
4   "jouets": null
5 }
```

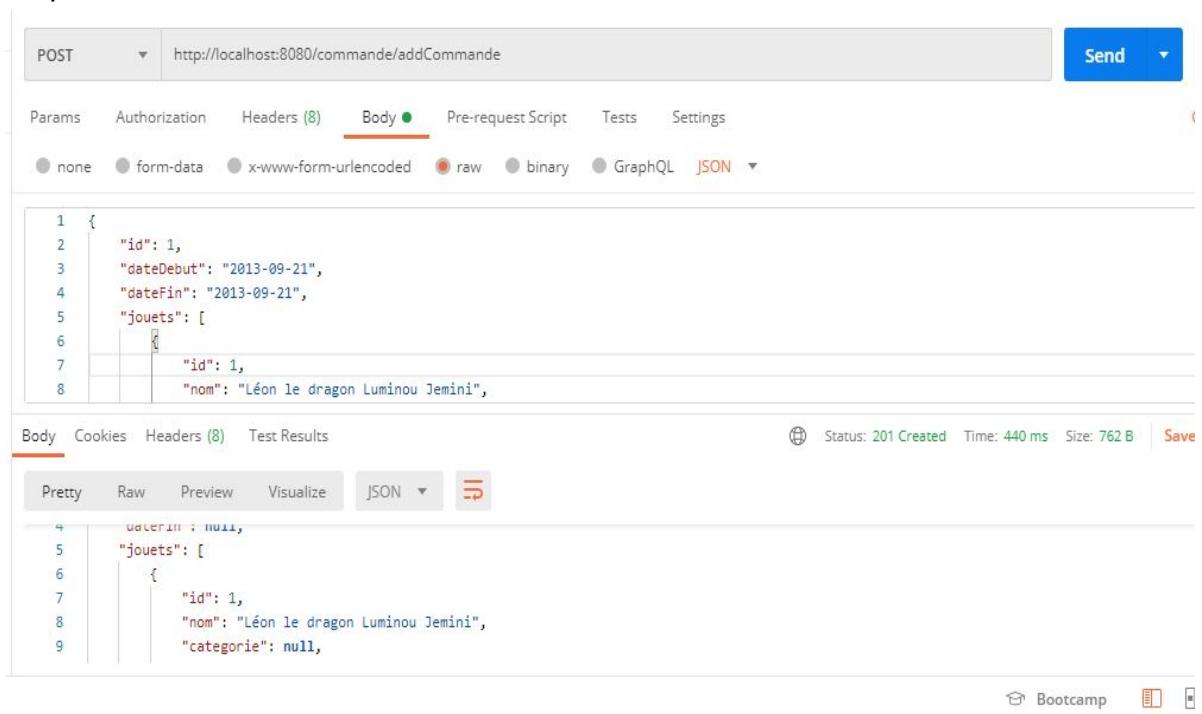
exemple d'ajouter un catégorie avec la méthode POST



exemple de listing des catégorie avec la méthode GET

- Gestion des compétences des lutins, et des compétences nécessaires à la fabrication des jouets.

L'exemple suivant montre l'ajout d'une commande avec un jouet, des lutins et leurs compétences associées.



exemple d'ajouter une commande avec la méthode POST

Nous avons envisagé de faire les deux points suivants dans la partie front avec la bibliothèque PDFMake.

- Export (PDF ou json ou XML ou csv) listing Lutins
- Export (PDF ou json ou XML ou csv) commandes avec état (à faire / en cours / réalisée)

7.2 Partie utilisateur

une fois un utilisateur est connecté, il peut accéder et modifier les ses informations, accéder aux informations des commandes en cours sur lesquelles l'utilisateur est affecté.

Dans la partie front-end, nous n'étions pas très avancés car c'est la première fois que nous utilisons ce framework et cela ne nous permet pas d'avancer plus vite sur le projet.

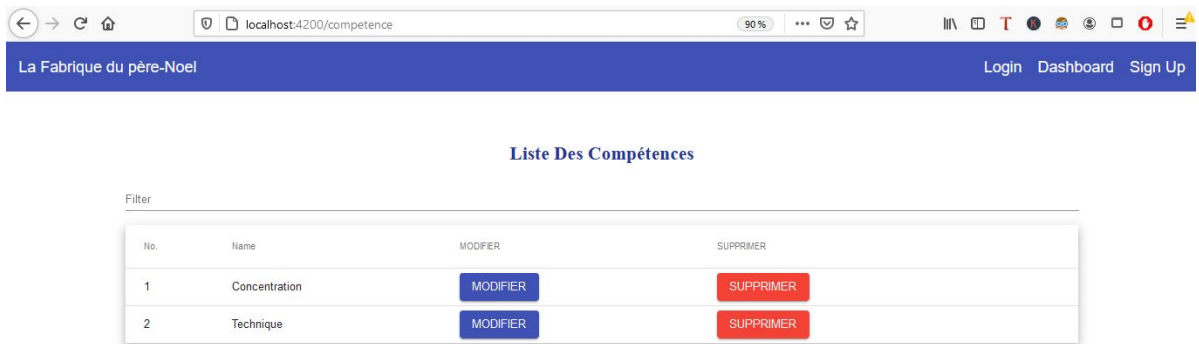
Nous n'avons pas encore mis l'intégration entre la partie front et back après la dernière mise à jour du projet. (les captures suivantes montre quelque exemples)

A white rectangular form with rounded corners. At the top, it has the title "Log in". Below the title, there are two input fields: the first is labeled "Username" and the second is labeled "Password". Below these fields is a pink button with the text "Log in".

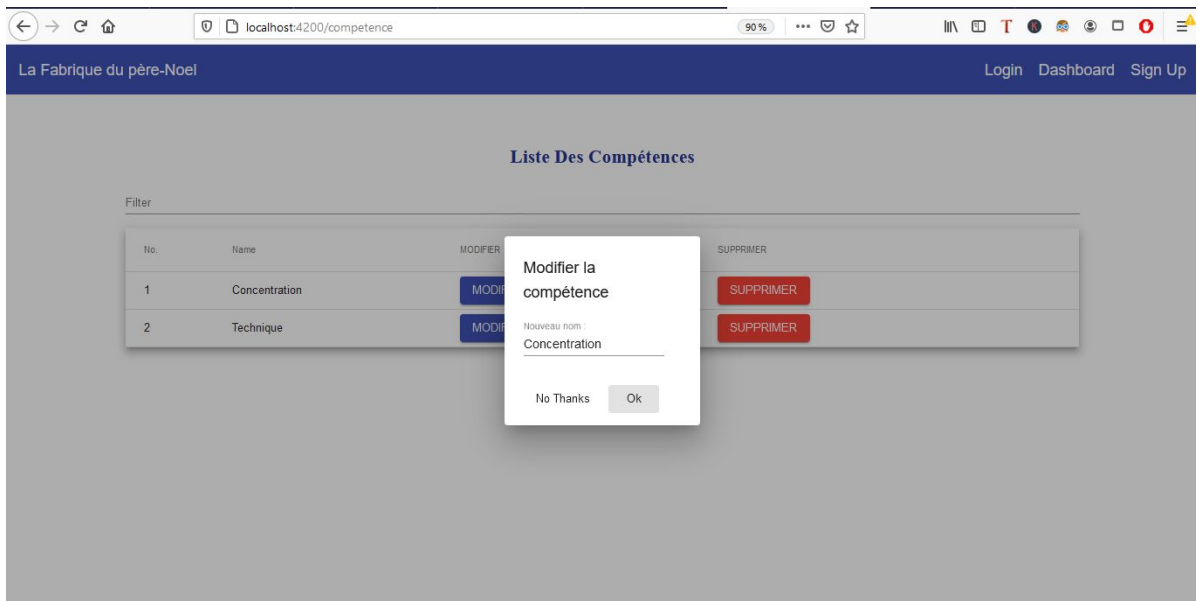
Exemple de page de connexion

A white rectangular form with rounded corners. At the top, it has the title "Register". Below the title, there are four input fields: "Username", "Email", "Password", and "Choose a role...". Below these fields is a pink button with the text "Register".

Exemple de page d'inscription



Exemple de page de liste de compétences



Exemple de page de modification d'une compétence



Liste Des Compétences

Filter

No.	Name	MODIFIER	SUPPRIMER
1	Concentration	<button>MODIFIER</button>	<button>SUPPRIMER</button>
2	Technique	<button>MODIFIER</button>	<button>SUPPRIMER</button>

Compétence bien modifié

Exemple de page de modification d'une compétence réussie



Liste Des Compétences

Filter
Conj

No.	Name	MODIFIER	SUPPRIMER
1	Concentration	<button>MODIFIER</button>	<button>SUPPRIMER</button>

Exemple d'utilisation de filtre sur la liste de compétence

8. Organisation de l'équipe

Concernant l'organisation des tâches, Nous avons défini pour tous les objectifs les grandes étapes à réaliser pour y parvenir. Nous avons ensuite découpé ces étapes en tâches. Une fois le tout ordonné, chacun des membres d'équipe a pris une tâche en fonction de disponibilité et de préférence pour la livrer dans un temps défini.

Et pour la gestion de version, nous avons mis à notre disposition un dépôt Git. Cet outil m'a permis de garder trace des différentes évolutions de celui-ci. L'un des avantages de Git est sa gestion aisée des branches. Cette fonctionnalité est très utile pour tester sans crainte certaines approches et d'en garder une trace même si en l'état ces modifications ne sont prêtes pour rejoindre le tronc commun

9. Difficultés rencontrées

9.1 Problèmes

Notre équipe est composée de quatre étudiants dont deux alternants, les alternants travaillent souvent juste le soir, ils articulent leur temps entre la charge de travail d'entreprise et la charge du projet. Les deux autres étudiants ont des tâches dans le projet annuel et d'autres mini-projets, préparation aux examens, (manque du temps) ce qui empêche l'avancement rapide du projet.

Au niveau matériel, nous avons utilisé nos PC qui ne fonctionnent pas forcément bien sur ce genre de projets.

utilisation de nouvelles technologies qui nécessitent un bon apprentissage avant de démarrer ce projet.

10. Conclusion

Le but pédagogique de ce jeu est de pouvoir transmettre deux types de savoir :
Le savoir théorique à travers l'apprentissage et l'évaluation des connaissances. Le savoir faire c'est à-dire mettre en pratique les différents technologies vu en cours, afin de concevoir un projet performant.

11. Références

- <https://spring.io/guides/gs/securing-web/>
- <https://openclassrooms.com/fr/courses/4668271-developpez-des-applications-web-avec-angular>