

DOUGUEDROIT AMANDINE

DOSSIER DE PROJET

DEVELOPPEUR WEB ET WEB MOBILE
De septembre 2020 à juin 2021

ITSO



SOMMAIRE

I.	Liste des compétences du référentiel couvertes par le projet.....	3
II.	Résumé du projet.....	3
III.	Spécifications fonctionnelles du « site Admin ».....	4
A.	Analyse de la fonctionnalité : Gestion des utilisateurs.....	4
1.	Quelques définitions.....	4
2.	Règles de gestion	6
B.	Diagramme des cas d'utilisation	7
	7
C.	Cas d'utilisation : Administrer les utilisateurs	8
D.	Cas d'utilisation détaillé : Gérer les utilisateurs	9
E.	Diagramme de classes.....	18
IV.	Spécifications techniques du projet	19
A.	Choix technologiques	19
1.	Back-end	19
2.	Front-end.....	20
3.	Base de données.....	20
B.	Accessibilité	20
1.	Compatibilité navigateur	20
2.	Types d'appareils.....	20
V.	Réalisations du candidat	21
A.	Les tests d'intégration sur le « site Admin »	21
1.	« UserManagerController »	21
2.	Jeu d'essai	21
3.	« UserManagerControllerTest »	25
4.	Résultats.....	26
B.	Une nouvelle approche méthodologique.....	28
1.	Nouvelles IHM.....	28
2.	Approche TDD.....	37

VI.	Veille sur les vulnérabilités de sécurité	42
A.	Mauvaise gestion de l'authentification et des droits d'accès	42
1.	L'authentification.....	42
2.	Les droits d'accès.....	42
B.	Injections SQL.....	43
C.	Cross-Site Scripting (XSS).....	43
VII.	Recherche à partir d'un site anglophone	44
VIII.	Traduction du site anglophone sélectionné durant la recherche.....	46
A.	Extrait du site anglophone	46
B.	Traduction.....	46
	Remerciements	47
	Annexes	48
	A1.....	48
	A2.....	49

I. Liste des compétences du référentiel couvertes par le projet

- **Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité :**
 - Maquetter une application
 - Réaliser une interface utilisateur web statique et adaptable
 - Développer une interface utilisateur web dynamique

- **Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité :**
 - Créer une base de données
 - Développer les composants d'accès aux données
 - Développer la partie back-end d'une application web ou web mobile

- **Compétences transversales**
 - Utiliser l'anglais dans son activité professionnelle en développement web et web mobile
 - Actualiser et partager ses compétences en développement web et web mobile

II. Résumé du projet

WeDeal est un fabricant de distributeurs automatiques de bouteilles de gaz. En plus de la fabrication des machines, WeDeal assure à ses clients la livraison, l'installation ainsi que la maintenance de celles-ci.

Pour réaliser au mieux ses missions, WeDeal souhaitait offrir à ses clients une interface de gestion des distributeurs en leur possession. De ces besoins sont nées deux applications web sur lesquelles les clients de WeDeal peuvent gérer leur parc de machines. La première, « WeDeal site Store », est à l'usage uniquement des magasins. La deuxième, « WeDeal site Admin » est à destination des administrateurs.

Le client de ces applications est la société WeDeal. Les utilisateurs finaux sont les employés de WeDeal ainsi que les différents employés des sociétés clientes de WeDeal.

La gestion des utilisateurs sur le « site Admin » est une part essentielle de l'application car l'accès aux diverses fonctionnalités du site est conditionné par des privilèges attribués aux utilisateurs en fonction de leur rôle.

ITSO m'a demandé d'initialiser la mise en place de tests automatiques sur la fonctionnalité de gestion des utilisateurs du « site Admin ».

J'ai donc organisé une réflexion sur le développement des tests dans l'application dans l'objectif de la transmettre aux employés de ITSO afin qu'ils puissent la poursuivre et l'approfondir. Cette réflexion a d'abord mené au développement de tests d'intégration sur la classe Controller de gestion des utilisateurs au sein de l'application. Puis au développement d'une petite application de gestion d'utilisateurs avec une méthode de développement piloté par les tests (Test Driven Development).

III. Spécifications fonctionnelles du « site Admin »

A. Analyse de la fonctionnalité : Gestion des utilisateurs

1. Quelques définitions

- Utilisateur (User) :

Un utilisateur est une personne physique amenée à se connecter à l'application.

- Société (Company) :

Une société est une entité qui va contenir plusieurs divisions. On peut la considérer comme le siège de l'entreprise. Aucun utilisateur n'est attaché à cette société, c'est uniquement une « capsule » qui permet d'englober les différentes divisions.

- Division (CompanyTyped) :

Aussi appelée « société typée ». Un ensemble de divisions compose une société. Chaque division représente un rôle différent que la société va prendre dans l'application. Les différentes divisions sont les suivantes :

- **Constructeur (Manufacturer) :** Le constructeur désigne la société qui construit les machines. A ce jour, seule la société WeDeal aura ce rôle mais il est prévu de pouvoir avoir plusieurs constructeurs s'il y en avait la nécessité un jour.
- **Propriétaire (Owner) :** Le propriétaire d'une machine désigne la société qui a acheté la borne et qui a un contrat avec le constructeur. Le propriétaire peut être un gazier qui achète des machines pour y vendre ses produits et du MDD ou une enseigne de GMD (Grande et Moyenne Distribution) qui achète ses propres machines pour mettre dans ses magasins ou encore un magasin indépendant qui achète sa propre borne pour vendre ses produits.
- **Magasin (Store) :** Le magasin désigne la société d'implantation d'au moins une machine.
- **Mainteneur (Maintainer) :** Le mainteneur désigne la société qui assure la maintenance d'un parc de machines.
- **Distributeur (Distributor) :** Le distributeur désigne la société qui distribue des produits dans une ou plusieurs machines. Toute société qui souhaite vendre des produits dans une machine ont le rôle de distributeur. Les gaziers sont distributeurs et les enseignes sont distributeurs de leurs MDD. Le rôle de distributeur procure le droit de définir les produits vendus, leur prix, leurs stocks, leur réapprovisionnement.

- Rôle (Role) :

Un rôle est un profil qui sera attaché à un utilisateur et qui permet de regrouper un ensemble de droits d'accès aux fonctionnalités commun à plusieurs utilisateurs. Un rôle s'inscrit dans une division car il serait illogique d'avoir par exemple un comptable d'un gazier qui ait les mêmes droits qu'un comptable de magasin.

Par exemple > Administrateur de magasin, caissier, directeur régional de gazier, directeur départemental de gazier, propriétaire de plusieurs magasins, etc.

- Privilège (Privilege) :

Les privilèges sont des autorisations d'accès à des pages, des fonctionnalités, des informations ou des actions dans l'application.

Par exemple > Voir l'historique des ventes, analyser l'historique des ventes, créer des utilisateurs, ajouter une promotion, voir le détail d'une promotion, changer le prix des articles, etc.

- Machine (Kiosk) :

Une machine, fabriquée par WeDeal, est composée d'un distributeur de bouteilles de gaz, ainsi que d'une borne de paiement du produit.

- Étiquette (Tag) :

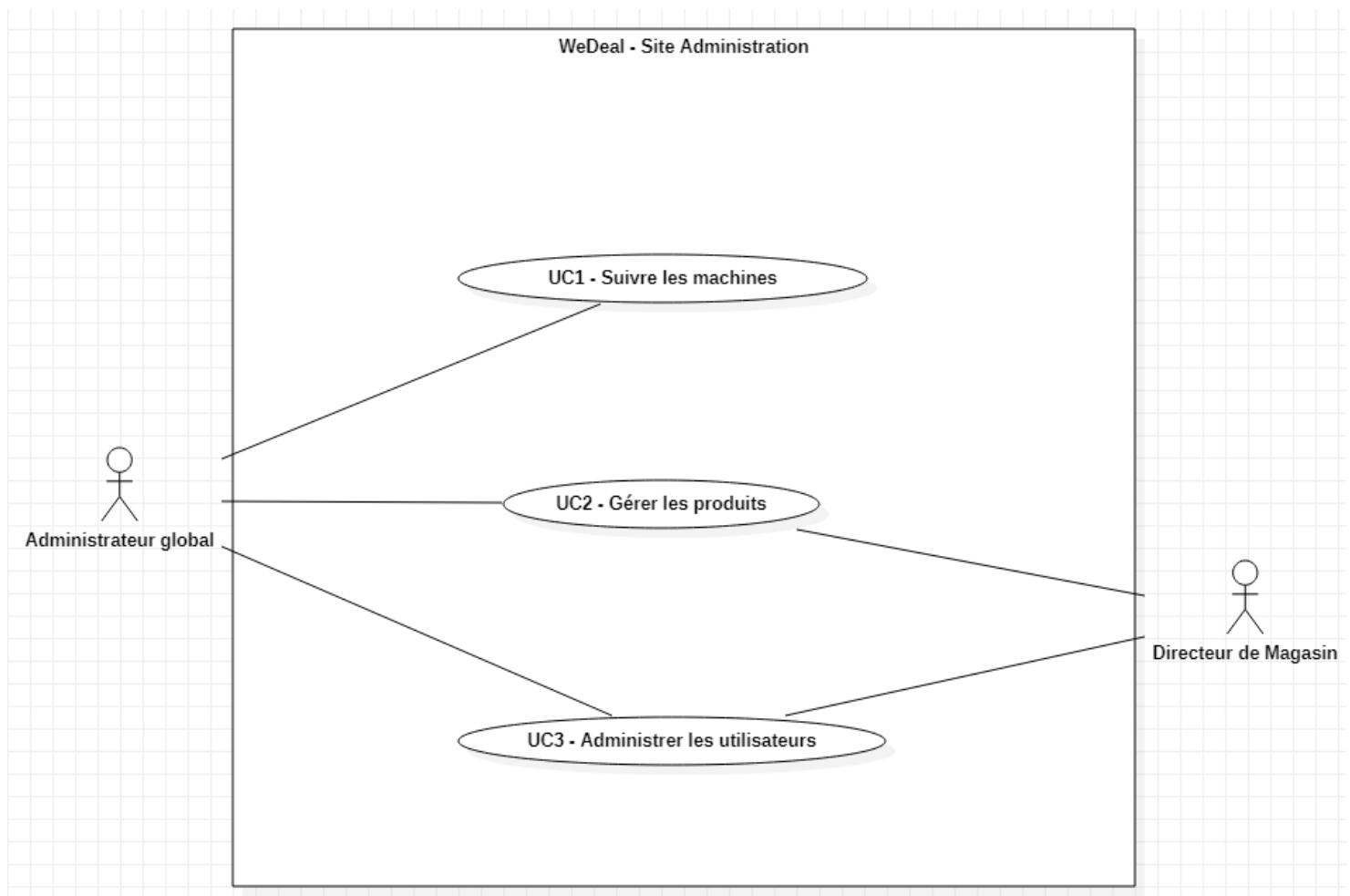
Une étiquette est assignée à un élément afin de définir ses caractéristiques ou ses critères. Elle peut-être assignée à une borne ou à un utilisateur. Si elle est assignée aux deux, cela ouvre les droits d'accès de cette borne à l'utilisateur. Elle répond au besoin de restreindre l'accès à certaines bornes pour chaque utilisateur et faciliter la sélection d'une ou plusieurs bornes à l'intérieur de ce périmètre limité afin d'y effectuer des actions.

2. Règles de gestion

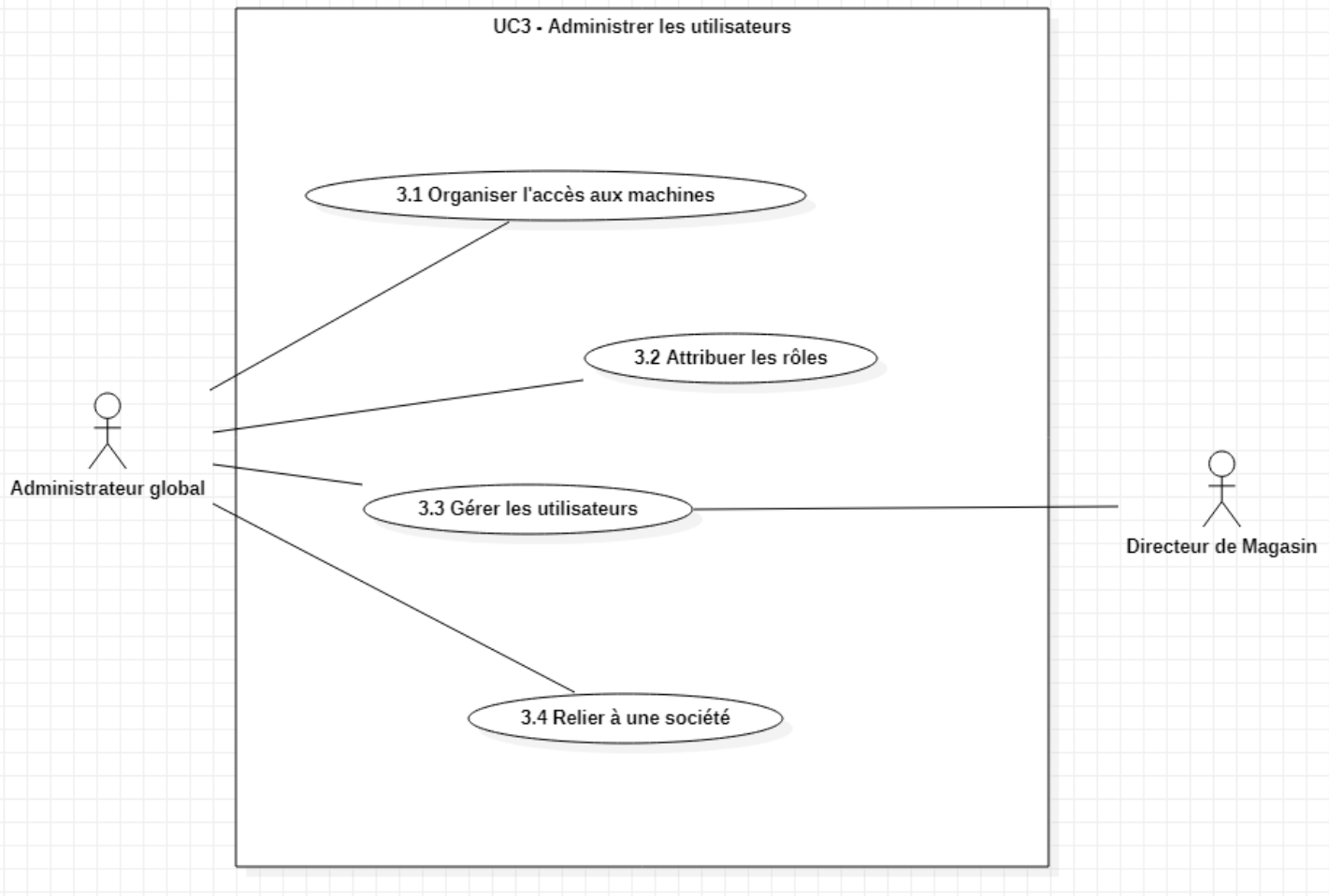
- Machine (Kiosk) :
 - Une machine a obligatoirement un Constructeur, un Propriétaire et un Magasin.
 - Une machine peut avoir 1 à n Distributeurs de produits.
- Utilisateur (User) :
 - Un utilisateur n'est attaché qu'à 1 et 1 seule Division.
 - Un utilisateur n'a qu'1 et 1 seul Rôle.
- Privilège (Privilege) :
 - Un privilège est attaché à plusieurs rôles et un Rôle est attaché à plusieurs privilèges.

- Étiquette (Tag) :
 - Une étiquette n'est définie que par son nom. Elle n'a aucune autre caractéristique.
 - Les éléments pouvant être étiquetés sont les bornes et les utilisateurs.
 - Une utilisateur n'a accès à une borne que si celle-ci possède toutes les étiquettes assignées à l'utilisateur.
 - Une borne a un nombre X d'étiquettes permettant de la définir (caractéristiques).
 - Un utilisateur a un nombre Y d'étiquettes qui délimitent (critères) le périmètre des bornes auxquelles il a un droit d'accès.
 - Un utilisateur qui n'a pas d'étiquette a accès à toutes les bornes de sa société

B. Diagramme des cas d'utilisation



C. Cas d'utilisation : Administrer les utilisateurs



D. Cas d'utilisation détaillé : Gérer les utilisateurs

- **Titre :** Gérer les utilisateurs
- **Acteurs :** Directeur de magasin
- **Le résumé :**

Ce cas d'utilisation décrit la mise à jour, dans l'application, par le directeur de magasin, qui en possède les droits, de la liste des utilisateurs des machines du magasin. L'ajout de l'utilisateur à la liste lui créera un compte au sein de l'application et permettra audit utilisateur de se connecter via ce compte.

- **Pré-condition :** Le directeur de magasin est connecté à l'application et est bien enregistré en tant que directeur de magasin.
- **L'événement déclencheur :**

Ce cas d'utilisation commence lorsque le directeur de magasin souhaite gérer la liste des utilisateurs des machines de son magasin.

- **Le scénario nominal :**

Action des acteurs	Réponse du système
1a. Le directeur de magasin souhaite consulter un utilisateur de la liste des utilisateurs.	
2a. Le directeur de magasin saisit le nom d'un utilisateur et valide sa saisie en cliquant sur un bouton.	3a. Le système affiche les utilisateurs en lien avec la recherche du directeur de magasin ainsi que leurs informations (société, division, nom, identifiant, email, rôle, niveau, service, état, tags).
4a. Le directeur de magasin indique la fin de la recherche.	

- **Les séquences alternatives :**

- Ajouter un utilisateur

Action des acteurs	Réponse du système
1b. Le directeur de magasin souhaite ajouter un utilisateur à la liste des utilisateurs.	
2a. Le directeur de magasin saisit le nom d'un utilisateur et valide sa saisie en cliquant sur un bouton.	3b. L'utilisateur n'existe pas. Le système affiche une liste vide.
4b. Le directeur décide d'ajouter l'utilisateur.	5b. Le système envoie le directeur de magasin sur un formulaire de création d'utilisateur.
6b. Le directeur de magasin saisit les informations et valide sa saisie.	7b. Le système enregistre le nouvel utilisateur. Retour sur 2a.

- **Post-condition :** L'utilisateur est bien enregistré dans le système.

Action des acteurs	Réponse du système
1b. Le directeur de magasin souhaite ajouter un utilisateur à la liste des utilisateurs.	
2a. Le directeur de magasin saisit le nom d'un utilisateur et valide sa saisie en cliquant sur un bouton.	3b. L'utilisateur n'existe pas. Le système affiche une liste vide.
4b. Le directeur décide d'ajouter l'utilisateur.	5b. Le système envoie le directeur de magasin sur un formulaire de création d'utilisateur.
6c. Le directeur de magasin décide d'annuler sa saisie et clique sur le bouton d'annulation.	7c. Le système n'enregistre pas le nouvel utilisateur. Retour sur 2a.

- **Post-condition :** L'état de l'utilisateur n'a pas changé dans le système.

○ Modifier un utilisateur

Action des acteurs	Réponse du système
1c. Le directeur de magasin souhaite modifier un utilisateur de la liste des utilisateurs.	
2a. Le directeur de magasin saisit le nom d'un utilisateur et valide sa saisie en cliquant sur un bouton. 3a. Le système affiche les utilisateurs en lien avec la recherche du directeur de magasin ainsi que leurs informations (société, division, nom, identifiant, email, rôle, niveau, service, état, tags).	3a. Le système affiche les utilisateurs en lien avec la recherche du directeur de magasin ainsi que leurs informations (société, division, nom, identifiant, email, rôle, niveau, service, état, tags).
4c. Le directeur de magasin décide de modifier l'utilisateur. Il clique sur le bouton de modification.	5c. Le système envoie le directeur de magasin sur la page de détails de l'utilisateur qui contient les informations modifiables le concernant (langue, nom, prénom, division, service, téléphone, email, rôle, état, mot de passe).
6b. Le directeur de magasin saisit les informations et valide sa saisie.	7d. Le système enregistre les modifications. Retour sur 2a.

- **Post-condition** : L'utilisateur est bien modifié dans le système.

•

Action des acteurs	Réponse du système
1c. Le directeur de magasin souhaite modifier un utilisateur de la liste des utilisateurs.	
2a. Le directeur de magasin saisit le nom d'un utilisateur et valide sa saisie en cliquant sur un bouton.	3a. Le système affiche les utilisateurs en lien avec la recherche du directeur de magasin ainsi que leurs informations (société, division, nom, identifiant, email, rôle, niveau, service, état, tags).
4c. Le directeur de magasin décide de modifier l'utilisateur. Il clique sur le bouton de modification.	5c. Le système envoie le directeur de magasin sur la page de détails de l'utilisateur qui contient les informations modifiables le concernant (langue, nom, prénom, division, service, téléphone, email, rôle, état, mot de passe).
6c. Le directeur de magasin décide d'annuler sa saisie et clique sur le bouton d'annulation.	7e. Le système n'enregistre pas les modifications. Retour sur 2a.

- **Post-condition** : L'état de l'utilisateur n'a pas changé dans le système.

- Supprimer un utilisateur

Action des acteurs	Réponse du système
1d. Le directeur de magasin souhaite supprimer un utilisateur de la liste des utilisateurs.	
2a. Le directeur de magasin saisit le nom d'un utilisateur et valide sa saisie en cliquant sur un bouton.	3a. Le système affiche les utilisateurs en lien avec la recherche du directeur de magasin ainsi que leurs informations (société, division, nom, identifiant, email, rôle, niveau, service, état, tags).
4d. Le directeur de magasin décide de supprimer l'utilisateur. Il clique sur le bouton de suppression.	5d. Le système ouvre une fenêtre modale.
6d. Le directeur de magasin valide la suppression.	7f. Le système enregistre la suppression. Retour sur 2a.

- **Post-condition** : L'utilisateur est bien supprimé du système.

Action des acteurs	Réponse du système
1d. Le directeur de magasin souhaite supprimer un utilisateur de la liste des utilisateurs.	
2a. Le directeur de magasin saisit le nom d'un utilisateur et valide sa saisie en cliquant sur un bouton.	3a. Le système affiche les utilisateurs en lien avec la recherche du directeur de magasin ainsi que leurs informations (société, division, nom, identifiant, email, rôle, niveau, service, état, tags).
4d. Le directeur de magasin décide de supprimer l'utilisateur. Il clique sur le bouton de suppression.	5d. Le système ouvre une fenêtre modale.
6e. Le directeur de magasin annule la suppression.	7g. Le système n'enregistre pas la suppression. Retour sur 2a.

- **Post-condition** : L'état de l'utilisateur n'a pas changé dans le système.

- Désactiver un utilisateur

Action des acteurs	Réponse du système
1e. Le directeur de magasin souhaite désactiver un utilisateur de la liste des utilisateurs.	
2a. Le directeur de magasin saisit le nom d'un utilisateur et valide sa saisie en cliquant sur un bouton.	3a. Le système affiche les utilisateurs en lien avec la recherche du directeur de magasin ainsi que leurs informations (société, division, nom, identifiant, email, rôle, niveau, service, état, tags).
4e. Le directeur de magasin décide de désactiver l'utilisateur. Il clique sur le bouton de désactivation.	5d. Le système ouvre une fenêtre modale.
6f. Le directeur de magasin valide la désactivation.	7h. Le système enregistre la désactivation. Retour sur 2a.

- **Post-condition** : L'utilisateur est bien désactivé dans le système.

Action des acteurs	Réponse du système
1e. Le directeur de magasin souhaite désactiver un utilisateur de la liste des utilisateurs.	
2a. Le directeur de magasin saisit le nom d'un utilisateur et valide sa saisie en cliquant sur un bouton.	3a. Le système affiche les utilisateurs en lien avec la recherche du directeur de magasin ainsi que leurs informations (société, division, nom, identifiant, email, rôle, niveau, service, état, tags).
4e. Le directeur de magasin décide de désactiver l'utilisateur. Il clique sur le bouton de désactivation.	5d. Le système ouvre une fenêtre modale.
6g. Le directeur de magasin annule la désactivation.	7i. Le système n'enregistre pas la désactivation. Retour sur 2a.

- **Post-condition** : L'état de l'utilisateur n'a pas changé dans le système.

- Maquettes :

3a.

weDeal

David ACCARIES
Directeur Magasin

Machine(s) 1 / 1

Accueil > Gestion > Gestion des utilisateurs

Gestion des utilisateurs

Bouton de création d'un utilisateur

Liste des utilisateurs des machines sélectionnées

Zone de saisie de recherche

Rechercher

Société

Division

Lignes par page : 50 sur 5

SOCIÉTÉ	DIVISION	NOM	IDENTIFIANT	EMAIL	ROLE	NIVEAU	SERVICE	ACTIF	TAGS	ACTIONS
Intermarché Monastier sur Gazeille	Magasin	ACCARIES David	monastieradmin	f.bensmail@itso.fr	Directeur Magasin	Niveau 2	Commerce	●		...
Intermarché Monastier sur Gazeille	Magasin	Employé Monastier	monastieremploye	f.bensmail@itso.fr	Employé Magasin	Niveau 4	Commerce	●		...
Intermarché Monastier sur Gazeille	Magasin	WD Monastier	monastierWD	f.bensmail@itso.fr	Administrateur Global	Niveau 1	Commerce	●		...
Intermarché Monastier sur Gazeille	Magasin	Monastier divAdmin	supervMonastier	f.bensmail@itso.fr	Superviseur de société	Niveau 1	Commerce	●		...
Intermarché Monastier sur Gazeille	Magasin	Monastier compta	comptamonastier	f.bensmail@itso.fr	Comptable Magasin	Niveau 3	Commerce	●		...

Liste des utilisateurs

Bouton de suppression ou désactivation de l'utilisateur

Bouton de modification de l'utilisateur

Aller à la page : 1

5b.

weDeal

David ACCARIES
Directeur Magasin

Machine(s) ?
1 / 1

Accueil > Gestion > Gestion des utilisateurs > Création d'un utilisateur

← Page précédente

Ajout d'utilisateur

Langue par défaut : Français

Nom* : Nom

Prénom* : Prénom

Société : Société

Division* : Division

Service : Service

Téléphone : Téléphone

Email* : Email

Rôle* : Rôle

Activation utilisateur : OFF ON

Identifiant* : Nom d'utilisateur.

Bouton de validation de la création

Bouton d'annulation de la création

Zone de saisie

Bouton d'activation ou de désactivation de l'utilisateur

Liste déroulante : sélection simple

5c.

weDeal

David ACCARIES
Directeur Magasin

Machine(s) 1 / 1

Accueil > Gestion > Gestion des utilisateurs > Modification d'un utilisateur

Page précédente

Modification d'utilisateur

Bouton d'activation ou de désactivation de l'utilisateur

Langue par défaut : Français

Nom* : Employé

Prénom* : Monastier

Société : Intermarché Monastier su

Division* : Magasin

Service : Commerce

Téléphone : Téléphone

Email* : f.bensmail@itso.fr

Rôle* : Employé Magasin

Activation utilisateur : OFF ON

Supprimer l'utilisateur

Identifiant* : monastieremploye

Mot de passe : Forcer la génération d'un nouveau mot de passe

Tags publics : Aucun tag lié

Tags privés : Aucun tag lié

Cet utilisateur n'est associé à aucun tag. Améliorez la gestion de vos utilisateurs et machines grâce aux tags !

Bouton de validation de la modification

Bouton d'annulation de la modification

Zone de saisie

Liste déroulante sélection simple

5d.

The screenshot shows the 'Gestion des utilisateurs' (User Management) page in the WeDeal system. A modal dialog titled 'Choisissez le type de suppression' (Choose the type of deletion) is open, prompting the user to select whether to deactivate or permanently delete a user. The modal has three buttons: 'Annuler' (Cancel), 'Désactiver l'utilisateur' (Deactivate user), and 'Supprimer définitivement l'utilisateur' (Permanently delete user). Annotations with arrows point to these buttons and the modal's close button.

Bouton d'annulation de la suppression ou de la désactivation (points to the 'Annuler' button)

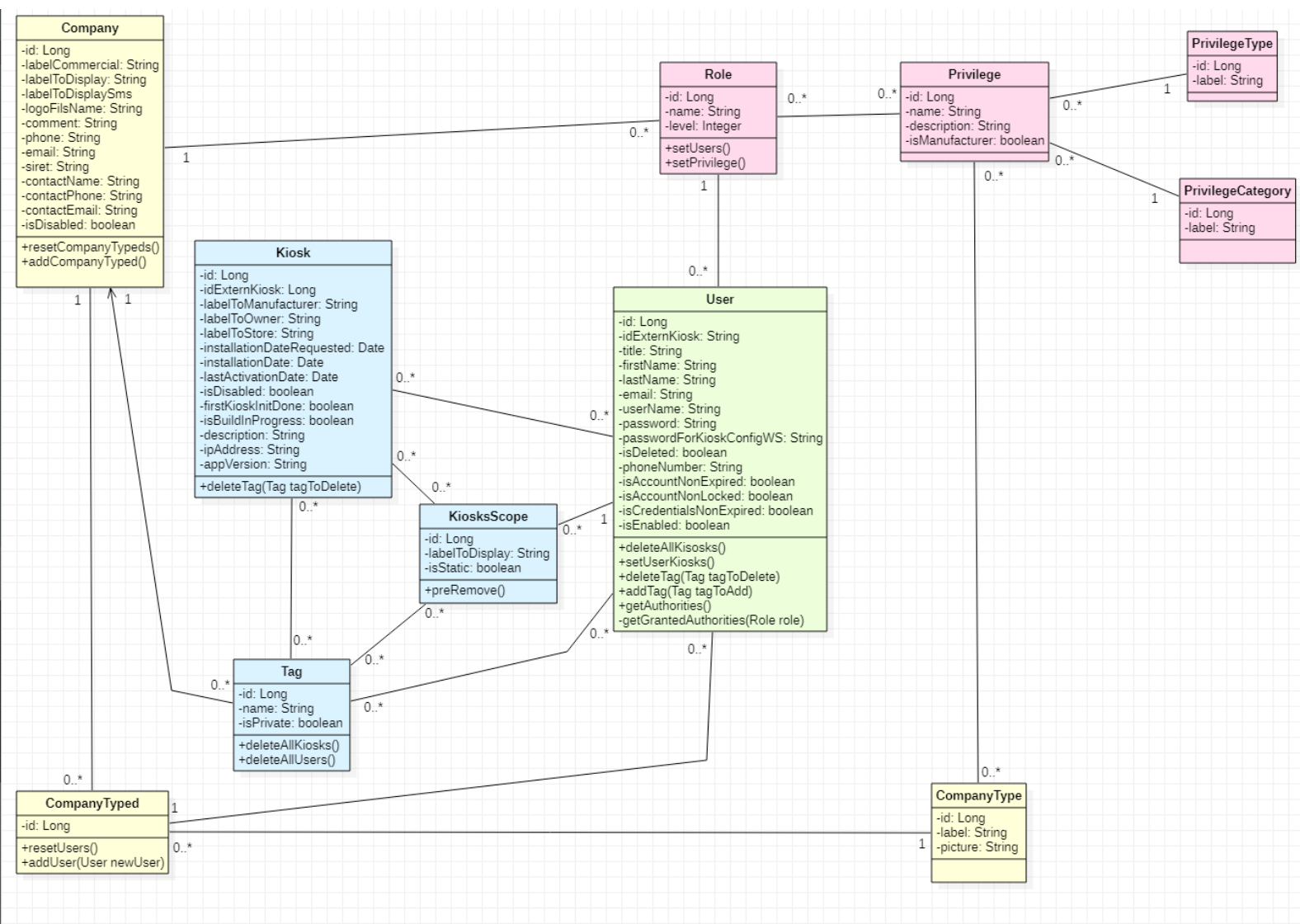
Bouton de fermeture de la modale (cause l'annulation de la suppression ou de la désactivation) (points to the close button in the top right of the modal)

Bouton de désactivation de l'utilisateur (points to the 'Désactiver l'utilisateur' button)

Bouton de suppression de l'utilisateur (points to the 'Supprimer définitivement l'utilisateur' button)

SOCIÉTÉ	DIVISION	NOM	IDENTIFIANT	EMAIL	ROLE	NIVEAU	SERVICE	ACTIF	TAGS	ACTIONS
Intermarché Monastier sur Gazeille	Magasin	ACCARIES David	monastieradmin							...
Intermarché Monastier sur Gazeille	Magasin						commerce	●		...
Intermarché Monastier sur Gazeille	Magasin						commerce	●		...
Intermarché Monastier sur Gazeille	Magasin						commerce	●		...
Intermarché Monastier sur Gazeille	Magasin						commerce	●		...

E. Diagramme de classes



IV. Spécifications techniques du projet

A. Choix technologiques

1. Back-end

a) *SpringBoot 2.5*



SpringBoot est un composant du framework MVC Spring, qui utilise le langage Java. Il a été choisi pour faciliter la configuration de Spring et réduire la taille du code.

b) *Hibernate*



Hibernate est un framework de gestion de la persistance des objets en base de données relationnelles. Les modules utilisés sont notamment l'Entity manager et le Validator.

c) *JUnit5*



JUnit5 est un framework open source de rédaction de tests unitaires et de non régression. Il est utilisé pour la mise en place des tests unitaires au sein de l'application.

2. Front-end

a) *React*



React est un framework de création d'interfaces utilisateurs qui utilise le langage JavaScript.

3. Base de données

a) *MySQL*



MySQL est un système de gestion de base de données relationnelles qui utilise le langage SQL.

B. Accessibilité

1. Compatibilité navigateur

L'application est compatible avec les navigateurs Chrome, Firefox et Safari. La compatibilité à Internet Explorer 11 est disponible via le module npm react-app-polyfill.

2. Types d'appareils

L'application s'utilise sur PC. Elle n'est pas adaptée pour les appareils mobiles.

V. Réalisations du candidat

A. Les tests d'intégration sur le « site Admin »

1. « UserManagerController »

La demande originelle d'ITSO était d'initier la mise en place de tests unitaires au sein du « site Admin ».

L'application étant étendue et complexe, la première réflexion s'est portée sur la fonctionnalité qu'il serait convenu de tester. Il est apparu plus évident de concentrer les tests autour de la gestion des utilisateurs dans l'application car il s'agit d'une fonctionnalité basique et parmi les moins difficiles à appréhender.

Seulement l'analyse de la structure du projet a révélé l'impossibilité de mettre en place ceux-ci. En effet, le « site Admin » ainsi que le « site Store » utilisent tous deux une librairie nommée « Librairie Extranet », composée pour la plus grande partie de « Helpers », mélangeant couches service et repository, dans de longues et complexes fonctions. Ainsi, il n'existe pas dans le « site Admin » de fonction qu'il soit possible de tester unitairement.

J'ai donc choisi de poser des tests sur un élément clef de la gestion des utilisateurs dans l'application, à savoir le controller, c'est-à-dire faire respecter le contrat API établi entre le back et le front. En effet, celui-ci a pour unique rôle de s'occuper de la couche HTTP et délègue les traitements métier à la couche service, ici dénommée « UserHelper ». De surcroît, dans une application web, et qui utilise un langage fortement typé tel que Java, le maillon le plus faible, donc le plus susceptible de générer des erreurs, est la couche de communication. Le controller que j'ai passé sous tests est donc le « UserManagerController ».

2. Jeu d'essai

```
@PostMapping("users/create")
@ResponseBody
@PreAuthorize("@customPermissionEvaluator.hasPrivilege(authentication, T(com.weddeal.lib_extranet.helper.util.constants.enumeration.PrivilegeEnum).AdminUsersManage.getName()))"
public ResponseEntity<> createUser(@Valid @RequestBody CompanyUserAdminCreateInputDto companyUserCreateInput)
{
```

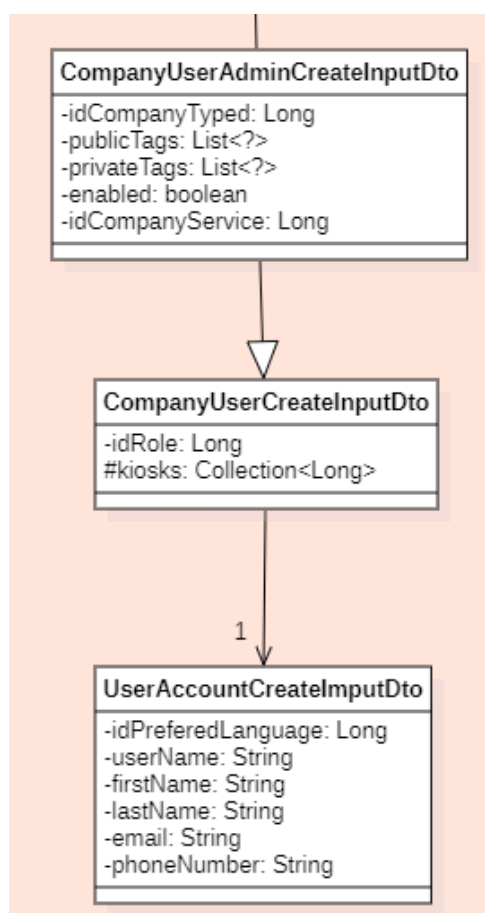
a) Classes d'équivalence et jeu de données

Avant d'écrire les méthodes de test pour la méthode `createUser()`, il convient en premier lieu de réfléchir au jeu de données.

Nous voyons dans la signature de la méthode ci-dessus, qu'elle prend en paramètre d'entrée un objet `CompanyUserAdminCreateInputDto` dont nous pouvons observer la composition ci-dessous.

Au sein de cet objet, les propriétés `idRole`, `idPreferredLanguage`, et `idCompanyTyped` sont des numériques positifs qui ne peuvent pas être `null`. Les propriétés `username`, `firstname`, `lastname` et `email` sont des chaînes de caractères qui ne peuvent pas être `null` et ne doivent pas également être seulement composées d'espaces. En outre, la propriété `username` doit être longue d'au minimum 1 caractère et au maximum 150 caractères.

En partant de ces règles de gestion, le jeu de données qui se trouve en annexes (Cf. A1) a été réalisé.



b) Fiche de tests

Une fois le jeu de données déterminé, j'ai rédigé la fiche de tests. Celle-ci comporte le jeu de données précédemment cité, la description de toutes les méthodes de tests de la méthode `createUser` ainsi que les résultats attendus. La fiche de tests est disponible en annexes (Cf A2).

Afin de réaliser les tests d'intégration sur le controller, j'ai effectué une recherche sur les tests d'intégration de la couche web avec SpringBoot. Cette recherche sera détaillée dans le huitième chapitre de ce dossier (Cf. VII. Recherche à partir d'un site anglophone). Il en est ressorti qu'un controller a six responsabilités, que je vais citer ci-dessous :

1. **Écouter les requêtes HTTP** : Le contrôleur doit répondre à certaines URLs.
2. **Désérialiser les données en entrée** : Le contrôleur doit analyser la requête HTTP entrante et créer des objets Java à partir de variables présentes dans l'URL, de paramètres présents dans la requête ou encore des données dans le corps de la requête, afin de les utiliser dans le code.
3. **Valider les données en entrée** : Le contrôleur est la première ligne de défense contre des mauvaises données entrées. C'est un endroit idéal pour opérer la validation des données.
4. **Appeler la couche métier** : Une fois les données changées en objets Java, le contrôleur doit encore les transformer en modèles, tels qu'ils sont attendus par la couche métier, et les transmettre à la couche métier.
5. **Sérialiser les données en sortie** : Le contrôleur récupère l'objet retourné par la couche métier et le sérialise dans la réponse HTTP.
6. **Traduire les exceptions** : Si une exception est levée pendant le traitement, le contrôleur doit la traduire en un statut HTTP et un message d'erreur clair pour l'utilisateur.

Ainsi, je teste que la méthode `createUser()` répond bien aux cinq premières responsabilités citées.

1. **testCreateUser** : vérifie que l'objet `CompanyUserAdminCreateInputDto` donné en entrée est correctement désérialisé, que la requête est bien écoutée et renvoie un statut HTTP 200.

2. **testCUroleKO** : vérifie qu'un objet `CompanyUserAdminCreateInputDto` donné en entrée, dont le *idRole* est *null*, retourne bien un statut HTTP 400.
3. **testCUcompanyTypedKO** : vérifie qu'un objet `CompanyUserAdminCreateInputDto` donné en entrée, dont le *idCompanyTyped* vaut 0, retourne bien un statut HTTP 400.
4. **testCUpreferedLanguageKO** : vérifie qu'un objet `CompanyUserAdminCreateInputDto` donné en entrée, dont le *idPreferredLanguage* vaut -1, retourne bien un statut HTTP 400.
5. **testCUusernameKO** : vérifie qu'un objet `CompanyUserAdminCreateInputDto` donné en entrée, dont le *username* est une chaîne de caractères vide, retourne bien un statut HTTP 400.
6. **testCUfirstnameKO** : vérifie qu'un objet `CompanyUserAdminCreateInputDto` donné en entrée, dont le *firstname* est composé de plus de 150 caractères, retourne bien un statut HTTP 400.
7. **testCUlastnameKO** : vérifie qu'un objet `CompanyUserAdminCreateInputDto` donné en entrée, dont le *lastname* est une chaîne de caractères qui ne contient qu'un espace, retourne bien un statut HTTP 400.
8. **testCUemailKO** : vérifie qu'un objet `CompanyUserAdminCreateInputDto` donné en entrée, dont le *email* est *null*, retourne bien un statut HTTP 400.
9. **testCUhelperCall** : vérifie que l'appel de la méthode *createAdminNewUser* du *UserHelper* s'effectue correctement.
10. **testCUserialization** : vérifie que l'objet retourné par le controller dans le corps de la réponse est bien sérialisé en JSON.

3. « UserManagerControllerTest »

```
/**
 * <b>UserManagerController's integration testing class.</b>
 *
 * The <i>@SpringBootTest</i> Annotation runs Spring Boot based tests. It offers, besides others, the following features :
 * "
 *   •Provides support for different webEnvironment modes,including the ability to start a fully running web server listening on a defined or random port
 *   •Registers a TestRestTemplate and/or WebTestClient bean for use in web tests that are using a fully running web server. "
 *
 * The <i>@AutoConfigureMockMvc</i> Annotation allows us to add a MockMvc instance to the application context.
 *
 * <b>The methods' annotations (in order of apparition)</b>
 *
 * The <i>@MockBean</i> Annotation mocks an instance of the UserHelper class.
 *
 * The <i>@BeforeEach</i> JUnit 5 Annotation executes the method before each test's run.
 *
 * The <i>@DisplayName</i> Annotation is used to display the name you give to the test in the JUnit view.
 *
 * The <i>@Test</i> Annotation sets the method as a JUnit test.
 *
 * The <i>@WithMockUser</i> Annotation mocks a user. It's needed when the method requires a user to be logged in.
 *
 * @author Amandine Douguedroit
 */
@SpringBootTest
@AutoConfigureMockMvc
public class UserManagerControllerTest {
```

Le framework **Spring** fournit un grand nombre d'annotations. Il en existe notamment qui permettent de déclarer des classes de tests.

Comme indiqué dans la documentation ci-dessus, **@SpringBootTest** est une annotation qui permet de charger le contexte de l'application Spring à chaque exécution des tests. Elle se déclare au dessus d'une classe de tests.

L'annotation **@AutoConfigureMockMvc** permet d'ajouter une instance de **MockMvc** dans le contexte de l'application. Cette instance est très utile puisqu'elle permet d'exécuter la requête à tester et d'effectuer des actions dessus.

```
// Instance properties

@Autowired
private MockMvc mockMvc;

@Autowired
private ObjectMapper objectMapper;

@Autowired
private WebApplicationContext context;

@MockBean
private UserHelper userHelper;
```

Parmi les propriétés d'instances, j'utilise un **@MockBean** qui me permet de simuler le comportement du *UserHelper*.

L'*ObjectMapper* me permet de sérialiser et désérialiser des objets en, et depuis, une chaîne JSON.

4. Résultats

Une fois les tests exécutés, nous remarquons qu'ils sont tous en succès, sauf un.

Runs: 10/10

Errors: 0

Failures: 1

```
▼ UserManagerControllerTest [Runner: JUnit 5] (5,855 s)
  ✓ requête /users/create : méthode createUser() => inputValidation_companyTypedKO (2,562 s)
  ✓ requête /users/create : méthode createUser() => inputValidation_emailKO (0,102 s)
  ✓ requête /users/create : méthode createUser() => inputValidation_roleKO (0,084 s)
  ✓ requête /users/create : méthode createUser() => outputSerialization_OK (1,596 s)
  ✓ requête /users/create : méthode createUser() => helperCall_OK (0,763 s)
  ✓ requête /users/create : méthode createUser() => httpMethod_RequestContentType_inputParsing_OK (0,243 s)
  ✓ requête /users/create : méthode createUser() => inputValidation_firstnameKO (0,053 s)
  ✓ requête /users/create : méthode createUser() => inputValidation_preferredLanguageKO (0,078 s)
  ✓ requête /users/create : méthode createUser() => inputValidation_usernameKO (0,060 s)
  ✗ requête /users/create : méthode createUser() => inputValidation_lastnameKO (0,313 s)
```

Failure Trace

```
! java.lang.AssertionError: Status expected:<400> but was:<200>
  at org.springframework.test.util.AssertionErrors.fail(AssertionErrors.java:59)
  at org.springframework.test.util.AssertionErrors.assertEquals(AssertionErrors.java:122)
  at org.springframework.test.web.servlet.result.StatusResultMatchers.lambda$matcher$9(StatusResultMatchers.java:627)
  at org.springframework.test.web.servlet.MockMvc$1.andExpect(MockMvc.java:196)
  at com.weddeal.webapi_admin.UserManagerControllerTest.testCUIlastnameKO(UserManagerControllerTest.java:468)
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

```

/**
 * tests that the controller's method createUser() responds 404 if the lastname is a String with only whitespaces
 *
 * @throws Exception
 */
@DisplayName("requête /users/create : méthode createUser() => inputValidation_lastnameKO")
@Test
@WithMockUser(username="monastieradmin",roles={"ADMIN"})
public void testCUlastnameKO() throws Exception
{
    UserManagerControllerTest.companyUserAdminTest.getUserAccountInput().setLastName(" ");

    mockMvc.perform(post("/users/create")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(UserManagerControllerTest.companyUserAdminTest)))
        .andExpect(status().isBadRequest());
}

```

Jetons donc un œil sur la méthode en échec : *testCUlastnameKO*.

Ici, afin de ne pas répéter l'instanciation du *CompanyUserAdminCreateInputDto* à mettre dans le corps de la requête dans chaque méthode de test, celui-ci est instancié avant chaque test dans une méthode annotée **@BeforeEach** (annotation fournie par JUnit) et est assigné à la propriété de classe *UserManagerControllerTest.companyUserAdminTest*. Comme cela est présenté ci-dessous.

```

/**
 * Instantiates a CompanyUserAdminCreateInputDto before each test
 * and put it in the companyUserAdminTest class property
 */
@BeforeEach
public void CompanyUserAdminCreateInputDtoInitializer()
{
    // instantiating a UserAccountCreateInputDto object
    UserAccountCreateInputDto userAccountTest = new UserAccountCreateInputDto();
    userAccountTest.setIdPreferredLanguage(2L);
    userAccountTest.setUserName("UserManagerControllerTest");
    userAccountTest.setFirstName("TEST");
    userAccountTest.setLastName("TEST");
    userAccountTest.setEmail("a.douguedroit@itso.fr");

    // instantiating a CompanyUserAdminCreateInputDto object
    CompanyUserAdminCreateInputDto companyUserAdminTest = new CompanyUserAdminCreateInputDto();
    companyUserAdminTest.setUserAccountInput(userAccountTest);
    companyUserAdminTest.setIdRole(2L);

    ArrayList<Long> kiosks = new ArrayList<Long>();
    kiosks.add(1L);
    companyUserAdminTest.setKiosks(kiosks);

    companyUserAdminTest.setIdCompanyTyped(4L);
    companyUserAdminTest.setEnabled(true);

    setCompanyUserAdminTest(companyUserAdminTest);
}

```

```
// Class properties

private static CompanyUserAdminCreateInputDto companyUserAdminTest;

// useful methods

public static void setCompanyUserAdminTest(CompanyUserAdminCreateInputDto companyUserAdminTest)
{
    UserManagerControllerTest.companyUserAdminTest = companyUserAdminTest;
}
```

Avant d'exécuter la requête, je change le *lastname* de mon *companyUserAdminTest* en une chaîne de caractères ne contenant que des espaces " ".

Les tests ont donc permis de révéler que le cas d'une chaîne de caractères contenant uniquement des espaces (pour *username*, *firstname*, *lastname* ou *email*) n'est pas pris en charge dans l'application.

B. Une nouvelle approche méthodologique

Outre l'initialisation de la mise en place de tests unitaires au sein de l'application, ITSO souhaitait pouvoir poursuivre la démarche d'automatisation des tests. Je ne pouvais donc pas négliger la demande de transmission de la méthodologie de tests à l'équipe. C'est pourquoi j'ai repris une partie de la fonctionnalité de gestion des utilisateurs en développant une nouvelle application selon la méthodologie « Test Driven Development ».

1. Nouvelles IHM

Pour ce faire, j'ai maqueté et conçu des interfaces utilisateurs plus simples, ce qui me permettait de concentrer mes efforts autour de la conception.


a) Les maquettes

Liste des utilisateurs


Intermarché David Accaries
Directeur Magasin















Machine(s)
8888 / 8888

Accueil > Utilisateurs

bouton créer → 

Liste des utilisateurs

Dupond  ← Zone de saisie

SOCIÉTÉ	DIVISION	NOM	IDENTIFIANT	EMAIL	RÔLE	TAGS	ACTIONS
WeDeal	Fabricant	François Dupond	DirInterFD	fdupond@email.com	Directeur	Région PACA	 
WeDeal	Fabricant	François Dupond	DirInterFD	fdupond@email.com	Directeur	Région PACA	 
WeDeal	Fabricant	François Dupond	DirInterFD	fdupond@email.com	Directeur	Région PACA	 
WeDeal	Fabricant	François Dupond	DirInterFD	fdupond@email.com	Directeur	Région PACA	 
WeDeal	Fabricant	François Dupond	DirInterFD	fdupond@email.com	Directeur	Région PACA	 
WeDeal	Fabricant	François Dupond	DirInterFD	fdupond@email.com	Directeur	Région PACA	 
WeDeal	Fabricant	François Dupond	DirInterFD	fdupond@email.com	Directeur	Région PACA	 

Liste des utilisateurs

Respectivement :
_ bouton modifier
_ bouton supprimer

création d'un utilisateur

we-Deal Intermarché David Accaries Directeur Magasin

Machine(s) 8888 / 8888

Accueil > Utilisateurs > Création d'un utilisateur

Bouton de confirmation de la saisie de création

Bouton d'annulation de la saisie de création

Création d'un utilisateur

Nom* :

Identifiant* :

Prénom* :

Mot de passe :

Société :

Confirmation :

Division* :

Email* :

Rôle* :

Zone de saisie

Liste déroulante : sélection unique

Modification d'un utilisateur

weDeal Intermarché David Accaries
Directeur Magasin

Machine(s)
8888 / 8888

Accueil > Utilisateurs > Modification d'utilisateur

Modification d'utilisateur

Zone de saisie

Nom* : Dupond

Prénom* : François

Société : WeDeal

Division* : Fabricant

Email* : fdupond@email.com

Rôle* : Directeur

Bouton d'annulation de la saisie de modification

Identifiant* : DirInterFD

Mot de passe : *****

Nouveau mpd :

Confirmation :

Bouton de confirmation de la saisie de modification

Tags : Région PACA

Liste des tags de l'utilisateur

Liste déroulante : sélection unique

b) Réalisation de l'interface web statique**(1) Les composants React**

Pour le développement des interfaces, j'ai utilisé le framework React. J'ai réalisé le style des pages avec le langage CSS.

Toute la spécificité de **React** réside dans le fait que le framework ne sépare pas artificiellement les technologies en mettant le balisage et la logique dans des fichiers séparés, mais sépare plutôt les préoccupations. Ainsi, **React** mêle le balisage et la logique dans de petites unités faiblement couplées appelées « composants ». L'avantage de ces composants est qu'ils sont créés sur mesure et sont réutilisables sur l'ensemble de l'interface.

```
3 export default function Header(props) {  
4  
5     return (  
6         <header>  
7             <nav>  
8                 <div className='connectedUser'>  
9                       
10                      
11                    <div className='userInfo'>  
12                        <p className='nav-text userName'>David Accaries</p>  
13                        <p className='nav-text status'>Directeur Magasin</p>  
14                    </div>  
15                </div>  
            </nav>  
        </header>  
    )  
}
```

Ci-dessus un extrait du premier composant de mon interface que j'ai développé avec **React**. Il s'agit du Header de mon interface. Celui-ci figurera sur chacune des pages de l'application.

Comme vous pouvez le remarquer sur la ligne 3, le composant est une fonction. Les fonctions composants acceptent en entrée un seul argument *props*, qui signifie *propriétés*, et renvoient un élément React, comme nous l'observons à partir de la ligne 5. Pour ce faire, j'utilise une extension syntaxique de JavaScript nommée **JSX**. Dans la mesure où JSX se compile en appels à *React.createElement*, il est nécessaire que la bibliothèque *React* soit présente dans la portée du code JSX.

Sur la ligne 3 également, les mots clefs *export default* permettront à notre composant d'être importé dans les autres composants de l'application. C'est ce qui rend le composant réutilisable.

(2) La navigation avec React router

```
1  import React from 'react';
2  import {
3    BrowserRouter as Router,
4    Switch,
5    Route,
6    Redirect
7  } from "react-router-dom";
8  import Header from './components/Header.js';
9  import Menu from './components/Menu.js';
10 import Home from './pages/Home.jsx';
11 import NewUser from './pages/NewUser.jsx';
12 import UpdateUser from './pages/UpdateUser.jsx';
13
14 function App() {
15   return (
16     <Router>
17       <Header/>
18       <Menu/>
19       <Switch>
20         <Route exact path="/">
21           <Redirect to="/users" />
22         </Route>
23         <Route exact path="/users" component={Home} />
24         <Route exact path="/users/new" component={NewUser} />
25         <Route exact path="/users/update/:userId" component={UpdateUser} />
26       </Switch>
27     </Router>
28   );
29 }
30
31 export default App;
```

Le *Router* est l'élément parent de l'application. Il est retourné par la fonction composant *App*. Il est composé des éléments enfants qui seront affichés. Ces éléments enfants sont des composants fournis par *react-router-dom*, tels que *Switch*, *Route* et *Redirect*, ainsi que des composants que j'ai développés. Ils ont été importés de la ligne 2 à la ligne 12. Nous retrouvons le composant *Header*, cité ci-avant, à la ligne 17. Les composants *Header* et *Menu* seront affichés dans toutes les pages de l'application. Les composants encapsulés dans le composant *Switch* seront affichés en fonction de l'URL qui leur est rattachée dans le composant *Route*.

Dans un fichier *index.js* situé à la racine du projet, le composant parent *App* est donné à la fonction `ReactDOM.render` qui mettra à jour le DOM afin de correspondre au JSX codé dans les composants.

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5
6  ReactDOM.render(
7    <React.StrictMode>
8      <App />
9    </React.StrictMode>,
10    document.getElementById('root')
11  );
```

(3) Le style avec CSS

```
32  /* Navbar */
33  nav {
34    display: inline-grid;
35    grid-template: 100px / 2fr 2fr 1fr;
36    justify-items: stretch;
37    align-items: stretch;
38    background-color: #169FED;
39    height: 100px;
40    width: 100vw;
41  }
42
43  .logo {
44    width: 150px;
45  }
46
47  .connectedUser {
48    grid-area: 1 / 1 / 1 / 1;
49    display: inline-grid;
50    grid-template: 1fr / 1fr 1fr 1fr;
51    justify-items: start;
52    align-items: start;
53    padding: 10px;
54    column-gap: 10px;
55    width: 500px;
56  }
57
```

Ci-dessus se trouve un extrait du fichier *style.css* situé dans le répertoire *public* du projet. Toute la stylisation de l'application a été réalisée avec le langage CSS, sans utilisation de

framework. Comme visible sur la ligne 34, j'ai utilisé le module CSS *Grid Layout* pour la disposition des éléments sur la page.

(4) Le référencement naturel

L'application développée par ITSO est un « Software as a Service » (SaaS). Un SaaS est un logiciel hébergé par un fournisseur tiers. Il est rendu disponible pour les clients par l'intermédiaire d'internet. Ce faisant, l'application n'a pas besoin d'être référencée sur internet.

c) Réalisation de l'interface web dynamique

(1) Accès à l'API avec Axios

```
1 import axios from 'axios';
2
3 export function getAllUsers() {
4   return axios.get('http://localhost:8080/users')
5     .then(response => response.data)
6     .catch((error) => console.log(error));
7 }
```

Afin de récupérer la liste des utilisateurs dans l'application, j'utilise la méthode *get* de *axios* qui exécute la requête HTTP selon le verbe précédemment cité sur l'URL donnée en paramètre. Cette méthode retourne une promesse de laquelle j'extrais les données via la méthode *then* et dont je gère les erreurs via la méthode *catch*.

J'effectue également des requêtes de création, modification et suppression grâce à *axios*.

```
const url = "http://localhost:8080/users";
axios.post(url, state)
```

```
const url = "http://localhost:8080/users/" + state.id;
axios.put(url, state)
```

```
axios.delete('http://localhost:8080/users/' + user.id)
```

(2) Les Hooks d'état

Dans les extraits de code ci-dessus, la constante *state* a été déclarée avant l'appel de la méthode d'*axios*. Cette constante, accompagnée de la méthode *setState*, est le retour de la méthode *useState*. C'est ce qui se nomme un *Hook d'état*. La constante *state* représente donc une variable d'état du composant. La méthode *useState* a pour unique argument l'état initial de la variable d'état. Ci-dessous, il s'agit d'un formulaire de création d'utilisateur dans le composant *createForm*.

```
6   export default function CreateForm() {
7
8     const [state, setState] = useState({
9       lastName : '',
10      firstName : '',
11      email : '',
12      username : '',
13      password : '',
14      confirmPw : ''
15    });
```

Nous observons que dans son état initial, le formulaire possède un champ *lastname*, un champ *firstname*, un champ *email*, un champ *username*, un champ *password*, ainsi qu'un champ *confirmPw* qui sont tous vides.

```
48      <FormInput
49        value={state.lastName}
50        onChange={handleChange}
51        label='Nom* :'
52        type='text'
53        name='lastName'
54        required={true}
55      />
```

Le composant *FormInput* est un composant du formulaire de création. Il prend la valeur du champ *lastname* du formulaire, à savoir un champ vide initialement. Sur l'événement *onChange*, j'exécute la méthode *handleChange* décrite ci-dessous.

```
17   const handleChange = (event) => {
18     const value = event.currentTarget.value;
19     const name = event.currentTarget.name;
20     setState({...state, [name] : value})
21   }
```

Je récupère la valeur ainsi que le nom du champ sur lequel l'événement se produit et je mets à jour la valeur du champ dans la constante *state*.

```
28     const handleSubmit = (event) => {
29         event.preventDefault();
30
31         const url = "http://localhost:8080/users";
32         axios.post(url, state)
33             .then(response => history.push('/'))
34             .catch(error => console.log(error));
35     };
```

Lorsque l'utilisateur valide la saisie du formulaire, nous retrouvons la méthode d'*axios* précédemment citée avec la constante *state* en paramètre, qui contient tous les champs du formulaire dûment complétés.

(3) La validation des données

Une partie de la validation des données s'effectue côté Front grâce à la propriété *required* qui prend la valeur *true* dans le *FormInput* (comme visible sur la ligne 54 de la capture d'écran précédente).

2. Approche TDD

a) *Réalisation de la partie Back-end*

(1) Test Driven Development

Le Test Driven Development est une démarche de développement piloté par les tests. Le principe est de produire un code qui suivra un plan fixé par les tests. Ceci permet de déterminer d'abord le besoin à développer en l'exprimant de façon claire, et ainsi de ne coder que ce qui est nécessaire à l'application.

La première étape est de déterminer le système à tester, ce qui se nomme en anglais le System Under Test (SUT). Ici, je cherche à tester la fonctionnalité de gestion des utilisateurs. J'ai donc un objet *User* qui contient des propriétés. La classe qui me permet de communiquer avec une application extérieure et d'effectuer des traitements sur mon utilisateur est le *controller*

UserManagementController. Ce controller doit permettre de gérer la visualisation, la création, la modification et la suppression d'un utilisateur.

Un utilisateur est défini par un *id*, un *firstName*, un *lastName*, un *username*, un *email* et un *password*. En dehors de l'*id*, toutes ces propriétés sont des chaînes de caractères qui ne doivent être ni *null*, ni vides, ni seulement remplies d'espaces.

Le controller doit remplir les responsabilités qui ont été citées dans la partie V. A. ► La méthode « *createUser()* » b) de ce même dossier.

De cette réflexion résultent le jeu d'essai ainsi que la fiche de tests qui ont permis de rédiger la classe de tests *UserManagementControllerTest*.

Une fois celle-ci codée, je me suis attelée à coder le controller et ses méthodes *getUsers*, *getUser*, *create*, *update* et *delete* dont nous voyons un extrait ci-dessous.

```
25 @RestController
26 @RequestMapping("/users")
27 public class UserManagementController
28 {
29     @Autowired
30     private UserService userService;
```

```
49     @PostMapping("")
50     @ResponseStatus(HttpStatus.CREATED)
51     public void create(
52         @Valid @RequestBody UserCreateDto userCreateDto
53     ) {
54         userService.create(userCreateDto);
55     }
```

L'annotation Spring **@RequestMapping** sur le controller permet de préciser le chemin de base de la requête vers le controller. Les annotations **@GetMapping**, **@PostMapping**, **@PutMapping**, **@DeleteMapping**, etc, sur les routes, en précisent le chemin ainsi que le verbe HTTP.

Grâce à l'annotation **@RestController**, qui comprend les annotations *@Controller* et *@ResponseBody*, chaque route encode automatiquement en JSON les objets renvoyés dans la réponse HTTP.

Ici, la méthode *create* reçoit en paramètre un objet *UserCreateDto* qu'elle transmet à la méthode *create()* du *UserService* afin d'ajouter un nouvel utilisateur dans la base de données. L'annotation **@RequestBody** transforme automatiquement le *userCreateDto* fourni au format JSON dans le corps de la requête en objet Java *UserCreateDto* afin de pouvoir le transmettre au *UserService* qui appellera les méthodes du *UserRepository*. L'annotation **@Valid** vérifie la validité de l'objet donné en paramètre en fonction de ce qui a été défini au niveau de sa classe.

J'ai donc poursuivi le développement en codant les *Datas Transfer Objects (DTO)*, à savoir *UserCreateDto* et *UserUpdateDto*, ainsi que l'objet *User*.

```
10 @Data
11 @Component
12 public class UserCreateDto {
13
14     @NotBlank
15     @Size(min = 1, max = 30)
16     private String username;
17
18     @NotBlank
19     private String firstName;
20
21     @NotBlank
22     private String lastName;
23
24     @NotBlank
25     private String email;
26
27     @NotBlank
28     private String password;
29
30 }
```

Toutes les propriétés du *UserCreateDto* sont annotées **@NotBlank**, ce qui indique que l'objet sera considéré comme valide uniquement si aucune des propriétés n'a la valeur *null*, et n'est une chaîne de caractères vide ou seulement constituée d'espaces.

b) Création de la base de données

```
13 @Data
14 @Entity
15 @Table (name = "users")
16 public class User
17 {
18     @Id
19     @GeneratedValue(strategy = GenerationType.IDENTITY)
20     @Column(name = "id")
21     private Long id;
22
23     @Column(unique = true)
24     @NotBlank
25     private String username;
26
27     @Column
28     @NotBlank
29     private String firstName;
30
31     @Column
32     @NotBlank
33     private String lastName;
34
35     @Column
36     @NotBlank
37     private String email;
38
39     @Column
40     @NotBlank
41     private String password;
42 }
```

Spring Data JPA est un module de Spring qui fournit un meilleur support d'utilisation de la **Java Persistence API**. La **JPA** est utilisée pour organiser l'accès aux données. L'annotation **@Entity** spécifie que la classe *User* est une entité persistante. L'annotation **@Table** permet de définir le nom de la table dans laquelle les instances de cette classe seront écrites. L'annotation **@Column**, comme son nom l'indique, précise que l'instance de cette classe est une colonne de la table. Les annotations **@Id** et **@GeneratedValue** indiquent que la propriété *id* représente l'*id* dans la base de données et qu'il est auto-généré.

```
11 # =====
12 # JPA / HIBERNATE
13 # =====
14 # Makes Hibernate generate better SQL for the chosen database
15 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
16 spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation = true
17
18 # Hibernate ddl auto (create, create-drop, validate, update)
19 spring.jpa.hibernate.ddl-auto = update
20 spring.jpa.open-in-view = false
21 spring.jpa.show-sql = false
```

Je configure, dans le fichier *application.properties* du sous-répertoire *ressources*, la création automatique des tables grâce à **Hibernate**.

c) *Développement des composants d'accès aux données*

```
1 # =====
2 # DATABASE => MySQL
3 # =====
4 spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
5 spring.datasource.url = jdbc:mysql://localhost:3306/wd_tdd
6 spring.datasource.username = root
7 spring.datasource.password = root
8 spring.datasource.initialization-mode = always
9 spring.datasource.sql-script-encoding = UTF-8
```

Egalement dans le fichier *application.properties*, je configure la connexion à la base de données.

```
12 @Repository
13 public interface UserRepository extends JpaRepository<User, Long> {
```

L'interface *UserRepository* est chargée de communiquer avec la base de données et d'effectuer les requêtes SQL.

```
@Query(value = "INSERT INTO users (username, first_name, last_name, email, `password`) VALUES (?, ?, ?, ?, ?)", nativeQuery = true)
@Modifying
int create(
    String username,
    String firstName,
    String lastName,
    String email,
    String password
);
```

La méthode *create* ci-dessus prend en paramètres d'entrée le *username*, le *firstName*, le *lastName*, l'*email* ainsi que le *password* de l'utilisateur et retourne un numérique qui correspond au nombre de lignes qui ont été impactées par la requête SQL. Elle est annotée **@Query**, afin de la définir comme requête SQL effectuée sur la base de données, ainsi que **@Modifying** pour préciser qu'il s'agit d'une requête qui va impacter la base de données. Cette dernière annotation est importante car le traitement de la requête n'est pas le même selon qu'il s'agisse d'une requête de lecture (*SELECT*) ou d'une requête de modification (*UPDATE*), création (*INSERT*), suppression (*DELETE*) ou encore définition (*CREATE TABLE*) des données.

VI. Veille sur les vulnérabilités de sécurité

Avec l'avènement d'internet et la démultiplication des applications web, le nombre de cyber-criminels a crû proportionnellement à la quantité de données personnelles et/ou professionnelles détenues par ces applications exposées au public. Mots de passe, adresses email, numéros de cartes bancaires, données santé, sont autant d'informations sensibles à protéger de l'avidité de personnes malveillantes. Les applications web, par leur publicité, sont naturellement susceptibles de porter des failles et, par conséquent, sont les cibles d'attaques variées. Il convient de les prémunir du maximum d'attaques possibles.

- Mots-clefs : *failles sécurité web*
- Site sélectionné : <https://www.vaadata.com/blog/fr/comment-renforcer-la-securite-de-vos-applications-web-pour-contrer-les-attaques-les-plus-courantes/>

A. Mauvaise gestion de l'authentification et des droits d'accès

1. L'authentification

- Mots-clefs : *password security*
- Site sélectionné : <https://www.mylogin.com/resources/password-strength-test/>

Le processus d'authentification permet de limiter l'accès d'un site à des utilisateurs enregistrés dans le site. Il nécessite d'entrer des données dans un formulaire de connexion. De façon générale, ces données de connexion sont un login et un mot de passe.

Lors de la création du mot de passe, un Password Meter permet d'évaluer la force du mot de passe en informant l'utilisateur par un code couleur. Plus la longueur et la complexité du mot de passe sont grandes, plus une attaque par force brute sera longue à découvrir celui-ci.

Une fois le mot de passe créé et avant son ajout en base de données, il doit être Hashé, ceci afin d'éviter le stockage du mot de passe en clair. Ce processus est permis grâce à une fonction particulière qui, à partir d'une donnée fournie en entrée, calcule une empreinte numérique servant à identifier rapidement la donnée initiale.

2. Les droits d'accès

- Mots-clefs : *spring access security*
- Site sélectionné : <https://www.baeldung.com/spring-security-acl>

Spring Security Access Control List est un composant de **Spring** qui permet de lier une liste de permissions à un objet. *Spring ACL* spécifie également quelle identité peut opérer quelle action sur l'objet spécifié.

B. Injections SQL

- Mots-clefs : *spring jpa sql injection*
- Site sélectionné : <https://www.baeldung.com/sql-injection>

Les injections SQL consistent à modifier une requête SQL en injectant des morceaux de code non filtrés. Cette opération s'effectue généralement par le biais d'un formulaire. Pour prévenir toute injection il est important de ne jamais faire confiance aux données fournies par les utilisateurs.

Un bon moyen de prévenir ces injections est l'emploi de requêtes paramétrées comme ci-dessous.

```
@Query(
    value = "UPDATE users "
        + "SET username = ?2, "
        + "first_name = ?3, "
        + "last_name = ?4, "
        + "email = ?5, "
        + "`password` = ?6 "
        + "WHERE id = ?1",
    nativeQuery = true
)
```

C. Cross-Site Scripting (XSS)

- Mots-clefs : *XSS react*
- Site sélectionné : <https://dev.to/spukas/preventing-xss-in-react-applications-5f5j>

Cette attaque a pour but d'injecter du code dans les pages en le rendant exécutable par le navigateur. Pour l'éviter, deux solutions sont à mettre en œuvre : filtrer les entrées et échapper les données en sortie.

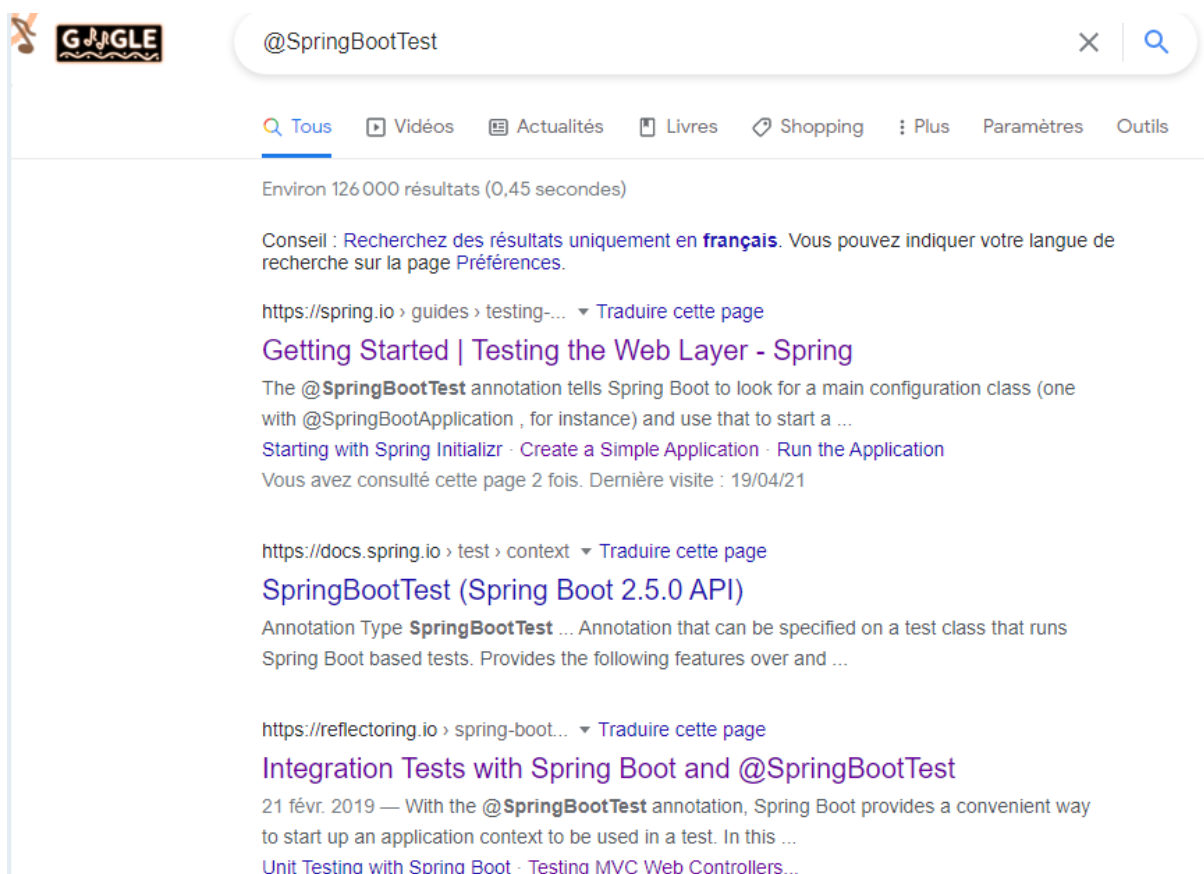
React prévient naturellement les injections de script en échappant tout ce qui n'est pas explicitement écrit dans l'application.

VII. Recherche à partir d'un site anglophone

Initier la mise en place de tests automatiques au sein du « site Admin » nécessitait pour moi l'acquisition des bases d'implémentation de tests avec **Spring Boot**. Après avoir déterminé que je souhaitais mettre en place des tests d'intégration sur le controller de gestion des utilisateurs, j'ai consacré une recherche sur ce sujet.

J'ai choisi d'utiliser le moteur de recherche **Google**. J'y ai entré le mot-clef **@SpringBootTest** car je savais, par le biais d'une recherche précédente, que cette annotation définissait une classe de tests dans l'application. Je cherchais à me renseigner sur la méthodologie des tests d'intégration, appliqués sur le controller d'une application web, et leur utilité au sein d'un développement.

Les pages web suivantes m'ont été proposées par **Google** :



J'ai d'abord ouvert la première page proposée, à savoir le site <https://spring.io> car il s'agit de la documentation officielle de Spring et, qui plus est, d'un guide d'utilisation de Spring. Ce site m'a aidée à mieux comprendre le fonctionnement des tests au sein d'une application développée avec Spring, mais ne répondait pas à ma recherche initiale, à savoir l'utilité des tests d'intégration sur le controller d'une application et la stratégie de mise en place de ces tests.

Integration Tests with Spring Boot and @SpringBootTest

21 févr. 2019 — With the `@SpringBootTest` annotation, Spring Boot provides a convenient way to start up an application context to be used in a test. In this ...

Unit Testing with Spring Boot · Testing MVC Web Controllers...

J'ai finalement opté pour la troisième page proposée car il s'agissait également d'un tutorial, et les liens sous la description semblaient indiquer que l'article approfondirait le sujet des tests sur un controller. J'ai donc visité le site <https://reflectoring.io>. L'article introduisait une série d'articles intitulée « The "Testing with SpringBoot" Series » et décrivait en effet les différences entre tests unitaires et tests d'intégration. Il indiquait quel type de tests préférer en fonction de ce que l'on souhaite tester, comment les mettre en place avec SpringBoot, etc, et pointait sur un article de la série, spécifiquement consacré aux tests d'intégration sur les controllers d'une application Spring MVC. Ce qui correspondait parfaitement à ce que je souhaitais mettre en place.

This tutorial is part of a series:

1. Unit Testing with Spring Boot
2. Testing Spring MVC Web Controllers with Spring Boot and `@WebMvcTest`
3. Testing JPA Queries with Spring Boot and `@DataJpaTest`
4. Integration Tests with `@SpringBootTest`

Je me suis donc dirigée sur l'article en question, deuxième dans la liste ci-dessus, qui a pour URL : <https://reflectoring.io/spring-boot-web-controller-test/>

Celui-ci décrivait en détails les responsabilités d'un controller REST au sein d'une application web et expliquait comment effectuer des tests qui couvraient toutes ces responsabilités. Je me suis appuyée sur cet article pour construire les tests du controller *userManagerController* au sein du « site Admin ».

VIII. Traduction du site anglophone sélectionné durant la recherche

A. Extrait du site anglophone	B. Traduction
<p style="text-align: center;">Unit or Integration Test?</p> <p>Do we write unit tests? Or integration tests? What's the difference, anyways? Let's discuss both approaches and decide for one.</p> <p style="text-align: center;">[...]</p> <p>In summary, a simple unit test will not cover the HTTP layer. So, we need to introduce Spring to our test to do the HTTP magic for us. Thus, we're building an integration test that tests the integration between our controller code and the components Spring provides for HTTP support.</p> <p>An integration test with Spring fires up a Spring application context that contains all the beans we need. This includes framework beans that are responsible for listening to certain URLs, serializing and deserializing to and from JSON and translating exceptions to HTTP. These beans will evaluate the annotations that would be ignored by a simple unit test.</p>	<p style="text-align: center;">Test unitaire ou test d'intégration ?</p> <p>Devons-nous rédiger des tests unitaires ou des tests d'intégration ? D'ailleurs, quelle est la différence entre les deux ? Nous allons les étudier et décider de ceux qu'il convient d'écrire.</p> <p style="text-align: center;">[...]</p> <p>En résumé, un simple test unitaire ne couvrira pas l'ensemble de la couche HTTP. Il nous faut donc introduire Spring à notre test afin qu'il opère la magie HTTP pour nous. Ainsi, nous construisons un test d'intégration qui vérifie l'interaction entre le code de notre controller et les composants que Spring fournit pour le support HTTP.</p> <p>Un test d'intégration avec Spring lance un contexte d'application Spring qui contient tous les composants logiciels dont nous avons besoin. Cela inclut des composants logiciels Spring qui s'occupent de certaines routes, et qui sont en charge de transformer un objet en JSON ou inversement, ainsi que de traduire les exceptions en HTTP. Ces composants logiciels prennent en compte les annotations qui seraient ignorées par un simple test unitaire.</p>

Remerciements

Je tiens à remercier chaleureusement l'entreprise ITSO qui m'a accueillie pour réaliser mon stage. Merci à Arnaud Fichot de m'avoir acceptée comme stagiaire et d'avoir veillé à mon intégration dans l'entreprise durant ces sept semaines. Merci à Fouad Bensmail, mon tuteur de stage, pour tout le temps et l'aide qu'il m'a apportés. Merci à tous les employés d'ITSO pour leur bienveillance et leur bonne humeur.

Je remercie également tous les formateurs de l'Ecole Pratique qui ont fait leur possible pour nous transmettre les connaissances nécessaires au métier convoité de développeur web.

A1

idRole, idCompanyTyped, idPreferredLanguage		
	Classes d'équivalence	Représentant
C1	Numérique positif	2,4
C2	0	0
C3	Null	null
C4	Numérique négatif	-1

username, firstname, lastname, email		
	Classes d'équivalence	Représentant
C5	String longueur [1 , 150]	"a.dougedroit@itso.fr", "TEST"
C6	String longueur < 1	""
C7	String longueur > 150	"abcdefghijklmnopqrstuvwxyz bcdefghijklmnopqrstuvwxyzabc defghijklmnopqrstuvwxyzabcde fghijklmnopqrstuvwxyzabcdefg hijklmnopqrstuvwxyzabcdefghi jklmnopqrstu"
C8	String seulement remplie d'espaces	" "
C9	Null	null

Tous paramètres confondus	
	Données de test
C1, C5	2, "a.dougedroit@itso.fr"
C2, C5	0, "TEST"
C3, C5	null, "TEST"
C4, C5	-1, "TEST"
C1, C6	4, ""
C1, C7	2, "abcdefghijklmnopqrstuvwxyza bcdefghijklmnopqrstuvwxyzabc defghijklmnopqrstuvwxyzabcde fghijklmnopqrstuvwxyzabcdefg hijklmnopqrstuvwxyzabcdefghi jklmnopqrstu"
C1, C8	2, " "
C1, C9	4, null