GRETA Suivi Séance

Mathias Bondoux

Dossier de Projet

Titre professionnel de Concepteur Développeur d’Application

Table des matières

[I. English summary 2](#_Toc105766585)

[II. Liste de compétences couvertes par le projet 3](#_Toc105766586)

[III. Expression des besoins 4](#_Toc105766587)

[IV. Gestion du projet 5](#_Toc105766588)

[V. Spécifications fonctionnelles 6](#_Toc105766589)

[VI. Spécifications techniques 7](#_Toc105766590)

[VII. Réalisation 9](#_Toc105766591)

[VIII. Jeu d’essais 41](#_Toc105766592)

[IX. Veille technologique 42](#_Toc105766593)

[X. Recherche sur une situation de travail 44](#_Toc105766594)

# English summary

The GRETA Tracking Session was made for our training organization to help them out with some administrative work. The application will be used by the coordinator and the trainers themselves, they will replace a manual task by completing a tracking sheet of each lessons of a session online. I have worked with two classmates, and we have decided to design a project each one, so we have worked alongside during all the conception and development of the project, but we did mostly everything in our respective project.

We started to talk about the conception with our clients also known as our coordinator and trainers on what they wanted for the application. I initially proceeded to make the basic conception with some wireframe, use case diagram and the user story. I managed to design the sketch and the database conception then I started to work on the code, and I did many refactoring and came back on the conception to match with the client changes in the product. I worked on this project at home with my computer, I used IntelliJ for the development and the conception with the PlantUML plug-in, Figma for the design conception, GitHub and GitLab to work with my classmates. I’ve made the application using JAVA with the IDE IntelliJ, I used the framework SpringBoot and the template engine/handler Thymeleaf. The difficulties I encountered during the development of this project was that I must learn and understand very quickly some technical terms, how a web application using an MVC architecture works and learn one of my first programming language. Hopefully I learnt that without online research I would have been stuck in the progress of making my application because I was stuck on the validation and my classmates didn’t get this issue.

My project isn’t finished due to the overwhelming and late changes that I couldn’t do at these times due to the deadline of the end of the traineeship. The project is functional and meets the basic requirement but at this stage a lot of time-saving and ergonomic technologies are missing to make the application more efficient than the actual written system, like for example the import/export it will build up more easily and quickly the overall data rather than typing them individually. The project does not have all the latest change to comply with our client needs due to time limitation quite short.

# Liste de compétences couvertes par le projet

CCP1 :

* Maquetter une application
* Développer des composants d’accès aux données
* Développer la partie front-end d’une interface utilisateur web
* Développer la partie back-end d’une interface utilisateur web

CCP2 :

* *Concevoir une base de données*
* Mettre en place une base de données

CCP3 :

* *Collaborer à la gestion d’un projet informatique et à l’organisation de l’environnement de développement*
* Concevoir une application
* Développer des composants métier
* Construire une application organisée en couches
* *Préparer et exécuter les plans de test d’une application*

# Expression des besoins

1. Les objectifs du projet

Dans le cadre de notre projet de fin d’année, deux camarades et moi avons choisi de concevoir une application pour le GRETA pour simplifier le renseignement de fiches de suivi des cours au sein de l’établissement.

L’application pourra permettre à l’utilisateur de se connecter en tant que formateur ou coordinateur et pourra effectuer des actions liées à leurs rôles respectifs.

Le formateur pourra remplir une fiche de suivi tandis que le coordinateur pourra modifier et supprimer les fiches, les centres, les formations et les sessions du GRETA.

1. Notre équipe

Mon groupe de travail avec qui j’ai conçu et développé l’application, est composé de deux autres stagiaires de la formation de Concepteur Développeur d’Applications :

* JEANDENANS Alexis
* GAGNANT Florian

1. Analyse de l’existant

Pour le moment le Greta n’utilise qu’un format papier qui est ensuite retranscrit sur des fichier Excel.

1. Les cibles

Les formateurs, les coordinateurs ainsi que l’administration des GRETA qui pourront se servir de cette application afin de rendre leur travail moins rébarbatif.

1. La portée du projet

Cette application permettra de supprimer une tâche manuelle redondante et de la simplifier informatiquement au sein de notre centre de formation mais cette application pourra être adaptée à une plus grande échelle.

# Gestion du projet

Nous avons, mes camarades et moi, définis les critères du projet avec notre client majeur : notre coordinateur. Nous avons donc commencé la conception commune de notre projet. Nous avons travaillé en utilisant la méthode Agile c’est-à-dire que nous sommes revenus énormément vers notre client à certaines étapes clefs du développement du projet.

La phase d’analyse du domaine métier et fonctionnel étant faite nous pouvons donc émettre de ces dernières les acteurs interagissant avec notre application, quant à nos fonctionnalités elles seront créées à partir de nos acteurs. Cela a déterminé nos classes métier que nous avons mises en place et les données à intégrer et sauvegarder dans notre base de données. Nous avons fait quelques wireframes et maquette validés pas notre client.

Puis nous avons commencé le développement, dans un premier temps nous avons commencé par analyser et choisir les technologies que nous allions utiliser pour notre projet. Nous avons donc sélectionné les technologies qui nous paraissaient les plus adéquates à notre projet. Dans un second temps nous avons commencé le développement du code en commençant par effectuer l’écriture du code par étape clef, par exemple écrire nos classes entités avant nos repositories, aussi banal que cela puisse paraitre. C’est aussi cette partie qui a nécessité le plus de recherche pour le fonctionnement de Spring JPA, Thymeleaf et les validateurs, pour ne citer que ces exemples. Notre source majeure de réponse à nos questions a été Baeldung.

Mon groupe et moi avons pris la décision que chacun crée sa propre application, nous avons développé ensemble mais différemment ce qui donne une structure similaire à trois applications crées mais avec des approches différentes aux problème rencontrés. Grâce à nos branches respectives sur GitLab et Discord nous avons pu travaillez en commun tout au long de notre stage.

# Spécifications fonctionnelles

## Use Case

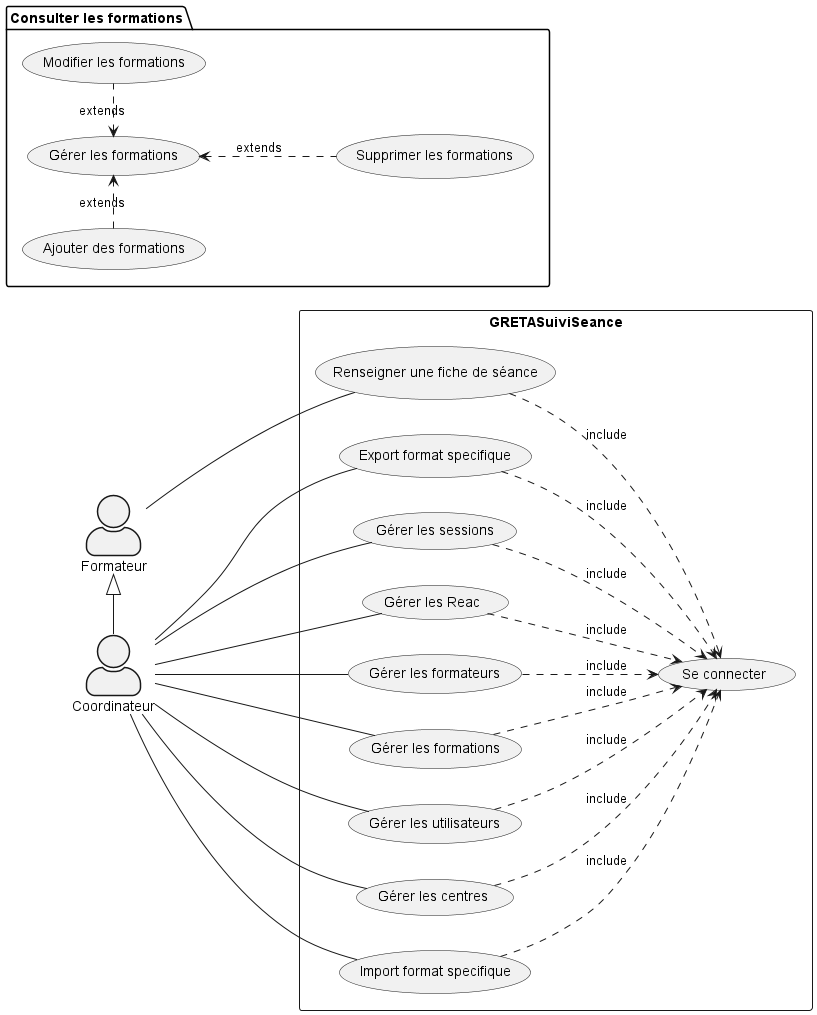


Figure 1 : Diagramme Use Case général

## User Story

User story des différents acteurs :

* **Le formateur** : est un utilisateur connecté qui peut remplir des fiches de suivi de ses cours.
* **Le coordinateur** : est un utilisateur connecté qui peut gérer l’application dans sa globalité, ajouter, modifier, supprimer ce qu’il souhaite dans l’application; cela comprend aussi les utilisateurs.

# Spécifications techniques

## Langages, Framework et SGBD

|  |  |  |
| --- | --- | --- |
|  | HTML/CSS | Langages de balise et de style qui nous a servi pour le développement front-end. |
|  | Bootstrap | Le framework CSS qui m’a permis de styliser mes vues front-end plus efficacement que de passer par une feuille de style CSS. |
|  | Java | Langage de programmation sur lequel le projet est basé car c’est un langage de programmation orienté objet. |
|  | SpringBoot | Le famework du langage Java qui nous a permis d’utiliser et d’intégrer la structure MVC et Spring Security via les dépendances. |
|  | Thymeleaf | Le moteur de templates Thymeleaf fonctionnant avec Springboot permettant de gérer les templates et la logique de présentation. |
|  | Tomcat | Le serveur built-in dans IntelliJ pour lancer l’application web en face de développement. |
|  | H2 | Le SGBD SQL que nous avons décidé d’utiliser pour notre application. |
|  | PlantUML | Ce plug-in d’IntelliJ m’a permis de faire la conception de notre application via du code qui formatait nos diagrammes.  De la même manière que les fichiers markdown sont interprétés. |

## Environnement et Logiciel de développement

|  |  |  |
| --- | --- | --- |
|  | IntelliJ IDEA | L’environnement de développement pour le langage Java sur lequel nous avons développé notre projet. |
|  | GitLab | Mes camarades et moi avons utilisé GitLab pour travailler en parallèle sur des branches différentes. |
|  | GitHub | J’ai utilisé GitHub en plus de GitLab car nous avons créé des projets qui sur le terme, n’était pas possible de merge en fin développement. |
|  | Discord | Mes camarades et moi avons utilisé Discord afin de travailler sur notre projet, s’aider durant le développement et se donner des tâches à accomplir.  Nous l’avons utilisé comme un Trello |
|  | Figma | Figma a été notre application sur laquelle nous avons pu établir et réaliser nos maquettages |

# Réalisation

## Analyse du domaine métier

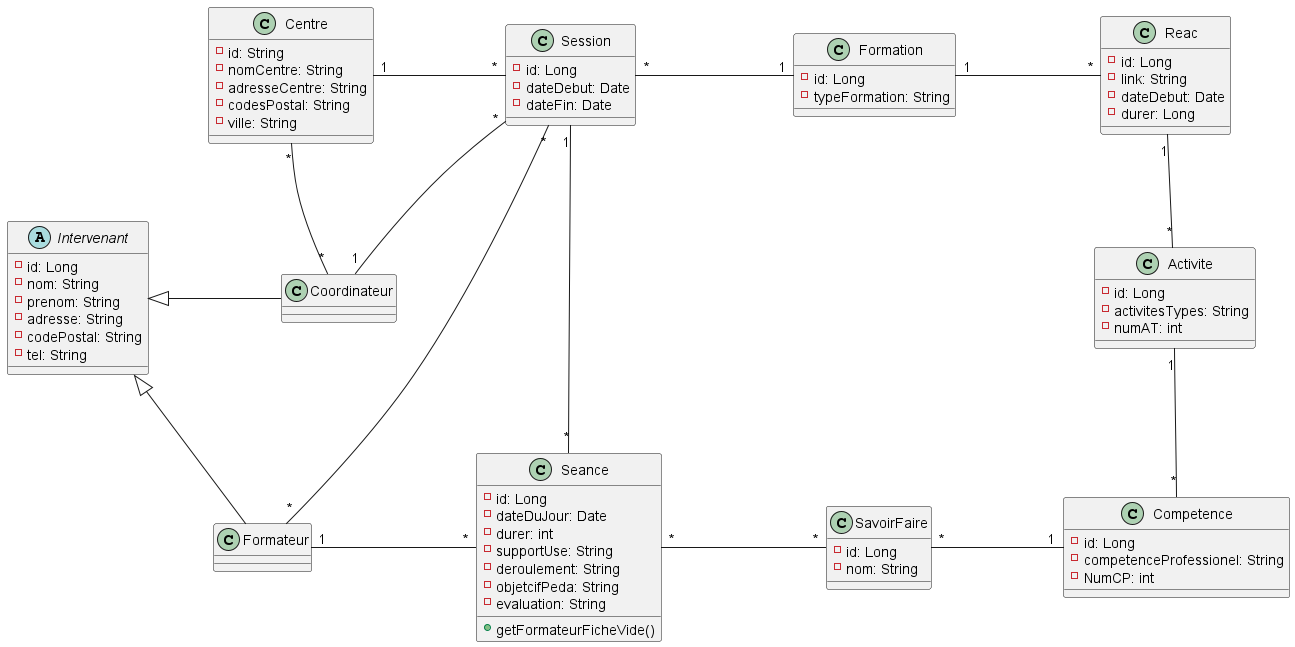


Figure 2 : Diagramme de classes

Un coordinateur et un formateur sont des intervenants qui possède une session de cours au sein du Greta.

Les coordinateurs peuvent avoir plusieurs centres. Les formateurs on des séances de cours qui compose des sessions de formation. Les formations sont régies par des référentiel (REAC) qui comporte des activités qui elle-même sont composées de plusieurs compétences qui comporte des savoir faires qui sont les points à abordé d’une séance.

## Maquettage

Depuis les demandes données par notre client principale pendant l’analyse fonctionnelle et métier, mes camarades et moi avons pu mettre en place quelque wireframe qui nous ont permis de faire le zonage pour visualiser à peu près quel et comment les éléments importants vont rendre lors de la production des vues. Grâce à Figma et le respect des règles de wireframe nous pouvons distinguer tous les éléments, leur types et places qu’ils prendront.

Ci-dessous un dashboard dans le cas présent celui des compétences, une page qui va être récurrente dans notre application web.

Une image contenant table

Description générée automatiquement

Figure 3 : Wireframe de la conception front-end

Ci-dessous nous voyons le wireframe qui a ensuite été maquetté qui est donc une version plus aboutie de la page et qui se rapproche le plus de la version finale que la page aura en production.

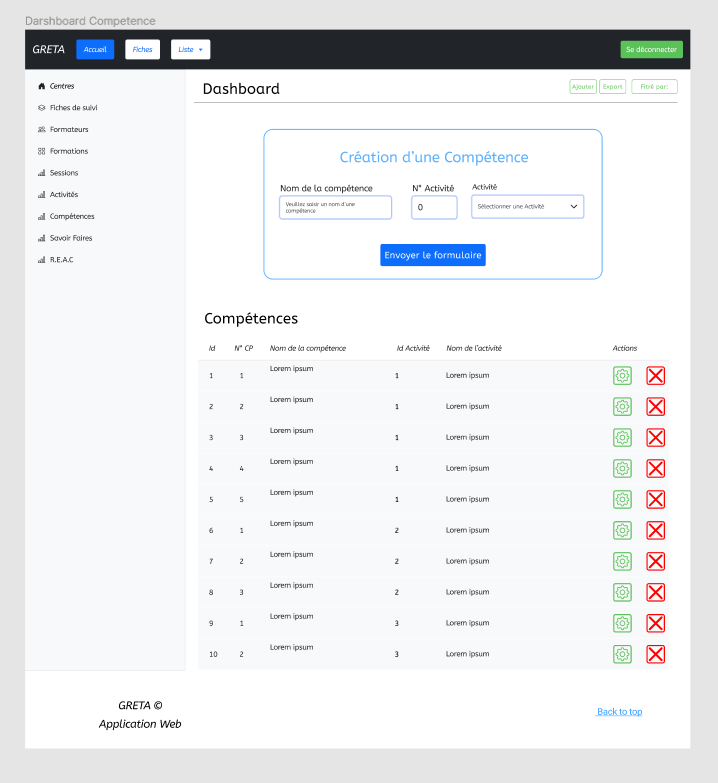


Figure 4 : Maquettage du wireframe

## Organisation en couche

Notre application est organisée en plusieurs couches :

* Une couche modèle : qui structure nos données indépendamment de l’interface utilisateur. Cette couche gère directement les données, la logique et les règles de notre application.
* Une couche DAO ou Repository : qui permet d’interagir avec la base de données via des interfaces.
* Une couche contrôleur : qui « écoute » l’utilisateur et lui « répond ». Le contrôleur commande la vue et la couche de transfert de données.
* La couche de transfert de données : permet de faire tampon entre les controller et le reste de l’application en recevant les données transmises pour les traitées correctement sans passé par l’entité directement
* Une couche vue : Qui utilise un format pour présenter les informations du modèle.
* Une couche moteur de template : qui est responsable seulement de l’interprétation de langages back-end en front-end pour rendre les vues dynamiques.
* Une couche service : qui est une couche intermédiaire (facultative) entre la couche modèle et la couche contrôleur : elle va chercher les données de la couche modèle et laisse le contrôleur utiliser ces données.

## La couche modèle

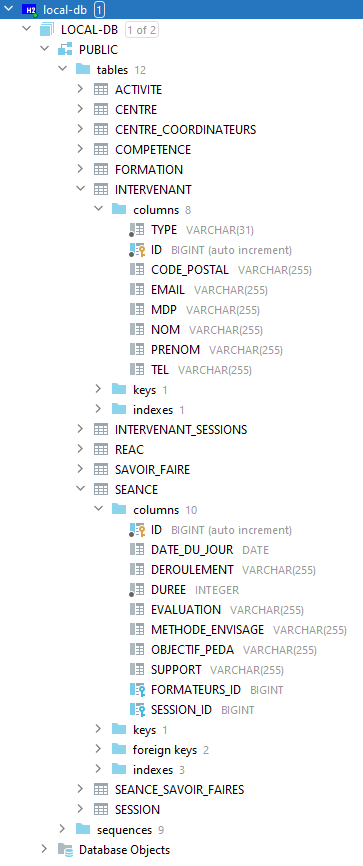
Les classes modèles sont importante dans la mise en place d’une base de données relationnelle comme la mienne.

Figure  : Arborescence de la base de données

Elle permette de créer les tables avec leurs colonnes respectives avec leurs relations avec d’autres tables et ce via les annotations de JPA.

Hibernate l’ORM (Object-Relational Mapping) va ensuite interpréter ces annotations pour créer les relations entre les tables et créer dans la structure de cette dernière. Lors de chaque compilation que j’effectue au cours du développement nous voyons Hibernate effectuer les requêtes SQL pour créer notre base de données à partir de mes annotations et importe des données que je lui ai fournis dans un fichier prévu à cet effet nommé import.sql et effectuera les requêtes données dedans.

Une image contenant table

Description générée automatiquement

Figure  : Exemple de requêtes SQL pour créer la BDD

J’ai respecté d’utiliser des modificateurs de visibilité adapté telle que mes attributs soient en « Private » et que j’interagis avec eux via des accesseurs « Getters » et mutateurs « Setters ».

Je passe par mes DTO pour éviter des injections HTML qui seront bloqué par mes validateurs lors de l’interprétation de mes méthodes dans mes services.

La classe Intervenant comporte mes annotations de JPA et Lombok. JPA nous permet via @Entity de créer et d’assigner cette classe en tant qu’entité il lui faut donc un attribut qui lui sert de clef primaire via @Id et @GeneratedValue. Lombok quant à lui me permet d’éviter d’écrire du code redondant en mettant @Getter et @Setter lors de la compilation il interprétera cette annotation pour créer nos getter et setter. J’ai aussi utilisé un type d’héritage dans ce cas présent, comme je n’ai aucun attribut différent pour nos coordinateur et formateurs j’ai décidé d’afficher une table dans la base de données qui différenciera ces deux entités par une colonne qui indiquera la classe enfant à laquelle chacun appartient.

Une image contenant texte

Description générée automatiquement

Figure 7 : La classe Intervenant

J’ai indiqué les relations entre les autres classes avec les annotations JPA dans les classes enfants ci-dessous. J’ai dû utiliser des paramètres à mes annotations de relations « mappedby » pour qu’il puisse se lier et indiquer les relations fortes.

Une image contenant texte

Description générée automatiquement

Figure 8 : La classe Coordinateur

Une image contenant texte

Description générée automatiquement

Figure 7 : La classe Formateur

Une image contenant texte

Description générée automatiquement

Figure 9 : Classe Séance 1

Une image contenant texte

Description générée automatiquement

Figure 10 : Classe Séance 2

## Propriétés de l’application

Pour configurer l’accès à ma base de données j’ai dû indiquer dans un fichier de configurations nommé application.properties, j’ai préféré utiliser un fichier secondaire comportant mes configurations de BDD et d’y indiquer le chemin dans le fichier principale ce faisant je décentralise mes données sensible de mon application. Dans le fichier nommé db.properties j’y indique le chemin de ma base de donnée pour la stocker en dure sur mon système ou un serveur en cas de déploiement. J’y renseigne également mes identifiants de connexion et si j’utilise ou non l’interface du SGBD, H2 dans mon cas.

Une image contenant texte

Description générée automatiquement

Figure 11 : Propriétés de l’application

Une image contenant texte

Description générée automatiquement

Figure 12 : Propriétés de la base de données

## La couche d’accès aux données

La couche d’accès aux données m’a servi à interagir avec ma base de données dans mon application par mes interfaces qui héritent de JPARepository qui hérite elle-même de CRUDRepository et me donne accès aux méthodes interagissant avec ma BDD.

Je dois préciser mes repository avec l’annotation @Repository.

Une image contenant texte

Description générée automatiquement

Figure 13 : Repository de Séance

Une image contenant texte

Description générée automatiquement

Figure 14 : Repository d’un Intervenant

## La couche contrôler

J’ai utilisé des classes contrôleurs, certaine générale et d’autre orienté par les CRUD de mes autres classes.

Chaque classe contrôleur a une annotation @Controller puis dans chaque méthode de mes vues celle de @GetMapping précisant en paramètre les URI qui mappera la méthode pour des requête http de type « get ».

J’ai donc comme contrôleur :



Et mes contrôleurs de CRUD :

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Figure 15 : Contrôleur de l’index principale

Ci-dessous le contrôleur de mon dashboard de mes fiches de suivis pour ses CRUD.

Dans mes contrôleurs de mes CRUD j’ai mis une annotation @RequestMapping qui indique que toutes les méthodes présentent dans cette classe devraient être précéder de l’URI indiquée en paramètre en plus du paramètre dans le @GetMapping. Par exemple figure 16 l’URI de la méthodes « read » des fiches ressembleras à ceci : <http://localhost:8080/formateur/dashboard/fiches>

Une image contenant texte

Description générée automatiquement

Figure 16 : Contrôleur des CRUD des Fiches 1

Une image contenant texte

Description générée automatiquement

Figure 17 : Contrôleur des CRUD des Fiches 2

Voici le contrôleur pour mes requêtes http de type post que les formulaires de modifications et de suppressions utilisent. Passer par des méthodes post est obligatoire lors d’envoi de formulaire.

Une image contenant texte

Description générée automatiquement

Figure 18 : Contrôleur des CRUD des Fiches 3

Ci-dessous ce trouve le contrôleur permettant de faire la pagination de notre dashboard des fiche suivis étant donné qu’il est sensé y avoir beaucoup de données.

Une image contenant texte

Description générée automatiquement

Figure 19 : Contrôleur des CRUD des Fiches 4

Voici mon contrôleur du dashboard de mes compétences imbriquant ma méthode « get » de modification et de « read » a la même URI donc sur la même vue qui est gérer via des exceptions.

Une image contenant texte

Description générée automatiquement

Figure 20 : Contrôleur des CRUD des Compétences

Une image contenant texte

Description générée automatiquement

Figure  : Méthode Save/Update et Delete du contrôleur de Compétence

Une image contenant texte

Description générée automatiquement

Figure  : Contrôleur de la méthode post de l’import de fichier Excel

Le contrôleur ci-dessus me permet d’importer un fichier Excel au sein de mon application web et d’être traité via la méthode post et de renvoyer un message de succès d’import ou d’erreur si aucun fichier n’est reçu.

## La couche de transfert de données

Voici les DTO (Data Transfer Object) de mes classes modèle, je m’en sers comme tampon pour ne pas toucher à mes classes modèle. Les contrôleurs, lorsqu’ils interceptent un formulaire qui est créé à partir de mon DTO suit les validateurs que j’ai imposé dedans au même titre que le format des données que je lui demande. Ensuite le formulaire créé via le DTO sera traité par le contrôleur qui effectuera les méthodes qu’ils lui sont données, tel que l’ajout ou la modification.

*Une image contenant texte

Description générée automatiquement*

Figure 23 : DTO de Séance

Dans l’exemple suivant j’ai donc utilisé des annotations pour utiliser des validateurs qui vérifierons si la donnée perçue correspond bien à celle qui est attendue. Mais aussi qu’un champ ne soit pas rempli que d’espace blanc dans un champs de type « String » c’est-à-dire contenant une chaine de caractère. Nous pouvons aussi réguler la taille minimum et maximum de ces champs d’entré.

Une image contenant texte

Description générée automatiquement

Figure 24 : DTO de Compétence

Une image contenant texte

Description générée automatiquement

Figure  : DTO d’un Intervenant

## La couche service

Les classes qui me servent de services fonctionnent avec l’annotation @Service et me permettent de mettre toutes mes méthodes CRUD à l’intérieur pour centraliser les méthodes et faire appelle uniquement aux services correspondant dans mes contrôleurs.

Ma couche service est composée de plusieurs classe qui catégorisent mes services.

Je possède mon service de JPAUserService qui me permet de charger un utilisateur avec ses identifiants. (Figure 30)

J’ai aussi mon service de mes Intervenant qui me permet de créer de nouveaux intervenant et d’encrypter leur mot de passe avec Bcrypt.

Et le reste de mes services me servent pour mes méthodes CRUD regroupé en trois grandes catégories.

Les deux premières classes fonctionnent grâce aux framework de Spring Security. J’ai donc suivi Baeldung et la doc officielle pour mettre en place ces dispositifs.

Les méthodes de mes services sont à utiliser au sein de leurs contrôleurs respectifs

J’ai « autowired » tous mes dépôts avec un constructeur cela évite une répétition d’annotation pour chaque dépôt.

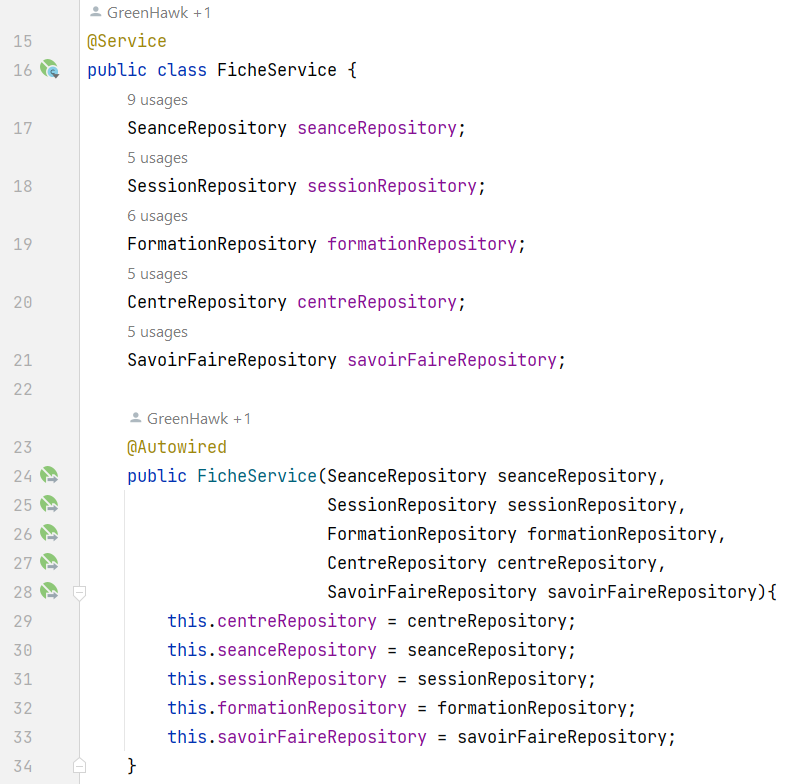


Figure 26 : Service des Fiches

Voici ci-dessous mes méthode CRUD pour mes fiches de suivi.

getAllFiches() : me permet d’avoir la liste de toute les fiches | Read

getFiche() : me permet d’avoir une fiche donnée | Read

getEmptyFiche() : me permet d’avoir la liste de fiches vides | Read

saveFiche() : me permet de sauvegarder ou modifier une fiche | Create et Update

deleteFiche() : me permet de supprimer une fiche | Delete

Une image contenant texte

Description générée automatiquement

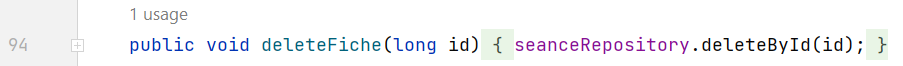


Figure 27 : Méthode CRUD des Fiches

Ci-dessous j’ai créé une méthode pour éviter la répétions de mon code entre la création d’un formateur ou d’un coordinateur. Puis j’ai écrit les méthodes CRUD.

J’ai mis les méthodes de classe trim() pour échapper les espaces avant et après la saisie pour évité les erreurs, ceci s’appelle préserver l’intégrité des données.

Une image contenant texte

Description générée automatiquement

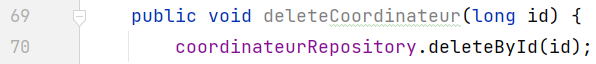


Figure 28 : CRUD de Coordinateur

Une image contenant texte

Description générée automatiquement

Figure 29 : CRUD de Formateur

Une image contenant texte

Description générée automatiquement

Figure 30 : Chargement d’un utilisateur par ses identifiants

## La couche des vues

J’ai utilisé Thymeleaf comme moteur de template. Les contrôleurs enverront dans mes templates, grâce à des indications dans leurs méthodes, des modelmap qui me permettent d’assigner des objets avec une clef en String et une valeur qui lui est un objet. Le template va faire interpolation de ces valeurs envoyées depuis le contrôleur et les incorporer dans la vue qui sera envoyé au tier Client de l’application web.

Les éléments du moteur de template sont les attribut « th » au sein des balises html.

J’utilise du HTML/CSS mais principalement du Bootstrap pour la partie CSS et JS pour la mise en forme et de donnée du style et mes balise HTML.

M’étant renseigné auprès de nos clients coordinateur et formateurs avec mes camarades nous avons donné de l’importance à l’ergonomie des vues car c’est une application de gestion administrative et une application qui n’est pas ergonomique n’aurait pas facilité l’efficacité de nos acteurs lors de leurs navigations dessus.

J’ai utilisé des fragments dans mon moteurs de template qui me permettent de créer des éléments à part dans un fichier spécial que je peux invoquer dans mes vues. Cela évite la réécriture sur plusieurs pages différentes si j’ai besoin de modifier ces fragments.

Une image contenant texte

Description générée automatiquement

Figure  : Vue de la page Welcome 1

Une image contenant texte

Description générée automatiquement

Figure 32 : Vue de la page Welcome 2

Sur l’exemple si dessous j’ai mis le bouton de déconnexion en formulaire pour éviter les attaque CSRF.

Une image contenant texte

Description générée automatiquement

Figure 33 : Navbar fragment

Une image contenant texte

Description générée automatiquement

Figure 34 : Footer fragment

Une image contenant texte

Description générée automatiquement

Figure 35 : Vue du Formulaire des compétences 1

Une image contenant texte

Description générée automatiquement

Figure 36 : Vue du Formulaire des compétences 2

Une image contenant texte

Description générée automatiquement

Figure 37 : Vue du Formulaire des compétences 2

Une image contenant texte

Description générée automatiquement

Figure 38 : Vue du Tableau des compétences 1

Une image contenant texte

Description générée automatiquement

Figure 39 : Vue du Tableau des compétences 2

Mes vues comportent des intercepteurs qui se déclencherons uniquement si une erreur est envoyée dans la vue et indiquera la source d’erreur suivant le message envoyé par le contrôleur grâce au « th:if » qui me permet de mettre des conditions dans mes vues.

L’attribut « th:classappend » vas choisir entre deux choix de classe en fonction de la conditions ce qui permet de styliser dynamiquement.

## La configuration de Spring Security

Spring Security est un framework de sécurité Spring. La sécurité d’une application, pour qu’elle soit efficace, doit être sur plusieurs niveaux. Dans mon cas elle servira de filtre pour diminuer la surface d’attaque, c’est-à-dire d’éviter au maximum d’éventuelle faille que pourrais utiliser des acteurs malicieux.

J’ai par exemple effectué plusieurs filtres restreignant l’accès d’un utilisateur aux fonctions qui lui sont attribuées et autorisées. Cela empêcherait certains acteurs d’avoir accès à des fonctionnalités qui n’étaient pas prévues pour eux.

Les préceptes importants de la sécurité sont l’authentification pour permettre d’identifier les utilisateurs, pour leur attribuer leurs autorisations et les autorisations qui permettent de limiter la portée d’action d’un utilisateur au sein du système.

Spring Security gère l’authentification de l’utilisateur grâce à la mise en place d’un formulaire de connexion avec un contrôleur qui invoquera la méthode loadUserByUsername() qui permet de charger les information correspondantes à l’utilisateur qui se connecte.

Spring Security permet également l’encodage du mot de passe grâce à BCrypt qui me permet d’encryptée une chaine de caractère avec un « sel ». Ensuite en utilisant le même mot de passe avec le sel on obtient la forme cryptée du mot de passe qui va être vérifié avec celui que possède l’application dans sa base de données via Spring Security, pour savoir s’ils correspondent bien.

Ci-dessous voici la classe de configuration des filtres que j’ai appliqué à mon application grâce au antMatchers() qui m’a permis d’assigner des URI avec des permissions, la définition de ma page connexion et celle de déconnexion, avec d’autres précisions supplémentaires.

Une image contenant texte

Description générée automatiquement

Figure 40 : Fichier de configuration de Spring Security

## Les tests de l’application

Au cours du développement de l’application j’ai, au fur et à mesure que je développais les composants et fonctionnalités, également développé des tests qui m’ont permis d’assurer le bon fonctionnement de l’application avant une éventuelle compilation du code.

Lorsque j’ai fait mes méthodes CRUD pour un de mes services j’ai donc testé mes méthodes avec JUnit en procédant avec des données de mon import.sql.

Pour faire des tests unitaires j’ai utilisé des assertions via la méthode assertEqual ou assertNotNull qui me permet de comparer le résultat obtenu et celui qui est attendue.

Si le test ne s’effectue pas correctement alors le test et nous donne les valeurs qu’il a obtenue et la raison de son arrêt pour trouver le problème.

Nous pouvons obtenir le même résultat avec l’outil de débogage de l’IDE.

Les tests s’effectuent sur une partie bien précise de l’application et qui lui est réservée. La classe de test utilise l’annotation @SpringBootTest ce qui crée l’environnement de test qui n’interfère pas avec le reste de l’application quand elle est lancée. Les annotations @Test est essentiel quant à l’annotation @Transactional elle va me servir lorsque mes méthodes vont modifier la BDD, cela indique que la base de données est volatile lorsque l’ont fait des tests et qu’aucun changement n’est conservé une fois le test effectué.

Sont présentés ici quatre tests :

* Un test pour avoir la liste de compétence.
* Un test pour avoir une compétence précise
* Un test d’ajout d’une compétence
* Un test de modification

Une image contenant texte

Description générée automatiquement

Figure 41 : Test de CRUD de compétences 1

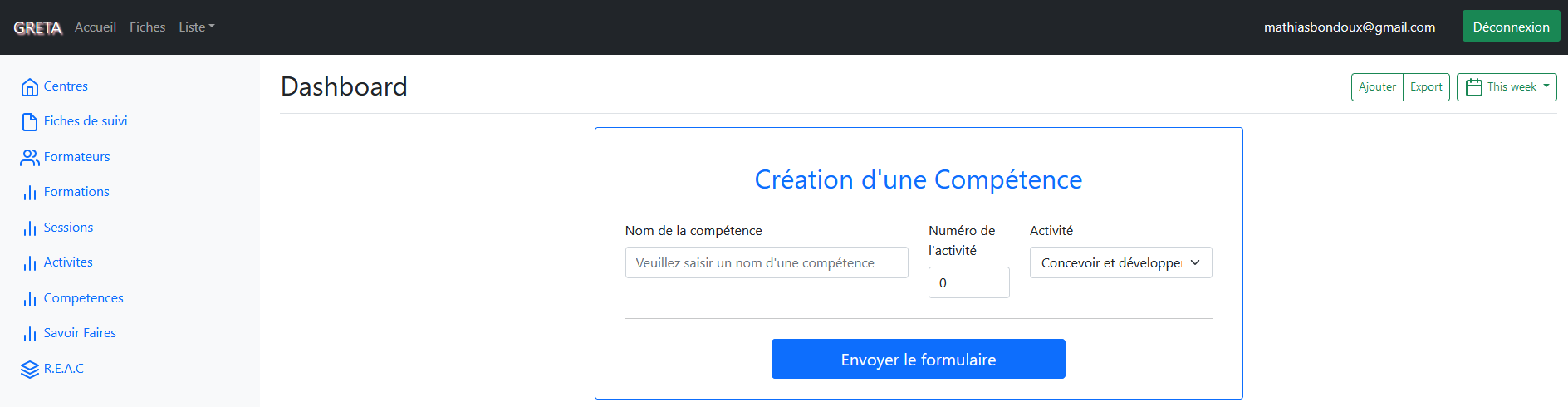
Une image contenant texte

Description générée automatiquement

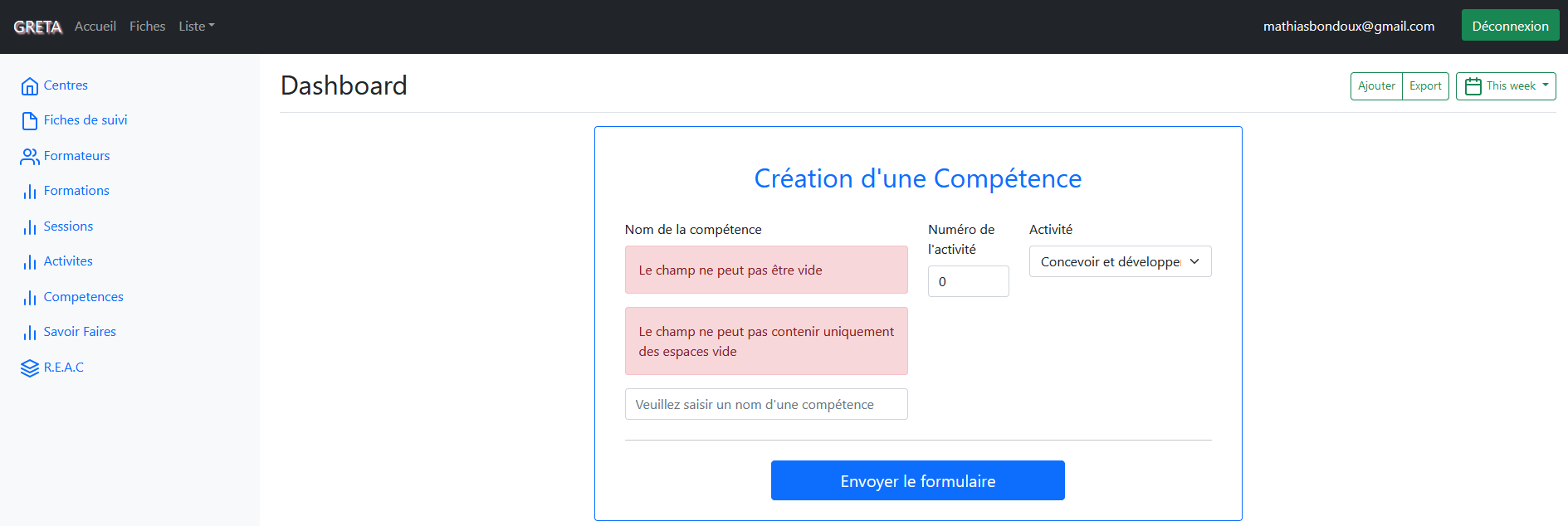
Figure 42 : Test de CRUD de compétences 2

# Jeu d’essais

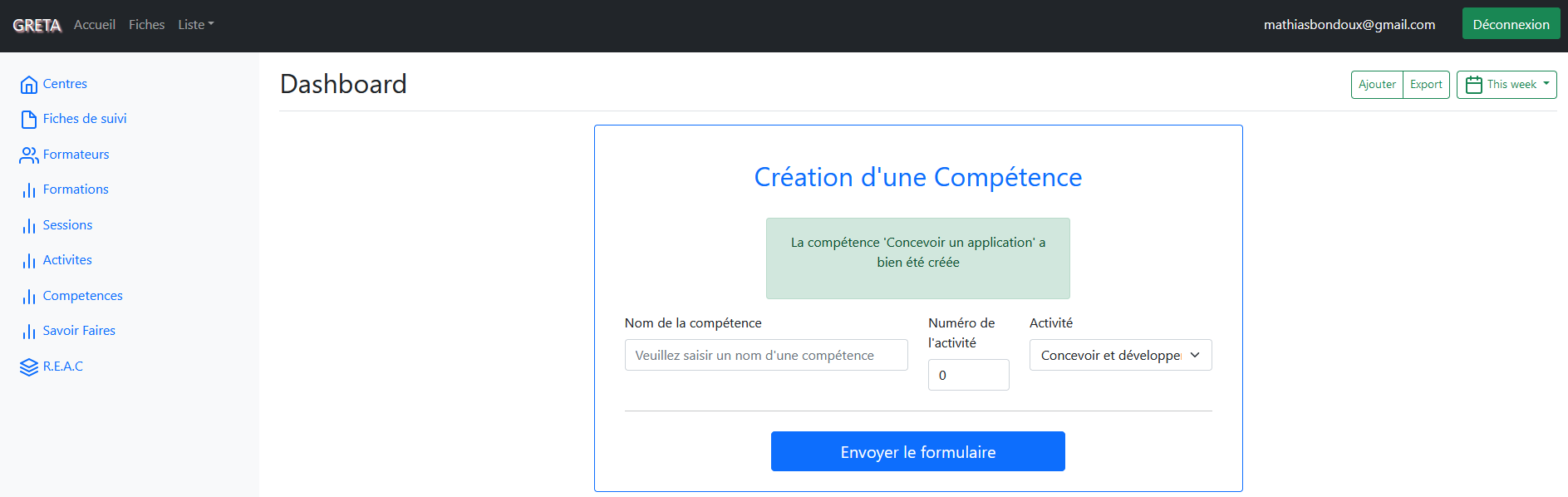
Dans le jeu d’essais je vais essayer de remplir un formulaire d’une compétence, le résultat attendu est que la compétence apparaisse dans le Dashboard si elle passe tous les validateurs.



Dans le jeu d’échec nous pouvons voir que les champs non complétés ou mal complétés seront accompagnés de messages d’erreurs permettant à l’utilisateur de savoir ce qui a bloqué l’envoi du formulaire.



Dans le jeu de réussite nous voyons donc qu’un message de réussite d’envoi du formulaire est présent pour indiquer à l’utilisateur que la compétence a bien été ajoutée.



# Veille technologique

Ma veille technologique porte sur l’élément de sécurité CSRF « Cross Site Request Forgery ».

L’objet de l’attaque CSRF est de forcer des utilisateurs à exécuter une ou plusieurs requêtes non désirées sur un site donné, forgées par un utilisateur malintentionné. La victime choisie aura les privilèges nécessaires à l’exécution de la requête, voire une session encore active.

Forger une requête revient à créer / falsifier une requête HTTP (par le biais d’une URL ou d’un formulaire principalement) pointant sur une action précise interne au site et néfaste pour la victime.

Concrètement, une ressource vulnérable aux attaques CSRF est représentée par toutes ressources disponibles directement ET sans étape intermédiaire (ex : génération de clé/jeton d’autorisation d’accès) sur un SI.

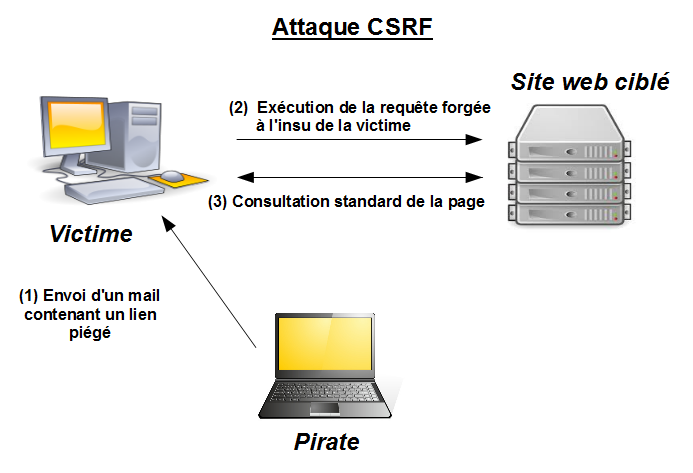


Figure 43 : Schéma montrant le fonctionnement de l’attaque CSRF

Via la méthode « Synchonizer Token Pattern » Il s’agit d’ajouter un paramètre obligatoire correspondant à un identifiant d’accès unique et non prédictible, regénéré pour chaque action utilisateur et par utilisateur, ou pour chaque session, cet identifiant d’accès est nommé “jeton CSRF”.

Ce mécanisme doit être implémenté sur chaque action et dans une balise « form » qui dépend de l’utilisateur qui l’exécute (déconnexion, modification/suppression de données en BDD, donc mes formulaires) et doit avoir une durée de validité limitée dans le temps.

Sans les champs cachés servant de token dans les formulaires pour le CSRF l’utilisateur peut subir à son issue des requêtes qu’il aurait commis à son insu. Des acteurs malicieux peuvent s’en servir pour détourner des fonds, faire du fishing, spamming, clickjacking et bien d’autre encore, pour un Webmail, avec un lien permettant de supprimer vos e-mails.

L’OWASP a placé cette faille dans son Top Ten de son classement annuel de faille informatique.

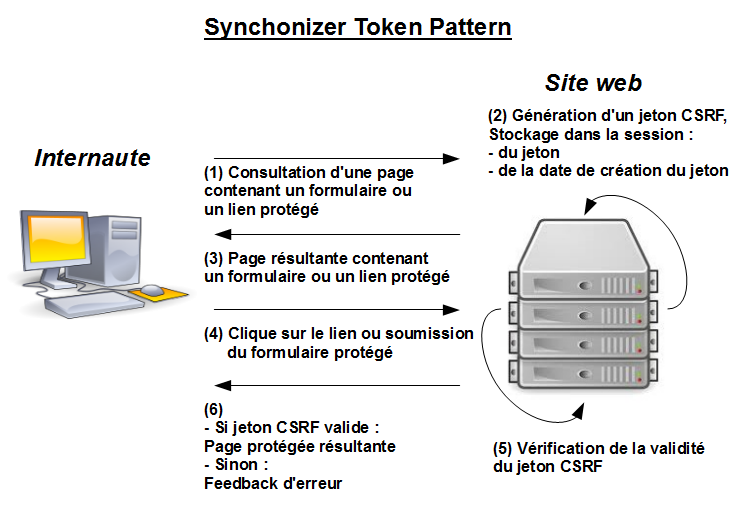


Figure 44 : Schéma montrant le fonctionnement du token CSRF

# Recherche sur une situation de travail

Après avoir effectué une recherche précédente sur comment l’on affectait les relations entre nos classes via les annotations JPA nous devions configurer le tout avec la base de données. C’est pour ce faire que je suis encore allé me renseigner sur Baeldung un web site d’aide sur Java pour le framework Spring.

Une image contenant texte

Description générée automatiquement

Figure 45 : Extrait Baeldung

Ici les indications m’ont permis de voir les dépendances éventuellement manquantes, ensuite les indications montrent le code à mettre dans les propriétés de l’application pour faire fonctionner la base de données.

Une image contenant texte

Description générée automatiquement

Figure 46 : Extrait Baeldung

Ensuite le code si dessus me permet de changer où la base de données va se situer et de la garder en « dure » plutôt quand mémoire qui ne perdure pas après l’arrêt de l’application.

Une image contenant texte

Description générée automatiquement

Figure 47 : Extrait Baeldung

Les commandes SQL ci-dessus et la ligne de code permettent de mettre en place un fichier de donnée SQL qui permet de rentrer des données de tester pour voir si les classes modèle sont bien renseignées et fonctionnent bien du côté de la base de données. On indique si Hiberante prend en compte ou non le fichier de données SQL via la commande des propriétés de l’application.

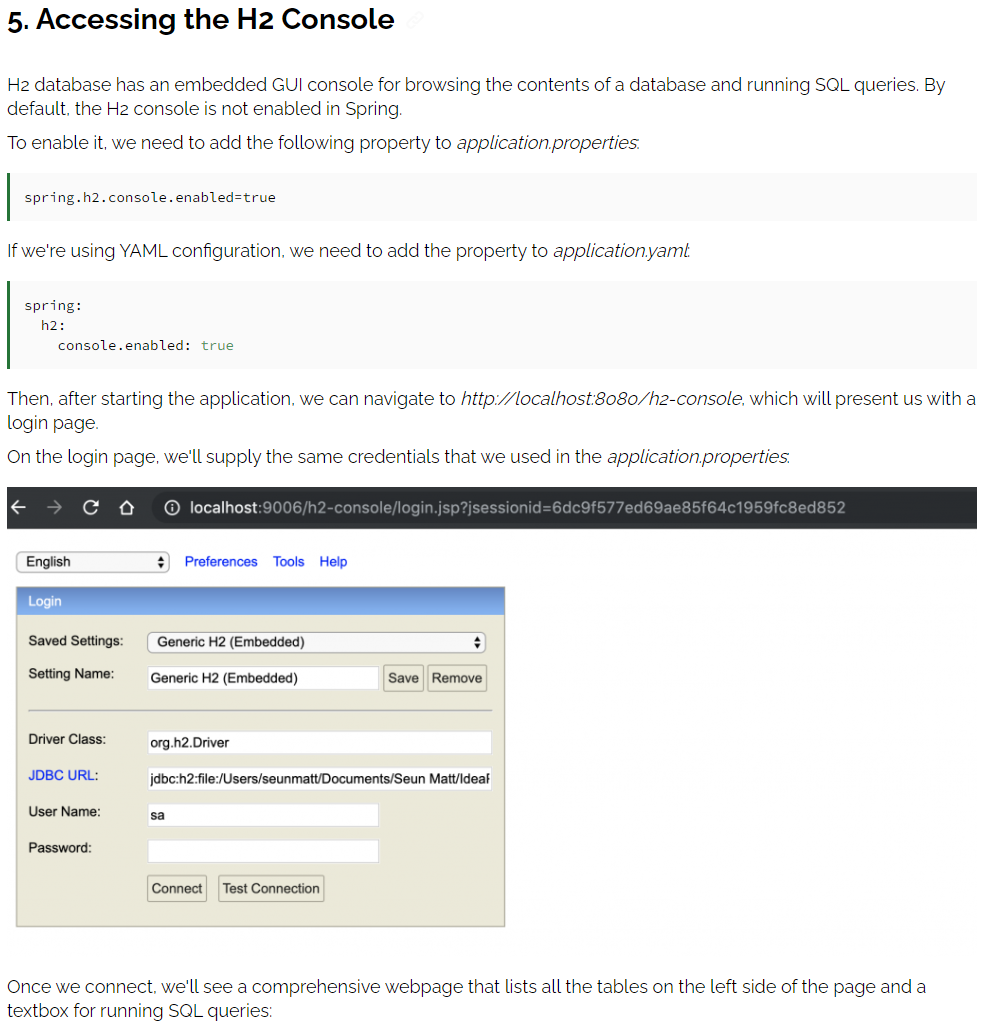


Figure 48 : Extrait Baeldung

Ici j’ai pu apprendre comment accéder à l’interface du SGBD H2 et donc regarder ma base de données et interagir avec elle et vérifier son intégrité.

# Remerciements

Je remercie les personnes m’ayant accompagné et soutenu durant le développement de ce projet en plus de mes camarades de travail :

Mes camarades :

* Patrick MOUCLE
* Loubna DAMRI

L’équipe enseignante :

* Mr Timothé MOULIN
* Mme Beatrice REZAG
* Mme Djamila RIANE

L’équipe administratif :

* Mr Éric SEJOURNE

L’équipe de correction :

* Mme Laura RENIER

## Annexe

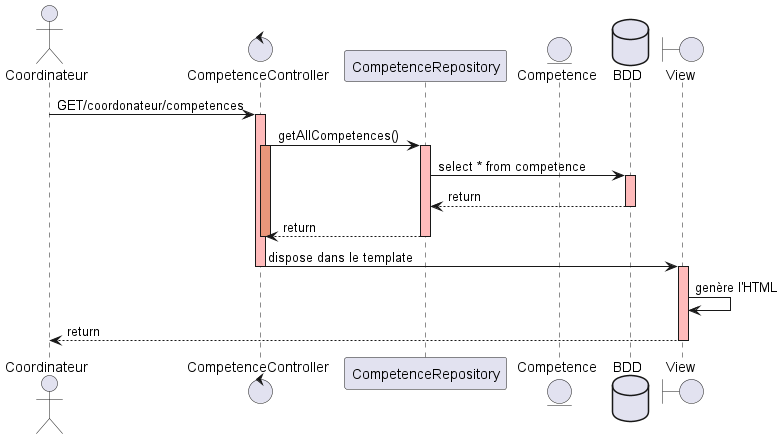


Figure 49 : Diagramme de séquence

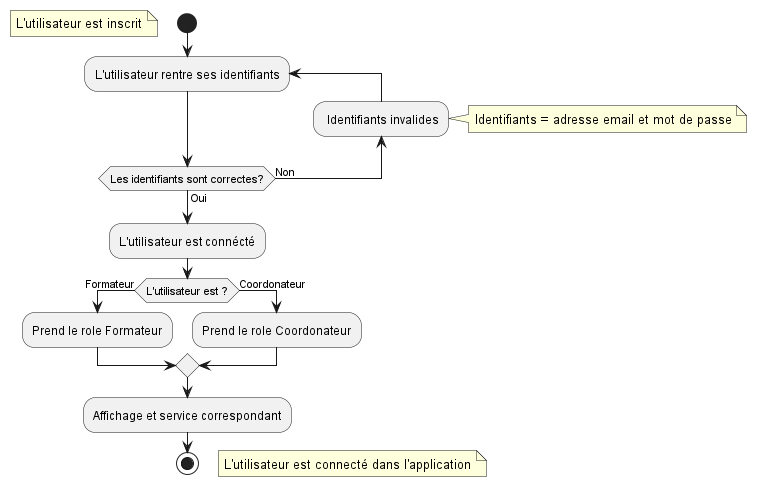


Figure 50 : Diagramme d’activité de la connexion

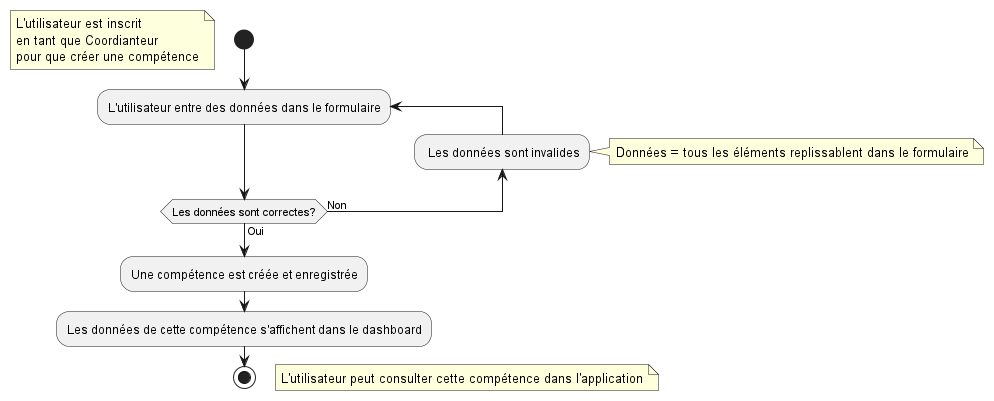


Figure 51 : Diagramme d’activité d’un remplissage de formulaire de compétence

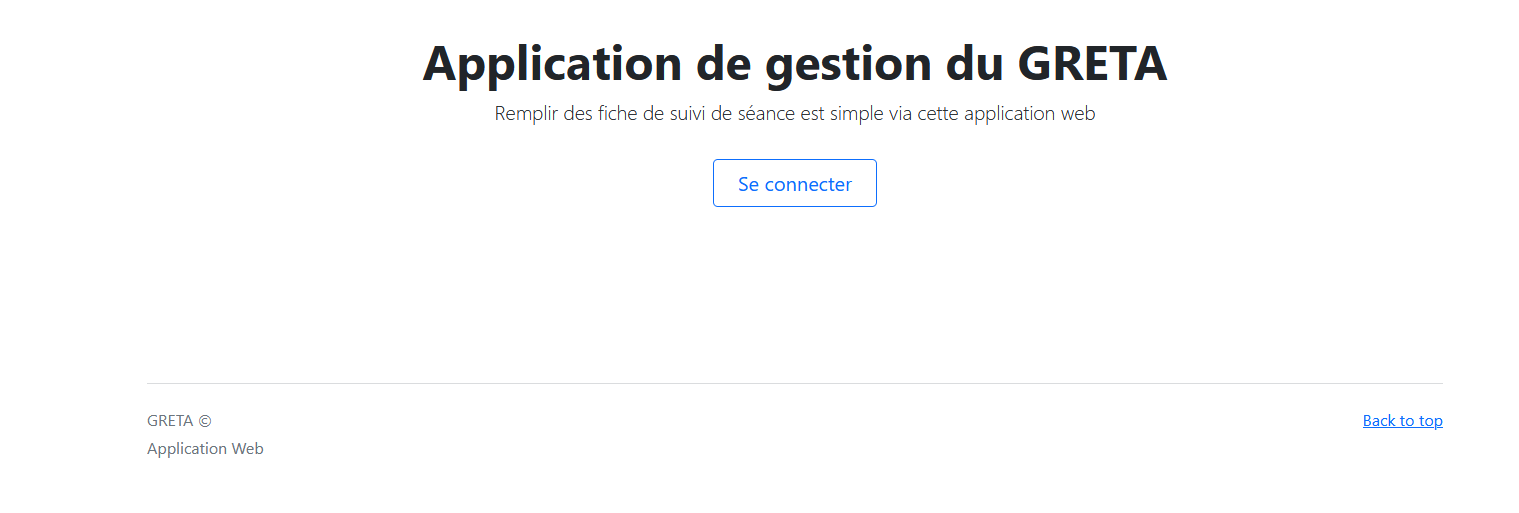


Figure 52 : Vue de la page home

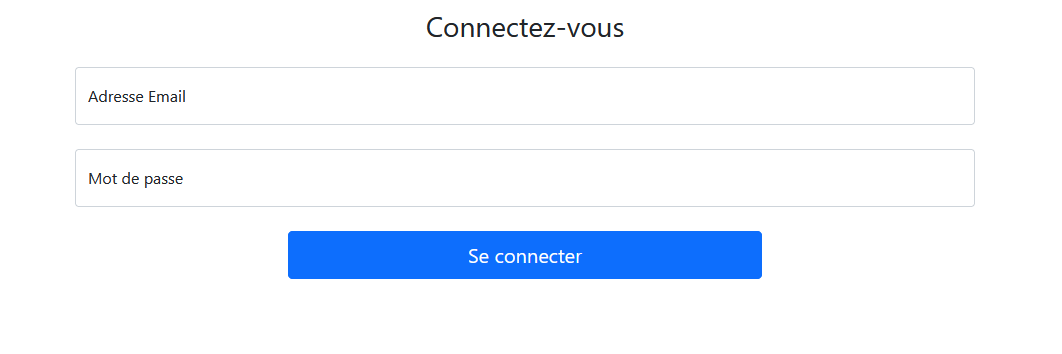


Figure 53 : Vue de la page login

Une image contenant texte

Description générée automatiquement

Figure 54 : Vue de la page Welcome par un Coordinateur

Une image contenant texte

Description générée automatiquement

Figure 55 : Vue de la page Welcome par un Formateur

Une image contenant table

Description générée automatiquement

Figure 56 : Vue de dashboard des Fiches de suivi

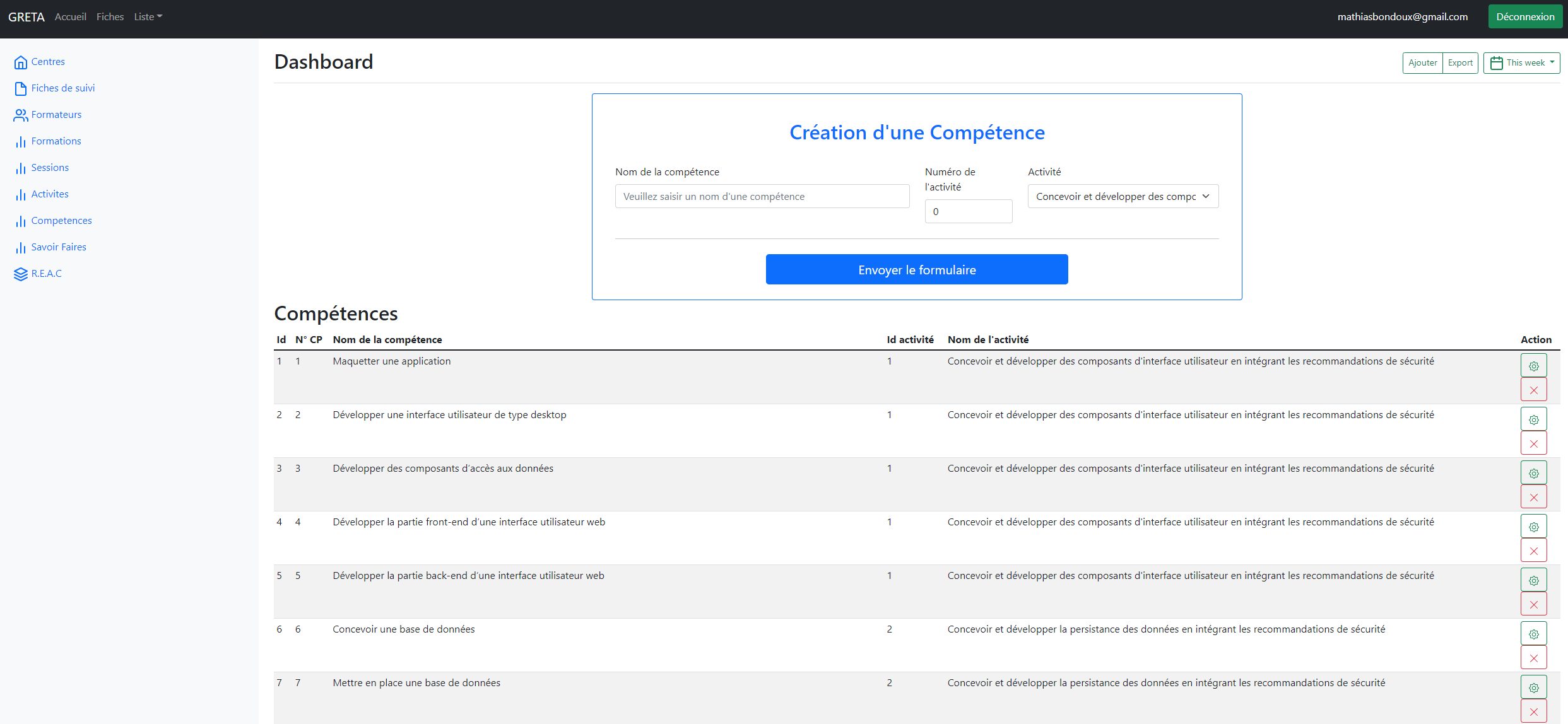


Figure 57 : Vue de dashboard des Compétences