

VERHASSELT Nina

Dossier Projet

ITSO TIME



Titre professionnel

Développeur Web et Web Mobile

Période de stage: du 19 Juin au 1er Septembre 2023

Tuteur en entreprise: CASTET Christophe

Formatrice au sein de l'AFPA: IMLYHEM Yasmina

Table des matières

1. Liste des compétences du référentiel couvertes par le projet.....	4
2. Présentation de l'entreprise.....	4
3. Résumé du projet.....	5
4. Expression des besoins et spécifications fonctionnelles du projet.....	6
5. Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité et le web mobile.....	11
5.1. Présentation de l'architecture.....	11
5.2. Présentation de la base de données.....	11
5.3. Présentation du SGBD.....	13
5.4. Le framework Spring Boot pour le back-end.....	13
5.5. Le framework React.js pour le front-end.....	14
5.6. Compatibilité navigateurs.....	15
5.7. Types d'appareils.....	15
5.8. Sécurité.....	15
5.8.1. Les failles XSS.....	15
5.8.2. React contre les failles XSS.....	16
5.8.3. Les injections SQL.....	16
5.8.4. Hibernate et JPA contre les injections SQL.....	16
6. Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité et le web mobile.	18
6.1. L'environnement de travail.....	18
6.1.1. Prérequis.....	18
6.2. Amélioration générale du style.....	19
6.3. La page de connexion.....	22
6.3.1. L'authentification côté front.....	22

6.3.2. Les jetons d'accès côté back-end.....	26
6.3.3. Le toggle.....	28
6.4. Le tableau de bord du profil administrateur.....	29
6.4.1. La partie front du tableau de bord.....	30
6.4.2. Le back-end du tableau de bord.....	33
6.5. La page Utilisateurs.....	37
6.5.1. La gestion de la navigation.....	37
6.5.2. La gestion du filtre de recherche.....	38
6.5.3. Le filtre de recherche côté back-end.....	39
6.5.4. Le popover du bouton Action.....	40
6.5.5. Le CRUD de la page Utilisateurs.....	42
6.6. La page projet.....	43
6.6.1. L'appel à l'API de Jira.....	44
6.6.2. La désérialisation.....	46
6.6.3. Le cron.....	47
6.6.4. Les RGPD.....	47
7.1. Test de l'ajout d'un nouvel utilisateur.....	49
7.2. Test des marqueurs des utilisateurs.....	49
7.3. Test de la connexion à l'API de Jira.....	50
10.1. Extrait de la documentation Atlassian.....	52
URI structure.....	52
10.2. Traduction de l'extrait.....	52
Structure de l'URI.....	52
11. Conclusion.....	53

1. 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.

- 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

2. Présentation de l'entreprise

La société ITSO Technologies a été créée le 3 juin 2015 par Mr Arnaud FICHOT, rejoint un an après, par son associé Mr Kevin BRETON. Au début, la société sous-traitait les développements et ne comptait aucun développeur jusqu'en décembre 2018.

Le pôle développement crée des applications web répondant au besoin d'une société. Le pôle infrastructure offre différents produits et services supplémentaires tels que la vente d'équipements et la maintenance.

En 2023, ITSO Technologies fusionne avec INERSIO, une société spécialisée dans la réalité virtuelle et augmentée fondée en 2018 par Raynaut ESCORBIAC,.

3. Résumé du projet

La gestion du temps est cruciale pour les petites et moyennes entreprises, optimisant les ressources, la satisfaction client et la croissance. Elle permet l'efficacité opérationnelle, la réactivité, et prévient l'épuisement des employés. Elle facilite aussi les décisions stratégiques et l'équilibre des rôles.

Le projet ITSO Time a pour but d'aider la direction à la gestion de l'entreprise, et de faire gagner du temps. Pour évaluer la rentabilité de l'entreprise et le suivi des tâches, le président d' ITSO Technologies souhaite avoir une vue en temps réel sur le temps consacré à chaque activité. Pour se faire, les développeurs renseignent manuellement dans un tableau Excel la répartition de leur temps de travail quotidien, en associant un nombre d'heures à des tâches prédéfinies ou non. Ceci permet entre autres de comparer le temps réel passé sur un projet au temps facturé au client.

ITSO Time est un outil interne qui permet de faciliter cette tâche. Les développeurs se connectent en tant qu'utilisateur et accèdent à un tableau de bord qui offre une visibilité rétrospective sur leur dernières tâches effectuées selon un nombre de jours donné. Il peut également créer une ou plusieurs tâches en choisissant une date, une durée, un projet, un tag (type de tâche) et une feature (ticket de type feature) depuis son profil.

Quant aux administrateurs, ils ont accès à la création, modification et suppression des utilisateurs, des projets, des tags, et des features. Leur tableau de bord permet de visualiser instantanément la durée de travail de chaque développeur par semaine, sur un nombre de semaines donné. Il permet également d'automatiser l'ajout de projets avec une connexion à l'API de Jira. Les projets peuvent donc être ajoutés manuellement, ou automatiquement. À terme, il en sera de même pour les features.

4.Expression des besoins et spécifications fonctionnelles du projet

À mon arrivée, j'ai dressé une liste de tâches grâce aux informations qui m'ont été transmises par l'ancien stagiaire et les demandes formulées par le président de l'entreprise.

1. Objectif principal :
 - a. Adapter le projet déjà commencé aux besoins spécifiques.
 - b. Améliorer le style
 - c. Assurer le bon fonctionnement global du logiciel.
2. Profil administrateur:
 - a. Page tableau de bord :
 - i. Passer d'un affichage par jour à un affichage par semaine.
 - ii. Afficher un marqueur en fonction du temps de travail des utilisateurs.
 - b. Page tâches:
 - i. Afficher toutes les tâches créées par les utilisateurs sur un nombre de jours donné.
 - c. Connexion à l'API de Jira:
 - i. Afficher les projets créés sur Jira sur la page projet de l'application.
3. Utiliser les outils et technologies suivants:
 - a. React.js
 - b. BitBucket
 - c. Jira
 - d. Slack
 - e. Spring Boot
 - f. PgAdmin

4. Utiliser les langages suivants:

- a. HTML (front-end)
- b. CSS (front-end)
- c. Java (back-end)
- d. JavaScript (front-end)
- e. SQL (back-end)

Le 4 Août, une liste de tâches précises m'a été donnée sous format Excel. Certains éléments ont été colorés en bleu pour marquer leur priorité moins élevée.

Page	Tâche	Statut
Connexion	Mettre Logo Inersio -> en pj	
Connexion	Texte champ de saisie identifiant/mdp trop petit	
Connexion	Erreur après connexion	
Connexion	Ajouter mot de passe oublié.	
Utilisateurs	Ordre colonnes : Nom - Prénom - Email - Identifiant - Pôle - Rôle - Statut - Action	
Utilisateurs	Ajouter colonne Pôle	
Utilisateurs	Ajouter colonne Statut (Actif/Désactivé)	
Utilisateurs	Ajouter colonne Identifiant prendra la valeur user.pseudo	
Utilisateurs	Action : Mettre liste déroulante à la place de pop-up	

Utilisateurs	Action : Renommer "Supprimer" en "Désactiver"	
Utilisateurs	Action : "Désactiver" mettre pop up de confirmation avant action.	
Créa / Modif Utilisateurs	- Mettre à jour champ "job" devient "pôle".	
Créa / Modif Utilisateurs	- Ajouter champ "Rôle" dans formulaire	
Créa / Modif Utilisateurs	Mettre à jour la liste des pôles : FrontEnd BackEnd Infra Moa Moe	
Projet	Bouton "Créer projet" devient "Créer client"	
Projet	Ordre colonnes : Client - Projet - Nb Feature - Statut - Action	
Projet	Renommer colonne "Nom" en "Projet"	
Projet	Ajouter colonne "Statut" (Nouveau /Actif/Désactivé/Terminé), Nouveau = lorsqu'un client est créé sans projet associé, dès qu'un projet est associé, passe en actif. Terminer possible uniquement en actif.	
Projet	Ajouter colonne "Nb Feature" -> count des features attention faire un distinct pour éviter les doublons liés au pôle	

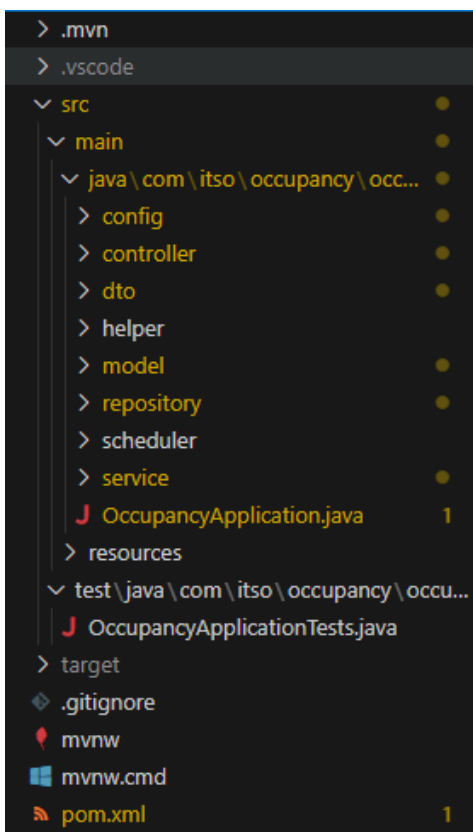
Projet	Action : Mettre liste déroulante à la place de pop-up	
Projet	Action : Renommer "Supprimer" en "Désactiver"	
Projet	Action : "Désactiver" mettre pop up de confirmation avant action.	
Projet	Liste des Actions : Récupérer projet_jira Créer projet manuel Désactiver Modifier Clôturer un projet (Permettre de rendre la saisie impossible)	
Menu	Ordre Menu : Mon Dashboard - Utilisateurs - Planning - Projets - Feature - Type de tâche - Tâches -> Vue Admin	
Menu	Renommer "Tags" en "Type de tâche"	
Menu	Supprimer Client	
Feature	Ajouter bouton "Exporter"	
Feature	Ordre colonnes : Projet - Code - Feature - Pôle - Version - Date échéance - Temps estimés - Temps réalisés	
Feature	Ajouter colonne "Pole" (Etiquette issu jira)	

Feature	Supprimer colonne "Début estimé"	
Feature	Supprimer colonne "Début réel"	
Feature	Ajouter colonne "Version" (issu Jira)	
Feature	Ajouter colonne "Date échéance" (issu Jira)	
Feature	Ajouter colonne "Temps estimés" = temps à la feature (issu Jira).	
Feature	Ajouter colonne "Temps réalisés" = cumul des temps saisis à la feature (issu jira).	
Feature	Ajouter sous tableau un Total global Temps estimés (A mettre à jour selon les filtres)	
Feature	Ajouter sous tableau un Total global Temps réalisés (A mettre à jour selon les filtres)	
Tâche	Date mettre par défaut sur la date du jour mais laisser le date picker.	
Client	Supprimer page	
Planning	Page à créer : Calendrier avec vision des absences collaborateurs	
Planning	Action pour admin de saisir les dates de cp/collaborateur	

5. Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité et le web mobile

5.1. Présentation de l'architecture

Le projet est réalisé en structure **MVC** (Modèle-Vue-Contrôleur). C'est une architecture de conception en couches orientée objet qui divise la logique du code en trois parties :

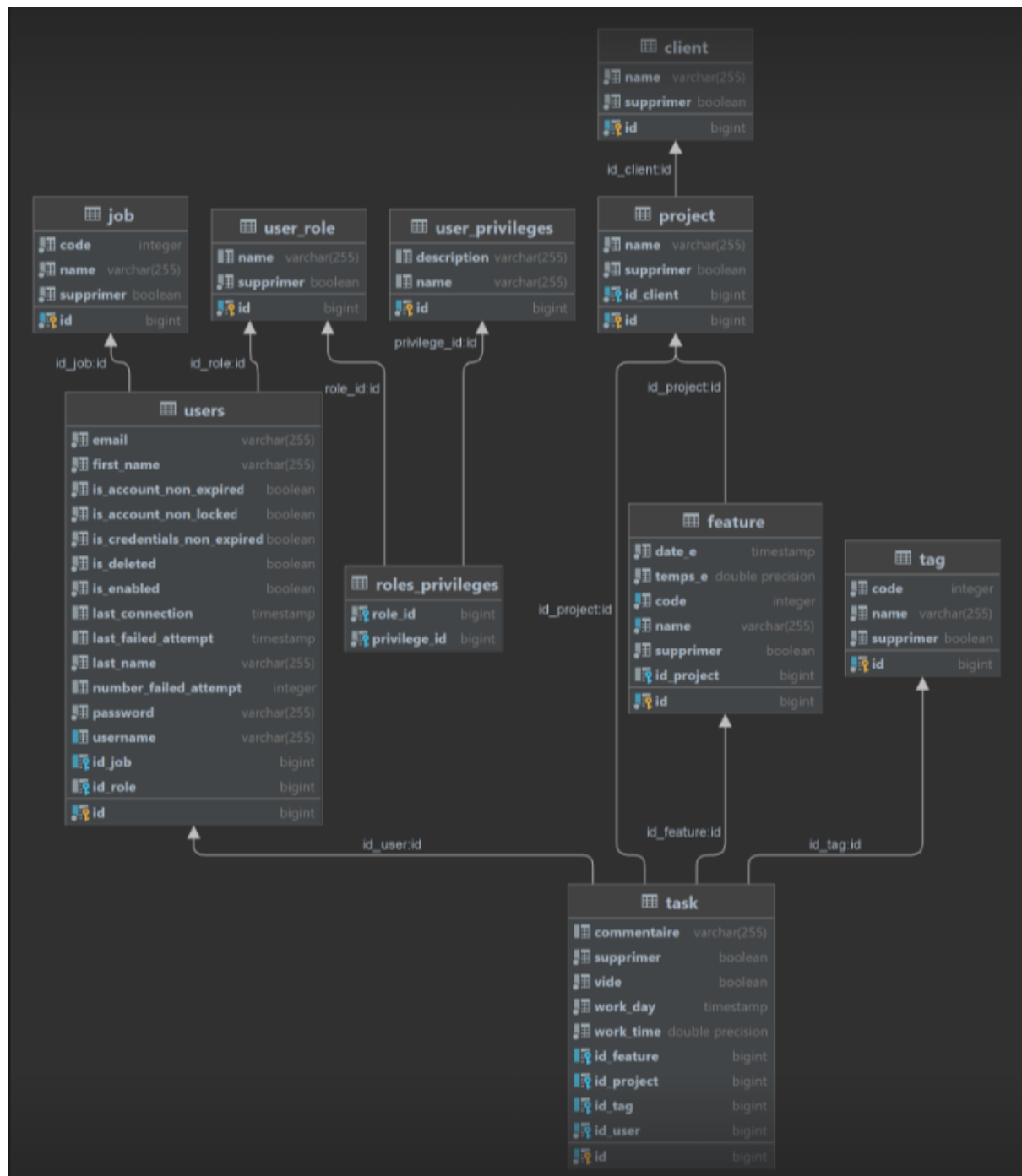


- **Modèle** : Ici, il s'agit des entités que l'on trouvera dans le dossier "model", et les requêtes à la base de données du dossier repository.
- **Vue** : C'est la partie front-end de mon application gérée en React.js, la partie visible par le client..
- **Contrôleur** : Dans le projet, les fichiers du dossier services qui s'occupent du traitement des données en font partie.

Cette architecture permet d'obtenir une application robuste et évolutive.

5.2. Présentation de la base de données

Schéma de la base de données:



C'est une base de données relationnelle qui contient dix tables. Il est important de noter que l'application a été réalisée en **Code First**, c'est-à-dire que les modèles de données ont été créés en premier à partir du back-end. Dans le schéma de la base de données on retrouve les entités, les propriétés, les clés étrangères et les relations entre les tables (ou entités).

5.3. Présentation du SGBD

PostgreSQL est un système de gestion de base de données open source très stable, qui prend en charge différentes fonctions de SQL, comme les clés étrangères, les sous-requêtes, les déclencheurs, ainsi que différents types et fonctions définis par l'utilisateur. C'est la deuxième base de données la plus utilisée après MySQL. Il permet donc la gestion des relations entre les tables et offre une sémantique ACID (Atomicité, Cohérence, Isolation et Durabilité) pour les transactions.

5.4. Le framework Spring Boot pour le back-end

Spring Boot est un framework fondé sur une autre framework appelé **Spring** utilisé pour le langage Java. Il réduit la complexité des configurations et accélère le développement d'applications. Il est simple à utiliser et s'intègre facilement avec d'autres technologies. Il est adapté aux besoins du projet ITSO Time, par exemple, sans l'utilisation du framework, la configuration d'un projet en MVC serait beaucoup plus longue. Il est comparable à Symfony ou Laravel qui sont utilisés pour le langage PHP.

Pour créer un projet spring boot, il faut se rendre sur le site spring initializr (<https://start.spring.io/>), compléter les champs et ajouter les dépendances souhaitées.

En cliquant sur generate, on lance le téléchargement du projet zippé.

On ouvre le projet dans L'IDE et on commence par créer la connexion à la base de données dans le fichier application properties.

Ensuite on créera les modèles, en commençant par la table utilisateurs pour la gestion d' authentifications, puis les controllers et les services.

Les modèles du back-end représentent les entités. On définit des index pour optimiser les performances des requêtes en accélérant la recherche de données dans une colonne donnée.

J'utilise également la bibliothèque **Lombok** pour faciliter la création des getters et des setters entre autres. En effet leur création est automatisée.

5.5. Le framework React.js pour le front-end

Pour la partie front-end, j'ai utilisé **React.js**, un framework **JavaScript**. C'est un outil qui permet de travailler de manière plus efficace et organisée. Il permet de développer des applications web robustes et efficaces.

Avec React, on développe une application en architecture simple page. C'est ce qu'on appelle une Single Page Application (SPA). Au lieu de faire une requête au serveur à chaque fois que l'on change de page, tout est envoyé au client, le comportement est le même, mais tout est géré par du JavaScript et on gagne en rapidité.

React est un framework orienté composant. On développe de multiples composants qui une fois assemblés tous ensembles constitueront une application à part entière. Les composants sont un assemblage de code HTML, de règles de style CSS et de fonction JavaScript permettant de leur implémenter des comportements.

Il est très populaire, on peut donc trouver de nombreux exemples de projets et de l'aide facilement. Il est créé et maintenu par Facebook, c'est une technologie de qualité qui sera toujours viable dans plusieurs années. L'utilisation du DOM virtuel n'interagit pas directement avec le DOM HTML de la page, ce qui le rend plus performant. On met à jour uniquement les parties de l'interface qui ont besoin d'être modifiées.

L'installation de l'environnement de travail est simple. On installe node. On ouvre VScode, et on tape "npx create-react-app nom-du-projet" dans le terminal. La

structure d'un projet React avec les dépendances nécessaire est automatiquement créée.

Dans ce projet, j'ai utilisé React.js avec **styled components** pour le style. C'est une bibliothèque JavaScript qui permet de définir des styles CSS directement dans les fichiers JavaScript en utilisant une approche basée sur les composants.

Il est possible de créer des composants React qui encapsulent à la fois leur logique et leur style.

5.6. Compatibilité navigateurs

- Google Chrome
- Firefox
- Microsoft Edge

5.7. Types d'appareils

- Ordinateur de bureau
- Ordinateur portable
- Tablette

5.8. Sécurité

La sécurité ne doit pas être négligée lors de la création d'une application web. Dans cette partie je vais aborder les failles XSS et les injections SQL qui sont des failles et attaques courantes qui peuvent être dangereuses. Les technologies choisies pour ce projet offrent des solutions pour les éviter.

5.8.1. Les failles XSS

XSS veut dire cross site scripting, c'est-à-dire du scripting inter site. L'attaquant va utiliser une vulnérabilité du client pour le forcer à envoyer des informations à un

poste pirate. Son objectif est de détourner la logique d'une application et permettre par exemple le vol de cookies ou de jetons de session.

5.8.2. React contre les failles XSS

React n'est pas épargné pour les failles XSS, cependant, l'utilisation de JSX aide à protéger de ce genres d'attaques.

JSX est une extension React de la syntaxe de langage JavaScript qui permet de structurer le rendu des composants à l'aide d'une syntaxe similaire au HTML. C'est ce que j'utilise dans le DOM virtuel.

On n'agit pas directement dans le HTML, et c'est ce qui représente un point positif pour la sécurité. De plus, React échappe les entrées (les inputs) des utilisateurs et les transforme en chaînes de caractères, ce qui fait que s'ils tentent d'injecter du code, cela ne fonctionnera pas. Si on a besoin de mettre en place un `innerHTML` dans le DOM, React met à disposition une propriété appelée `dangerouslySetInnerHTML` qui permet de le faire de manière sécurisé. Il existe également des bibliothèques élaborées pour renforcer la sécurité des applications React telles que **dompurify**.

Il est important de noter que React n'est pas totalement invulnérable, mais les points évoqués forment déjà une première barrière contre les attaques XSS.

5.8.3. Les injections SQL

L'injection SQL peut être encore plus grave que l'injection XSS, en effet, les données personnelles sont directement exposées. L'attaquant peut entrer du code SQL, faire des requêtes et accéder à la base de données.

5.8.4. Hibernate et JPA contre les injections SQL

L'utilisation d'Hibernate et JPA apportent une sécurité contre les injections SQL.

Hibernate est un framework open source qui gère la persistance des objets en base de données relationnelle. La persistance des données fait référence à la capacité de stocker et de conserver des données au-delà de la durée de vie d'une application ou d'une session en cours.

JPA est l'acronyme de Java Persistence API, c'est une API de Java qui définit une spécification pour la persistance des données dans une application Java.

Hibernate et JPA permettent d'exécuter des requêtes préparées qui forment un mécanisme de protection contre les injections SQL en paramétrant les valeurs dans la requête plutôt que de la concaténer directement dans le texte de la requête. J'ai deux types de requêtes dans mes fichiers repository.

Certaines sont faites sous forme de méthode en utilisant des noms de méthode définis dans Spring Data JPA. L'utilisation de noms de méthodes explicites et basés sur les conventions permet à Spring Data JPA de générer automatiquement des requêtes sécurisées en utilisant des requêtes préparées. C'est ce qui donne ces requêtes atypiques:

```
Optional<User>  
findByIdAndIsDeletedIsFalseAndJobSupprimerIsFalseOrJobSupprime  
rIsNullAndIsDeletedIsFalseAndId(Long id,Long id2);
```

D'autres sont faites en **HQL** (Hibernate Query Language), qui est un langage similaire au SQL conçu spécifiquement pour interagir avec des objets persistants gérés par Hibernate. En utilisant des paramètres positionnels ('?1', '?2', etc.) et les valeurs passées en arguments, je sépare la structure de la requête SQL du contenu des valeurs, ce qui contribue à éviter les injections SQL.

Il faudra également ajouter des conditions pour vérifier et filtrer les entrées, aussi bien dans le front que dans le back.

Ces mesures sont simples et bien qu'elles ne soient pas infaillibles, elles offrent une sécurité de base efficace contre les attaques courantes. De manière générale, les

frameworks offrent souvent une protection de base contre ces deux types d'attaques.

6. Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité et le web mobile

À mon arrivée, le projet était déjà bien avancé tant du côté front-end que du côté back-end. Je me suis rapidement imprégnée des technologies qui m'étaient inconnues. J'ai installé l'environnement de travail et entrepris une analyse approfondie de l'architecture en place. J'ai minutieusement étudié la documentation relative aux technologies impliquées et suivi des tutoriels pertinents pour renforcer mes compétences. Afin d'accélérer mon apprentissage, j'ai sollicité l'expertise des développeurs présents sur site lorsque leurs disponibilités le permettaient. Mon objectif était de comprendre le but de l'application et maîtriser au mieux les éléments nécessaires à la réalisation ou l'amélioration de ses fonctionnalités.

Je vais structurer ma présentation en débutant par une description des prérequis et des configurations essentielles pour mettre en place le projet en mode développement. Ensuite, je vais aborder les principales fonctionnalités sur lesquelles j'ai travaillé. Pour chacune d'entre elles, je commencerai par détailler la composante front-end, avant, le cas échéant, d'aborder la partie correspondante du back-end.

6.1. L'environnement de travail

Pour pouvoir lancer le projet, il faut s'assurer d'avoir certains outils, dépendances et configurations.

6.1.1. Prérequis

Les installations suivantes sont requises:

- Java Development Kit (JDK)

- Un IDE. J'utilise Visual Studio Code
- Le gestionnaire de dépendances Maven (grâce à la commande 'mvn install')
- Pg Admin pour la base de données
- node.js et yarn pour l'utilisation de React dans la partie front-end

Il est également nécessaire de configurer la connexion à la base de données dans le fichier "application.properties" avec le nom et le mot de passe de la base de données. Il faudra créer la base de données correspondante à cette configuration. Pour travailler en local, j'ai un fichier supplémentaire appelé "application-local.properties" avec les configurations nécessaires:

```
spring.datasource.url =  
jdbc:postgresql://localhost:5432/occupancy  
spring.datasource.password = admin
```

La partie back-end doit être démarrée en premier depuis l'IDE dans le tableau de bord Spring Boot. Il faudra s'assurer que le run a fonctionné, on doit apercevoir le message suivant dans le terminal : Started OccupancyApplication in 6.638 seconds (JVM running for 7.65).

La partie front-end pourra ensuite être compilée grâce à la commande 'yarn start' et s'ouvrir automatiquement dans le navigateur.

6.2. Amélioration générale du style

L'une des premières tâches qui m'a été assignée était l'amélioration globale du style de l'application. Bien qu'il n'y ait pas eu de maquette spécifique fournie, j'ai entrepris d'analyser la situation. J'ai pris la décision de me concentrer principalement sur la mise en forme des éléments centraux de l'affichage, laissant le menu latéral qui était déjà en place, et qui semblait approprié pour ce genre d'application. Je n'ai pas réalisé de maquette complète car il s'agissait seulement de peaufiner le style existant plutôt que de créer quelque chose de nouveau. J'ai cherché des exemples

de sites afin de m'inspirer en essayant d'appliquer les notions d'ergonomie que j'ai acquises grâce à l'ouvrage d'Amélie Boucher.

The screenshot shows a web application interface for a user named 'Utilisateur Test'. The page is titled 'Tâche' and 'Création de tâche'. The form is organized horizontally with three columns. The first column contains a date picker labeled 'Date de la tâche' and a dropdown for 'Durée'. The second column contains a dropdown for 'Projet' and a dropdown for 'Tag'. The third column contains a dropdown for 'Feature' and an orange button labeled 'Ajouter une tâche'.

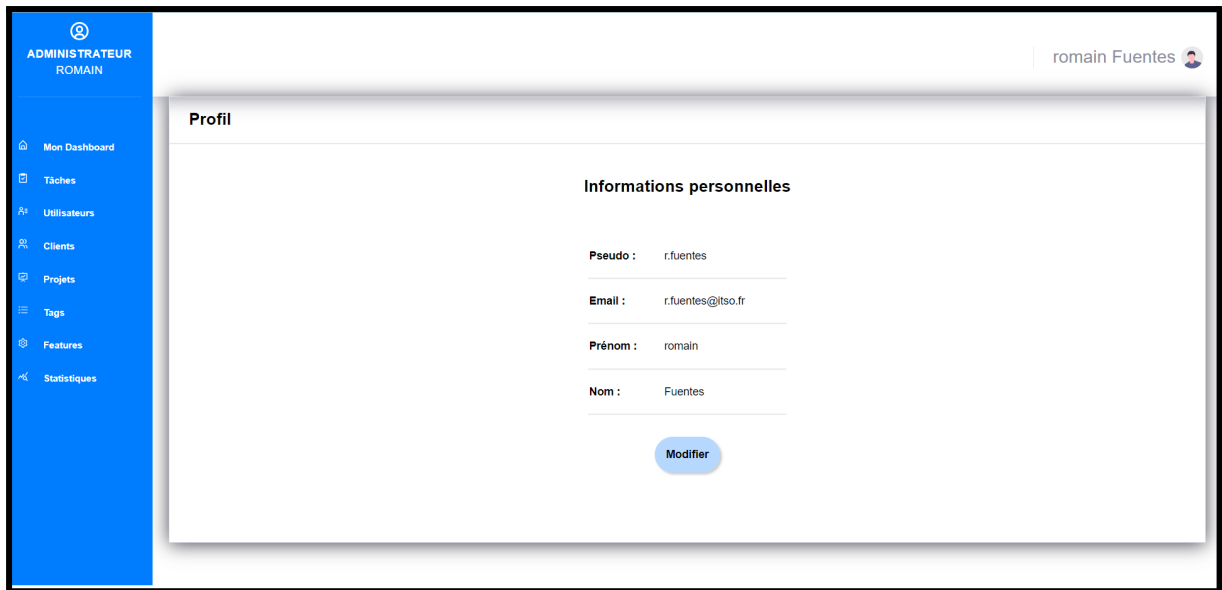
En ce qui concerne les formulaires, qui étaient initialement disposés horizontalement,

j'ai travaillé sur leur esthétique afin de suivre une cohérence visuelle tout en les présentant de manière plus verticale.

The screenshot shows the same 'Création de tâche' form, but with a vertical layout. The form fields are stacked vertically: 'Date de la tâche' (date picker), 'Durée' (dropdown), 'Projet' (dropdown), 'Tag' (dropdown), and 'Feature' (dropdown). The orange 'Ajouter une tâche' button is positioned at the bottom right of the form.

The screenshot shows a web application interface for an administrator named 'romain Fuentes'. The page is titled 'Paramètres' and 'Informations personnelles'. The form is organized vertically. It includes a section for 'Informations personnelles' with fields for 'Pseudo', 'Email', 'Prénom', and 'Nom'. Below this is a 'Modification' section with two orange buttons: 'Modifier le pseudo' and 'Modifier le mot de passe'.

J'ai procédé en suivant le même principe pour la page des informations personnelles.



Les tableaux arboraient un style minimaliste évoquant la disposition familière d'un tableau. Pour parvenir à l'aspect actuel, j'ai puisé des idées sur internet et sélectionné des designs de tableaux qui m'ont plu (voir images).

TARIF	START	BASIC	PREMIUM
PRICING TABLE	\$29.99	\$49.99	\$69.99
Collecteur adipiscing	X	✓	✓
Adapting consectetur	4 GB	16 GB	UNLIMITED
Pilemqueque	15 CONTACTS	30 CONTACTS	>50 CONTACTS
White barum	✓	✓	✓
Dolors aggestat nris	X	X	✓
Maurea erosqueque	✓	✓	✓

Date	numéro de commande	Nom	Prix
2020-06-04 01:22	200398	iPhone SE	\$399
2020-06-04 03:04	200399	iPhone 11	\$499
2020-06-04 04:00	200400	iPad Pro	\$999
2020-06-04 05:06	200401	iPhone 11 Pro	\$680
2020-06-04 06:00	200403	casque de musique	\$200

ADMINISTRATEUR ROMAIN

romain Fuentes

Utilisateurs

Ajouter un utilisateur

Filtre de recherche

Rechercher

Nom	Prénom	Email	Pole	Role	Statut	Action
RoleTest	UserRoleTest	mail@mail.fr	Backend	ADMIN	Actif	
Test	Utilisateur	userTest@mail.fr	Frontend	USER	Actif	
Fuentes	romain	r.fuentes@ltsa.fr	unknown	ADMIN	Actif	
Verhassett	Nina	nina.v@live.fr	Frontend	USER	Désactivé	

J'ai simplement supprimé la séparation des colonnes et séparé les rangées par des lignes blanches. J'ai introduit une légère touche de couleur en

appliquant un dégradé doux en arrière-plan, en alternant une ligne sur deux.

```
tr:nth-child(even) {  
    background-image: linear-gradient(to left, rgb(0, 119,  
254, 0.2), white);}
```

J'ai respecté une cohérence dans le style des pages que j'ai réalisée et porté une attention particulière à l'ergonomie. Le style est épuré et minimaliste car le but est d'afficher des informations clairement.

6.3. La page de connexion

La page de connexion ou Login constitue le point d'entrée initial de l'application. C'est depuis cette page que les utilisateurs et les administrateurs peuvent se connecter en entrant leurs identifiants. Elle est définie comme page d'accueil dans le fichier App.js qui contient toutes les routes de l'application.

```
<Route path="/" exact element={<LoginPage/>}/>
```

6.3.1. L'authentification côté front

Un rôle est attribué à chaque utilisateur, en effet, nous avons le rôle administrateur et le rôle utilisateur que l'on retrouve dans la table "user_role" de la base de données. On utilise un **token** ou **jeton d'accès** pour sécuriser leur authentification.

Lorsqu'un utilisateur tente de se connecter via la page de connexion, il envoie ses informations d'identification au serveur. Le serveur vérifie ces informations pour s'assurer que l'utilisateur est valide. Ici, dans la page de connexion Login du front-end, on collecte les informations d'identification de l'utilisateur pour les envoyer vers le endpoint "/auth/jwt" du back-end grâce à un appel à l'API.

Une **API** (application programming interface ou « interface de programmation d'application ») est une interface logicielle qui permet de « connecter » un logiciel à un autre, ou à un service afin d'échanger des données et des fonctionnalités. C'est

en quelque sorte un médiateur entre les utilisateurs ou clients et les ressources ou services web auxquels ils souhaitent accéder.

Dans le projet, j'utilise **Axios**, une bibliothèque JavaScript qui facilite les appels aux API en fournissant une interface simple. Elle permet de créer des instances Axios avec des options par défaut, d'effectuer des requêtes HTTP vers des endpoints spécifiques, de gérer les réponses et les erreurs, de définir des en-têtes personnalisés et d'envoyer des données dans le corps de la requête, entre autres.

Dans d'autres fichiers, je configure des instances Axios avec une URL de base selon les endpoints visés. Ici j'utilise "API_POST" définit comme suit:

```
export default function API_POST() {  
  return axios.create({  
    baseURL: 'http://localhost:8080/cw',  
    headers: {'Vary': 'Access-Control-Allow-Origin'}});  
}
```

C'est ce qui permet de communiquer avec l'API Spring Boot de mon back-end. J'utilise ensuite cette fonction dans le front pour faire y faire appel.

```
const customer = await API_POST()  
  .post('/auth/jwt', {  
    username: username,  
    password: password })
```

On récupère la réponse en intégrant un système de gestion d'erreur pour une meilleure visibilité en cas d'échec de la requête.

```
.then((res) => res.data)  
  .catch((e) => {  
    return { error: true, message:  
JSON.stringify(e) }}
```

S'il n'y a pas d'erreur, le token est mis dans une constante appelée "token", puis il est stocké dans le localStorage du navigateur grâce à la méthode "setItem()". Cela

permet de conserver le token entre les sessions et les rechargement de page, permettant ainsi à l'utilisateur de rester connecté. Ensuite la fonction retourne le token afin qu'il puisse être utilisé.

```
if (!customer.error) {  
    let token = customer.jwttoken  
    localStorage.setItem('token', token);  
    return token }  
}
```

Lorsqu' on clique sur le bouton "Se connecter", le gestionnaire d'événement appelle un événement défini dans la constante "onClickButton".

```
<cst.Bouton  
    type="submit" onClick={onClickButton}>Se connecter  
</cst.Bouton>
```

On a une condition qui récupère la valeur de l'identifiant du rôle de l'utilisateur stocké dans le token dans le stockage local. Si cette valeur est égale à 1, l'utilisateur est envoyé vers le tableau de bord de l'administrateur. Si elle est égale à 2, il est envoyé vers le tableau de bord de l'utilisateur, sinon, on aura un message d'erreur.

```
const onClickButton = async(event) => {  
    if (event) {  
        event.preventDefault();  
        let token = await getToken();  
        setTimeout(() => {  
            if (token !== undefined) {  
                let a = 1  
                localStorage.setItem('isAdmin', a);  
                const decoded = jwt_decode(token);  
                const Role = decoded.roleId;  
                if (Role === 1) {  
                    navigate('/Pages/Admin/AdminDashboard');  
                }  
                if (Role === 2) {  
                    navigate('/Pages/User/DashboardUser');  
                }  
            }  
        }, 1000);  
    }  
}
```



```

    }
    } else {
        toast.error("Nom d'utilisateur et/ou mot
de passe erroné");
    }}, 0);
}
};

```

Dans le fichier “BasicPageAdmin” qui définit l’affichage de la barre de navigation et de la top bar du profil Administrateur, j’ajoute un `useEffect` qui va vérifier que l’utilisateur est bien authentifié en tant qu’administrateur.

```

useEffect(() => {
    if (!isAuthenticatedAdmin()) {
        navigate('/');
    }
    if (a === "1") {
        const incrementedValue = Number(a) + 1;
        localStorage.setItem('isAdmin',
incrementedValue.toString());
        window.location.reload();
    }
}, [navigate, a]);

```

La fonction “`isAuthenticatedAdmin()`” du fichier “AuthenticationAdmin” du dossier “Fonction” renvoie un booléen à “true” si la condition est validée.

```

export default function isAuthenticatedAdmin() {
    const token = localStorage.getItem('token');
    if (token) {
        const decoded = jwt_decode(token);
        var Role = decoded.roleId
        if (Role === 1) {
            return true;
        } else {
            //localStorage.removeItem('token');
            return false;
        }
    }
}

```

```

    }
}
}

```

6.3.2. Les jetons d'accès côté back-end

Au niveau du back-end, les informations d'identification de l'utilisateur sont reçues dans le endpoint sous forme de DTO. Elles sont accessibles dans le service qui va effectuer un traitement. Dans le service, on appelle la méthode "Login" du fichier "LoginService". Cette méthode vérifie les informations d'identification fournies par l'utilisateur à l'aide de "l' AuthenticationManager". C'est une partie de Spring Security accessible dans Spring Boot qui gère l'authentification en comparant les informations à celles de la base de données.

On crée une instance de la classe "AuthenticationManagerImpl" puis on appelle sa méthode "authenticate" qui prend trois paramètres, une instance de la classe "UsernamePasswordAuthenticationToken" qui contient les informations de connexion de l'utilisateur, une instance de "passwordEncoder" et une instance de "userRepository".

```

// Check information
new AuthenticationManagerImpl().authenticate(
    new UsernamePasswordAuthenticationToken(
        authenticationRequest.getUsername(),
        authenticationRequest.getPassword(),
        passwordEncoder,
        userRepository);

```

La méthode compare donc les informations en paramètres, aux informations de la base de données à laquelle il a accès grâce à l'instance de "UserRepository".

Une fois que les informations sont vérifiées, on récupère les informations de l'utilisateur enregistrées dans la base de données grâce à une requête du repository. Grâce à la méthode ".orElseThrow(...)" on récupère l'utilisateur s'il existe, s'il n'est

pas trouvé, elle lance une "ElementNotFoundException" avec un message indiquant que l'utilisateur n'existe pas dans la base de données.

```
// Get User in Database
User user = userRepository
    .findByUsernameAndIsDeletedIsFalseAndJobSupprimerIsNullOrJobSupprimerIsFalseAndIsDeletedIsFalseAndUsername(
        authenticationRequest.getUsername(),
        authenticationRequest.getUsername()
    ).orElseThrow(() -> new ElementNotFoundException(
        String.format("Unable to find User [username = %s]",
            authenticationRequest.getUsername())
    ));
```

Si l'utilisateur est trouvé, la méthode "Login" crée une instance de "UserDetails", une interface Spring Security qui contient les détails de l'utilisateur authentifié dont le mot de passe crypté et les rôles.

```
final UserDetails userDetails =
    new org.springframework.security.core.userdetails.User(
        user.getUsername(),
        user.getPassword(),
        user.isEnabled(),
        user.isAccountNonExpired(),
        user.isCredentialsNonExpired(),
        user.isAccountNonLocked(),
        user.getAuthorities());
```

On génère ensuite un jeton JSON web token en appelant la méthode "generateToken" de l'instance de la classe "JwtAuthInputDto" définie dans un autre fichier. Cette méthode prend plusieurs paramètres dont "UserDetails". Elle utilise ces informations pour générer un token (ou jeton d'accès) sécurisé qui sera ensuite utilisé pour authentifier l'utilisateur lorsqu'il effectue des demandes à l'API backend.

```
// Generate token
return jwtTokenUtil.generateToken(userDetails, shortToken,
```

```
user.getRole().getId(),  
user.getFirstName(),  
user.getLastName(),  
user.getId());
```

Au niveau du endpoint, on place le résultat dans la variable appelée token. On le convertit en Dto adapté pour l'envoyer au front.

```
return ResponseEntity.ok(new JwtAuthPublicDto(token));
```

Pour conclure, l'utilisation de ce système est appropriée pour la gestion des authentifications pour ce projet. En effet, contrairement à l'utilisation des sessions, les tokens ne nécessitent pas de stocker des informations sur le serveur. Ils permettent une meilleure séparation entre le front-end et le back-end car toutes les informations nécessaires pour vérifier l'authenticité de l'utilisateur sont incluses dans le token. Ils sont plus légers en termes de gestion des ressources et offrent une meilleure sécurité.

6.3.3. Le toggle

Ma première tâche consistait à intégrer un bouton permettant de masquer ou de révéler le mot de passe sur la page de connexion. J'avais déjà quelques notions de React, car j'avais pris l'initiative de me préparer avant le début de mon stage. J'ai trouvé un tutoriel sur Youtube qui s'est avéré parfaitement adapté à l'effet souhaité ainsi qu'au code déjà en place.

Pour intégrer les icônes, j'ai utilisé le site **React Icons** et les ai importées dans le fichier de connexion. J'ai également introduit un état initial appelé "visible" avec une valeur booléenne "false", de façon à ce que le mot de passe soit masqué par défaut. Cet état est ensuite mis à jour exclusivement par le biais de la fonction "setVisible".

```
const [visible, setVisible] = useState(false);
```

J'ai placé un gestionnaire d'événements "onClick" dans la division concernée, ce qui permet de la rendre cliquable et de lui associer un ou plusieurs événement(s).

Lorsque l'on cliquera sur le contenu de la div (c'est-à-dire sur l'icône), la valeur "setVisible" sera inversée par rapport à sa valeur actuelle. Si elle est à "false", elle passera à "true", et vice versa.

```
<div onClick={() => setVisible(!visible)}>
```

Je mets une condition ternaire dans le composant de l'input dans le DOM virtuel. Le type d'entrée du champ saisi sera défini en fonction de la valeur de "visible" :

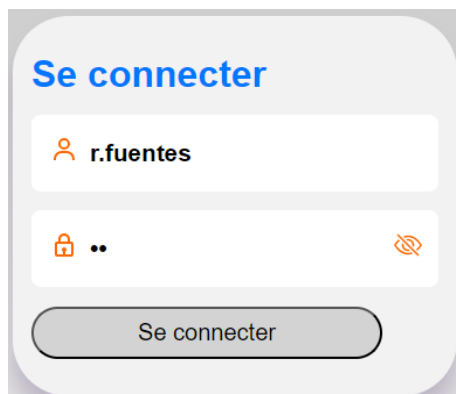
```
type={visible ? "text" : "password"}
```

Lorsque l'état "visible" est défini sur "true", le type sera automatiquement configuré en "text", tandis que s'il est à "false", le type sera attribué à "password", assurant ainsi la fonction de masquage.

J'applique le même système pour l'icône:

```
{visible ?  
<cst.Icône><BsEye/></cst.Icône> :  
<cst.Icône><BsEyeSlash/></cst.Icône>}
```

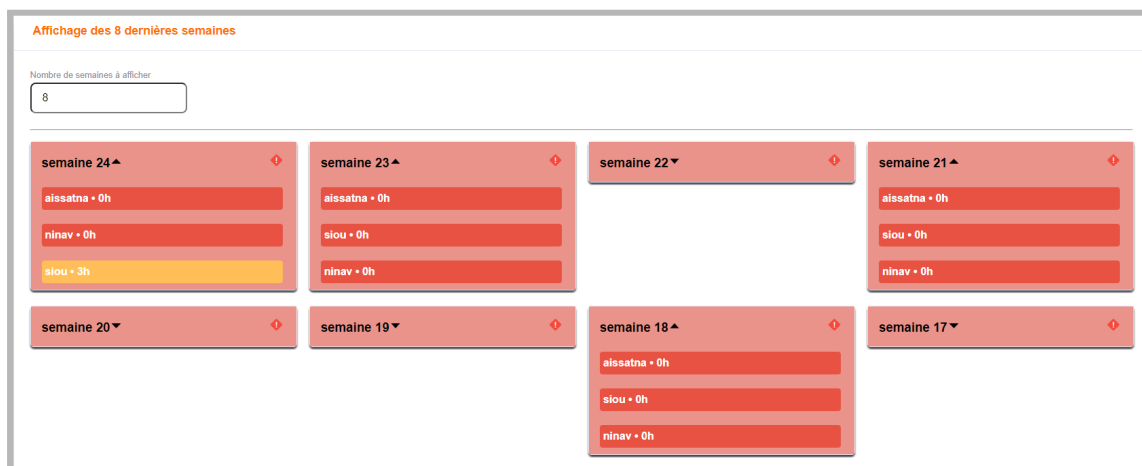
Ainsi, par défaut "visible" est à "false", donc le mot de passe est masqué et c'est l'icône de l'œil barré qui est cliquable. Lorsqu'on clique sur l'icône, le mot de passe devient visible et l'icône cliquable sera un œil simple.



C'est ce qu'on appelle un toggle, une technique fréquemment employée. J'ai utilisé ce mécanisme à plusieurs reprises par la suite dans différents contextes.

6.4. Le tableau de bord du profil administrateur

À l'origine, le tableau de bord de l'administrateur affichait un calendrier sous forme de cartes dépliantes contenant la liste des utilisateurs et leur heures de travail quotidiennes.



Le président de l'entreprise m'a demandé un affichage en semaines qui affiche les heures de travail hebdomadaires.

6.4.1. La partie front du tableau de bord

J'ai décidé de refaire l'affichage des cartes, en effet, modifier le code déjà en place était problématique car cela aurait impacté d'autres fichiers et rendu la compilation impossible. J'ai créé un traitement dans le back-end que je développerai plus en détail. J'ai fait en sorte de récupérer exactement les données qui m'intéressent dans l'endpoint.

J'ai créé le composant React qui contient toutes les cartes représentant les semaines.

```
const renderWeekList = () => (
  <cst.CardListContainer>
    {weekList.weeks.map((week, index) => renderWeek(week,
index))}
  </cst.CardListContainer>
)
```

J’ai appelé mon composant “renderWeekList”. “weekList” fait référence à l’objet que je reçois du back-end, qui contient une liste d’objets “week” appelée “weeks”.

Je parcours donc cet objet et pour chaque élément “week” je vais appeler la fonction “renderWeek” qui est le composant qui gère l’affichage d’une carte. Donc pour chaque “week” une carte s’affichera. On passe “week” en props afin que le composant enfant (c’est-à-dire “renderWeek”) ait accès aux propriétés de l’objet.

J’ai ensuite créé le composant “renderWeek”. Il s’agit d’une carte dépliant, c’est ce que l’on appelle un accordéon. Pour éviter de surcharger l’affichage, les cartes n’affichent que le numéro de la semaine. En cliquant sur cet élément, la liste des utilisateurs s’affichera.

```
const renderWeek = (week, weekIndex) => week.users?.length > 0 ? (
  <Accordion key={`week-${weekIndex}`}
weekNumber={week.weekNumber} >
    {week.users.map((user, userIndex) =>
renderUser(user))}
  </Accordion>
) : null
```

Ici mon composant “renderWeek” contient un autre composant appelé “Accordion”. Il est défini dans un autre fichier. Il représente la partie visible de la carte et nous retrouvons le système de toggle que nous avons déjà abordé. Dans ce cas là, il affiche ou non ses composants enfants (c’est-à-dire les utilisateurs “renderUser”).

“renderWeek” parcourt la liste “users” d’utilisateurs, et pour chaque “user”, il appellera un autre composant “renderUser”.

Dans “renderUser” on associe les valeurs de l’objet du back-end appropriées, aux propriétés du composant pour pouvoir les afficher. C’est grâce aux props que l’on peut accéder à ses valeurs de composant en composant.

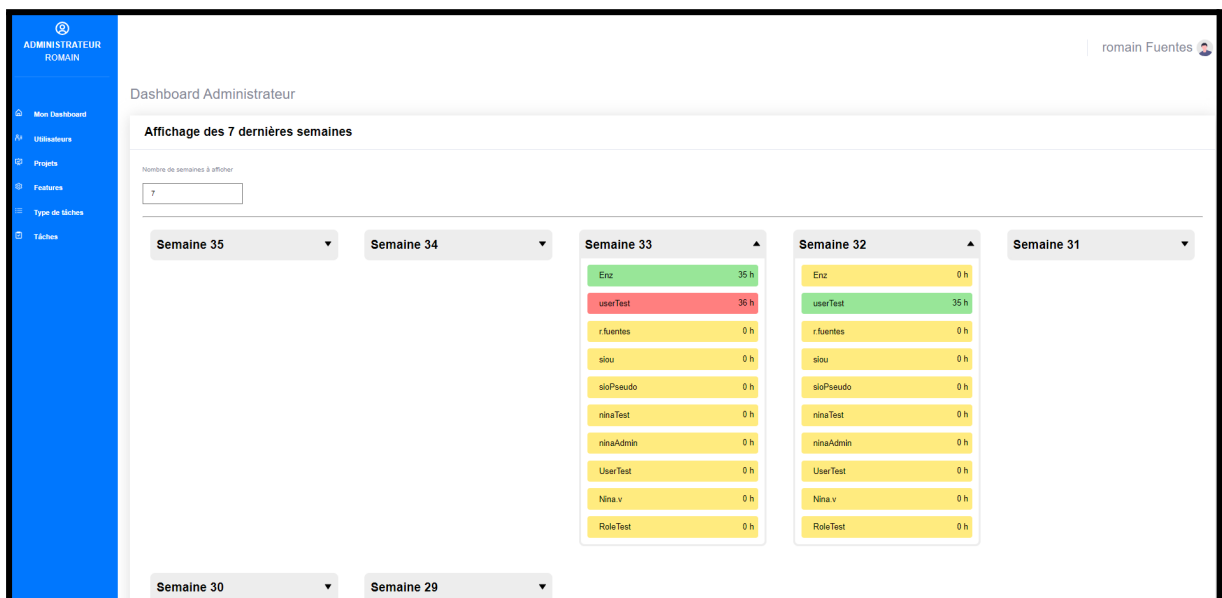
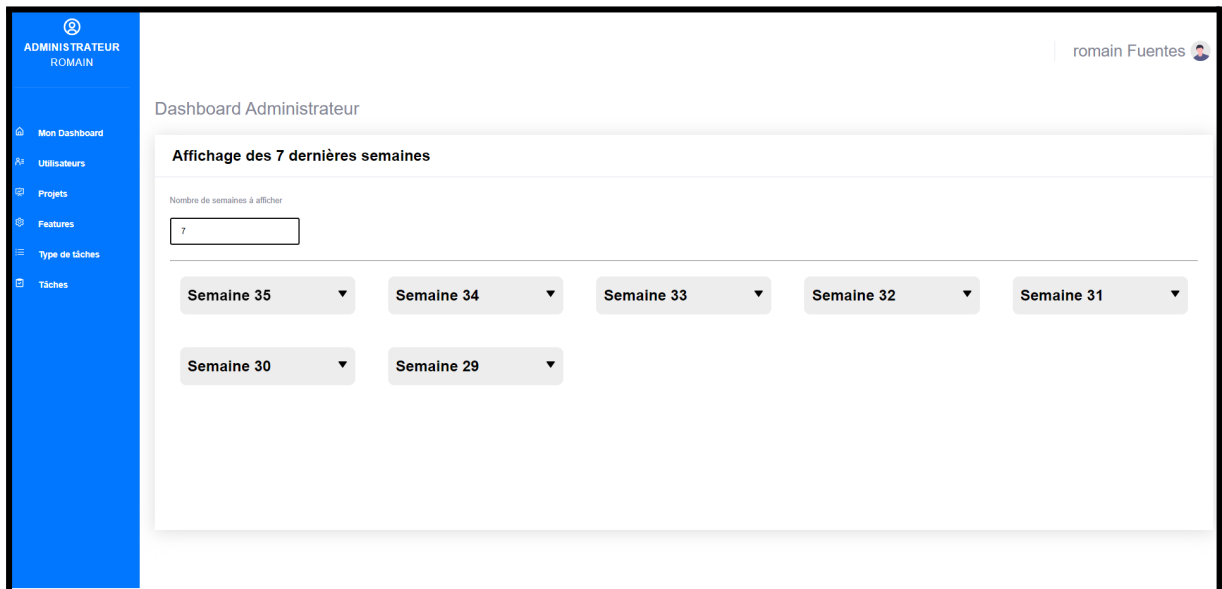
```
const renderUser = ({worktime, userName}) => (  
  <cst.UserContainer  
    key={`user-${userName}`}  
    worktime={worktime}  
    userName={userName}  
    displayRed={worktime > 35}  
    displayGreen={worktime == 35}  
    displayYellow={worktime < 35}>  
    <cst.UserName worktime={worktime} userName={userName}>  
      {userName}  
    </cst.UserName>  
    <div>{worktime} h</div>  
  </cst.UserContainer>  
)
```

Les propriétés “displayYellow”, “displayGreen” et “displayRed” permettent de paramétrer la couleur de fond. En effet, un des avantages de styled components est de pouvoir mettre des conditions dans le CSS très facilement.

```
export const UserContainer = styled.div`  
  background: ${(props) => {  
    if (props.displayRed) {  
      return "rgba(255, 0, 0, 0.4)";  
    } else if (props.displayGreen) {  
      return "rgba(50,255,13, 0.4)";  
    } else {  
      return "rgba(255, 255, 0, 0.4)";}}};  
`
```

De cette façon, la couleur de fond du composant “renderUser” sera jaune, verte, ou rouge selon le total d’heures effectuées par l’employé dans la semaine correspondante.

Le nombre de semaines à afficher est paramétré à 5 par défaut, mais l'utilisateur peut saisir un nombre choisi et l'affichage s'actualise en fonction de cette saisie. C'est l'intérêt des hooks en React. A chaque fois que "numberWeeks" est modifié, le "useEffect" le détecte et envoie la nouvelle donnée au back pour mettre à jour l'affichage.



6.4.2. Le back-end du tableau de bord

Pour cette fonctionnalité, c'est par la partie back-end que j'ai commencé. J'ai d'abord imaginé le DTO que doit recevoir le front.

DTO est l'acronyme de Data Transfer Object. C'est un objet qui permet de transférer des données d'une application à une autre. Ici, c'est la passerelle entre le back-end et le front-end.

Dans ce cas, mon DTO contient lui-même plusieurs DTO. En effet, on souhaite recevoir un objet ("WeekListDto") contenant une liste ("WeekDto") représentant les semaines identifiées par leur numéro de semaine calendaire, et une liste des utilisateurs ("UserWorkTimeDto"). Ce dernier DTO contient le nom de l'utilisateur, et la durée de ses heures de travail cumulées pour la semaine en question.

Le back-end effectue son traitement en fonction d'un nombre de semaines saisi dans le front, il attend donc une donnée. Ceci doit être paramétré dans le endpoint.

```
@PostMapping("/weeks")
@PreAuthorize("hasPermission(returnObject, 'export')")
public ResponseEntity<WeekListDto> postWorkTaskByWeek(@RequestBody
WeekTaskInputDto weekTaskInputDto) {
    return
workTaskService.getAllWorkTaskByWeek(weekTaskInputDto); }
```

Le endpoint "/weeks" attend un payload de type "WeekTaskInputDto" pour donner une réponse de type "UserTaskListDto". Le traitement se fait dans la méthode "getAllWorkTaskByWeek".

Dans le service, l'algorithme calcule une date de début et une date de fin en fonction du nombre de semaines donné.

```

@Override
public ResponseEntity<WeekListDto>
getAllWorkTaskByWeek(WeekTaskInputDto weekTaskInputDto) {
    Date nowDate = Date.from(LocalDateTime.now()
        .atZone(ZoneId.systemDefault())
        .toInstant());

    Date startDate = Date.from(LocalDateTime.now()
        .minusWeeks(weekTaskInputDto.getWeekCount())
        .atZone(ZoneId.systemDefault())
        .toInstant());

```

Il crée une liste des numéros de semaines de cette plage de dates.

```

List<Long> weekNumbers = new ArrayList<>(); for(int i = 0; i <
weekTaskInputDto.getWeekCount(); i++) {
    int weekNumber = startDate.toInstant()
        .atZone(ZoneId.systemDefault())
        .toLocalDate().plusWeeks(i).get(ChronoField.ALIGNED_WEEK_OF_YEAR);

    weekNumbers.add(Long.valueOf(weekNumber));
}

```

Il inverse la liste car on souhaite avoir un calendrier rétrospectif.

```

Collections.sort(weekNumbers, (d1,d2) ->{
    return (int) (d2 - d1);
});

```

Il récupère la liste de tous les utilisateurs, puis la liste de toutes les tâches.

```

WeekListDto weekListDto = new WeekListDto();
List<User> userList = userRepository.findAll();

```

```

for (Long weekNumber : weekNumbers) {
    WeekDto weekDto = new WeekDto();
    weekDto.setWeekNumber(weekNumber);

    for(User user : userList){
        UserWorkTimeDto userWorkTimeDto = new UserWorkTimeDto();

        List<WorkTask> userTasks = user.getWorkTasks();
    }
}

```

Il filtre ensuite la liste des tâches par utilisateur et par numéro de semaine.

```

List<WorkTask> filteredTasks = userTasks.stream()
    .filter(task -> task.getWorkDay().toInstant()
        .atZone(ZoneId.systemDefault())
        .toLocalDate()
        .get(ChronoField.ALIGNED_WEEK_OF_YEAR)
        == weekNumber)
    .collect(Collectors.toList());

```

Il fait la somme de la durée des tâches pour chaque semaine et attribue chaque valeur correspondante à la propriété du DTO approprié.

```

double totalWorkTime = filteredTasks.stream().mapToDouble(value ->
    value.getWorkTime()).sum();

    userWorkTimeDto.setUserName(user.getUsername());
    userWorkTimeDto.setWorktime((int)totalWorkTime);
    weekDto.getUsers().add(userWorkTimeDto);
}

```

Le tout est encapsulé dans le DTO de réponse attendu.

```

weekListDto.getWeeks().add(weekDto);
}

return new ResponseEntity<>(weekListDto, HttpStatus.OK);
}

```

J'ai utilisé Postman pour tester mon API.

Postman est une application permettant de tester des API. Elle permet par exemple de simuler l'appel qu'aurait fait le front-end au endpoint pour voir si la réponse est reçue. On peut lui envoyer des données en payload. Lorsque l'on travaille sur le front-end on peut également simuler une API dans un fichier qu'on appellera "mock".

6.5. La page Utilisateurs

La page "Utilisateurs" du profil Administrateur doit afficher la liste des utilisateurs sous forme de tableau. On doit avoir le nom, le prénom, l'email, le pôle, le rôle, le statut et un bouton permettant de modifier ou désactiver l'utilisateur. À l'origine, il n'y avait pas de colonne "pôle" et plusieurs appels à l'API étaient fait dans le front. En effet, comme les informations à afficher proviennent de différentes tables, le front appelait toutes les tables concernées et faisait le tri depuis le front.

Pour faire les choses de manière plus conventionnelle, j'ai modifié le code afin de ne faire qu'un seul appel, le tri est fait dans le back-end. Sur le même principe que la partie précédente, on reçoit un DTO contenant exactement les informations attendues.

6.5.1. La gestion de la navigation

React.js ne possède pas de système de navigation par défaut. Il existe une librairie qui permet d'en ajouter un à notre application. Elle se nomme **React-router-dom** et permet d'ajouter la navigation dans le DOM. Elle permet de revenir en arrière ou de rentrer directement une url dans la barre du navigateur. Il faut s'assurer qu'elle est présente dans le fichier package.json. On l'importe ensuite dans le fichier voulu.

On utilise le hook `useNavigate` fourni par la bibliothèque `react-router-dom` qui permet de gérer la navigation programmatique dans une application React. Dans mon fichier "UserListe.js" qui affiche la liste des utilisateurs, j'utilise ce hook pour rediriger l'utilisateur vers la page de modification dès qu'il clique sur la ligne de l'utilisateur.

J'importe le hook `useNavigate`.

```
const navigate = useNavigate();
```

Je crée une fonction `redirectToUserModif` qui pointe vers la page `UserModif` et lui envoie l'identifiant de l'utilisateur dans l'url.

```
const redirectToUserModif = () => {  
  navigate(`/Pages/Admin/User/UserModif/${user.id}`);  
};
```

Je place ensuite cette fonction dans un gestionnaire d'événement `onClick` à l'emplacement voulu, ici, sur chaque cellule du tableau. Ainsi, lorsque je clique, je suis redirigée vers la page en question.

C'est grâce à `useNavigate` que les routes sont définies dans le fichier `App.js`.

6.5.2. La gestion du filtre de recherche

Sur cette page nous avons un filtre de recherche, il était également géré dans le front, on avait une condition qui traitait d'abord le cas dans lequel il y avait une recherche saisie puis le cas dans lequel il n'y en avait pas. Cela entraînait une duplication excessive de code.

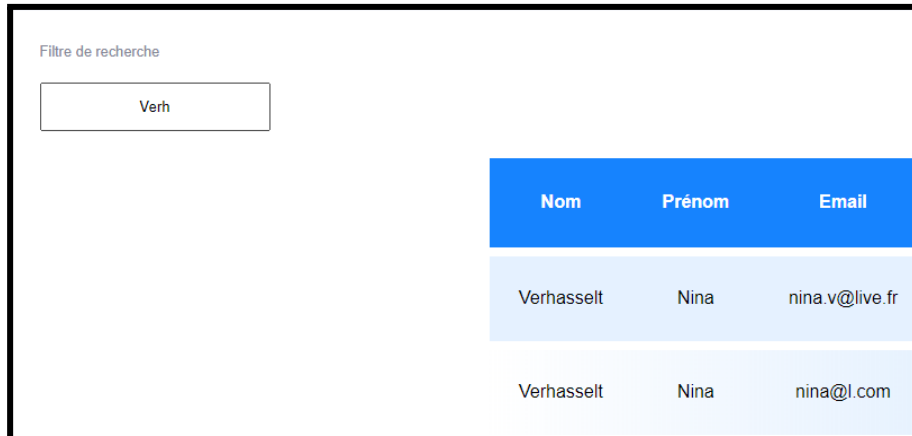
Le filtre peut se faire dans le back-end directement dans la requête à la base de données. C'est ce que je développerai dans un second temps.

Dans le front, on crée un hook `useState` appelé recherche que l'on initialise avec une chaîne de caractère vide pour le cas où aucune recherche n'est saisie. Dans le DOM virtuel, on met un `onChange` dans l'input de la recherche.

```
onChange={ (e) => setRecherche(e.target.value) }
```

Lorsqu'il y aura une saisie, la valeur de "recherche" sera mise à jour. Le "useEffect" détecte la modification et envoie la nouvelle chaîne au back dans son appel.

Après son traitement, le back-end renvoie le résultat filtré en fonction de la chaîne de caractères saisie.



The screenshot shows a web interface with a search filter and a table of results. The filter is labeled "Filtre de recherche" and contains a text input with the value "Verh". The table has three columns: "Nom", "Prénom", and "Email". It displays two rows of data, both for "Verhasselt" and "Nina".

Nom	Prénom	Email
Verhasselt	Nina	nina.v@live.fr
Verhasselt	Nina	nina@l.com

6.5.3. Le filtre de recherche côté back-end

Comme mentionné précédemment, le filtrage de recherche sur cette page est pris en charge côté serveur. Le côté front-end fait référence au endpoint "users/userList" et envoie les entrées de recherche via l'URL. Dans le contrôleur, il est spécifié qu'une valeur de type chaîne de caractères est attendue.

```
public ResponseEntity<UserListDto> getUserList(@RequestParam
String search) {
```

La méthode "getUserList" place le résultat de la requête "findAllUsersBySearch" dans une liste.

Dans la requête, on sélectionne toutes les données de la table Users que l'on appelle "u" pour plus de simplicité.

```
@Query("FROM User u " +
```

On fait une jointure avec la table “Role” et la table “Job” car nous allons récupérer des données de ces tables également.

```
"LEFT JOIN Role r ON r.id = u.role.id " +  
"LEFT JOIN Job j ON j.id = u.job.id " +
```

Puis on met en place un filtrage en fonction de la chaîne de caractère reçue en paramètre.

```
"WHERE " +  
    "LOWER(u.firstName) LIKE '%' || LOWER(?1) || '%'" +  
    "OR LOWER(u.lastName) LIKE '%' || LOWER(?1) || '%'" +  
    "OR LOWER(r.name) LIKE '%' || LOWER(?1) || '%'" +  
    "OR LOWER(u.email) LIKE '%' || LOWER(?1) || '%'" +  
    "OR LOWER(u.username) LIKE '%' || LOWER(?1) || '%'" +  
    "OR LOWER(j.name) LIKE '%' || LOWER(?1) || '%'"
```

Le champ “firstName” de la table “u” (alias précédemment donné à la table Users) est comparé avec la valeur du paramètre “?1”. La fonction “LOWER()” est utilisée pour convertir le texte en minuscules, ce qui permet de rendre la recherche insensible à la casse. Le “LIKE” indique qu’une correspondance partielle est recherchée, et les “%” entourant “LOWER(?1)” signifient que la valeur de “?1” peut apparaître n’importe où dans le champ “firstName”. Les lignes suivantes suivent le même principe.

Ensuite on convertit le résultat en DTO approprié dans le service pour l’envoyer au front via le endpoint.

6.5.4. Le popover du bouton Action

Initialement, dans la dernière colonne du tableau utilisateur, il y avait un bouton difficilement identifiable qui ouvrait une fenêtre pop-up proposant deux actions: Modifier et Supprimer. Le bouton “Modifier” menait à la page de modification et

“Supprimer” supprimait l'utilisateur de l'affichage puis dirigeait vers la page du tableau des utilisateurs.

Lors d'une réunion avec Arnaud Fichot et la cheffe de projet, il m'a été demandé de modifier le comportement de cette page.

J'ai modifié le style du bouton afin qu'il soit plus facilement identifiable.

Grâce à la bibliothèque **Radix**, j'ai mis en place un popover pour une meilleure ergonomie. Lorsque l'on clique sur le bouton, une fenêtre s'ouvre juste en dessous et affiche les actions possibles. Lorsque l'on clique sur “Modifier” on est redirigé vers la page de modification. Lorsque l'on clique sur “Désactiver”, on appelle la fonction “handleTriggerClick” qui prend en paramètre “user” (l'utilisateur de la ligne sélectionnée).

```
<cst.DropdownMenuLabelCustom
onClick={ () => handleTriggerClick(user) }>
    Désactiver
</cst.DropdownMenuLabelCustom>
```

Cette fonction stock les informations de l'utilisateur passé en paramètre dans un tableau “selectedUser” et passe “openModal” à “true”.

```
const handleTriggerClick = (user) => {
    setSelectedUser(user);
    setOpenModal(true);
};
```

Une fenêtre de dialogue en position absolue est mise en place dans le body du DOM. Elle ne sera visible que si sa propriété “open” prend la valeur booléenne “true”, c'est-à-dire, lorsque l'on clique sur l'option “Désactiver”.

```
<AlertDialog.Root open={openModal}
onOpenChange={ (value) => !value && setOpenModal(false) }>
```

“selectedUser” est utilisé pour afficher le nom de l'utilisateur dans le fenêtre de dialogue:

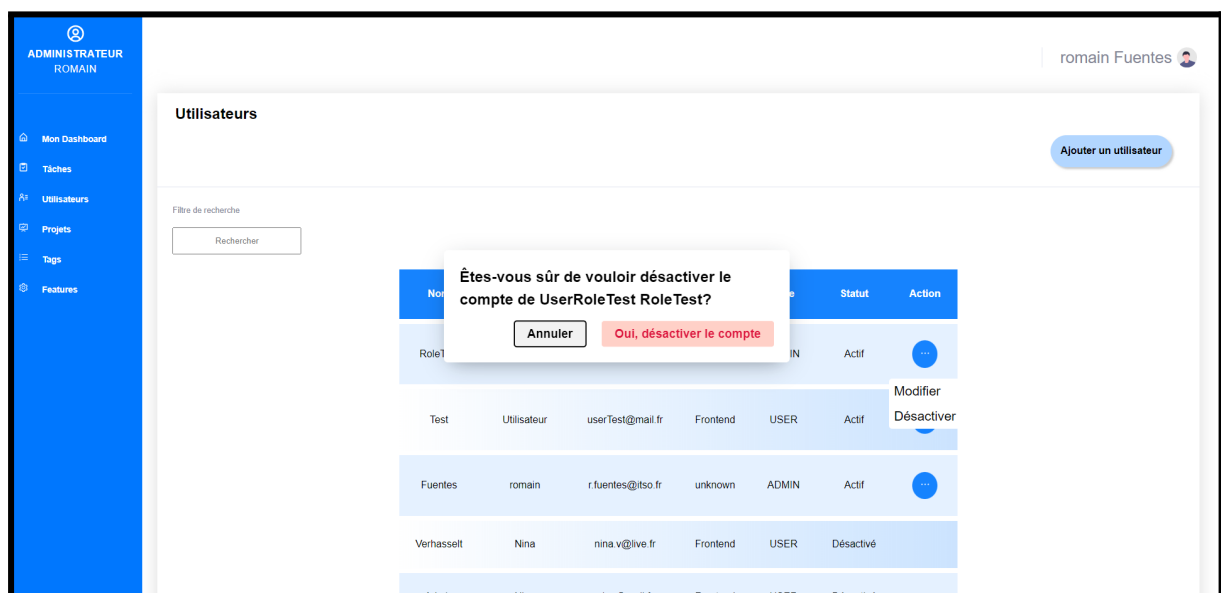
```
Êtes-vous sûr de vouloir désactiver le compte de  
{selectedUser.firstName} {selectedUser.lastName}?
```

Et mettre à jour “selectedId” ce qui déclenche la désactivation de l'utilisateur.

```
<AlertDialog.Action className="modalButton confirmButton"  
onClick={() => setSelectedId(selectedUser.id)}>
```

En effet, grâce à un “useEffect”, dès que “selectedId” est modifié, un nouvel appel à l'API est fait et des modifications sont faites au niveau de la base de données.

J'ai fait en sorte que le bouton d'action soit visible uniquement si le compte n'est pas désactivé. On souhaite garder une visibilité sur les utilisateurs qui ne font plus partie de l'entreprise même si plus aucune action n'est possible.



La bibliothèque Radix a été d'une grande aide pour obtenir le comportement recherché pour cette page. Elle est facile à implémenter, bien documentée et offre un large choix d'animations couramment utilisées et facilement personnalisables.

J'ai calqué ce système sur la page projet, qui fonctionnait avec la même logique que l'ancienne page Utilisateurs.

6.5.5. Le CRUD de la page Utilisateurs

Dans cette partie je vais aborder la notion le CRUD car c'est une approche fondamentale dans le développement d'applications qui impliquent la gestion de données.

Le **CRUD** est l'acronyme des opérations de base utilisées dans les applications de gestion de données (Create, Read, Update, Delete). De manière générale, on fait une requête à l'API selon l'opération voulue pour afficher des données ou mettre à jour la base de données.

On peut créer un nouvel utilisateur depuis le profil administrateur, il s'agit de l'opération **CREATE**. Après avoir récupéré les informations saisies dans le formulaire du front-end via une requête "post", le back-end ajoute le nouvel utilisateur grâce à la fonction `save()` qui est une méthode spécifique Spring:

```
userRepository.save(newUser);
```

L'affichage du tableau des utilisateurs correspond à l'opération **READ**. Nous avons vu le fonctionnement de cette opération avec le filtrage. C'est une requête "get", on récupère les données sans les modifier pour les afficher.

Pour modifier les données d'un utilisateur, on exécute l'opération **UPDATE** en utilisant une requête "put" et la fonction "save()" également.

Dans cette application, nous n'avons pas d'opération **DELETE** à proprement parler. Par exemple, dans l'entité "User" nous avons une propriété "isDeleted" qui contient un booléen. C'est une approche appelée "soft delete" (suppression douce) ou "logical delete" (suppression logique). C'est une pratique courante qui permet entre autres de restaurer les données et d'effectuer des statistiques et analyses si besoin.

On suivra donc le même processus que pour un update en modifiant uniquement ce booléen.

J'ai pris la page utilisateurs pour exemple, mais ces opérations sont utilisées de la même manière sur d'autres tables.

6.6. La page projet

Cette page reprend le fonctionnement et le style de la page Utilisateurs. Sa particularité réside dans la récupération des données. En effet, les projets affichés sont récupérés depuis l'API de Jira. La connexion à l'API se fait depuis le back-end.

Il est possible d'appeler une API externe depuis le front, mais ici je souhaite mettre à jour ma base de données avec les données récupérées depuis Jira. De plus, quand j'ai essayé de tester cette méthode, j'ai rencontré une erreur liée aux CORS.

CORS est l'acronyme de Cross-Origin Resource Sharing (Partage des ressources entre origines multiples). C'est une politique de sécurité mise en place par les navigateurs web pour contrôler les demandes d'accès aux ressources provenant d'un domaine (ou origine) différent de celui à partir duquel la page est chargée. En général, il suffit d'ajouter des en-têtes HTTP afin de permettre à l'agent utilisateur d'accéder aux ressources du serveur.

Depuis le back-end, la connexion est simple, dans cette partie, je vais développer les étapes qui m'ont permis de mettre à jour les données de la table Projet avec les données de l'application Jira. L'appel au niveau du front est un appel classique, je présenterai donc uniquement le back-end de cette fonctionnalité. Puis, j'aborderai la notion de CRON et expliquerai sa mise en place.

6.6.1. L'appel à l'API de Jira

Après une étude minutieuse de la documentation de Jira qui est très complète et bien faite, j'ai créé un jeton d'API depuis le compte de l'application. C'est un token

généralisé pour sécuriser l'accès aux données. J'ai créé un contrôleur qui contiendra tous les endpoints associés à l'API de Jira.

J'ai ensuite créé un helper "JiraApiUtility" qui permet d'établir la connexion et pourra être utilisée pour différents appels. Je place les identifiants dans des variables, il s'agit de l'url de base du compte Jira, l'adresse mail associée, et le jeton d'accès.

```
public class JiraApiUtility {  
    private static final String jiraUrl =  
"https://ninaverhasselt84.atlassian.net/rest/api/3";  
    private static final String jiraUsername =  
"nina.verhasselt84@gmail.com";  
    private static final String jiraPassword =  
"ATATT3xFfGF0taDhKhx8R-jR9o3PLzDRQIZPhLP8P47Aw-5LtrBUUAQnc-L0_  
O2iJy_Rji9uA2UnKPHOQVA157E5yWmeFmVE8mxxP-lb4kmDVWNqGfNd26K7Tv1  
luBE8ubraBADOWuhbEzEIAeiUmKv1XGi5pM9fqCFDr8o5VxttOHAhewquJ00=BD319AE  
2";
```

J'utilise une authentification Basic HTTP, c'est-à-dire par nom d'utilisateur et mot de passe. J'utilise "Base64" pour encoder ces informations avant de les inclure dans l'en-tête "Authorization".

```
public ResponseEntity<String> callJiraApi(String url) {  
    String auth = Base64.getEncoder().encodeToString((jiraUsername  
+ ":" + jiraPassword).getBytes());  
    HttpHeaders headers = new HttpHeaders();  
    headers.set("Authorization", "Basic " + auth);
```

Ensuite, j'ai créé une instance de "RestTemplate". C'est une classe Spring qui facilite l'interaction avec les services Web. Elle est principalement utilisée pour consommer des API RESTful. J'ai utilisé la méthode "exchange" de "RestTemplate" pour effectuer la requête HTTP en passant l'url et les en-têtes en paramètre. La réponse sera envoyée sous forme de chaîne de caractères.

```
RestTemplate restTemplate = new RestTemplate();
return restTemplate.exchange(url, HttpMethod.GET, new
HttpEntity<>(headers), String.class);
```

Je peux maintenant faire différents appels en passant le complément de l'url correspondant aux informations que je recherche en paramètre.

Pour recueillir tous les projets, d'après la documentation de Jira, je dois ajouter "/project" à l'url de base.

```
public ResponseEntity<String> projectsCall() {
    String projectUrl = jiraUrl + "/project";
    return callJiraApi(projectUrl);}
```

6.6.2. La désérialisation

À ce stade, nous connaissons l'utilisation des DTO. L'objet reçu n'étant pas très complexe, j'ai choisi d'utiliser cette méthode. J'ai donc créé un DTO "JiraProjectListDto" qui contient entre autres une liste de "JiraProjectDto" et tous deux respectent exactement la structure de l'objet reçu.

Une première méthode "getJiraProjects()" utilise l'appel défini précédemment, parcourt l'objet JSON reçu pour le désérialiser en objet "JiraProjectListDto" grâce à la classe ObjectMapper de la bibliothèque Jackson.

```
public ResponseEntity<List<JiraProjectDto>> getJiraProjects() {
    ResponseEntity<String> jiraResponse =
        jiraApiUtility.projectsCall();
    List<JiraProjectDto> jiraProjects;
    try {
        ObjectMapper objectMapper = new ObjectMapper();
        TypeReference<List<JiraProjectDto>> projectListType =
            new TypeReference<List<JiraProjectDto>>() {};
        jiraProjects = objectMapper.readValue(jiraResponse.getBody(),
            projectListType);
        return new ResponseEntity<>(jiraProjects, HttpStatus.OK);
    }
```

```

    } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

```

Une autre méthode “addProjects()” vérifie si le projet n’est pas déjà présent dans la base données et l’ajoute.

J’ai ajouté l’annotation `@JsonIgnoreProperties(ignoreUnknown = true)` à mon “JiraProjectDto” suite à un dysfonctionnement causé par une évolution de l’objet reçu. En effet, lorsque l’on travaille avec une API, les données peuvent évoluer et inclure des propriétés supplémentaires. Afin que cela n’affecte pas la désérialisation de l’objet Java, cette annotation permet d’ignorer les propriétés inconnues lors de cette étape.

6.6.3. Le cron

Afin que la base de données se mette à jour automatiquement avec les données de Jira, j’ai mis en place un cron.

Le système **cron** permet de programmer l’exécution automatique de tâches à des moments spécifiques.

Ici, je déclenche la méthode “addProjects” toutes les dix secondes afin d’avoir une liste à jour. J’utilise l’annotation Spring `@Scheduled` qui permet de planifier une tâche automatique.

```

@Scheduled(fixedRate = 1000000) // 10000 milliseconds = 10
seconds
public void executeAddProjectsEvery10Seconds() {
    ResponseEntity<List<Project>> response = addProjects();}

```

Il existe d’autres alternatives pour effectuer la mise à jour, par exemple l’utilisation des webhooks conviendrait parfaitement.

6.6.4. Les RGPD

En ce qui concerne les RGPD (Règlement Général sur la Protection des Données, l'application offre la possibilité de supprimer les données personnelles d'un utilisateur tout en le conservant pour les statistiques. Seul son identifiant sera conservé, le reste de ses informations seront remplacées par des "X".

Certains points obligatoires n'ont pas encore été mis en place mais sont à ajouter avant le déploiement de l'application. Même lorsqu'il s'agit d'une application interne, l'utilisateur doit être informé sur le traitement de ses données personnelles.

Dans notre cas, c'est l'administrateur qui ajoute un nouvel utilisateur, on pourrait prévoir l'envoi automatique d'un mail d'information, contenant un document de type politique de confidentialité, à chaque fois qu'un nouvel utilisateur est ajouté.

Lorsque l'utilisateur entre la durée et la nature des tâches réalisées, il s'agit également de données personnelles. On peut ajouter un premier niveau d'information en fin de formulaire et renvoyer à une politique de confidentialité ou une page vie privée sur l'application. On peut également mettre un lien qui envoie vers ce même document en pied de page sur toutes les pages de l'application.

Ce document doit impérativement informer les utilisateurs des points suivants:

- La raison pour laquelle sont collectées ses données
- Ce qui autorise légalement à collecter ses données
- Qui y a accès
- Combien de temps elles seront conservées
- Les modalités selon lesquelles les utilisateurs peuvent exercer leurs droits (moyen de contact)

Le traitement de ces données devra figurer dans le registre des activités de traitement de données qui participe à la documentation de la conformité.

7. Présentation du jeu d'essai de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)

Les jeux d'essai sont cruciaux dans le processus de développement et de test de logiciels, car ils permettent aux développeurs et aux testeurs de s'assurer que le logiciel fonctionne correctement, ils identifient les erreurs, les bugs et les vulnérabilités potentielles.

Je vais présenter le jeu d'essai réalisé pour tester l'ajout d'un nouvel utilisateur, les marqueurs de couleurs des utilisateurs dans les cartes hebdomadaires et la récupération des projets du compte Jira.

7.1. Test de l'ajout d'un nouvel utilisateur

données en entrée	données attendues	données obtenues
Nouvel utilisateur: Enzo Boudart, nom d'utilisateur: Enz, mot de passe: password.	L'utilisateur doit s'afficher dans le tableau.	L'utilisateur s'affiche dans le tableau.

L'ajout d'un nouvel utilisateur fonctionne et il est maintenant possible de se connecter à l'application via ce profil.

7.2. Test des marqueurs des utilisateurs

données en entrée	données attendues	données obtenues
5 tâches de 7h pour l'utilisateur Enzo Boudart	Enzo Boudart doit s'afficher en vert sur la semaine 33.	Enzo Boudart s'affiche en vert sur la carte de la

sur les jours de la semaine 33		semaine 33.
6 tâches dont la durée cumulée fait 36h pour l'utilisateur userTest sur les jours de la semaine 33.	userTest doit s'afficher en rouge sur la carte de la semaine 33.	userTest s'affiche en rouge sur la carte de la semaine 33.

Les marqueurs de couleurs fonctionnent.

7.3. Test de la connexion à l'API de Jira

données en entrée	données attendues	données obtenues
Identifiants du compte Jira de l'entreprise.	Les projets présents dans Jira doivent s'ajouter dans la base de données et donc dans le tableau de la page projets.	Les projets du compte Jira s'affichent.

La connexion à l'API de Jira et l'insertion automatique des projets du compte Jira dans la base de données fonctionnent.

8. Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité

Le monde de l'informatique est en constante évolution, c'est pourquoi il est important de s'informer afin de rester à jour et compétitif. Il en est de même pour les failles de sécurité. Les attaquants malveillants ne cessent de déceler les failles des technologies qui tentent d'y remédier dès que la faille est déclarée. Il est donc indispensable d'effectuer une veille technologique générale et sur la sécurité lorsque l'on crée des applications.

J'ai déjà parlé de sécurité avec les failles XSS et les injections SQL. L'utilisation des jetons d'accès est aussi un élément qui contribue à la sécurité de l'application. Ici, je parlerai principalement des sources fiables et des méthodes pour se tenir à jour.

Pour effectuer une recherche efficace, les mots clés doivent être choisis avec soin. Les recherches en Anglais peuvent apporter des réponses plus pertinentes.

Mots clés: Spring Boot vulnerabilities, React vulnerabilities.

La date de la source doit être récente (2 ans maximum). La nature de la source doit être vérifiée, en général les documentations officielles des technologies offrent des sources fiables et mettent en garde quand une partie du contenu est obsolète.

OWASP (Open Web Application Security Project) est un site fiable qui analyse les dernières vulnérabilités, et fournit des outils permettant d'analyser la sécurité et identifier les vulnérabilités d'un projet. Le site de l'ANSSI (Agence nationale de la sécurité des systèmes d'information) en est une autre. Il est important de croiser les informations même lorsque une source est vérifiée.

9. Description d'une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, à partir de site anglophone

Pour établir la connexion à l'API de Jira, j'ai cherché dans la documentation officielle d'Atlassian.

➤ Mots clés: Jira API Connexion

Je retrouve mes éléments de recherche dans la deuxième proposition qui vient du site Atlassian (<https://developer.atlassian.com/server/jira/platform/rest-apis/>).

Sur cette page anglophone, j'ai trouvé les informations nécessaires pour établir la connexion.

10. Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la traduction en français effectuée par le candidat sans traducteur automatique (environ 750 signes).

10.1. Extrait de la documentation Atlassian

URI structure

Jira REST APIs provide access to resources (that is, data entities) via URI paths. To use a REST API, your application makes an HTTP request and parse the response.

The Jira REST API uses JSON as its communication format and the standard HTTP methods like `GET`, `PUT`, `POST`, and `DELETE`. URIs for Jira REST API resource have the following structure:

```
http://host:port/context/rest/api-name/api-version/resource-name
```

Currently there are two API names available, which will be discussed later on this page:

- `auth`: – for authentication-related operations.
- `api`: – for everything else.

The current API version is `2`. However, there is also a symbolic version called `latest` that resolves to the latest version supported by the given Jira instance.

As an example, if you wanted to retrieve the JSON representation of issue `JRA-9` from Atlassian's public issue tracker, you would access:

```
https://jira.atlassian.com/rest/api/latest/issue/JRA-9
```

10.2. Traduction de l'extrait

Structure de l'URI

Les API REST de Jira offrent un accès aux ressources (c'est-à-dire aux entités de données) via les URI. Pour utiliser une API REST, votre application envoie une requête HTTP et analyse la réponse.

L'API REST de Jira utilise le format JSON comme format de communication et les méthodes HTTP standard telles que GET, PUT, POST, and DELETE. Les URI de l'API REST de Jira ont la structure suivante:

```
http://host:port/context/rest/api-name/api-version/resource-name
```

Actuellement, il existe deux noms d'API disponibles, qui seront expliqués ultérieurement sur cette page:

- auth: - pour les opérations liées à l'authentification.
- api: - pour tout le reste.

La version actuelle de l'API est la version 2. Cependant, il existe également la version symbolique appelée `latest` qui correspond à la dernière version compatible de l'instance de Jira donnée.

Par exemple, si vous souhaitez récupérer la représentation JSON du ticket JRA-9 depuis le suivi public des tickets d'Atlassian, vous accéderiez à l'adresse:

```
https://jira.atlassian.com/rest/api/latest/issue/JRA-9
```

11. Conclusion

Je n'avais que très peu de connaissances sur les technologies utilisées à mon arrivée dans l'entreprise. Il m'a fallu un temps d'adaptation pour me familiariser avec le projet et comprendre l'ensemble. Ma persévérance, ma passion pour le

développement et l'aide précieuse des développeurs en poste m'ont permis d'y arriver.

Le lead développeur a démissionné juste après mon arrivée, ses tâches ont été réparties entre les membres de l'équipe. Je n'ai pas eu de directives précises avant la réunion du 4 Août. J'ai sollicité l'équipe lorsqu'ils avaient du temps à m'accorder mais je n'ai pas sollicité le président et la MOA par peur de les déranger. C'est eux qui ont provoqué la réunion pour faire un point. A l'avenir, je n'hésiterai pas à solliciter la direction et prendre des initiatives.

En additionnant ce projet à ceux réalisés au cours de ma formation, j'ai pu acquérir l'ensemble des compétences attendues pour le titre professionnel de développeur web et web mobile.

Concernant la première activité type du titre, j'ai réalisé des maquettes graphiques et des wireframes. J'ai apporté des améliorations UX et UI au projet ITSO Time. J'ai réalisé des applications responsives et j'ai rendu une page du projet ITSO Time adaptable uniquement dans le but de pratiquer car cet outil n'est pas destiné à être utilisé sur mobile. J'ai appris à utiliser React qui permet de réaliser des interfaces dynamiques et réactives. J'ai également créé un blog grâce à wordpress.

Pour ce qui est de la seconde activité, j'ai créé des bases de données en passant par les modèles de conception et compris le fonctionnement des entités persistantes et la création de base de données en code first comme pratiquée dans mon entreprise d'accueil. J'ai manipulé les données en concevant des traitements dans la logique métier de l'application.

J'ai une appétence particulière pour les technologies que j'ai pratiquées sur ce projet à savoir React et Java. Cette période de stage s'est avérée extrêmement gratifiante à la fois d'un point de vue technique et humain. J'ai également apprécié travailler en PHP et Symfony pendant ma formation. J'adorerai monter en compétence sur ces technologies et en apprendre d'autres telles que C#, Python et bien d'autres.