**Une image contenant texte, graphisme, Graphique, Police

Description générée automatiquement**

**RAPPORT DE PROJET**

**Thomas HAMON**

Alternance Concepteur Développeur d’Applications

2022/2023

**Tuteur d’alternance :**

Mickaël HEBERT

**Formateur responsable de promotion :**

Denis SANCHEZ

1. REMERCIEMENTS

Avant tout développement sur cette expérience professionnelle il apparaît opportun de commencer ce rapport de Projet par des remerciements. Je souhaite remercier tout particulièrement mon tuteur de stage Mickaël Hebert, pour ses nombreuses qualités humaines, et pour la confiance qu’il m’a accordé durant année d’alternance.

Je tiens également à remercier Angelo BOURSIN, pour son accompagnent technique sur le projet « Plan De Charge », et pour tous les conseils en développement qu’il a pu m’apporter.

Je remercie en outre toutes les personnes qui m’ont accompagné cette année chez Adventiel, pour leur patience et leur aide précieuse.

Dans une tout autre mesure je tiens à remercier ma compagne Mathilde dont le soutien indéfectible et les sacrifices m’ont permis de mener à bien ma reconversion professionnelle.

Enfin, je remercie mes deux filles Alix et Charlotte pour leurs encouragements dans cette période charnière de ma vie professionnelle, même si parfois, elles comprenaient difficilement que leur papa aille toujours à l’école.

1. INTRODUCTION

Mon rapport présente le projet "Plan de Charge", sur lequel j’ai travaillé dans le cadre de ma formation en tant que concepteur développeur d'applications chez Adventiel et stagiaire à l’ENI de Chartre de Bretagne. Adventiel est une Entreprise de Services du Numérique (ESN) spécialisée dans le domaine de l'agriculture et de l'agroalimentaire.

A la suite de formation de « développeur web et web mobile », j’ai souhaité parfaire mes connaissances en continuant sur le parcours « concepteur développeur d’applications ». Durant ma période stage, j’ai eu la chance d’être recruté par l’entreprise Adventiel, afin d’y effectuer mon année d’alternance.

Outre la joie d’avoir trouvé une entreprise pour continuer mon parcours, c’est le secteur dans lequel s’est spécialisé Adventiel qui m’a séduit. En effet le domaine de l’agriculture comme énormément d’autre domaines s’équipe depuis maintenant plusieurs années de solutions numériques. L’idée de travailler dans le domaine informatique qui ne compte que quelques décennies d’existence, pour le mettre au service de l’agriculture, pilier millénaire de notre civilisation, m’inspirait énormément.

Au cours de cette année d’alternance, j’ai principalement travaillé sur le projet « Plan de charge » (PDC). Ce projet est porté par mon tuteur, Mickael Hebert, Chef du centre de compétences. PDC est un projet interne à Adventiel, Il a pour utilisation, l’organisation du travail au sein de l’ESN. Il est principalement utilisé pour repartir la charge de travail aux différents intervenants présents chez Adventiel. Ainsi les Managers chargent l’emploi du temps hebdomadaire du personnel productif qui peut en retour consulter son plan de charge. Par Ailleurs le projet permet aux managers la consultation de plusieurs indicateurs en lien avec la charge de travail des intervenants afin de piloter aux mieux les différents projets en cours. Il permet aussi de saisir les congés pour qu’ils soient répercutés sur le plan de charge.

Depuis quelques mois, PDC permet aux managers de gérer les « présences » pour les projets nécessitant la présence permanente d’au moins un des membres de l’équipe projet. En effet certains projets en production peuvent nécessiter une intervention rapide de la part de l’équipe de développement. C’est pour s’assurer de la disponibilité des membres d’un projet que cette fonctionnalité a été implémentée.

Je vous propose donc au fil de ce rapport de vous présenter le projet « Plan De Charge » et l’implémentation de la gestion des présences sur laquelle travaillé, et de manière plus générale, mon expérience durant cette année d’alternance.

1. LISTE DES COMPETENCES DU REFERENTIEL QUI SONT COUVERTES PAR LE PROJET
2. RESUME DU PROJET (en anglais)

During this year at Adventiel, I had multiple opportunities to work on the « Plan de charge » project.

Adventiel is a company specialized in digital services for the agriculture and agri-food industry.

The "Plan de charge" project was initiated in 2020 by Mickaël Hebert, the head of the competence center. PDC is an internal application developed by Adventiel, with the goal of replacing the Excel files previously used for human resource management in the company's various projects.

The application is used by all employees, managers, and supervisors to visualize and distribute workloads. Managers utilize it to assign tasks to team members, consult multiple indicators to facilitate resource management, and manage permanents for sensitive projects.

All users can access and view their workloads and assigned permanents. Additionally, all users are required to input their holidays information.

The project is developed using Angular for the front-end and Java/Spring Boot for the back-end. Data persistence is ensured through a PostgreSQL database. Project deployment across different environments is automated using Jenkins pipelines.

For several weeks, I specifically worked on the design and implementation of the Presence/Shift Management feature. During this time, I participated in creating the screen mockup, defining use cases, and evolving the database structure. Then, I contributed to developing all the necessary functionalities on both the back-end and front-end of the application while ensuring its security.

1. PRESENTATION DE L’ENTREPRISE

Adventiel avant d’être une ESN (Entreprise de Services du numérique) spécialisée dans le domaine de l'agriculture et de l'agroalimentaire est une ARSOE (Association Régionale de Services aux Organismes d'Élevage). Depuis plus de 50 ans, elle accompagne les entreprises des filières agri/agro dans leur transition vers des modèles plus durables et performants en proposant des solutions technologiques innovantes et adaptées.

Adventiel conçoit et développe des solutions numériques sur mesure pour répondre aux besoins des entreprises agricoles et agroalimentaires. Que ce soit pour améliorer la productivité des exploitations, faciliter le travail des techniciens sur le terrain, optimiser la gestion des activités ou simplifier les démarches administratives, Adventiel offre une expertise complète pour la transformation digitale de ses clients.

L'entreprise s'est également développée grâce à l'acquisition de deux sociétés spécialisées :

* SGPI, un éditeur de logiciels pour les entreprises de collecte et de transformation laitière, qui a rejoint Adventiel en 2016.
* 2DB Conseils, connue sous la marque Innovagro, qui développe des solutions logicielles dédiées aux organismes de contrôle et de certification des AOC et IGP viticoles, acquise en 2018.

Adventiel propose une large gamme de services pour accompagner ses clients dans leur transformation numérique :

* Services numériques sur-mesure : conception et développement de solutions adaptées aux besoins spécifiques des entreprises.
* Conseil : expertise pour définir les besoins en solutions digitales et accompagner les projets de transformation.
* Hébergement et infogérance : sécurisation des données, disponibilité des services, optimisation des systèmes d'information.
* Centre de services : support technique et fonctionnel aux utilisateurs, gestion des demandes, formation.

**Quelques chiffres** :

* Chiffre d'affaires de 18 millions d'euros en 2020.
* Plus de 600 heures consacrées à l'innovation.
* 185 salariés passionnés.
* Plus de 200 clients satisfaits.
* 120 systèmes d'information hébergés dans le datacenter de niveau Tier 3.
* Indice de l'égalité professionnelle entre les hommes et les femmes de 90/100 en 2021.
* 150 projets réalisés chaque année.

L'équipe de développement d'Adventiel, basée principalement sur le site de Pacé près de Rennes, compte environ 80 développeurs spécialisés dans différents langages. Chaque développeur fait partie d'une ou plusieurs équipes projet, placées sous la responsabilité de Mickaël Hebert, Chef du centre de compétence. Cette équipe est composée en majorité de développeurs expérimentés et de quelques développeurs juniors, travaillant ensemble pour concevoir et mettre en œuvre les solutions numériques proposées de l'entreprise.

5. Le Projet

1. Présentation

Le projet Plan de Charge (PDC) est un projet interne d’Adventiel, visant à transformer et améliorer la manière dont la planification des ressources est gérée au sein de l'entreprise. Auparavant, cette tâche était effectuée manuellement à l'aide d'un fichier Excel, ce qui pouvait entraîner des difficultés en termes de suivi, de coordination et de partage des informations.

Afin de remédier à ces problèmes, l'entreprise a développé l'application web PDC, qui offre une plateforme centralisée pour gérer la planification des ressources de manière digitale. Cette application est accessible à tous les acteurs de l'entreprise et aux ressources externes.

Chaque intervenant est associé à un rôle, un nom, un prénom, un trigramme, une fonction et un site, ce qui facilite l'identification et la gestion des ressources.

La planification est centrale dans le fonctionnement de PDC et regroupe les différentes entités telles que les responsables de projet, les chefs d'équipe, les clients, les contrats et les fonctions. Chaque planification est associée à une charge en jours répartie sur une ou plusieurs semaines du planning. Le plan de charge est ensuite calculé en faisant la somme de toutes les charges attribuées à un intervenant chaque semaine.

Dans le projet « Plan de charge », une attention particulière est également accordée à la gestion des absences. Une planification spécifique est prévue pour les absences des intervenants, ce qui permet de mieux anticiper les disponibilités et de planifier les ressources en conséquence.

La gestion des présences est une fonctionnalité récente, ayant pour finalité de s’assurer de la disponibilité d’une personne compétente sur un projet sensible, nécessitant une intervention rapide en cas d’anomalie sur les environnements de productions. Ainsi les managers peuvent positionner des intervenants sur l’ensemble du calendrier paramétrable, et d’être alerter visuellement sur l’écran des présences en cas de non-disponibilité de l’intervenant positionné, ou en cas d’absences de disponibilité des membres d’un projet à une date donnée. Un intervenant de rôle « utilisateur » peut à son niveau consulter les dates auxquelles il a été positionné.

Différents rôles et accès sont définis dans le « Plan de charge ». Les administrateurs et superviseurs, ont un accès complet à toutes les fonctionnalités de l’application. La configuration de l’application est uniquement accessible à l’administrateur à savoir Mickaël Hebert. Les managers, quant à eux, peuvent créer des planifications et des absences, exporter les données et consulter les planifications de l'ensemble des intervenants. Les utilisateurs, représentant les collaborateurs, ont la possibilité de consulter leur planning individuel et de soumettre des demandes d'absence personnelle, et consulter les dates de présence qui leurs sont affectées.

Le projet Plan de Charge (PDC) chez Adventiel marque une transition vers une gestion optimisée des ressources grâce à une application dédiée. Cette solution digitale permet de centraliser les informations, de faciliter la coordination et la communication entre les acteurs de l'entreprise, tout en offrant une vision claire et précise de la planification des ressources.

1. Expression des besoins

L'expression des besoins dans le cadre d'un projet web est une étape fondamentale pour définir les attentes et les objectifs du système à développer. Elle consiste à articuler de manière claire et concise les fonctionnalités, les contraintes et les exigences du projet. En comprenant les besoins des clients, on peut formuler des spécifications précises qui serviront de guide tout au long du processus de développement. Cette expression des besoins permet d'identifier les principales fonctionnalités à mettre en place et les contraintes techniques à prendre en compte. Cette partie cruciale de la vie d’un projet est sous la responsabilité de la « MOA ».

Dans le cadre du projet PDC, Mickael Hebert prend le rôle d’utilisateur final, client, MOA et product owner.

La gestion des présences est la principale fonctionnalité sur laquelle j’ai pu intervenir durant cette année d’alternance.

Les demandes de Mickael Hebert pour fonctionnalité étaient :

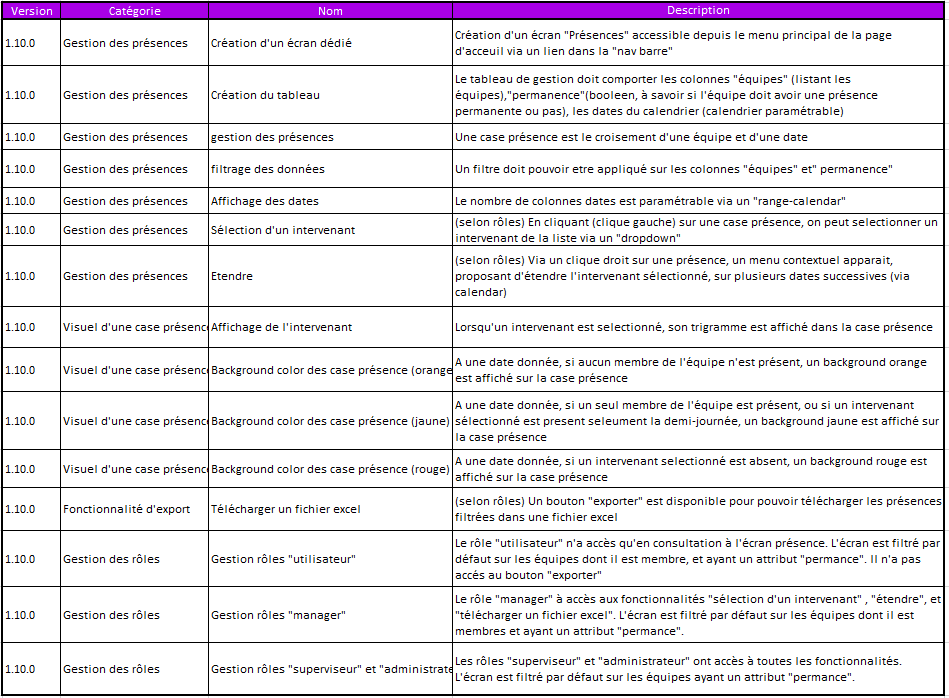
* + - * La gestion des présences doit se faire dans un écran dédié
      * L’écran des présences doit être présenté sous forme d’un tableau/calendrier
      * Le tableau doit croiser les dates et les équipes
      * Ainsi une case du tableau représentant une présence est la jonction entre une équipe et une date
      * Pour les rôles utilisateur l’écran est uniquement consultable
      * Les managers peuvent éditer les présences uniquement sur les équipes dont ils sont responsables.
      * Les superviseurs et administrateurs peuvent éditer toutes les présences
      * L’écran doit être visuel, ainsi des couleurs différentes apparaissent en background des cases *présence* quand il ne reste plus qu’un membre d’une équipe à une date donné, qu’aucun membre d’une équipe n’est disponible à une date donnée, ou que l’intervenant positionné est absent le jour de la présence, avec une nuance s’il est absent uniquement la demi-journée.
      * Un bouton « exporter » doit être disponible pour les rôles autres qu’utilisateur. Il permet de télécharger au formant excel, les informations affichées à l’écran.

1. Spécifications fonctionnelles

Dans un projet informatique, les spécifications fonctionnelles ont la charge de décrire, préciser les fonctionnalités de l’application

Les spécifications fonctionnelles sont, dans la manière où nous les représentons ci-dessus, une énumération des fonctionnalités indispensables de la fonctionnalité de gestion des présences.

Les spécifications fonctionnelles ci-dessous sont celles concernant la fonctionnalité de gestion des présences :



6. SPECIFICATIONS TECHNIQUES

1. Présentation

Être intégrer au projet « Plan de Charges » a été pour moi l’occasion de participer à l’une des fonctionnalités majeures du projet. La fonctionnalité de gestion des présences à due être implémentée sur toutes les couches de l’application et a nécessitée des modifications en base de données. A cette occasion j’ai participé à la rédaction des documents liés à la conception de cette fonctionnalité.

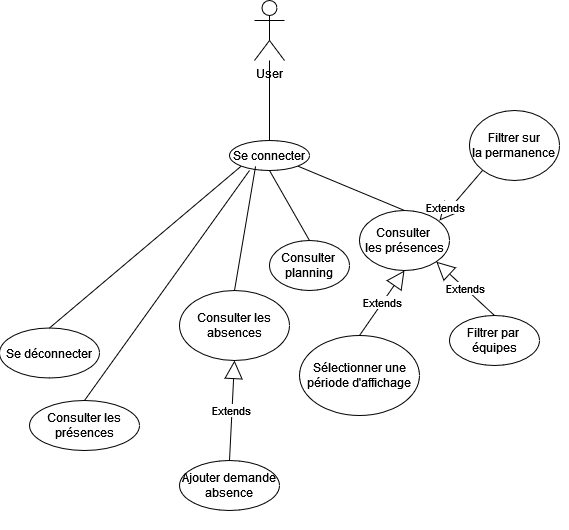
J’ai donc rédigé, en collaboration avec les autres membres de l’équipe et l’aide d’Angelo Boursin (lead dev), les use cases, le modèle conceptuel de données et les maquettes de l’évolution de « Plan de charge ».

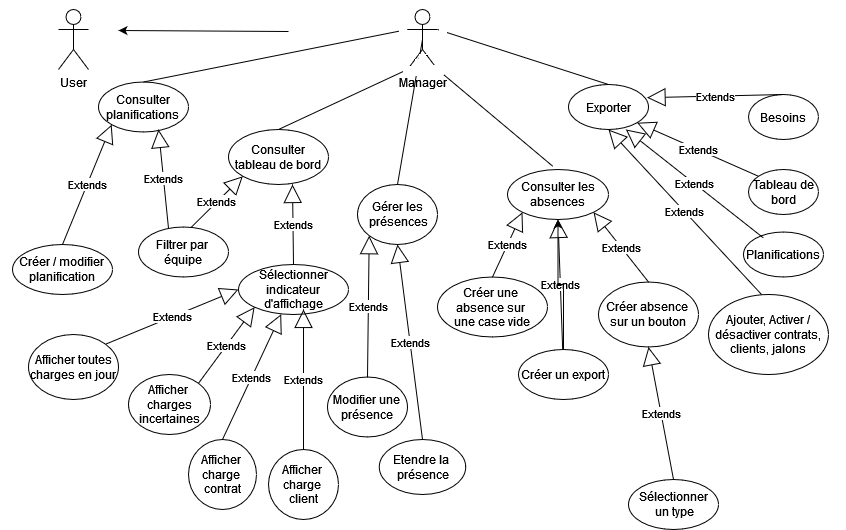
Je vous propose dans les onglets suivants de découvrir ces documents ainsi que les ressources logiciel et outils utilisés pour la réalisation de l'application.

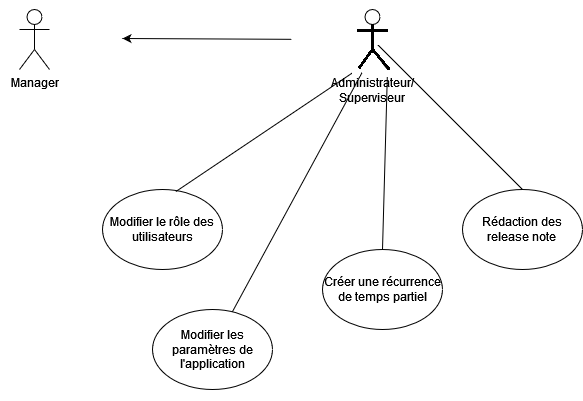
1. Les use case

Les uses case représentent les scenarios d’utilisation de l’application. Ils aident les développeurs, les concepteurs et les parties prenantes à visualiser comment le système sera utilisé dans le monde réel et à identifier les fonctionnalités et les interactions. Ils présentent les différents rôles que peuvent revêtir les utilisateurs et les fonctionnalités auxquels ils ont accès en fonction de ces rôles.

Ci-dessous, les use cases de l’ensemble de l’application « Plan de charge » :







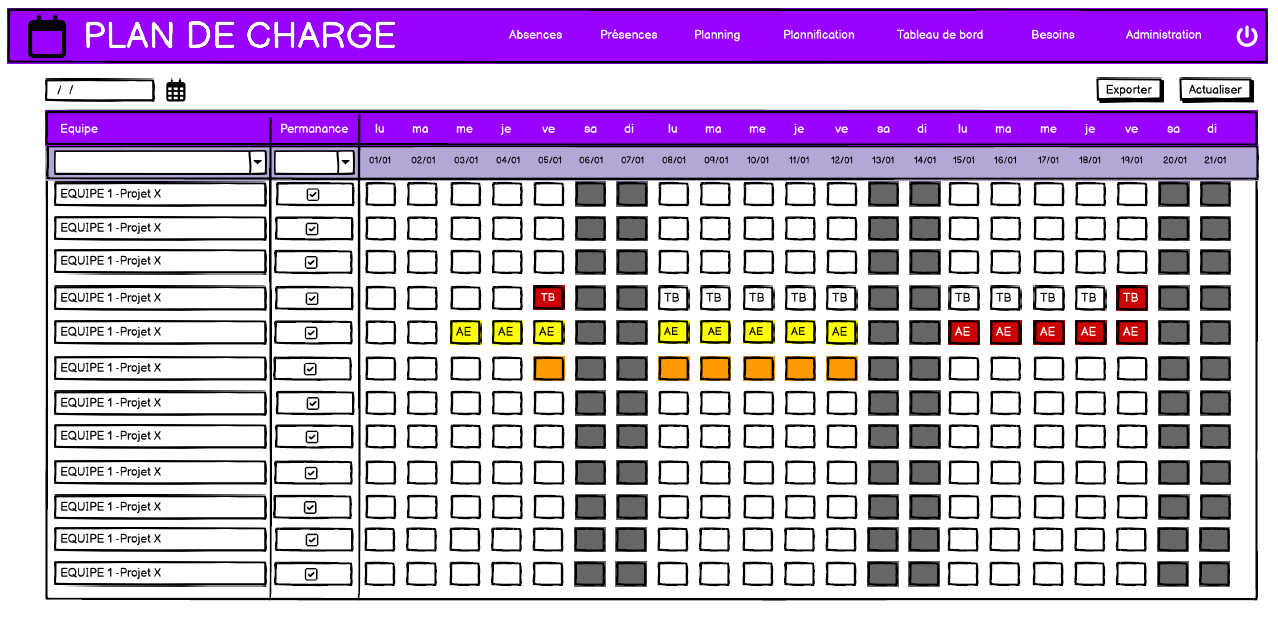
1. Les maquettes

Les Maquettes sont une ébauche graphique de l’interface utilisateur de l’application.

Elles permettent aux commanditaires de l’application et aux équipes de développement de s’accorder sur la chartre graphique générale de l’application et sur l’organisation de l’information avant le début du développement.

Dans le cas de PDC, j’ai participé à la conception de la maquette de l’écran Présence (via le logiciel Balsamiq mockup) avec les membres de l’équipe, en tenant compte des spécifications fonctionnelles, pour les faire valider par Mickaël Hebert avant le début du développement de la partie front-end.

Ci-dessous, la maquette réalisée pour l’écran de Présence :



Le modéle conceptuel de données (MCD)

Le MCD (Modèle Conceptuel de Données) est une représentation graphique des relations entre les données d’une application. C'est un outil de modélisation qui permet de décrire de manière abstraite la structure des données qui seront utilisées par une application. Le MCD est utilisé principalement dans le processus de conception de bases de données relationnelles, où il sert de support pour créer le schéma de base de données.

Dans le cadre de ce rapport, nous avons récréer le MCD à partir de l’extrait du schéma de base de données, présenté dans la prochaine partie.

Ci-dessous, le MCD concernant un extrait du schéma de base de données :

Une image contenant texte, capture d’écran, Police, carte de visite

Description générée automatiquement

1. Le schéma de base de données

Le schéma de base de données est la représentation graphique des tables de la base de données et des relations qui existent entre elles.

Le schéma de base de données (modèle physique de données) est très proche du diagramme de classe d’une application. Ainsi les tables représentent en base les entités (business object) de l’application, et les contraintes d’intégrités référentiels (foreign keys), ici modélisé par les flèches entre les tables, représentent les liens et les cardinalités existant entre les entités de l’application.

En réalité, une table de l’extrait du schéma de base de données ne serait pas présente dans le diagramme de classe. La table « pdc\_equipes\_membres », ne doit sa présence en base de données, que de la nécessité d’avoir une table d’association entre deux tables qui ont une relation « many to many ».

Ci-dessous vous pourrez observer un extrait du schéma de base de données de l’application PDC :

Une image contenant texte, capture d’écran

Description générée automatiquement

1. Ressources logicielles et environnements techniques

Le projet PDC a été élaboré grâce au concours de plusieurs langages de programmation, framework, librairie, et applications de support.

Ainsi pour ce projet, j’ai utilisé les ressources suivantes :

**Pour la partie Front-end :**



**Angular** : Angular est un framework client open source, basé sur TypeScript. Il facilite la création de "Single Page Applications", offrant ainsi une expérience utilisateur fluide sans rechargement de pages fréquent. Il permet une séparation claire entre les données, l'interface visuelle et les actions, favorisant ainsi la maintenabilité. Dans mes réalisations Front-End, j’ai fait usage des technologies Html5, Css3 et TypeScript. Je vais vous donner dans la présentation de mes réalisations, une description plus détaillée du fonctionnement des différentes couches et structures d'une application Angular.



**Rxjs**: bibliothèque de programmation réactive qui utilise des Observables pour simplifier la gestion des événements asynchrones. Les Observables sont utilisés pour représenter un flux ou une source de données qui peut évoluer au fil du temps. Ils sont généralement créés à partir d'événements.



**PrimeNg :** PrimeNG est une bibliothèque de composants d'interface utilisateur riche conçue spécifiquement pour Angular. Elle offre une large gamme de widgets prêts à l'emploi, tels que des tableaux de données, des calendriers, des graphiques, etc. Avec une documentation complète et une communauté active, PrimeNG facilite le développement d'applications web modernes et est largement utilisé dans la communauté Angular. L’application PDC utilise cette librairie dans de nombreux écrans, notamment pour l’affichage des tableaux et gérer le filtrage des colonnes.

**Pour la partie Back-end :**

**Java :** Java est un langage de programmation polyvalent, reconnu pour sa fiabilité, sa sécurité et sa portabilité. Il offre une syntaxe claire et prend en charge la programmation orientée objet. Avec sa gestion automatique de la mémoire et la possibilité de développer des applications multithread, Java est idéal pour créer des applications robustes. Sa vaste bibliothèque standard fournit des fonctionnalités étendues, ce qui en fait un choix privilégié pour le développement d'applications d'entreprise, de systèmes embarqués, d'applications Android et de services Web. De plus, grâce à sa plateforme Java Virtual Machine (JVM), les programmes Java peuvent s'exécuter facilement sur différents systèmes d'exploitation sans nécessiter de modifications majeures du code source.



**Spring Boot** : Spring Boot est un framework Java populaire et largement utilisé pour le développement d'applications web. Il se distingue par l'utilisation intensive d'annotations, qui simplifient la configuration et la gestion des composants. Grâce à ces annotations, les développeurs peuvent définir rapidement et efficacement des points d'entrée, des services, des contrôleurs, des requêtes et bien d'autres aspects de l'application. En offrant une approche basée sur les conventions, Spring Boot permet de gagner du temps et de se concentrer davantage sur la logique métier, ce qui en fait un choix prisé pour créer des applications robustes et hautement modulaires.

**JUnit 5 :** JUnit 5 est un framework de test unitaire pour Java, offrant des fonctionnalités avancées et une syntaxe plus moderne par rapport à ses versions précédentes. Il prend en charge les tests paramétrés, les assertions étendues, les hooks de cycle de vie et d'autres fonctionnalités qui facilitent l'écriture de tests efficaces et lisibles. JUnit 5 est devenu un choix populaire parmi les développeurs Java pour garantir la qualité et la fiabilité des applications.

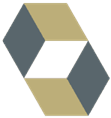


**Mockito :** Mockito est un framework de test open source pour Java. Le framework permet de créer facilement des objets simulés (mocks) pour les dépendances des tests unitaires.



**Maven :** Maven est un outil de gestion de projet largement utilisé en Java qui simplifie la construction, le packaging et la gestion des dépendances. Grâce à son modèle de projet basé sur des conventions, Maven permet une gestion efficace des dépendances et des tests, contribuant ainsi à l'automatisation des tâches de développement et à l'amélioration de la productivité des développeurs

**Pour la partie gestion des données :**



**Hibernate/jpa :** Hibernate est un orm (object-relational mapping) open source gérant la persistance des objets en base de données relationnelle. permettant aux développeurs de manipuler les données de manière orientée objet en les associant à une base de données relationnelle. En utilisant les annotations de la norme JPA (Java Persistence API), Hibernate facilite la gestion des opérations de persistance et offre des fonctionnalités avancées telles que la gestion des relations, le lazy loading et la génération automatique de requêtes SQL.

**PostgreSQL** : PostgreSQL est un système de gestion de base de données relationnelle (SGBD)

**Flyway :** Flyway est un outil de migration de base de données open source qui permet aux développeurs de gérer efficacement l'évolution de la structure de leur base de données. En utilisant des scripts SQL versionnés, Flyway facilite le suivi des changements apportés à la base de données et permet leur déploiement automatique dans différents environnements, garantissant ainsi la cohérence et la traçabilité des modifications

**Les applications supports :**



**Intellij** : IntelliJ IDEA est un environnement de développement intégré (IDE) populaire et puissant pour le développement de logiciels en Java et d'autres langages de programmation. Il offre des fonctionnalités avancées telles que l'autocomplétion, la refactoring, le débogage, la prise en charge des frameworks populaires et une intégration étroite avec les outils de versionning. Nous avons utilisé Intellij pour le développement de la partie Front-end et Back-end.

 ****

**Git/Gitlab :** Git est un système de contrôle de version distribué largement utilisé pour le suivi des modifications de code source et la collaboration entre développeurs. Il permet de gérer facilement les branches, les fusions et les historiques des projets.

GitLab, quant à lui, est une plateforme de gestion de dépôts Git qui offre des fonctionnalités supplémentaires telles que l'intégration continue, la gestion des tickets, la planification de projets et le suivi des problèmes. Il facilite le travail collaboratif et la gestion complète du cycle de vie des projets informatiques.



**Jenkins :** Jenkins est un outil d'intégration continue open source qui permet d'automatiser les tâches de construction, de test et de déploiement des logiciels. Il offre une interface conviviale, une large compatibilité avec différents outils et frameworks, et facilite la mise en place de pipelines de livraison continue pour garantir une livraison de logiciel fiable et rapide.

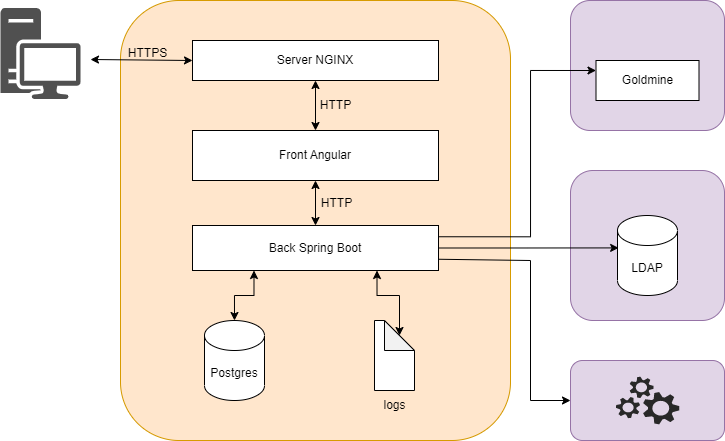
1. Architecture du projet

« Plan de charge » est une application web construite selon une structure modulaire et une architecture de type MVC à l’instar du schéma ci-dessous :

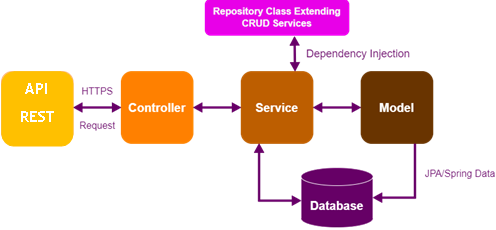


L'application repose sur une architecture conçue pour segmenter l'application en plusieurs modules distincts. Chaque module a une responsabilité spécifique et fonctionne de manière autonome, tout en étant connecté aux autres modules.

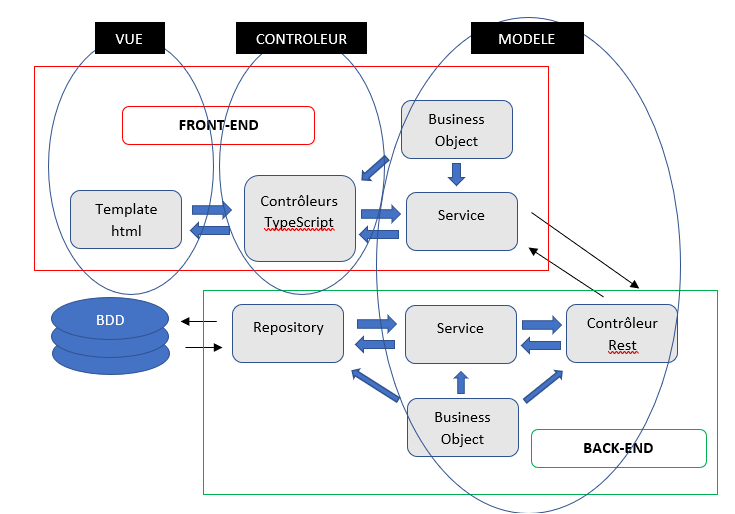
Le projet comprend un backend Spring Boot chargé de synchroniser les données à partir de Goldmine (une application d'annuaire interne), d’authentifier les utilisateurs via un serveur LADP et expose une API REST utilisée par un frontend Angular. La figure suivante présente un résumé de l'architecture générale choisie pour la mise en œuvre du projet PDC.



Ci-dessous le détail de la structure de la partie back-end en Spring-Boot :



Ci-dessous, mon interprétation du schéma MVC de notre application, sur la globalité des couches :



Dans la partie suivante je vous invite à découvrir l’architecture de PDC en suivi la réalisation d’une fonctionnalité à travers l’ensemble des couches de l’application.

6B. GESTION DE PROJET

La gestion de projet d'une application web est une étape cruciale pour assurer le succès du développement et du déploiement de l'application. Cela implique une planification minutieuse, une coordination efficace des ressources et une communication claire entre les différents acteurs du projet. Dans ce chapitre, je présente la méthodologie du travail adoptée pour la gestion de projet PDC.

La méthodologie de gestion de projet

Un projet doit être découpé en tâches pour faciliter sa planification. La méthodologie de gestion de projet correspond à la façon dont les tâches sont organisées au fur et à mesure du temps. Grâce à la disponibilité de Mickael Hebert, et pour avoir une décomposition fine des tâches de notre projet, ainsi qu’un retour fréquent sur le développement, une approche Agile SCRUM simplifiée a été adopté pour la gestion de du projet PDC.

L’approche Agile « SCRUM »

Les modèles traditionnels de gestion de projet, tels que le cycle en V ou le cycle en cascade, suivent une approche séquentielle où chaque étape doit être complétée avant de passer à la suivante, jusqu'à la fin du projet. Cependant, il peut être difficile de les appliquer strictement, et c'est souvent pendant la phase de codage que des problèmes sont identifiés. Par exemple, la méthode en V fixe les besoins en amont, ce qui peut la rendre rigide face aux changements.

En revanche, l'approche Agile propose de réduire, voire d'éliminer, cet effet tunnel en offrant une meilleure visibilité et en impliquant le client dès le début jusqu'à la fin du projet. Elle adopte un processus itératif et incrémental, considérant que les besoins ne peuvent être figés. L'approche Agile, notamment la méthode Scrum, favorise l'adaptation aux changements en intégrant des cycles de développement courts appelés "sprints" et en encourageant une communication constante entre l'équipe de développement et le client.

En mettant l'accent sur la collaboration et la flexibilité, l'approche Agile permet de mieux répondre aux besoins changements du client tout au long du projet. Elle favorise une livraison plus rapide de fonctionnalités fonctionnelles, tout en offrant des opportunités d'ajustement et d'amélioration continues. Cela permet d'éviter les retards et les problèmes majeurs qui pourraient survenir lors des phases finales du projet, comme c'est souvent le cas avec les modèles traditionnels.

Ci-dessous je vous présente quelques aspects essentiels de la méthode Agile « SCRUM » :

**User Story :**

Une US est une description simple précisant le contenu d’une fonctionnalité à développer. Chaque US est attribué à un membre de l’équipe lors du sprint.

Au sein d’Adventiel, la rédaction d’une US est formalisée par le schéma suivant :

« En tant que <qui> je veux <quoi> afin de <pourquoi> ». Le pourquoi n’est pas obligatoire mais il permet de cerner davantage le besoin du métier.

Les US ont été rédigées par Mickaël.

Angelo Boursin ainsi que l’équipe de développement avaient pour rôle de les découper en tâches et de les chiffrer.

**Le sprint planning :**

Le sprint planning avait lieu toutes les 3 à 4 semaines et permet à l’équipe d’inclure dans le sprint en préparation les User Story (US) qui seront traitées.

Le sprint planning est généralement dirigé par le Scrum Master (Angelo Boursin dans notre cas), qui est responsable de faciliter le processus. Le Product Owner (ici Mickaël Hebert) est également présent pour apporter des informations sur les objectifs et les priorités du produit. L'équipe de développement participe activement à la réunion en discutant des éléments du « backlog » (présenté ci-après), en partageant des estimations de charges de travail et en collaborant pour définir les tâches à réaliser pendant le sprint. L'objectif est d'aboutir à un consensus sur le contenu du sprint et de s'assurer que tous les membres de l'équipe comprennent clairement les attentes et les objectifs du sprint.



**Le Backlog :**

Le backlog, représente l'ensemble des fonctionnalités, des tâches et des exigences qui doivent être prises en compte lors du développement.

Il est généralement géré par le Product Owner. Les éléments du backlog sont généralement décrits sous forme de "user stories" qui représentent des besoins spécifiques des utilisateurs.

Le backlog est utilisé comme base pour la planification des sprints. L'équipe de développement sélectionne les éléments du backlog qu'elle s'engage à réaliser lors d'un sprint donné. Les éléments du backlog non sélectionnés peuvent être réévalués et repriorisés pour les futurs sprints.

**Le daily meeting :**

Le daily meeting est une réunion de planification qui permet aux développeurs de l’équipe de faire un point rapide sur les tâches à mener, leur coordination et éventuellement sur les difficultés rencontrées. D’une durée de 15 minutes maximum, chaque membre de l’équipe évoque tour à tour :

• Ce qu’il a réalisé la veille

• Ce qu’il compte réaliser aujourd’hui pour atteindre l’objectif du sprint

• Les obstacles éventuels qui l’empêchent d’atteindre l’objectif

Cette réunion journalière permet d’évaluer l’avancée de chacun, de synchroniser l’équipe et surtout de collecter les informations nécessaires à l’auto-organisation du projet. Ces moments d’échanges duraient plus souvent 30 minutes dans la cadre de plan de charge, en raison de la « juniorité » de l’équipe de développement et donc du nombre important d’obstacles rencontrés.

L’outil Redmine

Dans le cadre de mon alternance, nous avons adopté Redmine, un outil collaboratif sécurisé, pour suivre l'avancement des tâches. Redmine permet la gestion multi-projet et offre la possibilité de gérer les rôles d'accès aux projets en fonction des profils. L'une des interfaces clés est le tableau de bord "Carnet", qui affiche la liste des tâches planifiées avec leurs statuts actuels. Le Wiki est une autre partie essentielle où toute la documentation du projet est disponible. J'ai contribué à cette section en mettant à jour les informations existantes et en ajoutant la documentation correspondant à l'évolution du projet, notamment les tests d'acceptation. Une partie que j'ai particulièrement utilisée est la section des demandes, qui présente un tableau de tâches et les sous-tâches associées découlant de la découpe des User Stories.

Ci-dessous, un extrait de l’outils Redmine :

Une image contenant texte, capture d’écran, nombre, logiciel

Description générée automatiquement

Les environnements

Chez Adventiel, une attention toute particulière est portée sur les tests et la qualité de livraison des applications. Étant donné le nombre de collaborateurs utilisant l’application et l’intérêt stratégique, les environnements se succèdent de la manière suivante jusqu’à la livraison de l’application et sa mise en production :

* Environnement de développement (environnement local des développeur)
* Environnement de qualif (environnement de test de l’équipe projet)
* Environnement de recette (environnement de test du product owner ou du client)
* Environnement de production (environnement du client final)

Chaque environnement doit posséder son instance de base de données. Par ailleurs, chaque développeur possède son propre environnement de développement. Lors de cette année j’ai pu déployer régulièrement des développements sur la qualif puis sur la recette afin que le Product Owner puisse effectuer les tests fonctionnels et ainsi prévoir des mises en production régulières par l’équipe en charge.

Le versionning

Pour assurer le fonctionnement du plan de charge, chaque fois qu'une fonctionnalité (feature) ou une correction de bug (fix) est créée, une nouvelle branche Git est créée. L'équipe utilise GitLab pour permettre la synchronisation des modifications effectuées localement avec le dépôt de versionnage. Lorsqu'une branche est fusionnée (merge) avec la branche principale de développement (branche develop), elle nécessite l'approbation d'un chef développeur, qui était Angelo pour ce projet.

Un merge comprend plusieurs étapes :

1. Passage des tests validés en local + rebase sur la branche cible pour anticiper les conflits.

2. Création de la merge request.

3. Les tests sont rejoués par le pipeline Jenkins de GitLab.

4. Le lead Dev de l’équipe consulte la merge request (MR) et laisse d’éventuels commentaires sur les modifications apportées. Ceux-ci seront pris en compte par le développeur requêteur, qui se chargera de faire les modifications nécessaires.

5. Commentaires résolus + validation => Merge sur la branche Develop

Les déploiements

Lorsqu’une version de l’application est terminée sur l’environnement de développement, toutes les branches de fonctionnalités ou de corrections concernant cette version sont mergées sur la branche *dévelop.* L’outils Jenkins build une version « SNAPSHOT » de l’application en s’assurant que les tests et les sonar (audit de code) définis dans le projet soit conformes. Cette version est utilisée sur les environnements de qualif, puis de recette. Une fois la version « SNAPHOT » recettée, on merge la branche *develop* sur *master.* La branche *master* est l’image de la version déployer en production. Ainsi pour permettre un déploiement en production, toujours avec l’outils Jenkins, on build une version « RELEASE » (seule déployable en production). Un tag est alors créé et permet automatiquement la création d’un jar versionné et déposé sur le serveur d’artefact Nexus. Ce jar sera récupéré par l’équipe de déploiement pour préparer la mise en production.

7. REALISATION D’UNE FONCTIONNALITE REPRESENTATIVE

1. Présentation

Le Projet « Plan de charge » a été l’occasion pour moi de participer à l’évolution d’un projet web avec une architecture moderne.

En effet « Plan de charge » dispose d’une application front-end sur serveur web relié par une API Rest à une application back-end sur serveur d’application, qui elle-même communique avec la base de données se trouvant sur un serveur de base de données.

J’ai donc en autre tâche participé à la réalisation de la fonctionnalité permettant de de modifier l’affection d’un intervenant à une présence donné, depuis la vue client, jusqu’à sa persistance dans la base de données.

Je vous propose de suivre la réalisation de fonctionnalité au travers de l’architecture complète de l’application.

1. Réalisation partie front-end.

Cette partie a été pour ma part la plus compliquée à réaliser car il a fallu au préalable que je me forme sur le framework Angular et librairie PrimeNg que j’avais très peu utilisés auparavant. Les maquettes de cette page ont été déterminante pour choisir quel type de tableau nous allions utiliser et si nous allions nous service de la librairie PrimeNg comme c’est le cas dans de nombreuses page de l’application.

Angular utilise le principe de composants.

Ainsi un composant est en réalité fichier TypeScript faisant office de contrôleur auquel lui est associé un fichier html, auquel est associé un fichier css.

Ces composants sont appelés via un système de routing, et des composants peuvent en appeler d'autres directement dans le fichier HTML du composant appelant.

Dans cette partie je vais vous présenter la réalisation de la vue client via le fichier html, puis le contrôleur TypeSript qui lui est associé, pour finir par le service qui a la responsabilité de la liasion avec l’API Rest.

**La création du composant**

Le composant PresenceComponent est une classe Angular qui gère l'affichage et la gestion des présences dans un tableau.

Il permet de visualiser les plannings de présence des équipes pour une période spécifique. Le composant offre des fonctionnalités telles que le filtrage par équipe ou permanence, l'édition des présences,et l'extension des présences jusqu'à une date spécifique.

La modification d'une présence s'effectue via le tableau qui liste l'ensemble de ces dernières, en fonction des équipes filtrées et de la période demandée.

Il nous a fallu dans un premier temps créer le composant principal de la fonctionnalité et qui fera office de page dans l'application.

Ainsi il m'a fallu créer un composant nommé *presence* via l’interface en ligne de commande « Angular cli ».

Pour générer la base du composant, j’ai donc tapé « ng generate component presence » dans Angular cli.

**La réalisation de la vue.**

**Fonctionnement Général :**

Il est composé d'un tableau interactif qui affiche les équipes, les jours de la semaine et les présences associées. Les utilisateurs peuvent filtrer les données du tableau en fonction des équipes et de la permanence à l'aide de barres de recherche.

Le « range calendar » (calendrier) permet de choisir la date de début et la date de fin pour afficher les présences sur une plage de temps choisit.

L'affichage des jours non travaillés est géré de manière visuelle avec des indicateurs pour les jours fériés, les week-end, etc.

Le statut d’une présence (intervenant disponible ou non, membre de l’équipe disponible ou pas) est indiqué par différentes couleurs de « back-ground » (couleur de fond) dans les cellules du tableau.

**Gestion des Cellules de Présence :**

La partie la plus intéressante de notre composant réside dans la gestion des cellules de présence dans le tableau.

Ci-dessous, un extrait du fichier html de notre composant :

Une image contenant texte, capture d’écran, Police, menu

Description générée automatiquement

Chaque cellule du tableau représente une présence pour une équipe donnée à une date donnée.

L'affichage des cellules est dynamique et dépend des données de présence associées.

Pour cela, la présence est récupérée dans un tableau de type map, don la clef et l’association de l’id de l’équipe et de la date de la cellule grâce à la méthode findPresence().

Nous utilisons la méthode displayPresenceClass() dans le fichier TypeScript pour déterminer les classes CSS à appliquer à chaque cellule en fonction de la présence associée.

Par exemple, si un intervenant sélectionné est absent sur le jour sélectionné, la cellule peut être colorée en rouge pour alerter les utilisateurs.

L’accès au menu déroulant (<p-dropdown>) permettant de modifier l’intervenant de la présence, est géré par la méthode isAuthorizedToUpdate(). Cette méthode du fichier TypeScript renvoi un booléen en fonction de la date de la cellule, et en allant vérifier les droits de l’utilisateur, comme détaillé dans les spécifications fonctionnelles.

Si un changement est détecter ((onChange)) dans le menu déroulant, la méthode editPresence() est appelée en prenant en paramètre l’intervenant choisit dans le menu déroulant ($event), l’id de l’équipe,et la date de la cellule (col.field).

Afin, si un intervenant est sélectionné, son trigramme est affiché dans la cellule.

**Style et responsivité :**

Même si l’application « Plan de charge » est destinée uniquement au format desktop nous avons pris en charge la responsivité en utilisant les fonctionnalités intégrées de PrimeNG, ce qui permet à notre composant de s'adapter automatiquement à différentes tailles d'écran.

Le fichier presence.component.scss contient des classes CSS personnalisées pour harmoniser l'apparence de notre composant avec le reste de l'application.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Cet extrait de presence.component.scss montre les classes css qui sont appelées par la méthode displayPresenceClass() en fonction de la disponibilité d’un intervenant. Pour homogénéiser les couleurs dans l’application, les background-color font appel à des variable définit dans autre fichier sccs de l’application.

Ainsi l’association du template HTML de la librairie PrimeNg et du CSS donne un rendu sur navigateur web comme suit :

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

**La réalisation du contrôleur.**

Les fichier TypeScript des composants Angular, sont la cheville de la partie front-end de notre application.

En effet ce sont eux qui envoient les données à la vue (le template HTMl), récupèrent les données des formulaires, gèrent les interactions du client sur la vue.

En outre ils ont en charge de transmettre des données à d’autres composants, et d’appeler les services qui se chargerons de transmette ou de récupérer des données via l’API.

Je vous propose de vous présenter certains des éléments essentiel du contrôleur presence.component.ts.

Ci-dessous un extrait de la méthode ngOnInit() du composant :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

OnInit() est une méthode du cycle de vie d'un composant Angular. Lorsque vous créez un composant en utilisant Angular, il suit un certain cycle de vie avec différentes étapes. Cette méthode est appelée une seule fois, juste après que le composant ait été créé et que ses propriétés d'entrée (@Input) aient été définies. C'est la méthode idéale pour récupérer des données à partir des services, et effectuer les initialisations des attributs qu’ira chercher la vue pour l’affichage.

Dans la méthode on constate que j’appelle la méthode getPresenceBetwen() du présence service qui pour récupérer les présences entre deux dates. La méthode getPresenceBetwen() retournant un observable, je dois attendre le retour de l’observable (*result* qui la liste des présence retournée) pour effectuer l’initialisation des attributs de la classe.

Pour suivre le parcours de la modification d’une présence, je vous propose de détaillé la méthode editPresence() ci-dessous :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

La fonction prend trois paramètres :

event: C'est l'événement déclenché lorsqu'une modification est apportée à la cellule de présence. Sa valeur correspond à l’intervenant sélectionné.

equipeId: L'identifiant unique de l'équipe associée à la cellule.

colField: Un objet de type IDay représentant la colonne/jour concernée par la cellule.

La première étape consiste à construire une clé unique key pour la présence, en utilisant la méthode buildKey() de presenceBusinessService, qui prend l'identifiant de l'équipe *equipeId* et la date du jour *colField.date* pour construire la clé.

Ensuite, la présence associée à la clé est récupérée à partir de la map *this.presences* à l'aide de la méthode get().

Si la présence existe (*presence* non null) et que l'utilisateur a sélectionné un intervenant (event.value != null), la présence est mise à jour avec l'intervenant sélectionné *event.value* et la clé mise à jour est réinsérée dans la liste des présences. Cette étape sert uniquement à améliorer la réactivité de l’écran. Ainsi le trigramme de l’intervenant sélectionné est affiché avec le retour des traitements serveur déterminant la disponibilité de l’intervenant.

Ensuite, un appel est fait à presenceService.updatePresence, qui envoie une requête HTTP PUT au serveur pour mettre à jour la présence avec les modifications apportées. La méthode subscribe est utilisée pour attendre la réponse du serveur et, une fois la mise à jour effectuée, une nouvelle requête est faite pour récupérer toutes les présences mises à jour entre les dates *startDate* et *endDate*, et la liste des présences est mise à jour à l'aide de la méthode buildPresencesMap() de presenceBusinessService.

Si la présence existe (*presence* non null) et que l'utilisateur a supprimé l'intervenant de la cellule (event.value == null), la présence est également mise à jour avec l'intervenant sélectionné (null) et réinsérée dans la liste des présences (this.presences.set(key, presence)).

Comme pour la partie précédente, une requête est faite à presenceService.updatePresence() pour mettre à jour la présence avec l'intervenant supprimé. Une fois la réponse du serveur reçue, la liste des présences est mise à jour avec la nouvelle valeur de présence mise à jour (res).

**Le requêtage de l’api.**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

Les services dans une application Angular ont la charge de requêter l’Api.

En d’autres termes ils font le lien avec la partie back-end pour transférer des données.

Ci-dessus, je vous présente le PresenceService et la méthode *updatePresence()* qui ont pour finalité l’envoi d’un objet *Presence et de Id* vers la partie back-end, afin que l’objet

Soit modifié et persisté en base de données.

Ainsi on peut retrouver le décorateur @Injectable au-dessus de la classe qui nous a permis d’appeler PresenceService par injection de dépendance dans le contrôleur.

*La constante url* stock l’url de l’api à requêter.

Le constructeur de la classe PresenceService, *Constructor()* contient dans ses paramètres les services injectés nécessaires à la requête.

La méthode *updatePresence (),* prend donc en paramètre l’objet *presence* qui lui a été passé par le contrôleur et qui retourne un observable de type HttpResponse contenant dans son body, objet de type *Ipresence* (interface dont dépendent les objets de *Presence*)qui vient d’être persister en base de données.

Au final, la méthode *updatePresence()* réalise un PUT (qui dans le cas d’une api rest est utilisé pour la modifcication d’une entité) vers l’url de l’api à requêter avec en terminaison */${presenceId}.* L’ensenble de url associée à la méthode PUT correspondra, à une méthode appelée dans l’api, et *presence* qui sera l’objet envoyé dans le *body* de la requête.

1. Réalisation partie back-end.

Afin d’authentifier les utilisateurs, de contrôler les données envoyer par le client web, traité ces données pour assurer les fonctionnalités de l’application, et veiller à leur persistance, nous avons implémenter une application back-end divisé en plusieurs couches.

Ainsi pour la gestion des objets de type Presence, j’ai implémenté dans la partie back-end, une classe PresenceController, se chargeant de récupérer les requêtes venant des clients web (partie front-end), un service où se trouve la logique métier, ainsi qu’un repository se chargeant de faire la liaison avec la base de données.

Nous retrouverons PresenceController en contrôleur Rest, Presenceservice pour le service,et PresenceRepository pour le repository. D’autre classes que je ne présenterai pas dans ce rapport, comme les mapper (permettant de transformer des Entités en DTO), ou l’interface PresenceService ont été créé pour assurer le traitement des présences. Dans cette partie je présente les différentes classes et méthodes de l’application back-end que j’ai créé et implémenter, ainsi que les nombreuses annotations fournit par Spring Framework , pour faciliter en simplifier le développement.

**Le contrôleur REST**

Le contrôleur PresenceController reçoit les requêtes de la partie front-end. Ainsi pour la fonctionnalité permettant la modification d’une présence, j’ai créé en m’inspirant du code déjà écrit dans d’autres contrôleur (afin d’assurer une uniformité du code), un contrôleur comme suit :

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

**@RestController** est une annotation qui combine deux autres annotations : @Controller et @ResponseBody. Elle est utilisée pour indiquer que cette classe est un contrôleur REST qui gère les requêtes HTTP entrantes et renvoie les réponses sous forme de données JSON. L’annotation indique que la classe sera responsable de la gestion des requêtes et des réponses JSON, sans nécessiter d'annotations supplémentaires pour chaque méthode.

**@RequestMapping**("/api/presences") permet de définir le préfixe commun pour toutes les URL gérées par ce contrôleur. Dans ce controleur, toutes les requêtes HTTP destinées à ce contrôleur doivent commencer par "/api/presences".

L'injection de dépendances via **@Autowired** permettra à Spring de fournir automatiquement une instance de PresenceService au contrôleur. Grâce à cette injection, le contrôleur pourra accéder aux fonctionnalités de gestion des présences fournies par le service sans avoir à s'occuper de créer une instance de la classe PresenceService manuellement.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**@Secured**({SecurityRole.Code.MANAGER, SecurityRole.Code.SUPERVISOR, SecurityRole.Code.ADMIN}): spécifie que seuls les utilisateurs ayant les rôles de "MANAGER", "SUPERVISOR" ou "ADMIN" sont autorisés à accéder à cette méthode.

Le décorateur **@*PutMapping*** sert à lier l’url avec la méthode update().

Dans les paramètres de la méthode on retrouve le décorateur **@*valid*** qui vérifie que l’argument *edition* est bien conforme à un *business object* de type PresenceEditioDto. **@*RequestBody*** indique si le contenu du corps de la méthode est requis. La valeur par défaut est *true*, ce qui conduit à une levé d’exception s’il n’y a pas de de corps dans la requête reçut.

**@PathVariable**("id") indique à Spring de récupérer la valeur de l'identifiant (id) à partir de l'URL de la requête et de l'assigner à la variable id. Cela permet de préciser quelle présence doit être mise à jour.

Pour terminer, la méthode appel la updatePresenceDispo() du PresenceRepository en passant en paramètres, l’*id* de la présence à modifier, ainsi que l’objet *edition* de type PresenceDto (contenant uniquement l’id de l’intervenant à positionner sur la présence).

**Le Service**

Les services de notre application back-end sont le siège de la logique métier. Le PresenceServiceImpl est une classe de service de l’application qui gère les traitements liés aux présences. Il s'agit d'un composant Spring géré par le conteneur d'inversion de contrôle grâce à l'annotation **@Component**. Cette classe implémente l'interface PresenceService, qui définit les méthodes à implémenter pour gérer les présences.

Ci-dessous le début de la PresenceServiceImpl et les annotations @Component et @Autowired :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**@Autowired** a la même fonctionalité que dans le controleur.

Ici, comme PresenceService dans le contrôleur, Le presenceRepository est instancié par injection de dépendance.

La méthode updatePresenceDispo() du service PresenceServiceImpl a pour tâche de mettre à jour la disponibilité de l’intervenant d'une présence spécifique en fonction des absences de l’intervenant choisit.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Lorsqu'elle est appelée, la méthode prend deux paramètres : l'identifiant de la présence à mettre à jour (id) et un objet PresenceEditionDto (version allégée de PresenceEntity) contenant les informations de mise à jour, notamment l'identifiant de l'intervenant (ou pas) associé à cette présence.

Dans un premier temps, je vais chercher la présence correspondante dans la base de données à l'aide de l'identifiant fourni. Si la présence n'est pas trouvée, une exception PresenceDoesNotExistsException est levée pour indiquer que la présence avec cet ID n'existe pas. Ceci est possible grâce au type optional<PresenceEntity> qui fournira une liste vide plutôt que soulever un SqlException si l’id n’existe pas en base de données.

Ensuite, la méthode vérifie si l'identifiant de l'intervenant dans le presenceEdition est null. Si c'est le cas, cela signifie que l'intervenant associé doit être retiré de la présence. Dans ce cas, les champs intervenant et disponibilite de la présence sont définis sur null, indiquant qu'il n'y a pas d'intervenant associé à cette présence et que sa disponibilité n'est pas déterminée.

Si l'identifiant de l'intervenant est présent, la méthode suit ces étapes :

Je récupère toutes les absences de cet intervenant à l'aide du service AbsenceService et de la méthode findByIntervenant().

Ensuite, je récupère l'intervenant associé à cette présence en utilisant l'identifiant fourni avec le service IntervenantService et la méthode findById().

La méthode determinePresenceForIntervenant() du determineJourDePresenceService est utilisée pour déterminer la disponibilité de l'intervenant pour cette présence spécifique. La disponibilité est calculée en fonction des absences de l'intervenant et de la date de la présence. Le résultat est enregistré dans le champ disponibilite de la présence.

Enfin, l'intervenant est associé à la présence en mettant à jour le champ intervenant de l'entité de présence.

Une fois les mises à jour effectuées, la présence mise à jour est enregistrée dans la base de données à l'aide de la méthode presenceRepository.save(). Enfin, l'objet PresenceDto correspondant est renvoyé au contrôleur ou au service appelant.

**Le Repository, composant d’accès aux données**

Le Repository est la couche la plus basse de notre application. Sa fonction est de requêter la base de données pour y récupérer des informations, en insérer, les modifier ou les supprimer, par l’intermédiaire de requêtes SQL.

Notre application se sert de l’ORM(Object-Relational Mapping) JPA / Hibernate pour gérer les relations avec la base de données. Ainsi nous n’avons pas à créer directement de requête SQL dans le code. Il nous suffit d’appeler la méthode du repository correspondante en lui passant les paramètres nécessaires, pour qu’il se charge de requêter lui-même la base de données.

Ci-dessous le PresenceRepository :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

On constate en outre que le PresenceRepository, « extends » CrudRepository, et hérite donc de ses méthodes et notamment de la méthode *save()* appelée par la méthode *updatePresenceDispo()* du PresenceService. Ainsi lorsque l’on appelle la méthode, le corps de la méthode n’est pas détaillé.

Dans le cas de *save()* la requête SQL dans le SGBD serai par exemple:

Une image contenant texte, Police, capture d’écran, blanc

Description générée automatiquement

**Les tests**

Les tests sont essentiels pour assurer la qualité d’une application. Ils permettent de tester l’application à plusieurs niveaux pour garantir un fonctionnement éprouvé de cette derniére, et cela avant même sont déploiement en production. Les principaux types de test que nous avons réalisé sont les suivants :

* + - * **Les tests unitaires :** Vérifient le bon fonctionnement des méthodes implémentées, en les testant individuellement, tout en les isolant par le « mockage » de leur potentielle interaction avec le reste de l’application.
      * **Les tests d’intégrations :** Valident les interactions entre différents modules et vérifient la cohésion de l'ensemble de l'application.
      * **Les tests fonctionnels :** Évaluent le bon fonctionnement global de l'application en vérifiant si les fonctionnalités attendues se comportent comme prévu, en se basant sur les spécifications et les cas d'utilisation de l'application.
      * **Les tests de performance :** Mesurent les temps de réponse et la capacité de l'application à gérer des charges élevées pour évaluer ses performances.

**Les tests unitaires :**

Dans le cadre du développement de la méthode présenter précédemment, j’ai eu l’occasion (et le devoir) d’implémenter les test unitaires qui valide le fonctionnement de la méthode updatePrésenceDispo() du PresenceService. L’idée générale du test unitaire, est de tester le comportement d’une méthode, en fonction des différents paramètres quelle peut recevoir ou des retours d’autres méthodes. Pour s’assurer de son comportement dans des cas précis, on « mock » ces différents paramètres et interactions.

Ci-dessous, un de cas de test de la méthode updatePrésenceDispo() extrait de la classe de test PresenceServiceTest, puis sont explication :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

La classe de test PresenceServiceTest est utilisée pour tester les différentes méthodes de la classe PresenceServiceImpl. Elle utilise le framework de test JUnit 5 avec l'extension MockitoExtension pour les tests. Les dépendances de la classe PresenceServiceImpl sont simulées à l'aide de Mockito pour isoler les tests et éviter les interactions avec les dépendances.

La méthode test\_updatePresenceDispo\_SupressIntervenant() teste la fonction updatePresenceDispo de la classe PresenceService lorsque l'intervenant d'une présence est supprimé (champ intervenantId de PresenceEditionDto est null).

Dans cette méthode de test, nous effectuons les étapes suivantes :

Le « mockage » : Nous créons les données nécessaires pour le test. Nous créons une PresenceEditionDto avec un intervenantId null pour simuler le fait que l'intervenant a été supprimé de la présence. Ensuite, une instance de PresenceEntity est créée pour simuler une présence existante avec un intervenant. Le mock du repository presenceRepository est configuré pour retourner la présence simulée lorsque findById est appelé avec l'ID de la présence.

L’exécution : Nous appelons la méthode updatePresenceDispo() avec l'ID de la présence et la PresenceEditionDto mockés.

La vérification : Nous nous assurons que la méthode save() du mock presenceRepository est appelée en utilisant le captor. Le captor capture l'argument passé à la méthode save(), qui est la présence après la mise à jour. Nous vérifions ensuite que la disponibilité de la présence a bien été mise à null et que l'intervenant associé à la présence est également null, ce qui signifie que l'intervenant a été supprimé.

1. Réalisation des Entités et de la base de données

**La création des entités.**

Les entités font référence aux objets Java qui représentent les objets métiers de l’application et les données persistées dans la base de données. Ces entités sont des classes Java annotées avec des annotations pour définir leur comportement et leurs relations avec d'autres entités. Les entités et leurs annotations sont essentielles à l'ORM JPA utilisé par Spring Boot, pour qu’il puisse établir le mapping entre les objets Java et les tables de la base de données.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Voici des explications sur les principales annotations utilisées par la classe PresenceEntity:

***@Entity*** marque la classe **PresenceEntity** comme une entité JPA. Cela signifie que cette classe sera mappée sur une table de base de données et peut être gérée par JPA pour la persistance des données.

**@*Table*(name = "pdc\_presence")** permet de spécifier le nom de la table associée à cette entité dans la base de données.

***@Builder , @NoArgsConstructor, @AllArgsConstructor, @Data*** sont des annotations permettant de génrer automatiquement les méthodes basiques et essentielles d’une classe Java, comme la génération automatique d’un constructeur sans argument et avec tous les arguments, la création des getters et des setters, ainsi que le méthode toString().

**@*ManyToOne*** et **@*JoinColumn*** sont utilisées pour définir une relation "many to one" entre les entités PresenceEntity et IntervenantEntity, ainsi qu'entre les entités PresenceEntity et EquipeEntity. Ces annotations spécifient les colonnes dans la table de base de données qui seront utilisées pour stocker les clés étrangères vers les entités IntervenantEntity et EquipeEntity.

Ainsi il a été établi, que dans le modèle de données, une présence pouvait dépendre d’une seule équipe et d’un seul intervenant. Mais que ces dernières entités, pouvait être présent dans plusieurs présences.

**La génération de la base de données.**

Plan De Charge, est projet qui existe depuis quelques années maintenant, ainsi, la structure principale de la base de données existait déjà quand j’ai été intégrer au projet. Toutefois, je vous propose de vous présenter comment, la base de données a été généré et comment elle peut évoluer au cours de la vie du projet.

Notre projet utilise Flyway, un gestionnaire de schémas de base de données qui fonctionne avec un système de versionnage des fichiers de migration SQL.

Avec Flyway, nous avons réussi à gérer facilement les migrations de notre base de données pour notre application. Les fichiers SQL de migration sont organisés dans un dossier "db.migration" et ont des noms spécifiques comme "VO48\_\_presence.ddl.sql" pour les modifications de table ou "VO49\_\_presence.dml.sql" pour les créations de table. Le numéro dans le nom du fichier détermine l'ordre d'exécution lors de la construction de l'application. Grâce à cette approche, nous pouvons garder notre base de données à jour à avec l’évolution du modèle de données tout au long du développement de l'application.

Ci-dessous, un extrait des script Flyway :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Il est essentiel de comprendre que chaque fichier de migration comprend une ou plusieurs requêtes SQL pour créer ou modifier les tables. Il est fortement déconseillé de modifier un fichier de migration qui a déjà été exécuté lors d'une construction antérieure de l'application, car cela entraînerait des incohérences avec Flyway, ce qui pourrait bloquer la construction de l'application.

Cette approche nous a permis, de gérer facilement les changements dans la structure de la base de données tout en assurant la fiabilité des données. J'ai travaillé en collaboration avec l'équipe de développement pour créer un script Flyway dédié à la gestion des présences. Ce script SQL a permis de créer la table nécessaire pour répondre à ce besoin spécifique. En l'intégrant dans le dossier "db.migration" avec le bon nom et numéro de version, nous nous sommes assurés qu'il serait exécuté correctement lors du déploiement de l'application. Cette démarche simple et bien coordonnée nous a permis de mettre en place les changements nécessaires sans compromettre la stabilité de l'application, tout en facilitant la collaboration au sein de l'équipe de développement.

Pour gérer les relations entre les tables, nous faisons usage des clés étrangères dans les fichiers de migration, ce qui permet d'établir les liens souhaités entre les tables. Ensuite, lors de la création des entités, j'utilise les annotations de Spring pour définir ces relations de manière claire et concise. Cela nous permet de maintenir une base de données bien organisée et cohérente, favorisant ainsi des opérations efficaces sur les données et une meilleure compréhension de la structure de la base de données pour toute l'équipe de développement.

Ci-dessous le script de création de la table présence :

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

La fonctionnalité de gestion des présences, requière le concours de plusieurs tables de la base de données dont certaines ayant une volumétrie conséquente de données. Afin de s'assurer que l'implémentation de la fonctionnalité est suffisamment performante pour garantir une expérience utilisateur satisfaisante, ou dans un cas plus extrême, que le traitement nécessaire ne tombe pas en time out, il a été nécessaire de travailler sur notre environnement de développement avec une volumétrie de données équivalente à l'environnement de production.

J'ai donc réalisé, à plusieurs reprises des « dumps » de la base de données de production, qui sont des sauvegardes complètes de l'état de la base de données à un moment précis. Ces dumps nous ont été très utiles car ils nous ont permis de créer un environnement de développement similaire à celui de la production en reproduisant les mêmes données. Grâce à cela, nous avons pu tester l'application dans des conditions réalistes, ce qui a grandement facilité le processus de développement et de validation des fonctionnalités.

8. SECURITE

1. Présentation

Sécuriser une application est un élément essentiel de son développement. En effet de nombreuses attaques existes pour porter atteinte aux données que peut contenir l'application ou tout simplement usurpé des comptes des utilisateurs. Il est donc essentiel pour une application professionnelle, de se prémunir au maximum des attaques les plus courantes, afin d’éviter les fuites de donnés, une utilisation frauduleuse ou encore le blocage de l’application. « Plan de charge » étant créé à partir de plusieurs frameworks, je vous propose de découvrir ci-dessous les options de sécurité implémenter et configurer pour assurer la sécurité de notre application.

1. L’authentification

L'authentification joue un rôle essentiel dans la sécurité d'une application. Elle garantit que seules les personnes autorisées ont accès aux données et peuvent interagir avec elles. Voici comment cela fonctionne dans « Plan de charge » :

Lorsqu'un utilisateur souhaite se connecter à l'application, il fournit son nom d'utilisateur (login) et son mot de passe via le formulaire de connexion côté client. Ces informations sont ensuite envoyées à l'API de l'application. L'API utilise un web service pour interroger un serveur LDAP (Lightweight Directory Access Protocol) chargé de vérifier la cohérence du nom d'utilisateur et du mot de passe fournis. Si le web service renvoie une réponse positive, cela signifie que l'utilisateur est authentifié avec succès.

Une fois authentifié par l'application, celle-ci génère un jeton (token) sécurisé conformément à la norme JWT (JSON Web Token). Ce jeton contient des informations telles que le nom d'utilisateur (login) et les droits associés à cet utilisateur (rôle). Le serveur signe ce jeton, ce qui garantit son intégrité et empêche toute modification par l'utilisateur.

Le jeton est ensuite renvoyé à l'utilisateur et stocké dans son navigateur. Pour chaque requête ultérieure du client vers le serveur, le jeton est inclus dans l'en-tête de la requête. Le serveur peut ainsi vérifier l'authenticité du jeton, le décrypter à l'aide de la clé de signature et extraire les informations nécessaires pour autoriser ou refuser l'accès à certaines ressources ou fonctionnalités de l'application.

Cette approche basée sur les jetons offre plusieurs avantages. Elle permet une gestion fine des droits d'accès en associant des rôles spécifiques à chaque utilisateur. De plus, le jeton étant stocké côté client, cela réduit la nécessité de vérifier les informations d'authentification à chaque requête, ce qui améliore les performances de l'application.

1. Les attaques et nos réponses

Les attaques « brut force »

L’attaque brut force est l’une des attaques informatiques les plus simple dans son fonctionnement. Son but est d’obtenir l’accès à un compte utilisateur d’une application en trouvant le mot de passe de l’utilisateur.

Son fonctionnement est assez simple, car il consiste tester toutes les combinaisons possibles de mot passe, jusqu’à l’obtention de ce dernier.

Se prémunir de genre d’attaques est plutôt simple, car plus un mot de passe est long et complexe (alternance de caractères alphanumériques, de majuscules et de minuscules), plus l’attaque prendra du temps et plus elle a de chance d’être mise en échec.

Afin d’être moins vulnérable à cette attaque, chaque salarié doit avoir un mot de passe sur le serveur LDAP d’au moins 12 caractères avec au moins une majuscule et un caractère spécial.

Pour informations, un mot de passe respectant les conditions définies précédemment, mettrai plusieurs centaines d’année à être cracké.

Toutefois, une méthode efficace, serai d’implémenter dans la partie, back-end, une méthode empêchant la soumission d’une demande de connexion pendant plusieurs secondes, après un certain nombre d’authentifications refusées.

Les failles « xss »

Les attaques XSS (Cross-Site Scripting) sont des vulnérabilités qui permettent d'insérer du code JavaScript malveillant dans le DOM (Document Object Model) d'une application. Les conséquences de telles attaques peuvent être le vol de données utilisateur via la récupération de cookies de session, la redirection vers un autre site ou l'affichage de contenu malveillant.

Pour prévenir ces attaques, nous utilisons Angular, un framework de développement de la partie front-end de « Plan de charge ». Angular considère toute valeur insérée dans le DOM comme potentiellement non fiable. Ainsi, il applique un processus de nettoyage (souvent appelé "sanitization") à ces valeurs pour éviter l'insertion de scripts non prévus dans le fonctionnement normal de l'application.

En plus de l'utilisation d'Angular, il est également possible de se protéger contre les attaques XSS en appliquant une validation des données d'entrée. Cela peut être réalisé en utilisant des expressions régulières (regex) pour valider les formats attendus des données, ainsi qu'en mettant en place une liste noire de certains caractères, tels que les balises <script>, afin de les bloquer explicitement. Une autre mesure importante consiste à encoder les entrées utilisateur avant de les afficher dans le DOM, de manière à ce qu'aucun script indésirable ne puisse être exécuté.

Il est à noter que ces mesures de protection contre les attaques XSS n'ont pas encore été pleinement mises en place sur "Plan de Charge". Il est donc recommandé de les prendre en considération et de les implémenter pour renforcer la sécurité de l'application

Les injections SQL

Beaucoup d’applications disposent de formulaire et champs de saisie pour recevoir des informations de la part des utilisateurs. Ces informations comme par exemple, le mot passe et le login son envoyer vers la base de données pour être comparer et permettre l’authentification.

Dans le cas d’une injection SQL, à la place de mettre un login et ou un mot de passe dans le formulaire, un pirate saisi un morceau de code SQL qui une fois insérer dans la requête SQL envoyée à la base de données, modifiera son comportement initial.

Notre application utilise l’ORM Hibernate pour gérer les requêtes entre notre back-end et notre base de données. L’ORM ne protège pas en soit des injections SQL car il nous est toujours possible d’écrire du code vulnérable. En revanche l’intérêt de de l’ORM est d’utilisé son interface de requête et les méthodes Query avec ses paramétre nommés. Ainsi l’ORM crée une instruction préparée à l’aide d’un espace réservé pour le paramètre. Utilisant uniquement cette procédure pour requêter la base de données, nous protégeons l’application contre les injections SQL.

Ci-dessous, un exemple de requête préparée tiré du repository de « Plan de charge » :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement9. RECHERCHE EFFECTUEE

1. Le besoin d’information

Au cours de mon alternance j’ai eu l’occasion de travailler sur trois projets différents, don Plan de charge. La recherche que je vous présente, a été effectué dans le cadre du projetCap2er, qui est une application destinée à calculer l’impact carbone des exploitations agricoles. A l’instar de Plan de charge, Cap2er est développé en Angular pour sa partie front-end. Comme présenté précédemment dans le rapport, des composant Angular peuvent faire appel à d’autre composant. Dans ce cas on parle de composant parent et de composant enfant. Ces composant peuvent dans de nombreux cas, partagé des informations. Dans mon cas de recherche, mon composant enfant devait détecter des changements dans les attributs du composant parent passés en *@Input* pour mettre à jour ses informations.

1. La recherche

La méthode de recherche de la solution à ce problème a été d’effectuer une recherche ciblée par mot clef en anglais sur le moteur de recherche google.

Le choix de l’anglais pour cette recherche était de ciblée des ressources de portée internationale et donc de maximiser les chances de trouver une réponse fiable.

Les mots clef choisit devait être le plus équivoque possible :

« detect input changes» : la fonction recherchée.

« component» : le type d’élément sur lequel doit intervenir la fonctionnalité.

« Angular» : le framework dans lequel on recherche la solution.

Dès lors le moteur de recherche propose plusieurs résultats de site anglophone :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

De ces deux résultats, il convient de prendre trois paramètres en compte. Le premier étant l’extrait de contenu avec les mots clefs. Le second, la « renommé » du site web qui délivre l’information, pour s’assurer de la fiabilité du contenu. Le troisième, la date à laquelle l’information a été divulguer afin de prendre en compte la mise à jour de la méthode proposée.

Ainsi, j’ai fait le choix de consulter Stack Overflow qui jouit d’une certaine réputation auprès de la communauté des développeurs, malgré l’âge de l’information et angularjswiki pour la fraicheur de l’information.

Une image contenant texte, capture d’écran, Police, Page web

Description générée automatiquement

La solution plébiscité par la communauté est d’utiliser la méthode ngOchanges(), qui est une des méthodes du cycle des composant Angular. Pour m’assurer que la Méthode ne soit par déprécier avec la version Angular que j’utilise, je décide de consulter la documentation officielle d’Angular concernant cette méthode.

1. La solution

Ci-dessous, un extrait de la documentation sur la méthode onChanges() :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Angular .io confirme l’utilisation de cette méthode pour détecter des changements dans les *@Input* du composant enfant. Je décide de mettre en place cette solution en m’inspirant de l’exemple de la documentation.

1. La mise en œuvre

Ci-dessous, un extrait de du ficher TypeScript du composant enfant :

Une image contenant texte, capture d’écran

Description générée automatiquement

Ainsi pour mettre en place cette solution, il m’a fallu implémenter l’interface Onchanges.

Dans la condition *if*, j’isole les changements pour rappeler la méthode OnInit() uniquement si les changements dans les @Input interviennent sur ces attributs.

Le composant est ainsi réinitialisé si des changements interviennent sur les attributs du composant parent passer en @Input au composant enfant.

10. Bilan et perspective du projet

Plan de charge est aujourd’hui un des outils interne majeur concernant la gestion des ressources chez Adventiel. PDC est utilisé quotidiennement par une grande partie des personnes travaillant dans l’entreprise. En effet, en remplaçant la gestion par fichiers ecxel, l’application est désormais l’outils exclusif de gestion des ressources pour l’ensemble des projets informatique portés par Adventiel. L’écran de Gestion des présences a été présenté comme une aide à la gestion des permanences auprès des chefs de projets, sans faire l’objet d’une obligation d’utilisation. A ce jour, la gestion des présences est utilisée sur environ la moitié des projets nécessitant une permanence, en conservant une évolution à la hausse.

Ainsi, Plan ce charge fait l’objet d’une maintenance attentive et bénéficie d’ajout régulier de fonctionnalités, à l’initiative de Mickaël HEBERT ou suite à des retours d’autres utilisateurs. Plan de charge est en constante évolution depuis sa genèse. Une attention régulière est particulièrement apportée à la gestion des requêtes entre les différentes couches de l’application, afin d’apporter plus de performance à l’application et de diminué son impact environnemental. Dans cet esprit nous avons donc veillé lors du développement de la fonctionnalité de gestion des présences, à rationaliser notre code en essayant garder un équilibre entre économie des ressources et fiabilité des données affichées à l’utilisateur.

Aujourd’hui en version 1.12.2, PDC continue d’évoluer sous l’impulsion de Mickaël HEBERT. Les prochaines évolutions concerneront, les planifications et modification des charges, qui bénéficieront d’un historique de modification pour accroitre leurs capacités d’audit.

11. CONCLUSION

Cette année d'alternance a été une expérience particulièrement enrichissante pour moi, tant sur le plan professionnel que personnel. J'ai eu l'opportunité de travailler sur le projet « Plan de Charge" pendant plusieurs mois, dont deux mois sous la supervision d'un développeur senior. Cette période m'a permis de consolider mes compétences dans des technologies que j'avais déjà abordées lors de mon stage D2WM, telles que Spring Boot et Angular. De plus, j'ai également eu l'occasion de découvrir de nouvelles technologies telles que Docker et Jenkins, ainsi que commencer à acquérir de bonnes pratiques de développement, notamment en ce qui concerne la mise en place de tests unitaires.

Ce qui m'a particulièrement plu dans cette expérience d'alternance, c'est d'avoir pu travailler sur toutes les couches de l'application, ce qui m'a permis de consolider mes compétences en tant que développeur Fullstack. J'ai apprécié pouvoir m'immerger dans les différents aspects d'un projet informatique, et de me perfectionner dans les technologies et concepts indispensables à un développeur polyvalent.

Mon année d'alternance a également été marquée par la nécessité de « sortir le nez de mon IDE ». En effet, à la fin de ma formation DW2M, je n'avais pas encore pleinement conscience de l'environnement complexe dans lequel s'inscrit le développement d'un projet informatique. J'ai ainsi découvert la gestion de projet et j'ai eu l'occasion de collaborer avec d'autres professionnels tels que des architectes, des administrateurs de base de données, des administrateurs système et des chefs de projet. Cette expérience m'a permis de mieux appréhender les enjeux d'un projet informatique et de développer des compétences transversales.

Au-delà des compétences techniques, cette année d'alternance m'a également enseigné l'importance de la formation continue dans le domaine du développement. J'ai réalisé que même en ayant investi beaucoup d'efforts pendant ma formation, il est essentiel de rester constamment à l'affût des nouvelles technologies et des évolutions du métier. Cela m'a conforté dans mon choix de carrière en tant que développeur, et je suis reconnaissant envers Adventiel de continuer à me faire confiance en me proposant un contrat à durée indéterminée.