**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
2. Présentation
3. Liste des compétences du référentiel qui sont couvertes par le projet
4. Résumé du projet
5. Cahier des charges
6. Présentation
7. Expression des besoins
8. Spécifications fonctionnelles
9. Spécifications techniques
   1. Présentation
   2. Uses cases
   3. Maquettes
   4. Schéma de base de données
   5. Ressources logicielles
   6. Architecture du projet
10. Réalisation d’une fonctionnalité représentative
11. Présentation
12. Réalisation front-end
13. Réalisation partie back-end
14. Réalisation des entités et de la base de données
15. Sécurité
16. Présentation
17. Les attaques et nos réponses
18. Mise en œuvre
19. Conclusion
20. Recherche effectuée
21. Le besoin d’information
22. La recherche
23. La solution
24. La mise en œuvre
25. Traduction
26. Présentation
27. Traduction
28. Conclusion

Annexes

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 toute 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 leur (relative) sagesse dans cette période charnière de ma vie professionnelle.

1. INTRODUCTION

Le présent rapport présente le projet "Plan de Charge" réalisé dans le cadre de ma formation en tant que concepteur développeur d'applications chez Adventiel et l’ENI de Chartre de Bretagne, 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 compétence. PDC est un projet interne à Adventiel, Il a pour utilisation l’organisation du travail au seins 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 répercuté sur le plan de charge.

Depuis quelques mois, PDC permet aux managers de gérer les astreintes 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, de manière plus général, mon expérience durant cette année d’alternance.

1. LISTE DES COMPETENCES DU REFERENTIEL QUI SONT COUVERTES PAR LE PROJET

Le projet Florilège m'a permis de développer les compétences du référentiel ci-après :

* + - * Maquetter une application.
      * Développer une interface utilisateur web dynamique.
      * Réaliser une interface utilisateur avec une solution de gestion de contenu.
      * Créer une base de données.
      * Développer les composants d’accès aux données (via orm).
      * Développer la partie back-end d’une application web ou web mobile.
      * Elaborer et mettre en œuvre des composants dans une application de gestion de contenu.

1. RESUME DU PROJET
2. 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 filiéres 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 talentueuse 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 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 PDC, 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 ailleurs 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 intervenant 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 PDC. Les administrateurs et superviseurs, ont un accès complet à toutes les fonctionnalités de l'application 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 propre planning individuel et de soumettre des demandes d'absence personnelle, et consulter les dates de présence qui leurs sont affectées.

En résumé, le projet Plan de Charge (PDC) chez Adventiel marque une transition vers une gestion optimisée des ressources grâce à une application web 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 les 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 était :

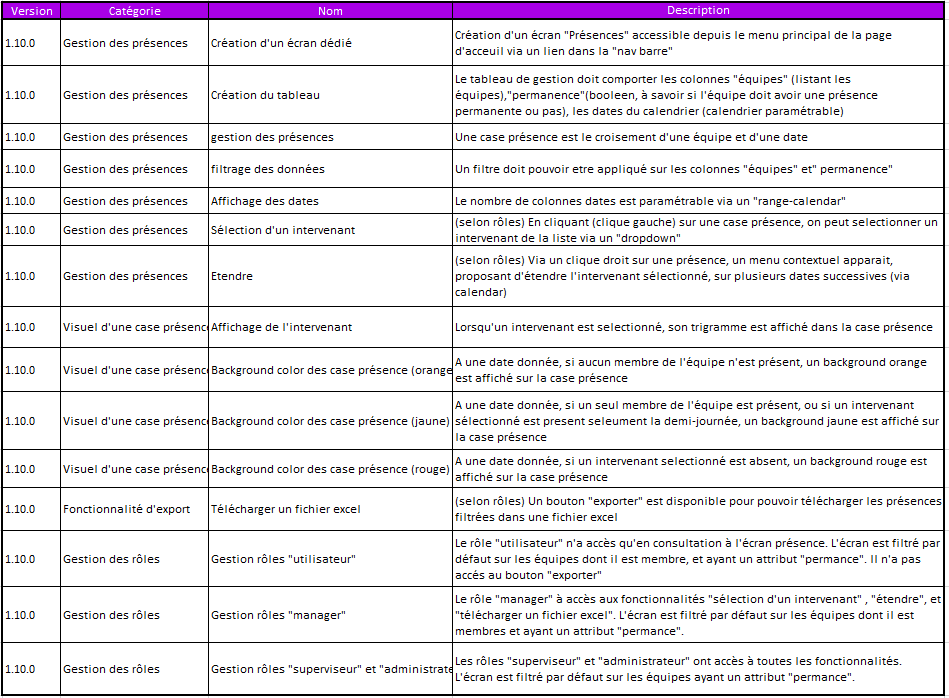
* + - * La gestion des présences doit se faire dans un écran dédié
      * L’écran des présences doit être présenté sous forme d’un tableau/calendrier
      * Le tableau doit croiser les dates et les équipes
      * Ainsi une case du tableau représentant une présence est la jonction entre une équipe et une date
      * Pour les roles 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 au fonctionnement de la fonctionnalité de gestion des présences.

Les spécifications fonctionnelles ci-dessus 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 implementé sur toutes les couches de l’application et à nécessité 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, les use cases et maquettes de l’application.

Nous avons aussi réfléchi en lien avec l’aide d’Angelo Boursin (lead dev) au schéma de base de données.

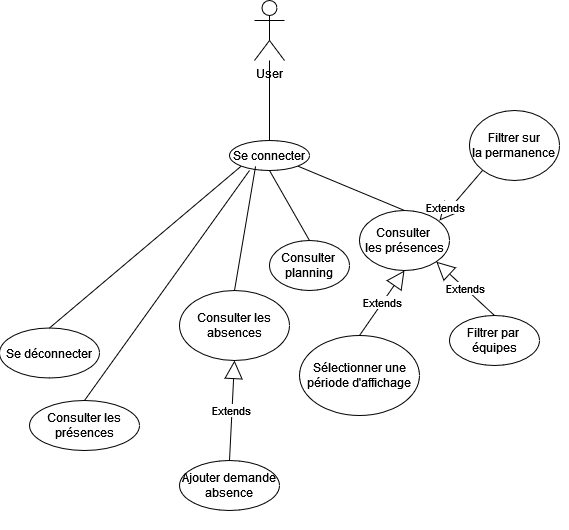
Je vous propose dans les onglets suivants de découvrir ces documents ainsi que les ressources logiciel et outils utilisés pour la réalisation de l'application.

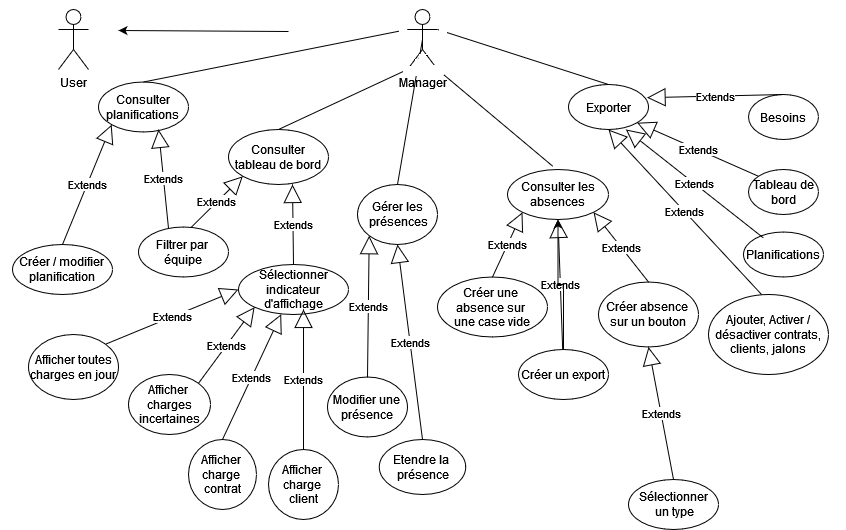
1. Les use cases

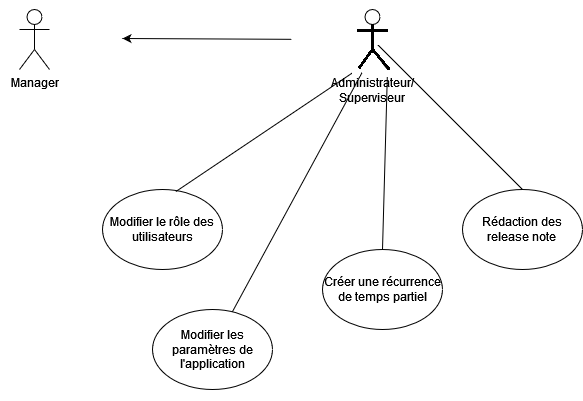
Les uses cases représentent les scenarios d’utilisation de l’application.

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.







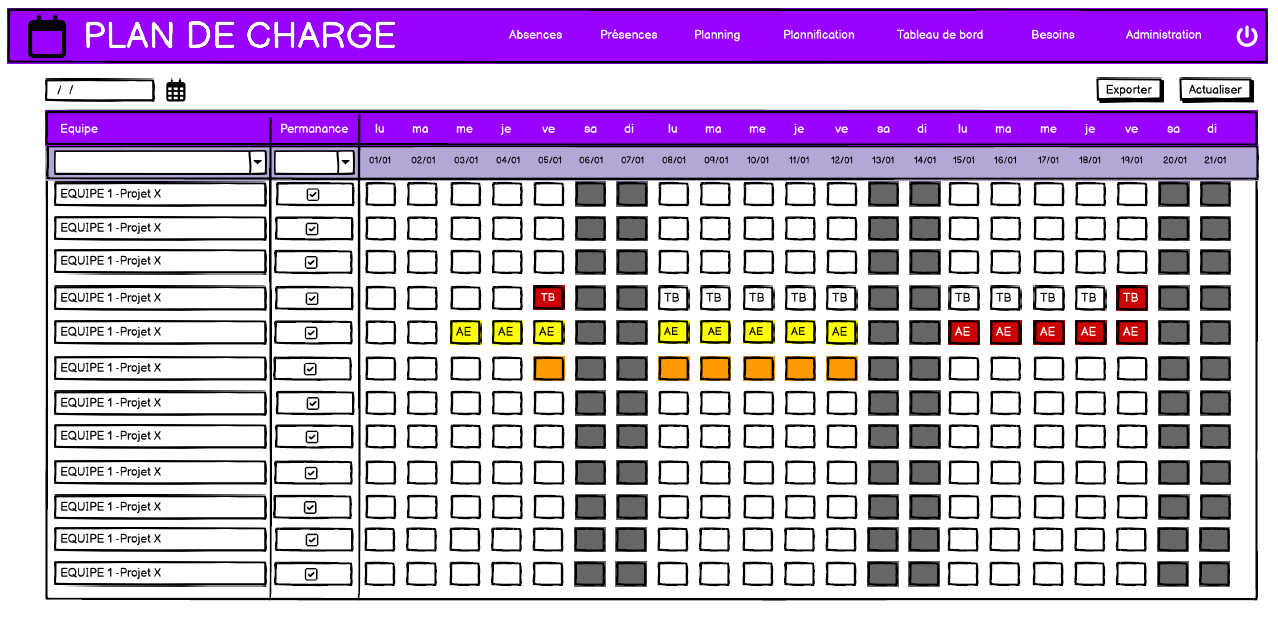
1. Les maquettes

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, 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 :



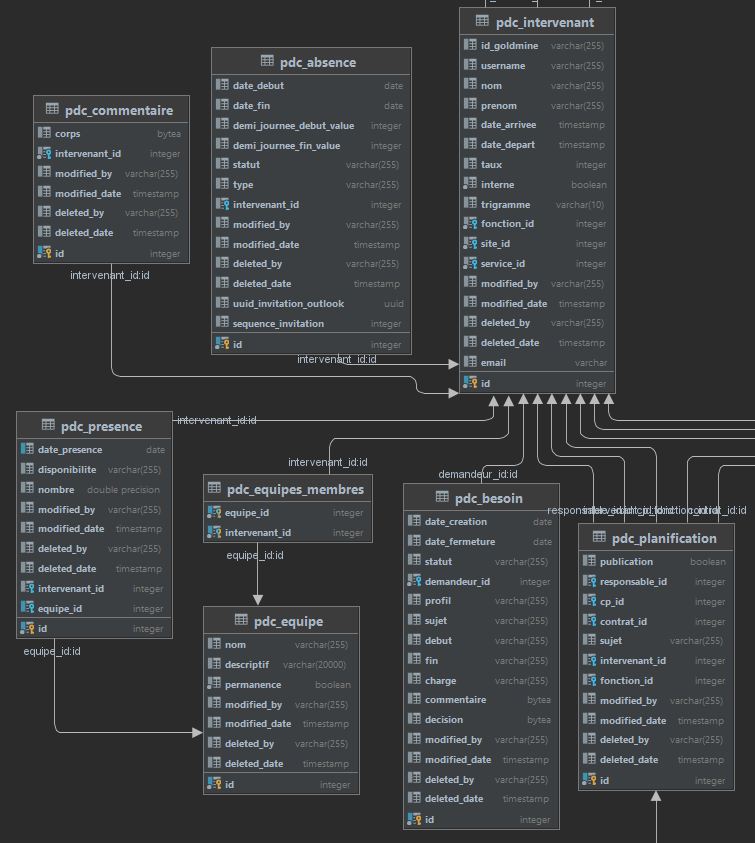
1. Le schéma de base de données

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 du schéma de base de données (cf annexe X) 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 :



1. Ressources logicielles et environnements techniques

Le projet PDC a été élaboré grâce au concours de plusieurs langages de programmation, framework, librairie, et application 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 fais 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 gérer le filtrage des tableaux.

**Pour la partie Back-end :**

**Java :** Java est un langage de programmation polyvalent et populaire, 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.

* + - * **Hibernate** : Hibernate est un orm (object-relational mapping) open source gérant la persistance des objets en base de données relationnelle.
      * **Framagit** : application de versioning de projet.
      * **PostgreSQL** : PostgreSQL est un système de gestion de base de données relationnelle (SGBD)
      * **Intellij** : Intellij IDEA est un environnement de développement intégré destiné au développement de logiciels informatiques reposant sur la technologie Java.

1. Architecture du projet

Florilèges est une application web construite selon une structure monolithique (une seule application backend-end et base de données), et une architecture de type MVC à l’instar du schéma ci-dessous :



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

MODELE

CONTROLEUR

VUE

**FRONT-END**

Business Object

**BACK-END**

Business Object

Repository

Contrôleur Rest

Service

Service

Contrôleurs TypeScript

Template html

7. REALISATION D’UNE FONCTIONNALITE REPRESENTATIVE

1. Présentation

Le Projet Florilèges a été l’occasion pour moi de suivre la genèse d’un projet avec une architecture moderne.

En effet Florilèges dispose d’une application 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 pu être en charge d’implémenter la fonctionnalité permettant de créer une *track* (entité contenant les informations d’un clip video) 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.

Notre application possède deux interfaces utilisateur.

La première, est la partie réservée aux administrateurs de Florilèges.

Cette interface est celle qui a été générée par Jhipster et permet uniquement d’interagir sur le contenu persisté en base de données.

La deuxième est destinée aux utilisateurs de l'application.

C’est dans cette interface que les fonctionnalités qui sont au cœur du projet sont implémentées.

C'est donc sur cette partie que j'ai essentiellement travaillé pour la partie front-end.

Les maquettes présentées en amont m’ont permis de créer plus facilement les ihms.

Il avait été décidé par Sylvain Letellier que la partie front end de l'application serait créé à partir du framework Angular.

Toutefois il nous a fallu trouver une librairie pouvons-nous aider à la réalisation des ihms.

Des librairies comme Ant et Prime Ng été testées dans un premier temps, mais ne nous ont pas apporté satisfaction faute de modularité.

C'est finalement la librairie à Ionic que nous avons choisi d'utiliser en raison de sa compatibilité avec les formats mobiles.

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 que je n'avais jamais utilisé auparavant.

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.

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 qui lui est associé pour finir sur la liaison avec l’API Rest.

**La création du composant**

L’ajout d’une nouvelle *track* se fait par l’intermédiaire d’un formulaire accessible depuis une playlist.

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

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

**La réalisation de la vue.**

Ce composant Angular a pour mission de proposer un formulaire permettant à un utilisateur de saisir les informations nécessaires à la création d’une nouvelle *track*.

Nous avons décidé que dans un premier le temps le formulaire proposerai la saisie de quatre informations, dont le « name » et l’ « url » qui sont des champs obligatoires en base de données.

J’ai fait le choix d’utiliser pour ce formulaire un formulaire type *reactive Form*.

Les formulaires réactifs sont d'abord **générés en TypeScript** , on vient ensuite **relier** les différents  *input*  du template à l'objet du formulaire.

Pour la responsivité du composant j’ai choisi d’utiliser un système simple et efficace pour être adaptable à la largeur des écrans qui est la méthode des lignes et des colonnes (row , col), à la maniére de BootStrap. Utilisant la librairie ionic, ici les balises sont appelées <ion-row> et <ion-col>.

Le principe est qu’une ligne (<row>) soit partagée en douze colonnes(<col>).

Ainsi j’encadre le code html du formulaire dans un jeu de balises <row> puis dans un jeu de balises <col>.

Pour que le formulaire soit *responsive,* j’ai indiqué dans la balise <col> le nombre de colonnes que devait prendre le formulaire en fonction de la taille de l’écran.



Le formulaire à une configuration assez classique à la différence qu’il ne possède pas d’attibut *method,* mais une liaison avec l’objet formulaire du fichier TypeScript.

Les différents *input* possèdent un attribut *formControlName* pour les lier aux différents attributs du formulaire TypeScript.

Lorsque le formulaire est complet et valide, il appelle la méthode *save()* du fichier TypeScript ((ngSubmit) correspond à un déclenchement sur un bouton de type *submit* à l’intérieur du formulaire).

Ci-dessous, un extrait de code du formulaire :

*Une image contenant texte

Description générée automatiquement*

Le composant prévoit aussi un fichier CSS qui est lié au template html par l’intermédiaire du fichier TypeScript afin d’enjoliver l’interface utilisateur.

Ci-dessous un extrait de code du fichier CSS :

Une image contenant texte

Description générée automatiquement

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

Une image contenant texte

Description générée automatiquement

**La réalisation du contrôleur.**

Les fichier TypeScript des composant 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.

Comme expliqué précédemment le composants *add-track* se charge de récupérer des informations du formulaire.

Je vous propose de voir les éléments essentiels du fichiers TypeScript permettant le traitement de ces données.

Une image contenant texte

Description générée automatiquement

Ci-dessus, un extrait de code montrant comment j’ai créé le formulaire.

La variable formulaire *addTrackform* est de type *FormGroup*, je lui passe un formulaire construit avec *fb* qui est de type FormBuilder. J’ai donc importé *Formbuilder* et *FormGroup* depuis *angular/core*.

*Artist, name, description, url* représentent les champs de saisie du formulaire. On peut constater dans les crochets, les options de ces champs.

*Null* signifie qu’aucune valeur n’est affectée par défaut, *validators.required* impose que ce champs soit rempli pour que le formulaire soit valide.

*Validators.pattern(this.urlRegex)* implique que la saisie doit respecter la structure d’une adresse URL pour être valide (*urlRegex* a été défini dans les attributs du composant).

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Ci-dessus vous pouvez observer les méthodes que j’ai implémentées à la soumission du formulaire par l’utilisateur.

On a pu voir dans la présentation de la vue que le bouton de soumission du formulaire appelait la méthode *save()* du fichier TypeScript.

Dans un premier temps la méthode appelle fonction *createFromForm()* qui va récupérer les données provenant du formulaire pour créer un objet de type *Track*. Ensuite le retour de cette fonction (l’objet de type Track) est stocké dans la variable *track*.

Dans un second temps, la méthode *save()* appelle la méthode *create()* du service *uiService* à qui je passe la variable *track* en paramètre.

J’ai fait appel à *uiService* grâce l’injection de dépendance.

Ainsi l’objet *track* va être transmis au service qui se chargera de l’envoyer à la partie back-end de l’application par l’intermédiaire de l’API.

**Le requêtage de l’api.**

Une image contenant texte

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 l’UiService et la méthode *create()* qui ont pour finalité l’envoi d’un objet *track* vers la partie back-end, afin que l’objet soit persisté en base de données.

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

*ressourceUrl* stock la première partie de l’url de l’api à requêter.

Le constructeur de la class Uiservice, *Constructor* contient dans ses paramètres les services injectés nécessaires à la requête.

La méthode *create(),* prend donc en paramètre l’objet *track* qui lui a été passé par le contrôleur et qui retourne un observable de type HttpResponse contenant dans son body, objet de type *Itrack* (interface don dépende les objet de *Track*)qui vient d’être persister en base de données.

Au final, la méthode *create()* réalise un POST (qui dans le cas d’une api rest est utilisé pour l’envoi d’une nouvelle donnée) vers l’url de l’api à requêter avec en terminaison */tracks* qui correspondra, à une méthode appelée dans l’api, et *track* qui sera l’objet envoyé dans le *body* de la requête.

1. Réalisation partie back-end.

Le générateur de projet Jhipster associé au framework Spring ont généré un crud sur l’ensemble des entités du projet.

Ainsi pour les objets de type Track, il a été généré dans la partie back-end une class TrackResource qui n’est autre que l’api, se chargant de récupérer les requêtes venant des applications web (partie front-end), un trackService où se trouve la logique métier, ainsi q’un trackRepository se chargeant de faire la liaison avec la base de données.

Sylvain Lettelier, a souhaité que l’on ne fasse aucune modification dans les classes déjà créées par Jhipster.

Ainsi pour implémenter des fonctionnalités particulières, Il m’a demandé de créer un nouveau contrôleur Rest concernant les nouvelles méthodes et un service qui contiendra la logique métier nécessaire aux nouvelles fonctionnalités. Ainsi nous retrouverons UiResource en contrôleur Rest et Uiservice pour le service.

Ci-dessous, un diagramme de séquence montrant le parcours d’ajout d’un objet track dans l’application back-end :



**Le contrôleur UiResource**

Le contrôleur UiResource reçoit les requêtes de la partie front-end. Ainsi pour la fonctionnalité permettant une persistance d’une nouvelle track, j’ai créé en m’inspirant du code déjà générer par jhipster dans d’autres contrôleur, une fonction comme suit :

Une image contenant texte

Description générée automatiquement

Le décorateur @*PostMapping* (appellé end-point dans ce cas) sert à lier l’url avec la méthode createTrack().

Dans les paramètres de la méthode on retrouve le décorateur @*valid* qui vérifie que l’argument *track* est bien conforme à un *business object* de type Track. @*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.

Le retour de la fonction est un *Mono* d’une *ResponseEntity* d’objet de type Track. Ainsi Mono correspond à un flux qui émet au plus un évènement. *ResponseEntity* est une extension de *HttpEntity* qui représente une réponse http comprenant le statut, les en-têtes et le corps. *Track* indique le Type de l’objet qui est renvoyé.

Le corps de la méthode *createTrack()* appel la méthode *saveTrack()* de l’*UiService* par injection dépendance en lui passant en paramètre l’objet *track* a traité.

*.map* sert à récupérer l’observable qui revient de la méthode *uiService.saveTrack(),* pour qu’il soit à l’intérieur du *try{}*, insérer dans une réponse http qui sera renvoyé vers l’application front-end.

**L’UiService**

Les services de notre application back-end sont le siège de la logique métier. Comme on peut le constater sur le schéma de base de données, *track* est relié à *track\_data*. Ainsi *track* contient les informations d’un clip vidéo et *track\_data* contient le clip vidéo en lui-même.

Pour pouvoir exploiter les informations indépendamment des données du clip (compte tenu de la taille importante de la vidéo), nous avons décidé de séparer les informations des données en créant deux table distinctes, et petite particularité, une *track* ne contient pas en attribut l’entité *track\_data,* mais un *trackDataId* qui fait référence à l’*id* d’une *trackData* . Par ailleurs si plusieurs utilisateurs décident de persister les mêmes vidéos, mais avec des informations différentes, cela permet de stocker qu’une seule fois la vidéo en question.

Les méthodes que j’ai implémentées permettent que quand un utilisateur veut insérer une nouvelle *track*, il soit vérifié avant de télécharger les données, que la *track\_data* correspondant à la *track* n’existe pas déjà en base de données.

Nb : la fonction *findByUrlorCreate()* qui fais partie de la présentation n’est pas complète, car au moment de rédaction de ce rapport, le *youtubedownloader* qui va nous permettre de télécharger les données de la vidéo n’est pas opérationnel.

Ci-dessous, les deux méthodes de *l’UiService* que j’ai réalisées pour cette fonctionnalité :

Une image contenant texte

Description générée automatiquement

Dans un premier temps la méthode *save()* appelle la méthode *findByUrlOrCreate()* en lui passant en paramètres l’url de la *track* à persister. La méthode *findByUrlOrCreate()* appel le trackDataRepository avec sa méthode *findByUrl()* afin de vérifier si une *trackData* avec cette url existe déjà en base de données.

Si l’observable retourné (*trackData*) est *null*, la méthode créer un nouvelle *TrackData* (*newTrackData*). *newTrackData* contiendra l’url de la vidéo et le youtubedownloader se chargera de remplir l’attribut data, avant que la fonction ne persiste *newTrackData* en base de données via la méthode *save()* du trackDataRepository.

Si le l’observable retourné est non *null*, c’est qu’une TrackData existe déjà en base de données avec cette url, la fonction retournera la *trackData.*

Ainsi dans les deux cas la fonction *findByUrlOrCreate()* retournera une *trackData* non *null* à la fonction *save().*

Dans un second temps, une fois qu’elle a récupéré une *trackData* non *null,* la méthode *save()* va charger l’attribut *trackDataId* de la *track* à persister avec l’*id* de la *TrackData* précédemment retournée.

Pour finir, la méthode *save()* appel la fonction la fonction *save()* du *trackRepository* afin qu’elle soit persistée en base de données.

**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 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 qui nous intéresse en lui passant les paramètres nécessaires, pour qu’il se charge de requêter lui-même la base de données.

Ci-dessous la méthode *save()* du TrackRepository :



Ci-dessous la méthode *findByUrl()* du TrackDataRepository :



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 pour remplir les champs obligatoire serai :

INSERT INTO track (name, url, track\_data\_id, user\_id)

VALUES (‘supersonic’, ’ <https://youtu.be/pbIRQR6FkhU>’, ‘3’,’1051’) ;

Dans le cas de *findByUrl()* la requête SQL dans le SGBD serai :

SELECT \* FROM track\_data WHERE url = https://youtu.be/kgx4WGK0oNU(exempled’url);

1. Réalisation des Entités et de la base de données

**La génération des entités.**

A l’instar du framework Symfony en Php, le générateur de projet Jhisper nous met à disposition un terminal permettant de définir les différent entitées don nous aurons besoin dans le projet et de définir les relations entre elles. Dans le cadre de Jhipster cette etape est préalable à la génération de la base, car jhipster, à partir de nos choix dans le terminal générera les entités dans la partie back-end et les fichiers Xml définissant la base de données, qui seront interprété par Liquidbase (outil de migration de base de données) pour la création de la base.

Ci-dessous, une partie du terminal pour générer l’entité playlist :

Une image contenant texte

Description générée automatiquement

Ainsi nous avons défini pour chaque Entités : Leurs attributs et leurs types, ainsi que les relations (many to many, many to one, etc..) les liants aux autres entités.

Ci-dessous, un extrait de code de l’entité playlist :

Une image contenant texte

Description générée automatiquement

Ainsi on retrouve dans nos entités la class Playlist qui fera partie des *business object* qui seront manipulé par nos contrôleurs Rest, service et repository.

Le décorateur @*table* indique le nom de la table associé, @*id* indique que l’attribut sera la clef primaire dans la table, @*column* fait le lien avec le nom des champs en base données.

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

En fonction de nos choix dans le terminal de génération de projet Jhipster, ce dernier génère des fichier xml contenant les informations nécessaires à la création de tables et leurs relations en base de données.

Ci-dessous, un extrait de fichier xml servant à la création de la table playlist :

Une image contenant texte, plaque, capture d’écran

Description générée automatiquement

Ce code xml va être interprété par Liquidbase et transformé par ce dernier en requête SQL pour créer la base de données.

Les fichiers xml contiennent aussi les contraintes de clef étrangère, comme suit :

Une image contenant texte

Description générée automatiquement

Si toutefois nous avions créé la base donnée manuellement directement en script SQL, j’aurai écrit ces deux extraits comme tel :

CREATE TABLE playlist

(

id INT NOT NULL CONSTRAINT PK\_playlist\_id PRIMARY KEY IDENTITY(1,1),

name VARCHAR(255) NOT NULL,

date DATE NOT NULL,

end\_date DATE NULL,

image LONGBLOB NULL

);

ALTER TABLE playlist ADD

CONSTRAINT fk\_playlist\_\_user\_id FOREIGN KEY (user\_id)

REFERENCES [jhi\_user](id) ;

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 les comptes des utilisateurs. Il est donc essentiel pour une application proposant de créer des profils d'utilisateurs et persistant des données, de se prémunir au maximum des attaques les plus courantes. L'application florilèges étant créé à partir de plusieurs frameworks, je vous propose de découvrir ci-dessous les options de sécurité Générer par Jhipster pour notre application, et que nous avons configuré par la suite.

1. L’authentification

L’authentification est l’un des aspects élémentaires de la sécurité d'une application il permet en effet de s'assurer que l'utilisateur ai les droits nécessaires pour avoir accès aux données proposées et pouvoir interagir avec ces dernières.

Le générateur de projets Jhipster génère automatiquement de nombreux éléments de sécurité. L'un des plus intéressants est qu’à la génération de l'entité *user*, un *crud* complet soit proposé sur cette entité tout en implémentant le protocole d'authentification nécessaire à la sécurité du compte et des sessions. Deux comptes son proposer à la génération de l’entité :

* + - * « user, qui est compte utilisateur normal avec l’autorisation « ROLE\_USER » . Son mot de passe par défaut est « user ».
      * « admin, qui est compte utilisateur admin avec l’autorisation « ROLE\_USER »  et « ROLE\_ADMIN ». Son mot de passe par défaut est « admin ».

Avant le déploiement de l’application web réserver aux administrateurs de Florilège, nous avons changé ces mots de passe par défaut.

La connexion au compte utilisateur se fait grâce à une comparaison des login et passwords stocké uniquement en base de données. En outre les passwords font l'objet d'un hashage avant leur persistance en base de données, ce qui évite leur exploitation en cas de fuite de ces données sensibles.

Une fois authentifier par l’application, celle-ci renverra un token , qui sera stocké dans le navigateur de l’utilisateur, et réutilisé pour chaque requête du client vers le serveur . Le token est sécurisé selon la norme JWT (JSON Web Token) qui contient le nom de connexion et les droits de l’utilisateur. Comme le token est signé par le serveur, il ne peut pas être modifié par l’utilisateur.

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 de sensibiliser nos utilisateurs à ce problème, l’application Florilege propose au moment de la création de profil et notamment du mot de passe de donner une indication quant à la résistance de ce dernier.

Une image contenant texte

Description générée automatiquement

Les failles « xss »

Les attaques xss (cross site scripting) ont pour but d’insérer du code Javascript malveillant dans le DOM ( document object model ) de l’application pour par exemple, voler des données utilisateur par la récupération de cookie de session, rediriger vers un autre site ou afficher du contenu malveillant.

Pour contrer ses attaques, Angular, framework que nous utilisons pour développer la partie front-end de Florilèges considère comme non fiable toute valeur insérer dans le DOM. Ainsi il assainit toutes ces valeurs pour éviter l’insertion de script non prévu par le fonctionnement normale de l’application.

En outre, il possible de se protéger des failles xss en appliquant une validation des donnés d’entrée par l’utilisation de regex ou en blacklistant certains caractères comme les balises script, ainsi qu’en encodant les entrées afin qu’elles n’exécutent aucun script, éléments qui n’ont pas encore été mis en place sur Florilège.

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.

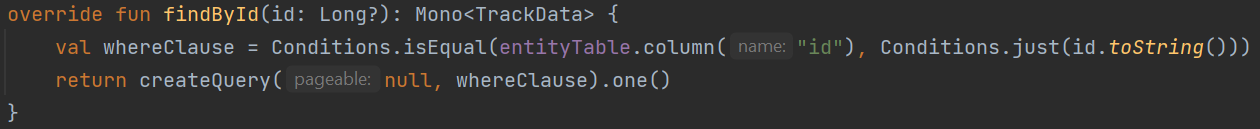
Dans le cas d’une injection SQL, à la place de mettre un login et ou un mot de passe dans le formulaire, un pirate saisi un morceau de code SQL qui une fois insérer dans la requête SQL envoyée à la base de données, modifiera son comportement initial.

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 les méthodes createQuery. 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 Florilège :

Une image contenant texte

Description générée automatiquement



Une image contenant texte

Description générée automatiquement

9. RECHERCHE EFFECTUEE

1. Le besoin d’information

Une des fonctionnalités essentielles de l’application est le classement des clips dans une playlist en fonction des votes alloués par les utilisateurs ayant été invité sur cette playlist.

Une partie des données des clips vidéo de l’application (nommé « *track* » dans le code) au moment de l’écoute son stocké dans un tableau. C’est au final l’index de chaque *track* qui va déterminer l’ordre passage de cette dernière. Il était donc indispensable que l’ordre de chaque *track* dans le tableau soit représentatif de son «rank »(calcul de la différence entre les votes « pour » et « contre ») dans la playlist. Ainsi plus une *track* a un *rank* élevé plus son index dans le tableau doit être faible.

Il était donc nécessaire de trouver une méthode pour trier le tableau d’objet de type Track en fonction d’un attribut numérique de cet objet.

1. La recherche

La 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 international et maximiser les chances de trouver une réponse fiable.

Les mots clef choisit devait être le plus équivoque possible :

« sort » : la fonction recherchée.

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

« TypeScript » : le langage dans lequel on recherche la solution.

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

Une image contenant texte

Description générée automatiquement

De ces trois 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 dans en priorité le site web « developer.mozilla.org ».

1. La solution

Une image contenant texte

Description générée automatiquement

Le site web propose plusieurs méthodes de trie de tableau. Par défaut les méthodes proposées, trient le tableau par ordre croissant en fonction des éléments sélectionné pour le tri. Pour sa simplicité de syntaxe et l’adaptabilité à notre situation. Je choisis d’essayer la méthode ci-dessus.

1. La mise en œuvre

La fonction « .sort((a,b) => a - b) » va trier les éléments d’un tableau en fonction de la valeur de retour de l’opération a – b. Cette opération va s’effectuer sur tous les éléments adjacents du tableau (représentés par a et b). Cette boucle sur le tableau va s’effectué jusqu’à ce qu’il n’y ait est plus de changement a faire dans le tableau.

Dans notre cas nous voulons trier des objets par rapport à un de leur attribut qui est un élément de type numérique. A savoir des objets de type « Track » sur leurs attribut « track.rank ».

La deuxième modification est d’inverser l’ordre de tri. Si le tri par ordre croissant s’effectue par le retour de l’opération de « a-b », le retour de « b-a » inverse le résultat et donc l’ordre de tri (celui recherché dans notre cas, est un ordre décroissant).

Une image contenant texte

Description générée automatiquement



Ainsi nous retrouvons, le tableau « this.tracks » à trier après un vote sur une track. Sur lequel on applique la méthode « .sort » en inversant les paramètres dans l’opération.

Cette méthode nous a donc permit de trier la playlist en lecture a chaque vote et donc de s’assurer que l’ordre de lecture des clips soit respecté.

10. TRADUCTION

1. Présentation

A défaut d’avoir trouvé une documentation suffisamment explicite sur la fonctionnalité précédente, pour pouvoir proposer une traduction dans le cadre du rapport de stage. Je vous propose une traduction sur la présentation de l’architecture MVC tiré du site web : « developer.mozilla.org ».

Texte original :

**MVC** (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic. It emphasizes a separation between the software's business logic and display. This "separation of concerns" provides for a better division of labor and improved maintenance. Some other design patterns are based on MVC, such as MVVM (Model-View-Viewmodel), MVP (Model-View-Presenter), and MVW (Model-View-Whatever).

The three parts of the MVC software-design pattern can be described as follows:

1. Model: Manages data and business logic.
2. View: Handles layout and display.
3. Controller: Routes commands to the model and view parts.

As a web developer, this pattern will probably be quite familiar even if you've never consciously used it before. Your data model is probably contained in some kind of database (be it a traditional server-side database like MySQL, or a client-side solution such as [IndexedDB [en-US]](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API).) Your app's controlling code is probably written in HTML/JavaScript, and your user interface is probably written using HTML/CSS/whatever else you like. This sounds very much like MVC, but MVC makes these components follow a more rigid pattern.

In the early days of the Web, MVC architecture was mostly implemented on the server-side, with the client requesting updates via forms or links, and receiving updated views back to display in the browser. However, these days, more of the logic is pushed to the client with the advent of client-side data stores, and XMLHttpRequest allowing partial page updates as required.

Web frameworks such as [AngularJS](https://en.wikipedia.org/wiki/AngularJS) and [Ember.js](https://en.wikipedia.org/wiki/Ember.js) all implement an MVC architecture, albeit in slightly different ways.

1. Traduction

Texte traduit en français :

MVC (Model-View-Controller) est un modèle en conception de logiciels couramment utilisé pour implémenter des interfaces utilisateur, des données et logique de contrôle. Il met l'accent sur une séparation entre la logique métier du logiciel et l'affichage. Cette "séparation des préoccupations" permet une meilleure division du travail et maintenance améliorée. Certains autres modèles de conception sont basés sur MVC, tels que MVVM (Model-View-Viewmodel), MVP (Model-View-Presenter) et MVW (Model-View-Whatever).

Les trois parties du modèle de conception logicielle MVC peuvent être décrites comme suit :

1. Modèle : gère les données et la logique métier.

2. Vue : manipule la disposition et l'affichage.

3. Contrôleur : achemine les commandes vers le modèle et les parties de la vue.

En tant que développeur Web, ce modèle vous sera probablement assez familier, même si vous ne l'avez jamais utilisé consciemment auparavant. Votre modèle de données est probablement contenu dans une sorte de base de données (peut-être une base de données traditionnelle côté serveur comme MySQL ou une solution côté client comme IndexedDB].) Le code de contrôle de votre application est probablement écrit en HTML/JavaScript, et votre interface utilisateur est probablement écrite en HTML/CSS/qu’importe ce que vous voulez d’autre. Ça ressemble beaucoup au MVC, mais le MVC fait que ces composants suivent un modèle plus rigide.

Au début du Web, l'architecture MVC était la plupart du temps implémentée côté serveur, avec le client qui demande des mises à jour via des formulaires ou des liens, et recevant des vues mises à jour pour afficher dans le navigateur. Cependant, ces temps-ci, une plus grande partie de la logique est poussée vers le client avec l'avènement des stockages de données côté client et XMLHttpRequest permettant des rafraichissements partiels des pages selon les besoins.

Les frameworks Web tels que AngularJS et Ember.js implémentent tous une architecture MVC, bien que de manière légèrement différente.

11. CONCLUSION

Ces 2 mois de mois de stage ont été particulièrement formateurs. J’ai de couvert le fossé qu’il pouvait exister entre les compétences apprises pendant les 6 mois de formation, et ce qu’il pouvait être demandé dans un contexte professionnel. Le plus compliqué pour moi ont été les premières les 4 premières semaines de stage pendant lesquelles ma méconnaissance du framework Angular et des concepts d’asynchronisme m’ont demandé un intense travail personnel.

Je ravi d’avoir participé à ce projet dès la rédaction du cahier des charges, des documents et d’avoir pu proposer mon point de vue sur certains points.

J’ai principalement apprécié de pouvoir travailler sur toute les couches de l’application, et ainsi découvrir et me perfectionner sur les technologie et concepts nécessaire à un développeur d’application Fullstack.

Le télétravail était un point qui m’inquiétait en début de stage, mais les outils de communication ainsi la disponibilité des membres de l’équipe nous ont permis travailler efficacement tout on long du projet. Cela m’a aussi permis de progresser sur les applications de versionning.

Je suis vraiment satisfait de ces 2 mois de stages même si le temps nous a manqué pour terminer certaines des fonctionnalités principales de l’application.

Ce projet m’a montré que malgré mon investissement pendant cette formation, le métier de développeur nécessite de se former en permanence. Le stage m’a confirmé que le développement est le domaine dans lequel je souhaite évoluer ces prochaines années. Ainsi je poursuis ma formation avec le cursus cda, pour parfaire mes connaissances et intégré sereinement le développement web professionnel.