



RAPPORT D'ACTIVITES



Table des matières

INTRODUCTION ET PRESENTATION DE L'ENTREPRISE.....	3
• <i>Remerciements</i>	3
• <i>Présentation de l'entreprise</i>	3
• <i>Présentation du poste et de son environnement technique</i>	3
LEXIQUE ET DEFINITION.....	4
LE(S) PROJET(S).....	6
• <i>Liste des compétences du référentiel qui sont couvertes par le(s) projet(s)</i>	6
• <i>Résumé du projet</i>	7
• <i>Cahier des charges ou expression des besoins du projet</i>	8
• <i>Analyse et Gestion de projet (planning et suivi, environnement humain et technique, ...)</i>	12
• <i>Spécifications fonctionnelles du projet</i>	17
• <i>Spécifications techniques, y compris pour la sécurité</i>	19
• <i>Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité</i>	31
1- <i>Création des parties Back End et Front End du projet</i>	31
2- <i>Connexion à la base de données</i>	33
3- <i>Générer le modèle de données</i>	34
4- <i>Implémentation des composants d'accès aux données</i>	35
5- <i>Connexion du l'API Rest au Front avec ReactJS</i>	39
6- <i>Présentation d'une des fonctionnalité du front de l'application SIGEE</i>	39
• <i>Présentation du jeu d'essai élaboré par le candidat de la fonctionnalité la plus représentative</i>	47
• <i>Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité</i>	50
• <i>Description d'une situation de travail ayant nécessité une recherche et effectuée par le candidat durant le projet</i>	51
BILAN ET CONCLUSION	52
• <i>Bilan et perspectives du (ou des) projet(s) présenté(s)</i>	52
• <i>Conclusion</i>	52
Annexes	53
Sélection des colonnes des Engins	55
Avancement.....	59
ModelSerializer	60
• <i>Webographie</i>	61

INTRODUCTION ET PRESENTATION DE L'ENTREPRISE

- **Remerciements**

Je souhaite exprimer ma gratitude envers les membres de l'équipe d'Ubidreams pour m'avoir donné l'opportunité de faire mon alternance au sein de leur agence. Je tiens tout particulièrement à remercier Monsieur Max Tessier, Directeur Général de l'agence, pour m'avoir accepté en cours d'année. Je suis également reconnaissant envers Ahmed, Lead Tech Backend et Ingénieur Data, Mike, Ingénieur IOT et Systèmes Embarqués, ainsi que toute l'équipe qui a été bienveillante et attentive à mes besoins tout au long de mon parcours dans l'entreprise.

- **Présentation de l'entreprise**

Ubidreams est une agence digitale basée à Aytré en France. Fondée en 2008, elle est spécialisée dans le développement d'applications mobiles, la conception de sites web, le design UX/UI, la création d'objets connectés, la cybersécurité et la conformité RGPD. Ubidreams accompagne les entreprises et les assureurs dans l'amélioration durable de leurs sites web, la maintenance et la création d'applications web/Mobile sur mesure. Elle dispose d'une équipe compétente et bienveillante, constituée d'experts techniques, de designers et de chefs de projets, tous dévoués à la satisfaction de leurs clients. La société utilise des technologies de pointe pour offrir des solutions sur mesure répondant aux besoins spécifiques de chaque client.

- **Présentation du poste et de son environnement technique**

En tant que développeur d'application, j'utilise un ordinateur professionnel équipé d'un SSD de 512 Go, de 8 Go de RAM et d'un processeur Intel. Pour communiquer avec mes collègues, nous utilisons la plateforme de communication Teams et nous organisons des réunions hebdomadaires pour faire le point sur l'avancement de chaque projet. Pour les visioconférences, nous utilisons GoogleMeet et pour la gestion de projet, nous utilisons Jira.

En ce qui concerne les langages de programmation, j'utilise principalement ReactJS, Python avec le Framework Django, ainsi que PHP. Pour chacun de ces langages, j'utilise un environnement de développement intégré (IDE) spécifique : PHP Storm pour PHP, WebStorm pour ReactJS et Visual Studio Code pour mes projets en général.

En matière de systèmes de base de données, j'ai l'habitude d'utiliser Oracle Database, MariaDB, MySQL et PostgreSQL. Pour gérer les transferts de fichiers, j'utilise le terminal et parfois un client FTP tel que Filezilla.

Comme mentionné précédemment, j'utilise encore PHP pour certains projets et pour les faire tourner en local, j'utilise MAMP. En ce qui concerne la gestion de versionning, j'utilise Git et plus particulièrement la plateforme GitHub pour gérer tous les projets de nos clients.

En somme, je suis équipé des outils nécessaires pour mener à bien mes projets de développement d'application. J'utilise des langages de programmation variés et des IDE spécifiques pour chacun d'eux, ainsi que des plateformes de communication, de gestion de projet et de versionning pour faciliter la collaboration avec mes collègues et clients.

LEXIQUE ET DEFINITION

- Backlog est un terme informatique qui définit une liste fonctionnelle de travail qui doit être fait pour compléter un produit ou un service informatique. C'est un terme largement utilisé dans la méthode Scrum et d'autres méthodologies agiles.
- Une API (interface de programmation d'application ou « interface de programmation d'application ») est une interface logicielle qui permet à un logiciel ou à un service de se « connecter » à un autre logiciel ou service pour échanger des données et des fonctions.
- Django REST Framework (DRF) est une bibliothèque Python qui permet de créer des API RESTful (Representational State Transfer) pour les applications Django. DRF fournit des outils pour simplifier la création d'API, y compris la validation des données, la gestion des autorisations, la pagination et la sérialisation des données.
- ReactJS est une bibliothèque JavaScript open-source développée par Facebook. Elle est utilisée pour la création d'interfaces utilisateur interactives et réactives pour les applications web
- Les attaques XSS impliquent l'injection de code malveillant dans des sites Web par ailleurs dignes de confiance. Une attaque XSS se produit lorsque des cybercriminels injectent des scripts malveillants dans le contenu d'un site Web ciblé, qui est ensuite inclus dans le contenu dynamique reçu par le navigateur de la victime. Il est impossible pour le navigateur de faire la distinction entre les balises valides et les balises pirates, il les remplit donc simplement.
- CSRF (Cross-Site Request Forgery) est une technique d'attaque informatique qui exploite la confiance entre un utilisateur authentifié et un site web. L'attaque CSRF se produit lorsqu'un utilisateur est induit en erreur pour effectuer une action sur un site web sans son consentement
- L'injection SQL (parfois abrégée en SQL) est une vulnérabilité dans laquelle un hacker utilise du code SQL (Structured Query Language) pour manipuler

une base de données et accéder à des informations potentiellement sensibles. Il s'agit de l'une des attaques les plus courantes et les plus menaçantes, car elle peut potentiellement être utilisée pour compromettre tout un site Web ou application utilisant une base de données SQL (ce qui est la plupart).

- Clickjacking (ou "UI redress attack") est une technique d'attaque informatique qui exploite la confiance de l'utilisateur envers les interfaces graphiques des sites web. L'attaque de clickjacking consiste à superposer une page web malveillante sur une page web légitime, de manière à tromper l'utilisateur pour qu'il clique sur des éléments qui sont en réalité situés sur la page malveillante, mais qui sont visuellement masqués par la page légitime.
- SSL/HTTPS : SSL (Secure Sockets Layer) et HTTPS (Hyper Text Transfer Protocol Secure) sont deux protocoles de sécurité pour les communications sur internet.

SSL est un protocole de sécurité qui permet de chiffrer les communications entre un client (comme un navigateur web) et un serveur (comme un site web). SSL assure également l'authentification du serveur, c'est-à-dire qu'il permet de vérifier que le site web que l'utilisateur visite est bien celui qu'il prétend être. SSL a été remplacé par TLS (Transport Layer Security), mais les deux termes sont souvent utilisés de manière interchangeable.

HTTPS est un protocole de communication sécurisé qui utilise SSL ou TLS pour chiffrer les données échangées entre un client et un serveur web. Contrairement à HTTP qui transmet les données en texte clair, HTTPS assure la confidentialité, l'intégrité et l'authenticité des données échangées entre un client et un serveur. Les sites web qui utilisent HTTPS affichent un cadenas vert dans la barre d'adresse du navigateur.

LE(S) PROJET(S)

- Liste des compétences du référentiel qui sont couvertes par le(s) projet(s)

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles
1	Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	1	Maquetter une application
		2	Développer une interface utilisateur de type desktop
		3	Développer des composants d'accès aux données
		4	Développer la partie front-end d'une interface utilisateur web
		5	Développer la partie back-end d'une interface utilisateur web
2	Concevoir et développer la persistance des données en intégrant les recommandations de sécurité	6	Concevoir une base de données
		7	Mettre en place une base de données
		8	Développer des composants dans le langage d'une base de données
3	Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité	9	Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
		10	Concevoir une application
		11	Développer des composants métier
		12	Construire une application organisée en couches
		13	Développer une application mobile
		14	Préparer et exécuter les plans de tests d'une application
		15	Préparer et exécuter le déploiement d'une application

Selon les exigences du référentiel de certification, le projet implique les compétences suivantes :

- Pour l'activité 1 : « Concevoir et Développer des composants d'interface utilisateur en intégrant les recommandations de sécurité » :
 - Maquetter une application
 - Développer des composants d'accès aux données

- Développer la partie front-end d'une interface utilisateur Web
- Développer la partie back-end d'une interface utilisateur Web
- Pour l'activité 2 : « Concevoir et développer la persistance des données en intégrant les recommandations de sécurité » :
 - Mettre en place une base de données
 - Développer des composants dans le langage d'une base de données
- Pour l'activité 3 : « Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité » :
 - Concevoir une application
 - Développer des composants métier
 - Construire une application organisée en couches
 - Préparer et exécuter les plans de tests d'une application (dans le planning)
 - Préparer et exécuter le déploiement d'une application (dans le planning)

- Résumé du projet

- En français

Dans le cadre de la validation de mes compétences en tant que Concepteur Développeur d'Application, j'ai travaillé sur la refonte et l'amélioration du back-office et de l'interface de gestion des engins de l'application SIGEE pour les Marins-Pompiers de Marseille. L'application initiale avait été réalisée en PHP et je l'ai reconstruite en utilisant le Framework Django Rest pour le Back-End et ReactJS pour le Front-End. Ce projet présentait plusieurs contraintes telles que la non-modification de la base de données existante car elle était en communication avec d'autres bases de données via un ETL, ainsi que la connexion avec Active Directory et l'ERP de Systel.

N'ayant pas accès à toutes les documentations du projet initial, j'ai dû concevoir toutes les informations nécessaires à partir du code PHP qui m'a été fourni, ainsi que d'une partie des documents de la version 1 de SIGEE qui était sous WinDev et toujours en exploitation. J'ai également eu une rencontre avec le client à Marseille pour obtenir les listes des droits des utilisateurs et de me faire une démonstration de l'application SIGEE sous PHP. J'ai créé le modèle conceptuel de données (MCD), les diagrammes de classe, le diagramme de séquence, le modèle relationnel de données (MRD) et les uses cases pour comprendre chaque fonctionnalité de l'application et voir comment les mettre en avant dans le back-office afin qu'un utilisateur puisse facilement l'utiliser.

Le projet a été réalisé en respectant les contraintes initiales tout en apportant des améliorations significatives au niveau du back-office et de l'interface de gestion des engins. J'ai utilisé Django Rest Framework pour créer des API performantes et réactives, et j'ai utilisé ReactJS pour concevoir une interface utilisateur moderne et intuitive.

➤ En Anglais

As part of validating my skills as an Application Designer and Developer, I worked on redesigning and improving the back-office and equipment management interface of the SIGEE application for the Marins-Pompiers de Marseille. The original application was developed in PHP, and I reconstructed it using Django Rest Framework for the back-end and ReactJS for the front-end. The project had several constraints, such as not modifying the existing database as it was communicating with other databases via an ETL, as well as the connection with Active Directory and Systel's ERP.

Since I did not have access to all of the initial project documentation, I had to create all the necessary information from the provided PHP code, as well as some documents from version 1 of SIGEE, which was developed in WinDev and still in use. I also had a meeting with the client in Marseille to obtain user rights lists and to receive a demonstration of the SIGEE application in PHP. I created the conceptual data model (MCD), class diagrams, sequence diagrams, data relational model (MRD), and use cases to understand each functionality of the application and how to showcase them in the back-office to ensure user-friendliness.

The project was completed while adhering to the initial constraints while bringing significant improvements to the back-office and equipment management interface. I used Django Rest Framework to create high-performance and responsive APIs, and I used ReactJS to design a modern and intuitive user interface.

- ***Cahier des charges ou expression des besoins du projet***

- 1- **Contexte du projet**

L'application de gestion des Engins (SIGEE) du bataillon Marin Pompier de Marseille (BMPM) est en cours de développement en utilisant PHP comme langage de programmation. L'application utilise une base de données Oracle pour stocker les données et communique avec d'autres bases de données via un ETL (Extract Transform Load).

- 2- **Expression des besoins du projet**

- Présentation des fonctionnalités actuelles du back-office

Actuellement, le back office permet de gérer (créer, modifier, supprimer, afficher et rechercher) :

- Les engins, incluant la gestion affectée des :
 - Informations opérationnelles et ordonnancement
 - Informations administratives
 - Comptabilité
 - SIC
 - Catalogue
 - Équipements
 - Réparations
 - Interventions
 - Affectations ADM/PHY
 - Garage
 - Documents
 - Les états des commandes
 - Des réparations
 - Des catégories d'engins
 - Des commandes
 - Liste et recherche avec filtrage et sélection des attributs pour engins
 - Liste et recherche avec filtrage des réparations
 - Un service d'enregistrement des cartes vertes des engins avec accès en liste
- Identification des fonctionnalités manquantes ou à améliorer
 - Gestion automatique des duplicatas. Afin d'éviter la création de doublons dans la base de données, gérer la détection des doublons et permettre la fusion des données des doublons de manière automatique.
 - Une analyse automatique des documents scannés des cartes vertes pour les relier automatiquement à un engin grâce à l'immatriculation.
 - Gérer l'affichage en fonction des droits de l'utilisateur en cours d'utilisation avec un popup. Ceci permettrait de faciliter la limitation d'accès aux données et à la gestion.
 - Recherche non intuitive et problématique au niveau de la recherche avec date.
 - Champ à ajouter sur les engins : Document Crit'Air
 - Permettre de générer une fiche imprimable pour l'affichage d'un détail engin.
 - Accès PDF aux documents liés aux engins en suppléments des affichages images.
 - La recherche en liste des engins nécessite 3 filtres par défaut pour la sélection d'attribut avec deux possibilités de filtres enregistrés par l'admin.

- Nécessité d'amélioration ergonomique des vues, faciliter l'utilisation pour les pompiers et utilisateurs avec un accès plus fluide et un affichage visuel des champs des affichages selon les droits de modification.
- Étude des contraintes techniques
 - Contraintes base de données : Oracle
 - Contraintes de système d'authentification : Active Directory
 - Interdiction de supprimer et modifier les champs existant dans la base de données.
 - Possibilité d'ajouter des champs si nécessaire
 - Les données doivent pouvoir être communiqué via leur ETL
 - Besoin d'utilisation d'une machine Linux avec si possible une image de la base de données pour représenter le cas d'utilisation classique.
- ETL

Chaque vue est sujets à des accès et interdictions suivant le rôle de l'utilisateur. L'interfaçage entre SIGEE et l'ERP de Systel est dans un seul sens, l'ETL s'en chargera de la récupération des données à partir de la base de données de SIGEE afin de les enregistrer dans le Data Warehouse. Donc si on ne touche pas à cette base de données, ce mécanisme ne sera pas touché ou impacté, et d'après le besoin actuel de BMPM je n'ai pas besoin de toucher à cette base de données ni à sa structure, et même si on refait la solution avec d'autres langages de développement on peut toujours garder la même base de données et donc ne rien modifier au niveau ETL.

Sauf pour une demande supplémentaire de leur part et qui nécessite une modification du modèle de données par exemple (ajouter d'une ou plusieurs tables, ajouter un lien entre deux tables.) et que cette modification impacte la logique de l'ETL, dans ce cas il faut appliquer des modifications au niveau de l'ETL, et donc l'intervention de l'entreprise qui gère l'ETL concerné (Systel ou autres).

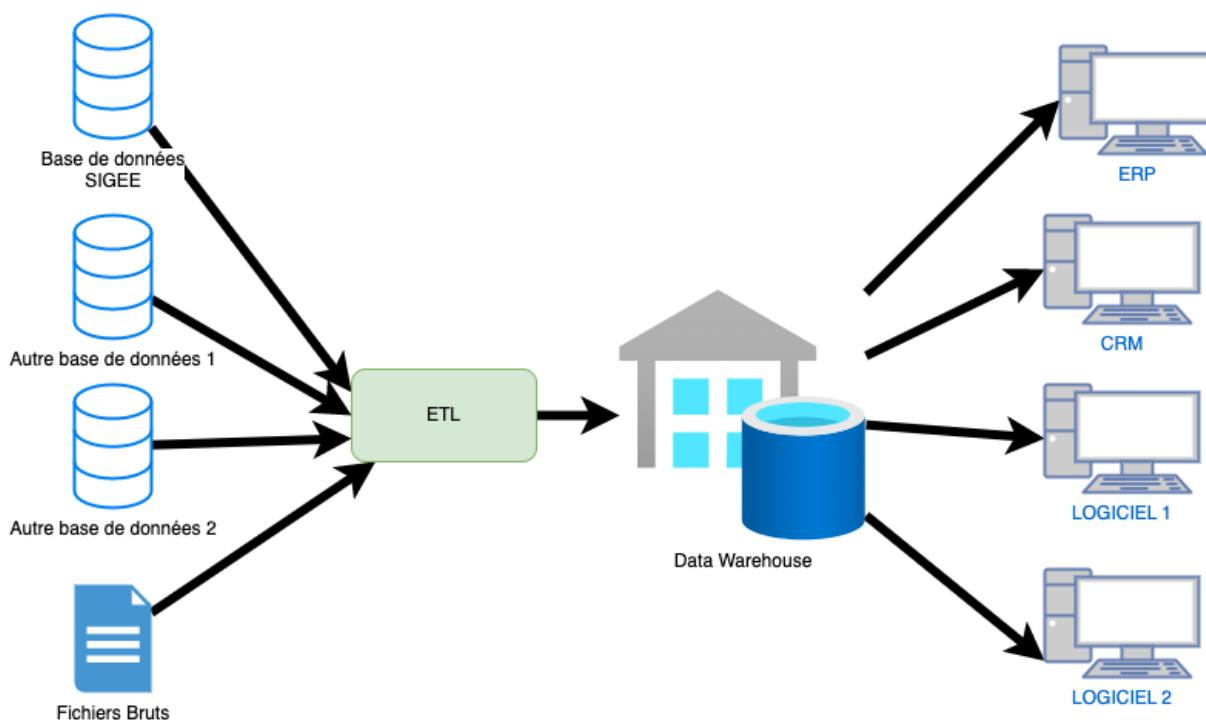


Fig. 1 Système ETL

3- Les rôles des différents utilisateurs du projet

	Consultation	Extraction	Ecriture	Insertion doc	Création	Droits
ADMIN	X	X	X	X	X	Droit A
COTEC	X	X	X	X	X	Droit B
GARAGE CIS	X		X			Droit C
GARAGE GPT	X		X			Droit C
TRANS	X		X			Droit D
TRANS CIS	X		X			Droit D
INGENIEUR FLOTTE	X		X			Droit E
CTB	X		X	X		Droit F
CTX	X		X	X		Droit G
OPS	X		X			Droit H
BCM	X		X			Droit I
TOUS BMPM	X					Droit J

Le tableau ci-dessus répertorie les différents droits d'accès attribués à chaque utilisateur pour les différentes actions telles que la consultation, l'extraction, l'écriture, l'insertion de documents et la création de fichiers. L'utilisateur ayant les droits les plus élevés est l'administrateur, qui a accès à toutes les actions possibles.

L'utilisateur COTEC ont la possibilité de consulter, extraire, écrire, insérer des documents et créer des éléments, à l'exception des champs SIC et OPS. Cela signifie qu'ils peuvent accéder et modifier tous les autres champs disponibles.

Les utilisateurs GARAGE CIS et GARAGE GPT ont la possibilité de consulter, et créer des éléments, pour les champs Position, Kilométrage et Horamètre. Mais, cela signifie qu'ils ne peuvent pas accéder et ni modifier tous les autres champs disponibles.

Les utilisateurs TRANS et TRANS CIS ont des droits de consultation et d'écriture, pour le champ SIC. Mais, ils ne peuvent pas accéder et ni modifier tous les autres champs disponibles.

L'utilisateur INGENIEUR FLOTTE a des droits de consultation et d'écriture pour le champ garage Engin. Ils ne peuvent pas accéder et ni modifier tous les autres champs disponibles.

Les utilisateurs CTB ont la possibilité de consulter, insérer des documents et créer des éléments, pour les champs Ordonnancement, Réparation Engins, Conformité/Levage.

Les utilisateurs CTX ont des droits d'insérer des documents, d'écriture et de création, pour les champs Date de fin de Validité Assurées, Place Assurées, mais en plus ils peuvent ajouter les Cartes Vertes de la compagnie d'assurances.

Les utilisateurs OPS dispose uniquement de droits de consultation, d'écriture et d'extraction pour les champs OPS, l'export et désapparier pour Start.

Les utilisateurs BCM a uniquement des droits de consultation et d'écriture pour les champs de comptabilités.

Enfin, l'utilisateur TOUS BMPM a seulement le droit de consulter et d'accéder à certaines parties de l'application.

- Analyse et Gestion de projet (planning et suivi, environnement humain et technique, ...)

- Planning et Suivi

Le projet est composé en plusieurs étapes qui se déroule comme suit :

- Etape 1 : Réunion de Lancement avec Mon maitre d'apprentissage, les administrateurs de SIGEE, et le centre technique pour l'environnement technique utilisé ;
- Etape 2 : Rencontre Client à Marseille, pour voir l'application existante opérationnel, et d'identifier leurs différents besoins, amélioration voulue.

- Etape 3 : Mise en place d'un environnement de développement et d'un serveur virtuel pour faire les tests avec le code existante et de la copie de la BDD ;
- Etape 4 : Création d'un nouveau projet Front End en utilisant le Framework ReactJS ;
- Etape 5 : Création d'un nouveau projet Back End, sans toucher à leur base de données vu qu'il communique avec un ETL et d'autre base de données, en utilisant le Framework Django Rest Framework.
- Etape 6 : Test et Déploiement

L'avancement de mon travail étant suivi grâce au logiciel Jira dont l'utilisation sera détaillée dans la partie suivante

1- La méthode Agile Scrum

Les projets de l'entreprise sont mis en œuvre selon la méthodologie Agile Scrum. Cette approche de la gestion de projet est basée sur les principes de collaboration, de flexibilité et d'itération, qui permettent de répondre rapidement et efficacement aux besoins du projet et à l'utilisateur final.

La méthode Agile Scrum divise le projet en plusieurs itérations appelées sprint, cela peut durer entre 2 à 4 semaines. Chaque Sprint débute par une réunion de planification, au cours de laquelle les membres de l'équipe décident les tâches à accomplir pour atteindre les objectifs fixés pour le sprint. A la fin de chaque sprint, une réunion de rétrospective est organisée entre le Product Owner (BMPM), le Scrum Master et l'équipe de développement, pour évaluer les développements réalisés et identifier les axes d'améliorations pour les prochains sprints. (Annexe Fig. 42)

Les avantages d'utiliser la méthode Agile Scrum :

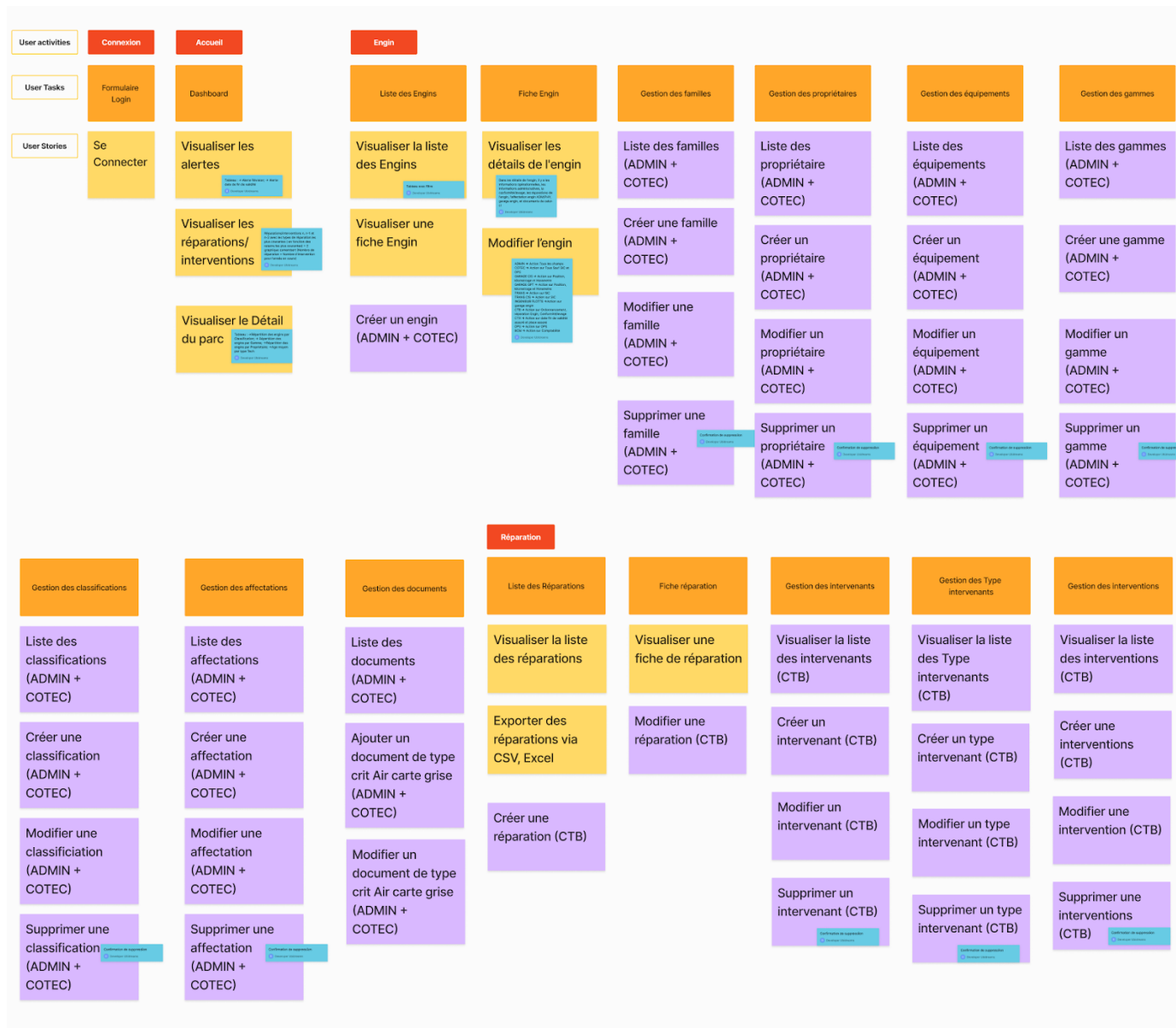
Premièrement, cette approche permet une collaboration très étroite entre les membres de l'équipe et les utilisateurs finaux, qui sont encouragés à participer tout au long du projet.

Des itérations courtes garantissent également que les besoins des utilisateurs finaux sont pris en compte rapidement et que les modifications nécessaires peuvent être apportées sans affecter l'ensemble du projet ;

Enfin, la transparence, la flexibilité et l'agilité de la méthode Agile Scrum permettent à l'équipe de s'adapter facilement aux changements et aux imprévus tout en restant concentré sur les objectifs du projet.

2- Story Mapping

Le Story Mapping est le processus d'organisation des fonctionnalités du produit en une série d'histoires d'utilisateurs ou de scénarios. Ces histoires sont ensuite organisées hiérarchiquement pour créer une carte de l'ensemble du produit. À l'aide de cette carte, il est possible de mieux comprendre les besoins du client, de prioriser les activités selon leur importance et de suivre leur évolution tout au long du projet.



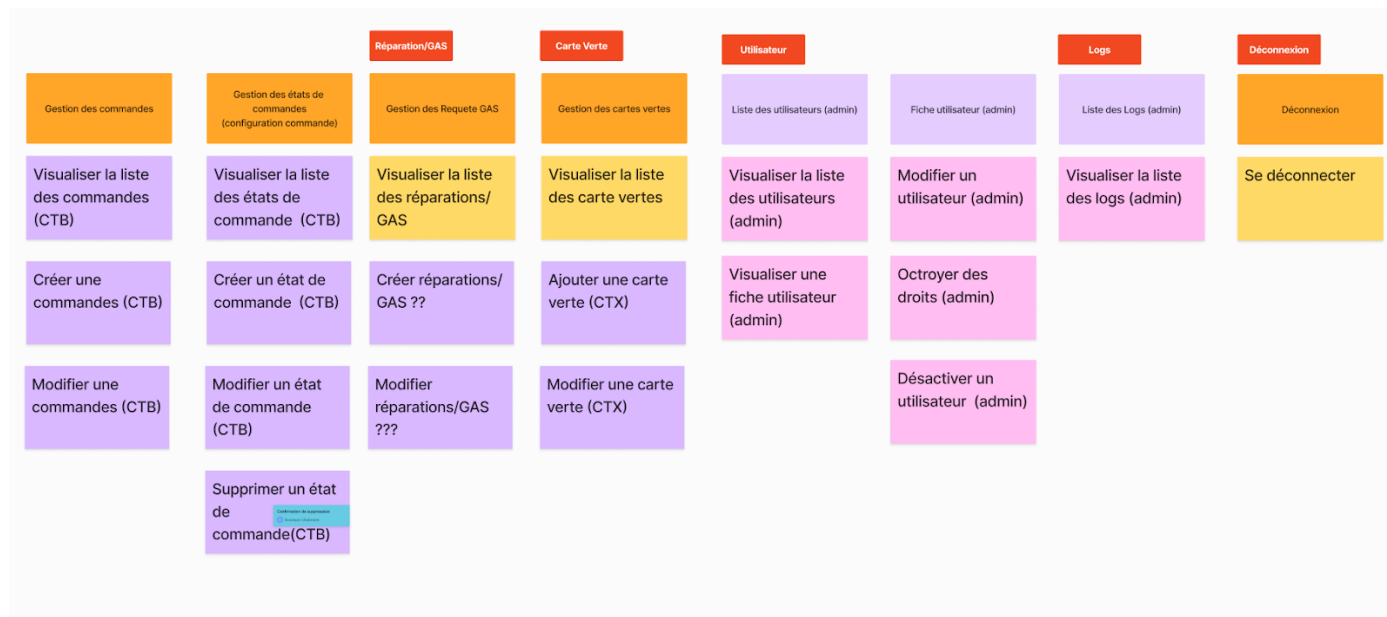


Fig. 2 Story Mapping de SIGEE

3- Planification

Vous trouverez tous les détails des tâches à faire pour la partie développement du projet dans le Backlog (Annexe Fig. 40) et planification Jira (Annexe Fig. 41)

4- Suivi de l'avancement

Le logiciel Jira (Annexe Fig. 47) est utilisé pour suivre l'avancement des différentes tâches. Cet outil permet aux équipes de s'organiser efficacement, de communiquer de manière durable et de visualiser le projet en un coup d'œil grâce à des tableaux de bord personnalisés.

Les tableaux de bord des différents sprints comportaient plusieurs colonnes : À FAIRE, EN COURS, EN REVUE et TERMINÉ.

Traditionnellement, une tâche suit les étapes suivantes : Au début du sprint, elle est positionnée dans la colonne "À FAIRE". Lorsqu'un membre de l'équipe commence à travailler sur la tâche, il doit la déplacer dans la colonne "EN COURS". Pour effectuer la tâche, le développeur doit créer une branche portant le numéro de ticket correspondant.

Une fois la tâche terminée, elle doit être placée dans la colonne "EN REVUE". Les tâches placées dans cette colonne doivent être relues par une tierce personne. Cette relecture de code est systématique.

Après la relecture, la tâche a deux possibilités : si des corrections sont nécessaires, le relecteur la replace dans la colonne "À FAIRE" et ajoute des commentaires expliquant les modifications à apporter et pourquoi. Si la tâche ne nécessite pas de corrections, le relecteur la place dans la colonne "TERMINER" et le code peut être fusionné dans la branche "principal".

- **Préparation des environnements de travail**

Durant cette période nous travaillons avec l'équipe infrastructure de BMPM afin de préparer les environnements nécessaires pour commencer les développements tel que la configuration serveur, l'accès à la base de données, la création des utilisateurs AD de tests ... etc.

- **Sprint 1**

Élément à développer :

- Authentification
- Gestion des utilisateurs

- **Sprint 2**

Élément à développer :

- Gestion des catégories d'engin
- Gestion des propriétaires d'engins
- Gestion des équipements d'engins
- Gestion des gammes d'engins
- Gestion des classifications d'engins
- Gestion des affectations d'engins
- Gestion des Marques
- Gestion des appellations commerciale
- Gestion des Type OPS
- Gestion des Type Technique
- Gestion des Type Servitude
- Gestion des Équipeurs

- **Sprint 3**

Élément à développer :

- Gestion des Type Attelage
- Gestion des équipements
- Gestion des documents de l'engin
- Gestion des réparations engins dans la fiche engin
- Gestion des Interventions engins dans la fiche engin
- Gestion des affectations engins ADM/PHY dans la fiche engin
- Gestion des garages engins dans la fiche engin
- Gestions des cartes verte (amélioration à faire avec script de récupération information)
- Gestion des Conformité/Levage dans la fiche engin
- Gestion des interventions liée à une réparation
- Gestion des intervenants
- Gestion des commandes
- Gestion des états de commandes
- Gestion des Requêtes GAS
- Gestions des types Intervenants

- **Sprint 4**

Élément à développer :

- Page d'accueil
 - Gestion des Engins
 - Gestion des Réparations
 - Gestion des logs
- **Recette et mise en production**
 - **Spécifications fonctionnelles du projet**

Les utilisateurs de SIGEE qui sont préalablement connectés via Active Directory seront les cibles de la partie Gestion Engin.

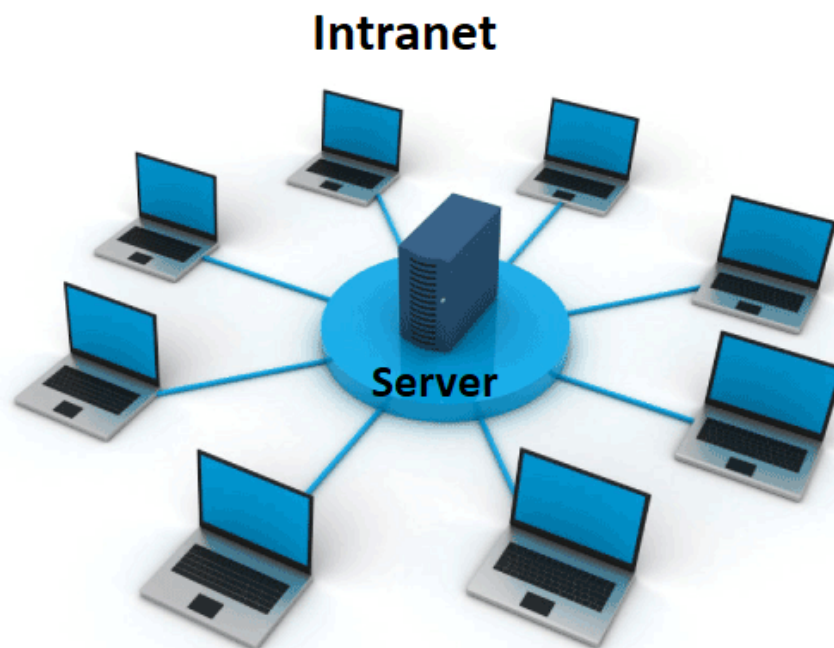


Fig. 3 Système intranet

L'application SIGEE sera accessible uniquement en interne (intranet), et pas en externe (extranet).

Tous les utilisateurs pourront accéder à la page Liste Engins, mais seuls les utilisateurs ayant des droits d'administration et Cotec pourront ajouter et enregistrer un engin sur la page Création d'un Engin.

La création d'un engin nécessitera 7 étapes :

- Étape 1 : les champs requis seront le type d'engin, les dates de la 1ère mise en circulation, d'entrée en service et de fin de validité, la durée de vie théorique de l'engin, son immatriculation et sa carrosserie.
- Étape 2 : les champs requis seront la catégorisation de l'engin, son année, son numéro, le propriétaire de l'engin, le type de classification et le type de gamme.

- Étape 3 : les champs requis seront la marque, l'appellation commerciale, l'état de l'engin, le type OPS, le type technique, le type de servitude, le numéro de dossier, le numéro de folio, le statut technique et le statut OPS.
- Étape 4 : les champs requis seront les affectations administrative et physique, la référence, le numéro d'inventaire Ville, le numéro, l'équipement, le numéro de série, le numéro BCM, la catégorie d'affectation et le type d'attelage.
- Étape 5 : les équipements devront être listés avec leur date de montage sur l'engin.
- Étape 6 : les champs requis seront le nombre de places assurées, la puissance fiscale, le prix d'achat, la longueur, la largeur et la hauteur en cm, la vitesse en km/h et le PTAC en kg.
- Étape 7 : les doubles de clé et les documents à importer (titre, date de fin de validité - qui ne sera pas obligatoire).

Il est nécessaire de remplir chaque champ obligatoire avec les informations relatives aux documents de l'engin et de répondre aux exigences suivantes :

- Pour le champ "Type d'engin", l'utilisateur doit choisir s'il s'agit d'un engin terrestre ou nautique.
- Pour les champs de dates, le format doit être YYYY/MM/DD, sinon un message d'erreur s'affichera indiquant que le format de la date ne correspond pas.
- La durée de vie théorique de l'engin doit être un nombre supérieur à zéro.
- Pour le champ "Immatriculation", le format doit être de la forme AB-123-CD pour les nouveaux engins sortis depuis la réforme de 2009 et ne doit pas dépasser 10 caractères.
- Pour le champ "Carrosserie", aucune exigence particulière n'est mentionnée.
- Les champs "Catégorisation", "Propriétaire", "Type classification", "Type gamme", "Marque", "Appellation commerciale", "Etat", "Type OPS", "Type technique", "Type servitude", "Statut Tech", "Statut OPS", "Affectation physique/administrative", "Catégorisation affectation" et "Type d'attelage" sont des listes déroulantes contenant les données saisies précédemment.
- Pour le champ "Année", il doit contenir quatre caractères numériques.
- Les champs "N°Dossier", "N°Folio" et "N°BCM" ne doivent pas dépasser 20 caractères alphanumériques et peuvent contenir des majuscules et des espaces.
- Le champ "N°Série" correspond au numéro VIN de la carte grise, un identifiant unique de 17 caractères attribué à chaque véhicule automobile produit.
- Pour ajouter un équipement, l'utilisateur doit cliquer sur le bouton correspondant pour faire apparaître une liste déroulante contenant les équipements enregistrés précédemment et renseigner la date de montage au format YYYY/MM/DD.
- Le champ "Places assurées" peut contenir 2 caractères numériques, indiquant le nombre de places de l'engin.

- Le champ "Puissance fiscale" est utilisé à des fins administratives et peut contenir 2 caractères numériques.
- Les champs "Longueur", "Largeur" et "Hauteur", exprimés en cm, peuvent contenir 3 caractères numériques. En cas d'erreur, un message s'affichera si la longueur ou la largeur est inférieure à 100 cm.
- Le champ "PTAC", exprimé en kg, peut contenir 5 caractères numériques. En cas d'erreur, un message s'affichera si le PTAC est inférieur à 1000 kg.
- Le champ "Double des clés" n'est pas obligatoire à remplir, car l'engin peut ne pas posséder de double de clé.
- Le champ "Importation des documents" doit comporter un titre qui ne doit pas dépasser les 26 caractères alphanumériques, une date de fin de validité qui n'est pas obligatoire, et les formats des documents acceptés sont PDF, jpg et png.

Après avoir rempli toutes les étapes de création d'un nouvel engin, l'utilisateur peut l'enregistrer et le voir apparaître automatiquement dans la liste des engins existants. En fonction de leurs droits d'accès, les utilisateurs peuvent également modifier les informations d'un engin déjà enregistré en accédant à sa page de modification. Certains champs, tels que ceux liés aux renseignements de la radio, ne peuvent être modifiés que par les utilisateurs ayant les droits TRANS ou TRANS CIS. Cette fonctionnalité permet de garantir la mise à jour des informations et de limiter les erreurs potentielles.

• Spécifications techniques, y compris pour la sécurité

- Use Case

Un "cas d'utilisation" ou "use case" décrit une série d'actions entreprises par un système en interaction avec des acteurs dans le but d'atteindre un objectif précis. En d'autres termes, il s'agit d'un modèle qui permet de concevoir l'affichage des fonctionnalités et des interfaces en fonction des droits d'accès des utilisateurs.

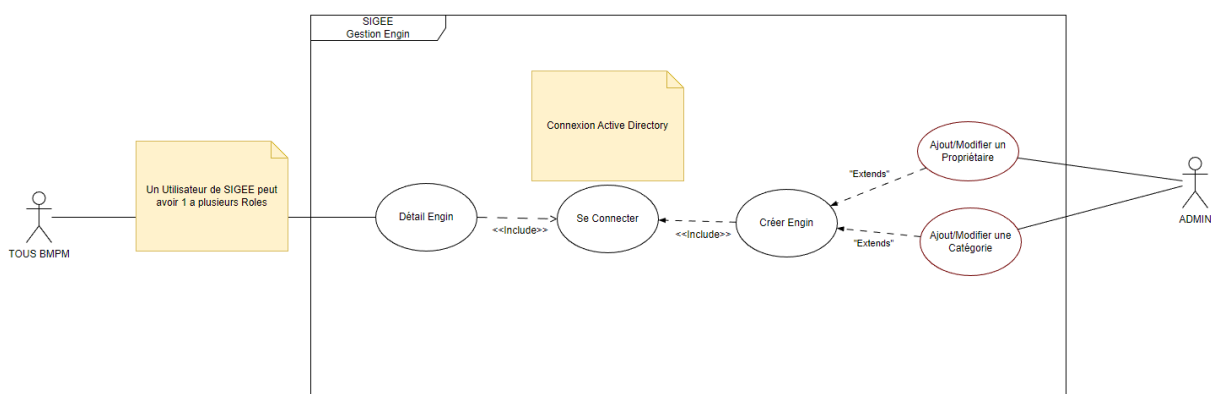


Fig. 4 : Diagramme Use Case correspondant au gestion Engin

Le schéma de cas d'utilisation relatif à la section "gestion Engin" de l'application SIGEE pourrait être représenté par le diagramme présenté sur la figure 4

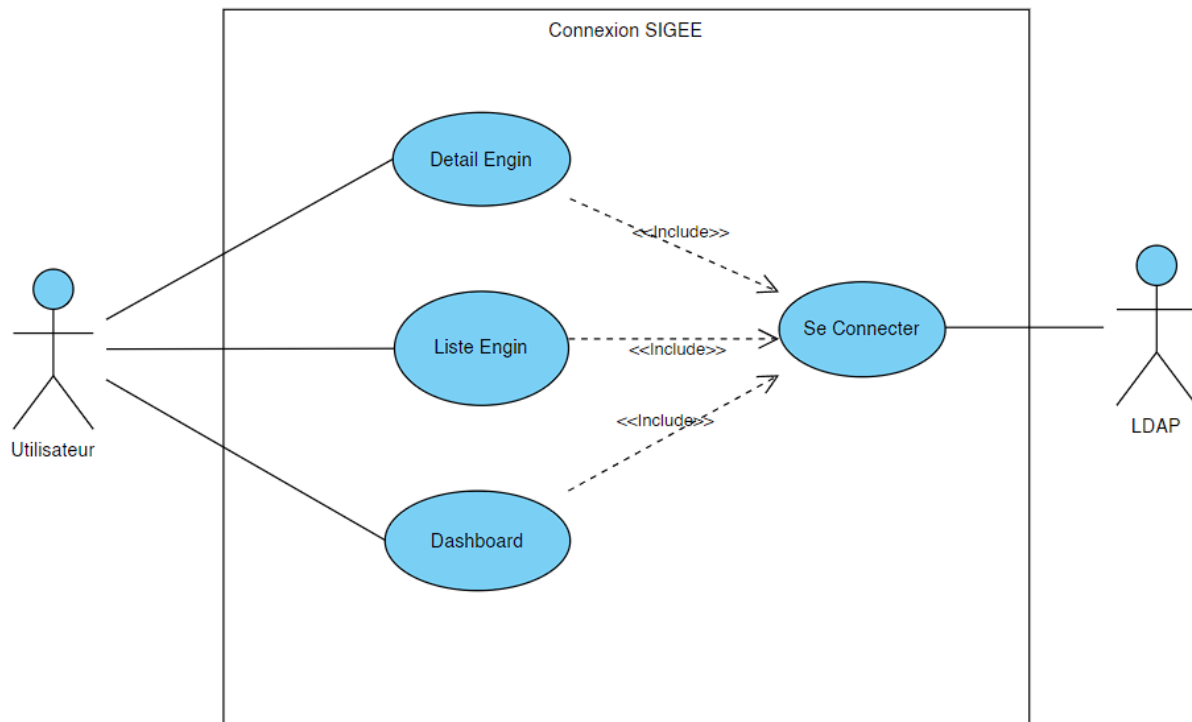


Fig. 5 Diagramme de cas utilisation correspondant à la connexion

L'accès à l'application de SIGEE, requiert que l'utilisateur dans la base de données de SIGEE et dans Active Directory pour pouvoir se connecter.

- Diagramme de Séquence

Un diagramme de séquence est un type de diagramme d'interaction qui aide à décrire l'interdépendance de plusieurs objets. Il montre comment les objets interagissent les uns avec les autres pour effectuer une tâche spécifique et montre l'ordre chronologique des messages échangés. Un diagramme de séquence peut être utilisé pour modéliser des processus métier ou des scénarios d'utilisation du système, il m'aide à comprendre les différentes étapes d'un processus et les interactions entre les objets impliqués.

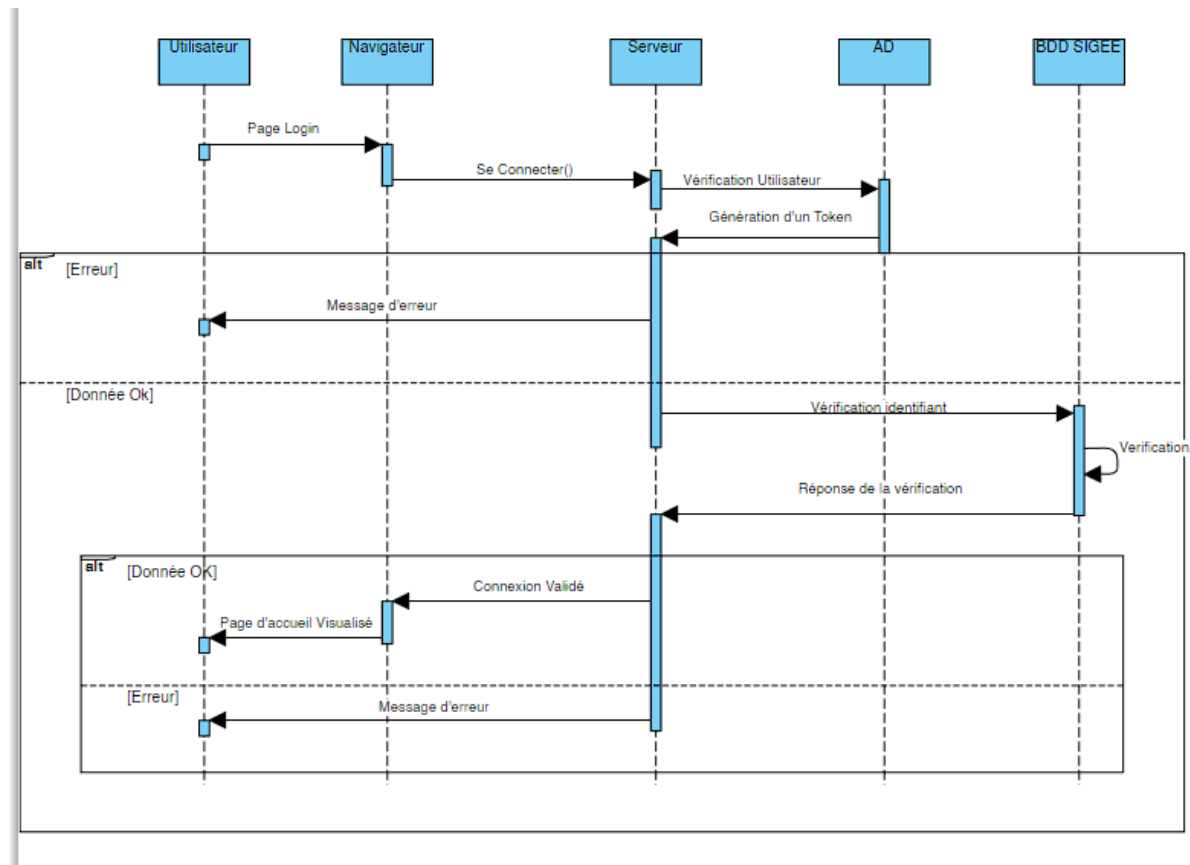


Fig. 6 Diagramme de séquence correspondant à la connexion d'un utilisateur

Lorsque l'utilisateur souhaite accéder à l'application, il doit d'abord se connecter en saisissant ses identifiants sur la page de login. Les données saisies par l'utilisateur sont envoyées par le navigateur au serveur. Le serveur doit alors vérifier les informations de l'utilisateur en les comparant avec les données stockées dans Active Directory. Si l'utilisateur est authentifié avec succès, un token est généré par Active Directory et envoyé au serveur pour validation.

Si l'utilisateur n'existe pas dans Active Directory, un message d'erreur est envoyé au serveur qui à son tour envoie un message d'erreur à l'utilisateur. Si l'utilisateur est authentifié avec succès, le serveur envoie l'identifiant de l'utilisateur à la base de données SIGEE pour vérifier s'il possède un compte. Si l'utilisateur possède un compte dans la base de données SIGEE, la connexion sera validée et l'utilisateur sera redirigé vers la page d'accueil de l'application.

L'accès à d'autres fonctionnalités de l'application dépendra des droits de l'utilisateur. Si l'utilisateur n'a pas de compte dans la base de données SIGEE, il sera redirigé vers la page de login avec un message d'erreur approprié.

En résumé, le processus d'authentification dans l'application implique plusieurs étapes de vérification des informations de l'utilisateur, depuis la saisie de ses identifiants jusqu'à la validation de son compte dans la base de données SIGEE. Les erreurs éventuelles sont gérées à chaque étape et l'utilisateur est informé en cas de problème.

- Diagramme d'activité

Un diagramme d'activité est un diagramme de comportement UML qui permet de représenter le déclenchement d'événements en fonction des états du système et de modéliser un comportement parallèle.

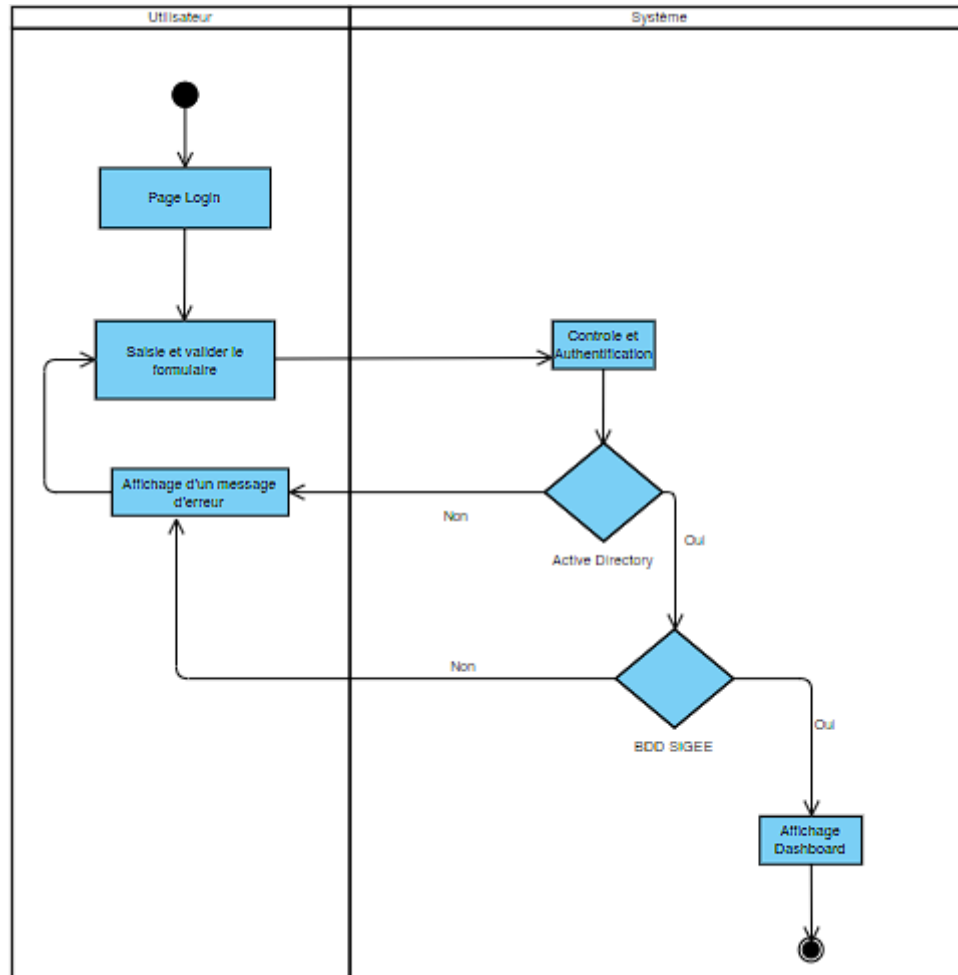


Fig. 7 Diagramme d'activité pour le cas d'utilisation « s'authentifier »

Le diagramme d'activité d'authentification illustre les processus internes du système, lorsqu'un utilisateur tente de se connecter à l'application SIGEE. Après avoir rempli le formulaire de connexion sur la page login avec son identifiant et son mot de passe, le système vérifie la validité de ces informations.

Cette vérification est effectuée en deux étapes : d'abord, le système se connecte à Active Directory pour voir si l'utilisateur possède un compte valide, puis il interroge la base de données de SIGEE pour confirmer l'existence de l'utilisateur et valider ses droits d'accès.

Si les informations d'authentification sont correctes, le système redirige l'utilisateur vers la page d'accueil. Dans le cas contraire, un message d'erreur s'affiche pour l'informer que l'identification a échoué. Le diagramme d'activité d'authentification permet donc de visualiser les étapes clés du processus d'authentification du système SIGEE.

- Maquettage

La réalisation de maquettes en amont du développement m'a permis de mieux visualiser la page d'engin que je devais réaliser. Celles-ci ont été créées avec le logiciel WireframePro et Figma.

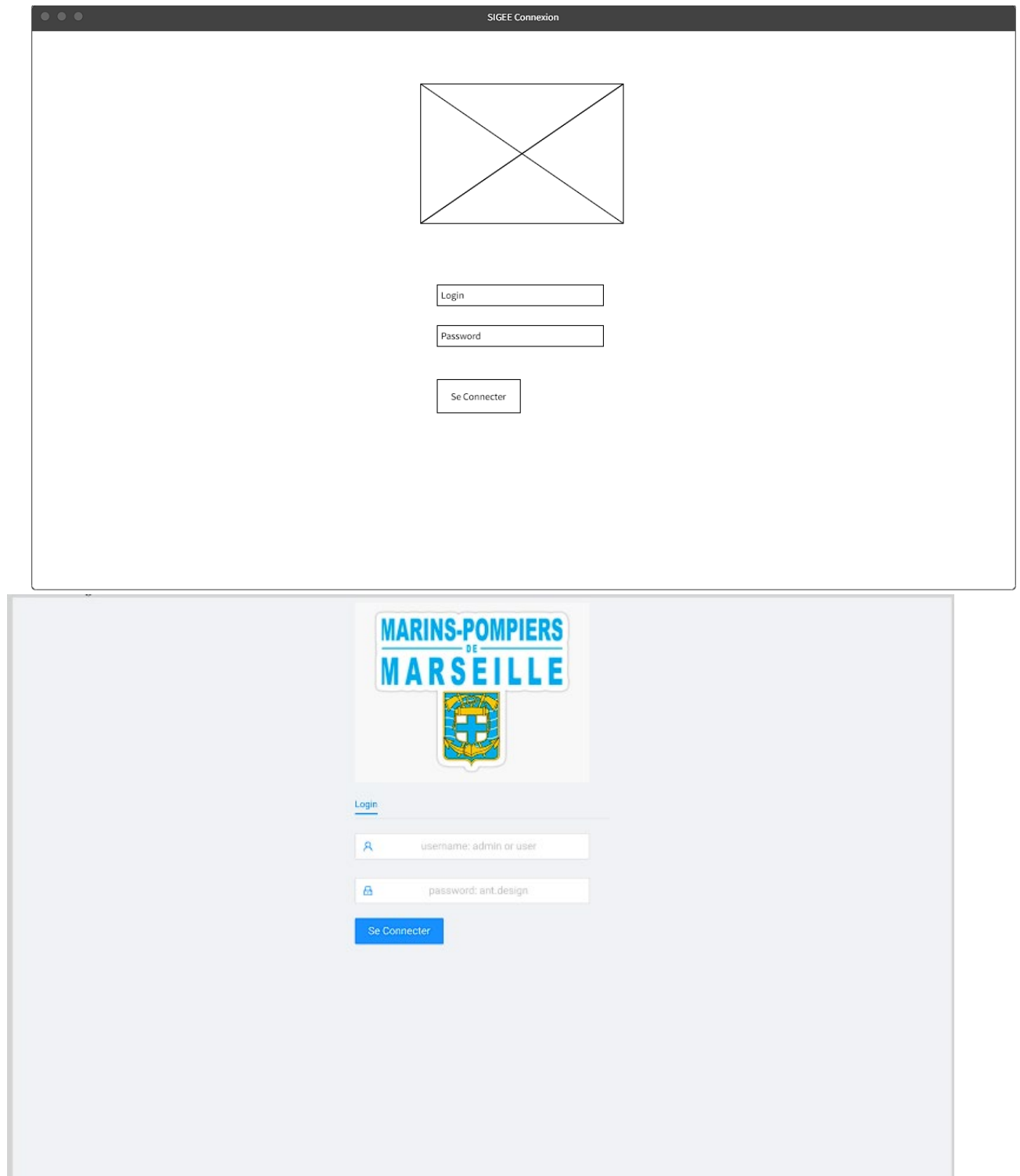


Fig. 8-9 Page de Connexion, pour qu'un utilisateur puisse accéder à l'application, il faut qu'il possède un compte Active Directory.

La figure 8 représente la maquette de la page de connexion pour le format ordinateur, tandis que la figure 9 représente l'aspect visuel de cette maquette.

Fig. 10 Visuel Maquette Modal de création Engin

La figure 10 représente la modal de création d'engin dont seuls les utilisateurs qui possèdent les droits ADMIN et COTEC peut y accéder. Lors d'un ajout d'un engin, celui-ci possède 7 étapes. Ci-dessus, je montre la 1^{ère} étape, il contient le type d'engin, les dates de la première mise en circulation, entre en service et la fin de validité Admin, puis ensuite la durée de vie théorique de l'engin, son immatriculation et sa carrosserie.

- BDD

La base de données SIGEE était déjà opérationnelle, mais il n'y avait pas de documentation associée à la conception de la base de données. J'ai donc utilisé SQLDeveloper et SQLDataModeler pour créer différentes conceptions de la base de données, en passant par le MLD (Modèle Logique de Données) ou MRD (Modèle Relationnel de Données), puis le MPD (Modèle Physique de Données), suivi du MCD (Modèle Conceptuel de Données), du diagramme de classe et enfin du dictionnaire de données. Cette démarche m'a permis de documenter la conception de la base de données existante et de mieux comprendre son fonctionnement.

Engin					TYPE_ENGIN				
Field name	Data type	Field length	Constraint	Description	Field name	Data type	Field length	Constraint	Description
ID_ENGIN	Number	10	Primary Key	Id Engin auto generated	IDTYPE_ENGIN	Number	10	Primary Key	IDType_engin auto generated
TYPE_DEPLACEMENT	Number		Not Null		ID_TYPE_GENRE	Number		Not Null	
IMMATRICULATION	Varchar2	10	Unique	Char	ID_TYPE_GAMME	Number		Not Null	
CODE_TAG	Number	5	Not Null		ID_ENGIN	Number		Not Null	
CODE_VEHICULE	Number	5	Not Null		MASSE_ENGIN	Varchar2	20	Not Null	Byte
D_ENTREE_SERVICE	Date		Not Null		ANNEE	Varchar2	20	Not Null	Byte
D_SORTIE_SERVICE	Date		Not Null		NUMERO_ENGIN	Varchar2	20	Not Null	Byte
ID_APELLATION	Number	10	Not Null		ID_TYPE_SERV	Number		Not Null	
TEMOINAUTEUR	Varchar2	50	Not Null						
TEMOINDATE	Date		Not Null						
LIMITE_NAVIGATION_ETAT_MER	Number		Not Null						
CERTIF_IMMAT	Blob		Not Null						
NO_MINE	Varchar2	20	Not Null	Char					
TYPE_MINE	Varchar2	20	Not Null	Char					
NO_DOSSIER	Varchar2	10	Not Null	Char					
NO_FOLIO	Varchar2	20	Not Null	Char					
NO_INVENTAIRE_VILLE	Varchar2	10	Not Null	Char					
D_SORTIE_INVENTAIRE_VILLE	Date		Not Null						
DUREE_VIE_THEORIQUE	Number		Not Null						
P_ACHAT_CHASSIS	Float		Not Null						
P_ACHAT_EQUIPEMENT	Float		Not Null						
CARROSSERIE	Varchar2	15	Not Null	Char					
P_FISCALE	Number	5	Not Null						
NB_PLACE_ASSURANCE	Number		Not Null						
COMPTEUR1	Number		Not Null						
NO_ORDRE_TYPE	Number		Not Null						
OBSERVATION_RADIO	Varchar2	100	Not Null	Byte					
DEPLACEMENT	Float		Not Null						
VITESSE_MAX	Number	5	Not Null						
ARMEMENT_POMPIER	Number		Not Null						
ARMEMENT_FLOTTE	Number		Not Null						
LIMITE_NAVIGATION_DIST	Float		Not Null						
NO_DEPART	Varchar2	3	Not Null	Byte					
ID_STATUT_TECH	Number	10	Not Null						
COMPTEUR2	Number		Not Null						
NO_BCM	Varchar2	50	Not Null	Char					
D_FVASSURANCE	Date		Not Null						
D_FVADMINISTRATIVE	Date		Not Null						
OBSERVATIONS	varchar2	1000	Not Null	Char					
TRACTEUR	Number	1	Not Null						
LONGUEUR	Number	5	Not Null						
LARGEUR	Number	5	Not Null						
HAUTEUR	Number	5	Not Null						
PTAC	Number		Not Null						
DOUBLE_CLEF	Number	1	Not Null						
IDFAMILLE	Number		Not Null						
IDPROPRIETAIRE	Number		Not Null						
NO_INVENTAIRE_VILLE2	varchar2	50	Not Null	Byte					
IDETAT_ENGIN	Number		Not Null						
D_MISE_CIRCULATION	Date		Not Null						
NO_SERIE	varchar2	20	Not Null	Byte					

AFFECTATION				
Field name	Data type	Field length	Constraint	Description
ID_AFFECTATION	Number	10	Primary Key	ID_Affectation auto generated
LIBELLE	Varchar2	20	Unique	Char
LIBELLE_LONG	Varchar2	100	Not Null	Char
TEMOINAUTEUR	Varchar2	50	Not Null	Char
TEMOINDATE	Date		Not Null	
LIBELLE_SIGALE	Varchar2	20	Not Null	Char
ALERTE_CTX	Number	3	Not Null	
CODE_CENTRE	Varchar2	10	Not Null	Char
ACTIVATION	Number	1,2	Not Null	

AFEC_HISTO_PHY				
Field name	Data type	Field length	Constraint	Description
ID_HISTO_PHY	Number	10	Primary Key	ID_Histo_phy auto generated
D_AFFEC	Date			
TEMOINAUTEUR	Varchar2	50		Char
TEMOINDATE	Date			

AFEC_HISTO_ADM				
Field name	Data type	Field length	Constraint	Description
ID_HISTO_ADM	Number	10	Primary Key	ID_Histo_adm auto generated
D_AFFEC	Date			
TEMOINAUTEUR	Varchar2	50		Char
TEMOINDATE	Date			
REFERENCE	Varchar2	50		Char
MASQUER_ALERTE	Number	1,1		

RADIO				
Field name	Data type	Field length	Constraint	Description
ID_RADIO	Number	10	Primary Key	ID_radio auto generated
RFGI	Varchar2	12		char
DATE_MAINTENANCE	Date			
OBSERVATIONS	Varchar2	100		Char
TEMOINAUTEUR	Varchar2	50		Char
TEMOINDATE	Date			
SERIALNUM	Varchar2	20		Char
ID_MAGALI	Number	10		

Fig. 11 Extrait Dictionnaire de donnée

Ce dictionnaire de données est une liste de références qui sont nécessaires pour concevoir une base de données relationnelle. Il clarifie également les termes qui seront utilisés dans la base de données, ainsi que la nature de chaque élément et sa longueur. L'exemple fournit un aperçu rapide de la façon dont les données peuvent être présentées ou traitées dans la base de données.

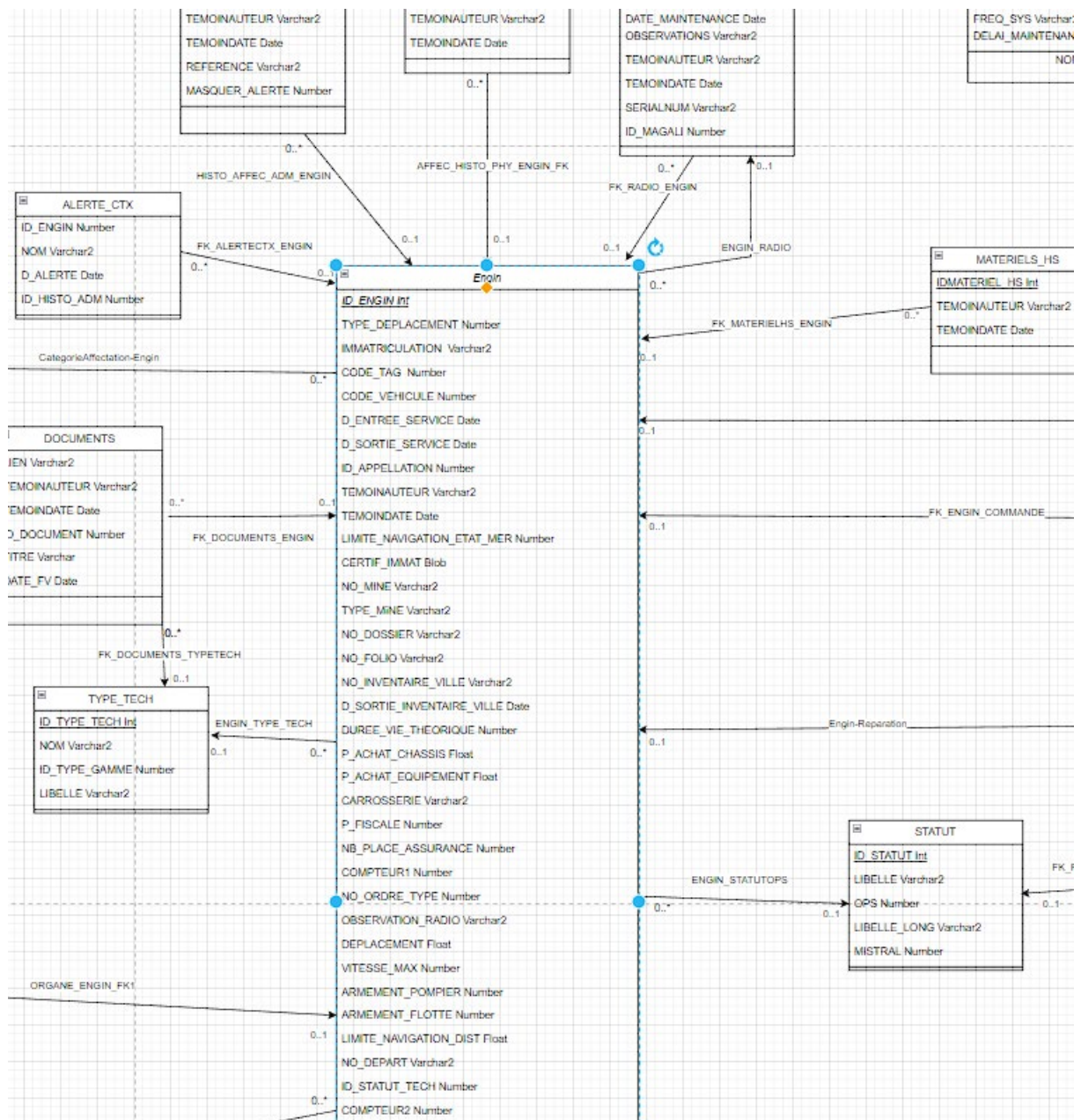


Fig. 12 Extrait diagramme de classe

La représentation visuelle présentée est un élément central de la conception, décrivant de manière statique le système à développer. Chaque classe est responsable des données et des traitements liés à elle-même et aux autres classes. Cette vue permet au client de visualiser facilement l'ensemble des composants d'une classe.

- Modèle Conceptuel de données, Modèle Physique de donnée et le Modèle Logique de Données

Grâce à tous ces éléments, nous avons pu réaliser la conception finale de l'application en créant un MCD sur Looping. Ce processus est crucial et nécessite beaucoup de temps pour éviter toute erreur lors de la création de la base de données.

Le MCD contient différents types d'éléments, tels que les entités, qui regroupent des objets de même nature et doivent être identifiées par leur identifiant.

Les relations, qui sont des associations entre deux ou plusieurs occurrences d'entités, peuvent également avoir des propriétés (ou attributs).

Les cardinalités, qui traduisent la participation des occurrences d'une entité aux occurrences d'une relation, sont représentées par les termes "0, 1 et n" et sont composées de deux termes pour chaque entité de la relation. Les possibilités de cardinalités sont limitées à (0, 1), (0, n), (1, 1) et (1, n).

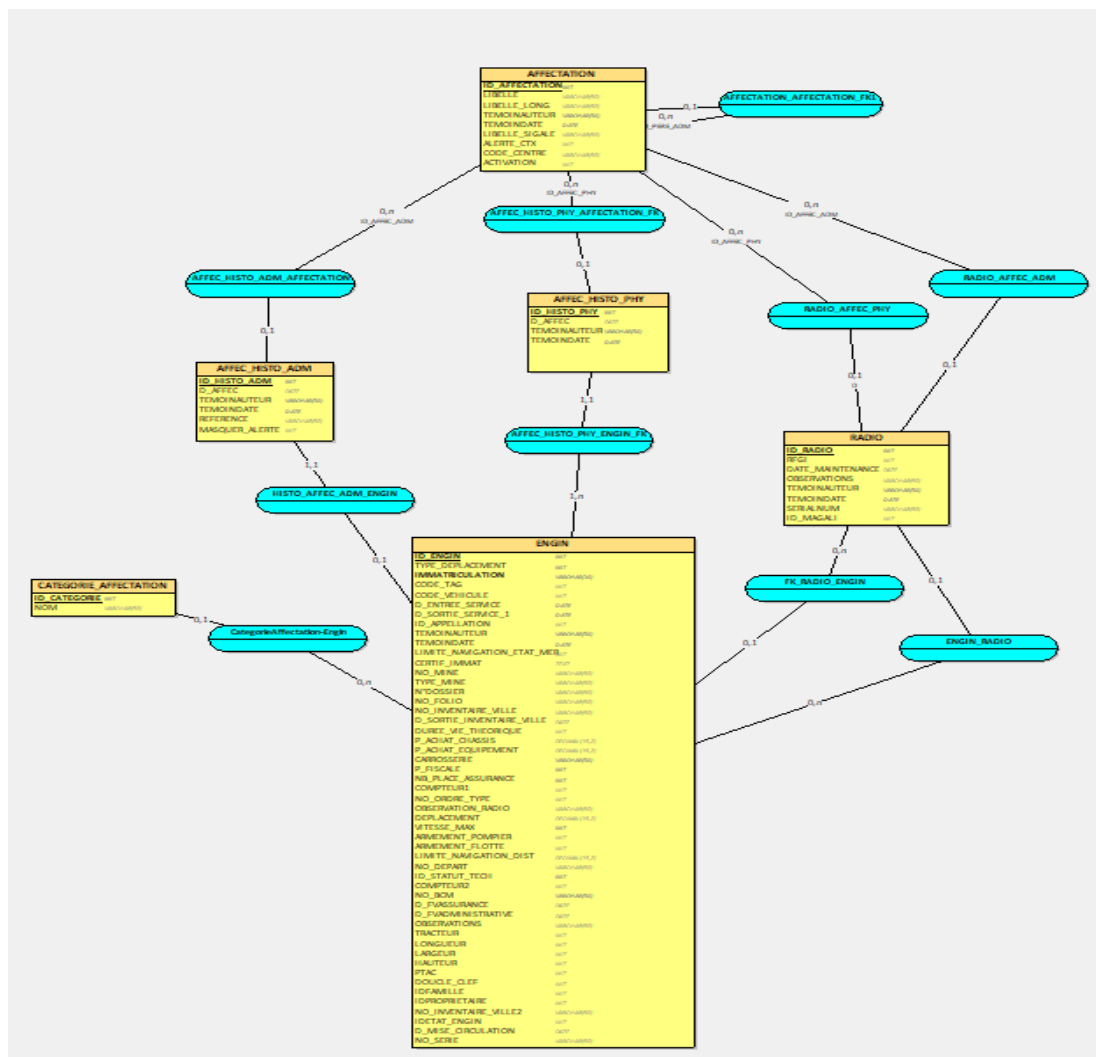


Fig. 13 Extrait Modèle Conceptuel de donnée

ENGIN = (ID_ENGIN INT, TYPE_DEPLACEMENT INT, IMMATRICULATION VARCHAR(10), CODE_TAG INT, CODE_VEHICULE INT, D_ENTREE_SERVICE DATE, D_SORTIE_SERVICE_1 DATE, ID_APPELLATION INT, TEMOINAUTEUR VARCHAR(50), TEMOINDATE DATE, LIMITE_NAVIGATION_ETAT_MER INT, CERTIF_IMMAT TEXT, NO_MINE VARCHAR(50), TYPE_MINE VARCHAR(50), N_DOSSIER VARCHAR(50), NO_FOLIO VARCHAR(50), NO_INVENTAIRE_VILLE VARCHAR(50), D_SORTIE_INVENTAIRE_VILLE DATE, DUREE_VIE_THEORIQUE INT, P_ACHAT_CHASSIS DECIMAL(15,2), P_ACHAT_EQUIPEMENT DECIMAL(15,2), CARROSSERIE VARCHAR(50), P_FISCALE INT, NB_PLACE_ASSURANCE INT, COMPTEUR1 INT, NO_ORDRE_TYPE INT, OBSERVATION_RADIO VARCHAR(50), DEPLACEMENT DECIMAL(15,2), VITESSE_MAX INT, ARMEMENT_POMPIER INT, ARMEMENT_FLOTTE INT, LIMITE_NAVIGATION_DIST DECIMAL(15,2), NO_DEPART VARCHAR(50), ID_STATUT_TECH INT, COMPTEUR2 INT, NO_BCM VARCHAR(50), D_FVASSURANCE DATE, D_FVADMINISTRATIVE DATE, OBSERVATIONS VARCHAR(50), TRACTEUR INT, LONGUEUR INT, LARGEUR INT, HAUTEUR INT, PTAC INT, DOUCLE_CLEF INT, IDFAMILLE INT, IDPROPRIETAIRE INT, NO_INVENTAIRE_VILLE2 VARCHAR(50), IDETAT_ENGIN INT, D_MISE_CIRCULATION DATE, NO_SERIE VARCHAR(50), #ID_RADIO_*);

AFFEC_HISTO_ADM = (ID_HISTO_ADM INT, D_AFFEC DATE, TEMOINAUTEUR VARCHAR(50), TEMOINDATE DATE, REFERENCE VARCHAR(50), MASQUER_ALERTE INT, #ID_ENGIN, #ID_AFFEC_ADM*);

AFFEC_HISTO_PHY = (ID_HISTO_PHY INT, D_AFFEC DATE, TEMOINAUTEUR VARCHAR(50), TEMOINDATE DATE, #ID_ENGIN, #ID_AFFEC_PHY*);

RADIO = (ID_RADIO INT, RFGI INT, DATE_MAINTENANCE DATE, OBSERVATIONS VARCHAR(50), TEMOINAUTEUR VARCHAR(50), TEMOINDATE DATE, SERIALNUM VARCHAR(50), ID_MAGALI INT, #ID_ENGIN*, #ID_AFFEC_PHY*, #ID_AFFEC_ADM*);

CATEGORIE_AFFECTATION = (ID_CATEGORIE INT, NOM VARCHAR(50), #ID_ENGIN*);

- Présentation de l'architecture technique proposée

J'ai proposé une architecture technique qui utilise ReactJS et Django REST Framework pour améliorer l'application existante en PHP natif. Cette approche permettrait d'obtenir une meilleure performance, une plus grande flexibilité et une maintenabilité accrue. ReactJS serait utilisé pour la partie front-end de l'application, offrant une expérience utilisateur plus interactive et conviviale. Django REST Framework serait utilisé pour la partie back-end, permettant une API RESTful plus rapide, plus sûre et plus évolutive. En utilisant cette architecture, je pourrai également intégrer facilement d'autres fonctionnalités, tels que des outils de surveillance, des tests automatisés et une mise en cache plus efficace pour améliorer encore la performance de l'application.

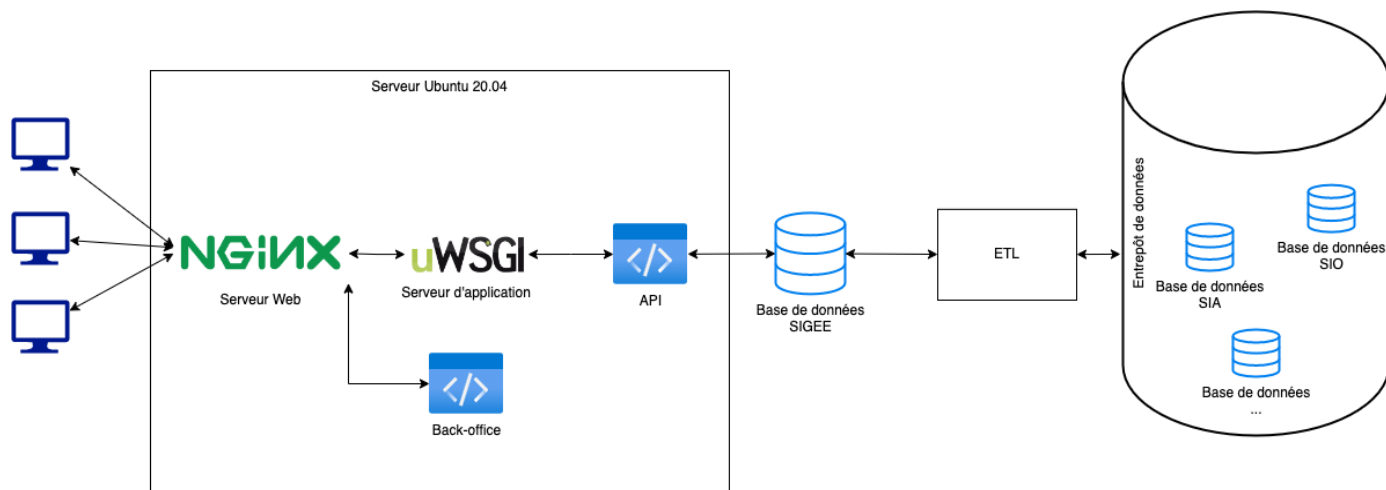


Fig. 16

Pour mettre en place la solution, j'ai opté pour l'utilisation d'un serveur Ubuntu 20.04, qui sera hébergé sur le réseau. Afin d'exposer l'application sur le web, j'ai choisi d'utiliser le serveur web NGINX. Le serveur d'application uWSGI a été choisi pour assurer la mise à disposition de l'API.

Afin de tester les API, j'ai opté pour l'utilisation de l'application Postman, qui facilite la réalisation des requêtes HTTP via une interface graphique utilisateur.

Le développement de la solution s'est appuyé sur le langage de programmation Python et le Framework Django Rest pour la mise en place de l'API. Quant au back-office (la partie visuelle de l'application), il a été développé en JavaScript avec l'utilisation du Framework ReactJS.

La décision a été prise de conserver la base de données Oracle existante afin d'éviter toute perturbation du fonctionnement normal de l'ETL en place, en utilisant le reverse engineering de python pour créer les modèles. En utilisant cette architecture, j'ai réussi à implémenter les fonctions identifiées avec efficacité et sécurité, tout en assurant des performances optimales de l'application.

L'utilisation de la plateforme GitLab a été privilégiée pour contribuer à plusieurs projets en cours, permettant ainsi de sauvegarder, partager et suivre les modifications apportées. En résumé, la mise en place de cette solution a été soigneusement planifiée pour assurer une exécution fluide et performante de l'application, tout en offrant une expérience utilisateur optimale.

- Sécurité

Je vais vous parler de la sécurité de connexion des utilisateurs à l'aide de ReactJS, Django Rest Framework et Active Directory en fonction des droits des utilisateurs. Lorsque les utilisateurs essaient de se connecter à l'application, des mécanismes d'authentification et d'autorisation solides doivent être mise en place pour garantir une sécurité maximale.

Pour assurer la sécurité des utilisateurs, les données des utilisateurs sont transmises au Django Rest Framework, puis à Active Directory pour authentifier et valider les droits d'accès aux applications. Cela garantit que les utilisateurs dans le cadre de leurs droits, on accès aux ressources demander.

Je suis conscient qu'il existe des risques potentiels tel que les attaques par force brute, injection SQL, les attaques de session et les attaques de pishing.

Pour réduire ces risques, j'applique des pratiques de développement sécurisées, telles qu'une gestion appropriée des informations d'identification, le chiffrement des données sensibles, l'utilisation de sessions sécurisées et le contrôle d'accès des utilisateurs.

Ensuite, je veille régulièrement sur les mises à niveau des codes utilisé lors du développement pour une meilleur sécurité, telle qu'une gestion des informations d'identification approprié, le chiffrement des données, le contrôle d'accès des utilisateurs et de l'utilisation des sessions sécurisé.

Je suis très vigilant sur la sécurité des connexions utilisateurs utilisées par ReactJS, Django Rest Framework et Active Directory. Je m'assure de mettre en œuvre des mesures de sécurité solides pour protéger les informations sensibles et les informations des utilisateurs.

Je suis également très vigilant, sur la sécurité des connexions des différents utilisateurs utilisé via ReactJS, Django Rest Framework qui est lié à la base de données Oracle et a Active Directory, donc pour cela je mets en place des mesures de sécurité solides pour protéger les informations sensibles et des utilisateurs.

- Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité

- 1- Création des parties Back End et Front End du projet

Comme mentionné précédemment, le back-end du projet a été réalisé avec Django Rest Framework et le front-end avec ReactJS. J'ai donc d'abord créé les projets correspondants.

- Création une API RestFul avec le Framework Django Rest Framework (DRF)

L'API a été créée à l'aide de python, une interface de ligne de commande qui permet d'initialiser, de développer et de gérer les applications python. Une fois npm et python installé, j'ai pu créer un projet python en exécutant les commandes suivantes dans mon terminal :

pip install djangorestframework

pipenv install django-cors-headers

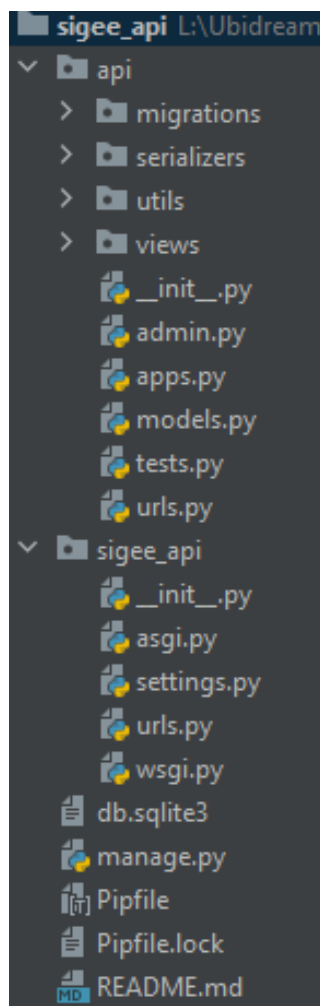
Pipenv est un outil polyvalent de gestion de virtualenv pour Python. Il offre une compatibilité étendue avec différents systèmes et facilite l'intégration entre pip, pyenv et virtualenv.

Puis ajouter 'rest_framework' et 'corsheaders' dans le fichier settings.py a la liste INSTALLED_APPS.

La première commande permet d'installer la bibliothèque Python « Django Rest Framework », cela permet d'utiliser le développement API RestFul en python.

La deuxième commande permet d'installer la bibliothèque django-cors-headers, qui est un environnement virtuel. Cela permet de gérer les requêtes Cross-Origin Resource Sharing (CORS) pour l'application SIGEE.

Une fois que le projet créer, ce répertoire contient un fichier à la racine manage.py et ainsi qu'un répertoire le nom du projet donc SIGEE_API qui contient des fichiers de base de django.



Dans le package API, il contient :

Le fichier apps.py est un fichier de configuration qui permet de définir les métadonnées de l'application, telles que le nom, la description et la configuration de l'application.

Le fichier test.py est un fichier qui contient les tests unitaires pour l'application.

Le fichier urls.py est un fichier de configuration qui permet de définir les URL (Uniform Resource Locator) de l'application.

Dans le package SIGEE_API, il contient :

Le fichier asgi.py est un fichier de configuration qui permet de spécifier le serveur ASGI (Asynchronous Server Gateway Interface) utilisé par l'application

Le fichier setting.py est un fichier de configuration qui contient les paramètres de configuration de l'application, tels que les informations de base de données, les clés secrètes, les fichiers statiques, les fichiers média, les middlewares, les applications installées, etc.

Le fichier wsgi.py est un fichier de configuration qui permet de spécifier le serveur WSGI (Web Server Gateway Interface) utilisé par l'application.

Fig. 17 Répertoire projet API

Et à la racine des packages, il contient

Le fichier `manage.py` est un fichier exécutable qui permet de gérer l'application à partir de la ligne de commande.

Certains fichiers comme "`package.json`" (pour gérer mes dépendances), "`README.md`" (pour ajouter une page d'explication) ou encore «. `gitignore`" (pour préciser ce qui ne peut pas être GitHub) sont automatiquement inclus lorsque le fichier est créé également du dossier.

Dans les parties suivantes, j'expliquerai en détails les packages Serializers et View de l'application `Sigee_API`. Les Serializers sont des composant DRF qui permet de transformer les objets Python en JSON tandis que les View sont utilisé pour les traitements de requêtes http pour API DRF. Les 2 packages sont essentielles pour le développement de l'API RESTful de Django et sont étroitement liés.

○ Création du projet en ReactJS

Pour initialiser le projet REACTJS, j'ai navigué jusqu'au répertoire sélectionné, puis j'ai exécuté la commande suivante :

`npx create-react-app mon-app`

Cette commande crée trois dossiers (Figure 18) :

- Dossier `Node_module` : C'est là que toutes les bibliothèques externes pour mon code seront installées.
- Le dossier `public` qui contient le fichier `index.html`.
- Dossier `Src` : la plupart des fichiers que je crée et modifie se trouvent dans ce dossier

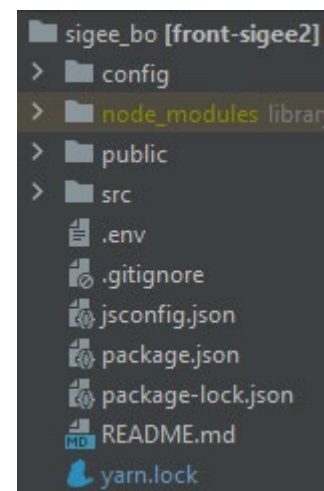


Fig. 18 Projet ReactJS SIGEE

Comme lors de la création d'un projet avec `SIGEE_API`, les fichiers "`package.json`", "`README.md`" et "`.gitignore`" sont automatiquement inclus.

2- Connexion à la base de données

Afin de me connecter l'API Restful a la base de données SIGEE existante, j'ai utilisé oracle server pour qu'il soit compatible avec celui-ci.

Pour que cela fonctionne avec DRF, j'ai commencer par installer les dependnaces requise a l'aide de la commande :

`pip install cx_Oracle --upgrade`

Cette commande permet d'installer ou de mettre a jour le pilote cx_Oracle qui permet de faire la connexion a la base de données et d'exécuter les requetes.

Dans le fichier settings.py, il faut importer `oracledb.version = "8.3.0"`

`sys.modules["cx_Oracle"] = oracledb`, cela permet d'établir une connexion avec la BDD Oracle. Et ajouter dans la Database :

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.oracle',
        'NAME': '(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521)))'
        '(CONNECT_DATA=(SERVICE_NAME=orcl)))',
        'USER': 'sigee',
        'PASSWORD': 'xxxx',
    }
}
```

Fig.19 Connexion a la BDD d'oracle

La clé `ENGINE` spécifie le type de base de données que j'utilise, dans ce cas `django.db.backends.oracle` pour Oracle. La clé `NAME` spécifie l'adresse de la base de données Oracle à laquelle je me connecte, en utilisant une chaîne de connexion Oracle.

3- Générer le modèle de données

```

class Engine(models.Model):
    id_engine = models.BigIntegerField(primary_key=True)
    type_deplacement = models.BigIntegerField(blank=True, null=True)
    immatriculation = models.CharField(unique=True, max_length=10, blank=True, null=True)
    code_tag = models.IntegerField()
    code_vehicule = models.IntegerField(blank=True, null=True)
    d_entree_service = models.DateField()
    d_sortie_service = models.DateField(blank=True, null=True)
    id_appellation = models.IntegerField(blank=True, null=True)
    id_categorie = models.ForeignKey(CategorieAffectation, models.DO_NOTHING, db_column='id_categorie', blank=True, null=True)
    id_type_ops = models.ForeignKey('TypeOps', models.DO_NOTHING, db_column='id_type_ops', blank=True, null=True)
    id_marque = models.ForeignKey('Marque', models.DO_NOTHING, db_column='id_marque', blank=True, null=True)
    temoinaire = models.CharField(max_length=50, blank=True, null=True)
    limite_date = models.DateField(blank=True, null=True)
    limite_navigation_etat_mer = models.BigIntegerField(blank=True, null=True)
    certif_immat = models.BinaryField(blank=True, null=True)
    no_mine = models.CharField(max_length=20, blank=True, null=True)
    type_mine = models.CharField(max_length=20, blank=True, null=True)
    no_dossier = models.CharField(max_length=10, blank=True, null=True)
    no_folio = models.CharField(max_length=20, blank=True, null=True)
    no_inventaire_ville = models.CharField(max_length=10, blank=True, null=True)
    d_sortie_inventaire_ville = models.DateField(blank=True, null=True)
    duree_vie_theorique = models.BigIntegerField(blank=True, null=True)
    p_achat_chassis = models.FloatField(blank=True, null=True)
    p_achat_equipement = models.FloatField(blank=True, null=True)
    carrosserie = models.CharField(max_length=15, blank=True, null=True)
    p_fiscale = models.IntegerField(blank=True, null=True)
    nb_places_assurance = models.BigIntegerField(blank=True, null=True)
    compteur1 = models.BigIntegerField(blank=True, null=True)
    no_ordre_type = models.BigIntegerField(blank=True, null=True)
    observation_radio = models.CharField(max_length=100, blank=True, null=True)
    deplacement = models.FloatField(blank=True, null=True)
    vitesse_max = models.IntegerField(blank=True, null=True)

```

Fig.20 Extrait d'un modele de donnée

En exploitant la base de données, j'ai employé une méthode de reverse engineering qui implique la création d'un modèle de données. À cet effet, j'ai utilisé la commande de django "**python manage.py inspectdb**" qui permet d'examiner la structure de la base de données et générer automatiquement le modèle de données correspondant. En évitant la conception manuelle de modèles de données, j'ai pu simplifier le processus grâce à l'utilisation de la technique de reverse engineering.

Cependant, même si cette méthode génère automatiquement le modèle de données, des incohérences peuvent exister dans la base de données, ainsi que des éléments manquants tels que des index ou des contraintes qui ne peuvent être générés automatiquement. J'ai donc dû effectuer des corrections manuelles pour résoudre ces problèmes et ajouter les éléments manquants nécessaires au modèle de données.

Le modèle de données est l'un des éléments clés de toute application qui doit stocker et manipuler des données.

Dans la figure 20, ce modèle de données qui représente une table dans la base de données "Engin". Cette table est considérée comme étant l'une des plus importantes du projet, car elle contient 61 colonnes et joue comme élément central.

Le modèle possède différentes colonnes qui comporte chacun un type de champs différents.

Par exemple, le type de champ "BigIntegerField" permet de stocker de très grands nombres comme les clés primaires, tandis que le type de champ "CharField" représente des chaînes de caractères de taille fixe ou variable.

Par exemple l'immatriculation est définie sur 10 caractères, ce qui signifie que seuls les immatriculations ayant exactement 10 caractères pourront être stockées.

Le type de champ "IntegerField" permet de stocker des nombres entiers tels que le code_vehicule, tandis que le type de champ "DateField" permet de stocker des dates.

La colonne "ForeignKey" représente une relation de clé étrangère entre deux modèles. Telle que la colonne "id_categorie" dans la classe "Engin" est une clé étrangère qui est liée à la classe "Categorie" par son champ "id_categorie" en tant que clé primaire.

4- Implémentation des composants d'accès aux données

L'utilisation d'un queryset permet de manipuler une collection d'objets provenant d'une base de données. Il offre la possibilité d'appliquer des filtres pour restreindre les résultats de la requête en fonction des paramètres fournis, ce qui est similaire à l'utilisation d'un SELECT avec des clauses WHERE ou LIMIT. Un queryset utilise le Manager associé à chaque modèle, qui est généralement le manager par défaut appelé "Objects".

Afin de manipuler les données, j'ai utilisé les serializers, cela permet de transformer des données complexes comme un ensemble de requêtes ou d'instance de modèle, pouvant être convertis en JSON, XML ou d'autres types de contenu. Les sérialiseurs ont également pour fonction d'effectuer la désérialisation des données. Après une première validation des données entrantes, ils permettent de convertir les données analysées en types complexes.

```
from rest_framework.serializers import ModelSerializer

from api.models import ProprietaireEngin

# Meysson17Alexis
class ProprietaireGetSerializer(ModelSerializer):
    # Meysson17Alexis
    class Meta:
        model = ProprietaireEngin
        fields = '__all__'

# Meysson17Alexis
class ProprietaireSerializer(ModelSerializer):
    # Meysson17Alexis
    class Meta:
        model = ProprietaireEngin
        fields = ('code_proprietaire', 'libelle')
```

Fig.21 Sérializers Proprietaire (proprietaire.py)

La première ligne de code permet d'importer la classe `ModelSerializer` qui est une bibliothèque de Django REST Framework, qui offre plusieurs fonctionnalités pratiques pour la création d'API REST. Cette bibliothèque permet notamment la conversion automatique des modèles en JSON et la validation des données entrantes. La classe `ModelSerializer`, quant à elle, facilite la création de sérialiseurs pour les modèles Django. Elle permet de créer des sérialiseurs avec des champs correspondant aux champs du modèle, tels que le modèle `ProprietaireEngin`. La classe `ModelSerializer` fournit également des implémentations par défaut pour les méthodes `.create()` et `.update()`, ce qui simplifie considérablement la création d'API REST pour les modèles Django.

Pour générer la vue et voir le bon fonctionnement du Serializers, j'ai créé un fichier `view.py`, par exemple `ProprietaireView.py` dans le dossier `views`.

```

from api.serializers.proprietaire import ProprietaireSerializer, ProprietaireGetSerializer
from api.models import ProprietaireEngin
from api.utils.pagination import PaginationViewSet

# Views proprietaire
# Meysson17Alexis
class ProprietaireViewSet(PaginationViewSet):
    """
    A simple ViewSet for viewing and editing accounts.
    """
    queryset = ProprietaireEngin.objects.all()
    serializer_class = ProprietaireGetSerializer

    action_serializers = {
        'create': ProprietaireSerializer,
        'update': ProprietaireSerializer,
        'partial_update': ProprietaireSerializer
    }

```

Fig.22 Views Proprietaire

Ce code permet de définir une classe ProprietaireViewSet qui hérite de PaginationViewSet permet d'affiche et/ou de modifie de l'instance ProprietaireEngin.

La variable queryset spécifie l'ensemble d'instance de modèle utilisée dans la vue. Dans ce cas, elle contient toutes les instances de ProprietaireEngin.

La variable serializer_class définit le sérialiseur à utiliser pour la réponse de la vue. Dans ce cas, il s'agit de ProprietaireGetSerializer, qui est utilisé pour la méthode HTTP GET.

La variable action_serializers définit les sérialiseurs utiliser pour les autres méthodes HTTP telles que POST, PUT et PATCH. Dans ce cas, ProprietaireSerializer est utilisé pour les méthodes de création, de mise à jour complète et de mise à jour partielle des instances ProprietaireEngin.

Et enfin, définir la route utilisée par lequel l'API pourra y accéder.

```

router.register(r'categories', CategorieViewSet, basename='public_data')
router.register(r'proprietaires', ProprietaireViewSet, basename='public_data')
router.register(r'equipements_engin', EquipementEViewSet, basename='public_data')
router.register(r'equipements', EquipementViewSet, basename='public_data')
router.register(r'gammes', GammeViewSet, basename='public_data')
router.register(r'classifications', ClassificationViewSet, basename='public_data')
router.register(r'affectations', AffectationViewSet, basename='public_data')

```

Fig.23 routes API dans urls.py

Ce code, permet l'utilisation de la fonction register du routeur Django REST Framework pour accéder à la vue ProprietaireViewSet avec une URL spécifique.

La méthode register, permet l'enregistrement automatique des URL nécessaire pour chacune des méthodes utilisées. Par exemple, pour la méthode POST, l'URL affichera '/Proprietaire/create/'.

La méthode register possède 3 arguments :

- Le 1^{er} représente l'URL,
- Le 2nd, la vue utilisée,
- Le 3^{ème}, qui est optionnel, si elle est spécifique, il lui faudra le définir un nom de base pour générer la vue.

Si tous ces conditions sont respectées, alors en utilisant cette commande

python manage.py runserver 0.0.0.0:8000

On accède à cette page : <http://localhost:8000>

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "engins": "http://localhost:8000/affectationHistoGisement/",
  "users": "http://localhost:8000/affectationHistoGisement/",
  "categories": "http://localhost:8000/affectationHistoGisement/",
  "proprietaires": "http://localhost:8000/affectationHistoGisement/",
  "equipements_engin": "http://localhost:8000/affectationHistoGisement/",
  "equipements": "http://localhost:8000/affectationHistoGisement/",
  "gammes": "http://localhost:8000/affectationHistoGisement/",
  "classifications": "http://localhost:8000/affectationHistoGisement/",
  "affectations": "http://localhost:8000/affectationHistoGisement/",
  "equipeurs": "http://localhost:8000/affectationHistoGisement/",
  "marques": "http://localhost:8000/affectationHistoGisement/",
  "appellationsCommerciales": "http://localhost:8000/affectationHistoGisement/",
  "typeOPS": "http://localhost:8000/affectationHistoGisement/",
  "typeTech": "http://localhost:8000/affectationHistoGisement/",
  "typeServ": "http://localhost:8000/affectationHistoGisement/",
  "typeAttelage": "http://localhost:8000/affectationHistoGisement/",
  "documents": "http://localhost:8000/affectationHistoGisement/",
  "reparationEngin": "http://localhost:8000/affectationHistoGisement/",
  "interventionEngin": "http://localhost:8000/affectationHistoGisement/",
  "affectationHistoPhy": "http://localhost:8000/affectationHistoGisement/",
  "affectationHistoAdm": "http://localhost:8000/affectationHistoGisement/",
  "affectationHistoGisement": "http://localhost:8000/affectationHistoGisement/"
}
```

Fig.24 Liste Urls API Root de Django Rest Framework

Et en utilisant Swagger, il permet de voir en détails les méthodes et les paramètres utilisées dans les vues, j'expliquerai son utilisation lors des jeux d'essais.

Swagger permet une visualisation et une manipulation des données échangées entre le client et le serveur.

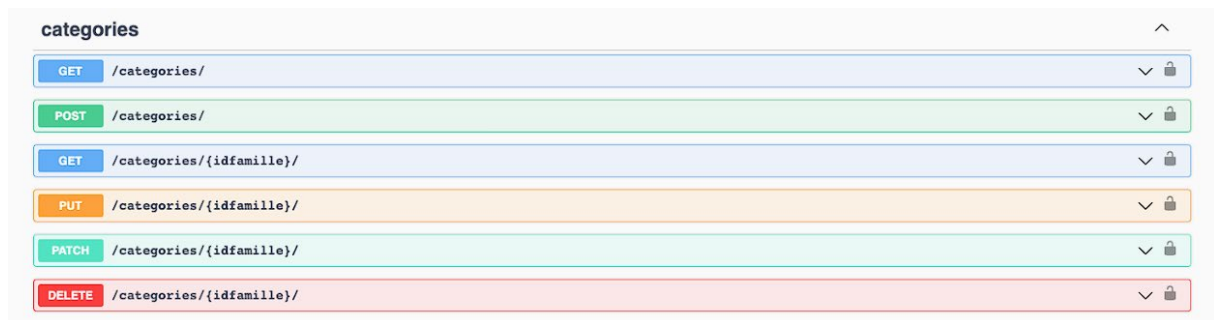


Fig.25 Swagger

5- Connexion du l'API Rest au Front avec ReactJS

Pour permettre la communication entre l'API et le projet Front avec ReactJS, j'ai créé un fichier .env dans le projet qui contient la configuration de l'environnement de l'application SIGEE. Cela m'a permis de configurer globalement l'application en définissant le nom de l'application (REACT_APP_WEBSITE_NAME=SIGEE BO) ainsi que sa couleur principale (REACT_APP_MAIN_COLOR=#093731). J'ai également configuré le port utilisé (PORT=3000) conformément aux spécifications de Create React App (CRA). En plus de cela, j'ai ajouté une configuration additionnelle qui permet de connecter l'application à l'API via l'URL du serveur (REACT_APP_SERVER_URL=http://localhost:8000). Cette configuration est adaptée à une utilisation locale et devra être mise à jour lors du déploiement de l'application sur un serveur distant.

6- Présentation d'une des fonctionnalité du front de l'application SIGEE

L'application SIGEE comprend un grand nombre de fonctionnalités que j'ai développées. Chaque fonctionnalité est essentielle pour la gestion des engins, car elle couvre l'intégralité du cycle de vie d'un engin, de sa création à sa modification, jusqu'à son archivage ou sa suppression s'il n'est plus disponible.

Par exemple pour connaître le propriétaire d'un engin, j'ai élaborer la fonctionnalité de recuperation des données, la creation et la modification d'un propriétaire.

```

name: 'proprietaires',
url: `${baseURL}/proprietaires`,
credentials: 'include',
actions: {
  get: {
    method: 'GET',
    gerundName: 'fetching',
    assignResponse: true,
    url: './',
  },
  create: {
    method: 'POST',
    gerundName: 'creating',
    url: './',
  },
}

```

Fig.26 Extrait ProprietairesRessource.js

Pour la récupération des données sur les propriétaires, j'utilise la méthode GET et avec une action asynchrone qui permet d'envoyer la requête au serveur. Elle est souvent utilisée pour mettre à jour l'interface utilisateur en affichant un indicateur de chargement. Le "name" est utilisé pour spécifier le nom de la ressource avec laquelle on souhaite interagir via les appels API. L'URL est l'adresse qui permet d'effectuer ces appels, et utilise une variable "baseURL" qui est définie avec l'URL de Django. La propriété "credentials" s'assure que l'utilisateur est bien authentifié et autorisé à accéder à l'API, en envoyant les informations d'authentification (comme les cookies ou les tokens) avec chaque requête. Enfin, les "actions" permettent d'utiliser les méthodes GET, POST, PATCH, PUT et DELETE, en utilisant une URL correspondante pour chaque action.

```

const ProprietaireTableData = {
  proprietaires: {
    pagination: {
      position: 'both',
      showSizeChanger: true,
    },
    rowKey: 'id',
    columns: {
      name: {
        title: I18n.t( key: 'Code Propriétaires'),
        key: 'code',
        dataIndex: 'code_proprietaire',
        sorter: (a, b) => {
          defaultTo(get(a, path: 'code'), defaultValue: '').localeCompare(
            defaultTo(get(b, path: 'code'), defaultValue: '')
          ),
          sortDirections: ['ascend', 'descend'],
        },
      },
      proprietaire: {
        title: I18n.t( key: 'Libelle Propriétaires'),
        key: 'proprietaire',
        dataIndex: 'libelle',
        sorter: (a, b) => {
          defaultTo(get(a, path: 'proprietaire'), defaultValue: '').localeCompare(
            defaultTo(get(b, path: 'proprietaire'), defaultValue: '')
          ),
        },
      },
    },
  },
}

```

Fig. 27 Extrait ProprietaireTableData.js

J'utilise une constante `ProprietaireTableData` qui permet de créer une table en imposant son contenu, elle contient les informations nécessaires pour permettre de définir sa structure.

L'objet `Proprietaire` renferme plusieurs informations, telles que la pagination, la clé primaire `"rowKey: 'id'"` qui s'applique à chaque ligne de la table, et les colonnes qui contiennent des noms spécifiques, tels que `"Code Propriétaire"` et `"Libellé Propriétaire"`. Les données de chaque colonne proviennent des `"dataIndex"` correspondants, qui correspondent aux colonnes de la base de données. En d'autres termes, cela équivaut à une requête SQL `"SELECT code_proprietaire, libelle FROM Proprietaire"`. Chaque colonne est également assortie d'une fonction de tri, grâce à la méthode `"sorter"`, qui permet de trier les données dans l'ordre croissant ou décroissant.

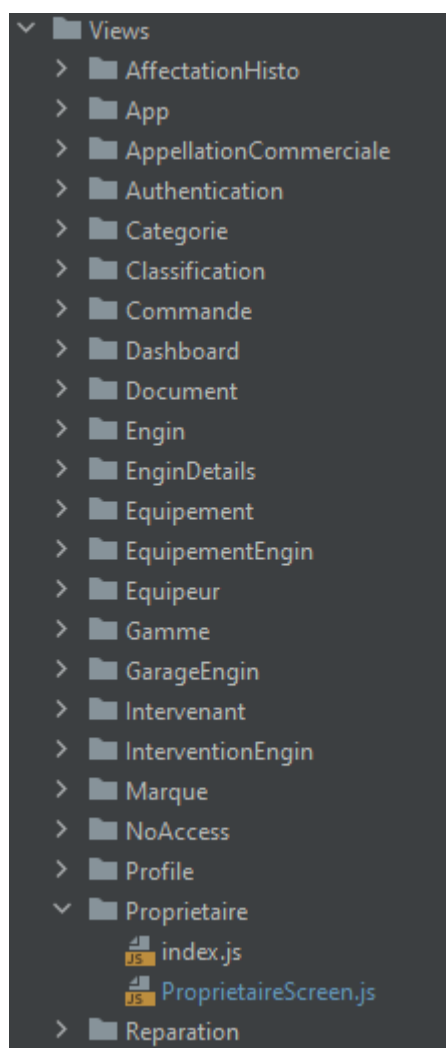


Fig. 28 Arborescence dossier views

Dans le dossier "view", il y a un dossier pour chaque fonctionnalité de l'application. Par exemple, le dossier "Proprietaire" contient deux fichiers : `"index.js"` et `"ProprietaireScreen.js"`.

`Index.js` c'est un point d'entrée, il permet d'exporter les fonctions qui sont utilisées pour la gestion des propriétaires.

```

useEffect( effect: () => {
  setButtons( value: {
    create: {
      label: I18n.t( key: 'common.create'),
      icon: <PlusOutlined />,
      action: showProprietaireModal,
    },
  })
  tables.proprietaires.onRow = (record, id) => ({
    onClick: () => updateProprietaire(record, id),
  })
  getProprietairesData() //Fait un appel pour récupérer les dernières données
}, deps: [])

```

Fig.29 Extrait ProprietaireScreen.js

Lors de l'initialisation du composant, l'useEffect est utilisé pour effectuer plusieurs actions. Tout d'abord, la fonction setButtons() est appelée pour mettre à jour l'état du bouton. Cela permet d'ajouter l'objet create, qui contient le label "Créer", une icône et une action à exécuter lors du clic. Dans cet extrait, cela permet d'accéder à la modal de création d'un Propriétaire.

Ensuite, j'ai créé une fonction qui permet à l'utilisateur de cliquer sur une ligne de la table. Cette fonction utilise la méthode onRow et prend en paramètre la ligne de la table (record) et son id pour afficher les données de la ligne et les mettre à jour avec la fonction updateProprietaire.

Enfin, la fonction getProprietairesData est appelée pour récupérer les dernières données de la table Propriétaire.

```

/**
 * Récupération des données des propriétaires
 */
3 usages  ⚙ Meysson17Alexis
const getProprietairesData = () => {
  setFetching( value: true)
  actions
    .getProprietaires()
    .then(({ body: proprietaires }) => {
      // Met à jour la table
      setTables( value: {
        ...tables,
        proprietaires: {
          ...tables.proprietaires,
          rows: proprietaires,
        },
      })
      setFetching( value: false)
    })
    .catch(() => {
      message.error(I18n.t( key: 'errors.proprietaires.fetch'))
    })
}

```

Fig. 30 Extrait ProprietaireScreen.js

Dans cet extrait de code, j'ai créé la constante `getProprietairesData`, pour récupérer les données de la table Proprietaire. Le `setFetching` permet d'indiquer que les données sont soit `True` pour dire que les données sont en cours de chargement soit `False` et dans ce cas là il affiche un message d'erreur pour dire échec de la récupération de données.

Quand les données sont récupérées, la fonction met à jour la table en utilisant la méthode `setTables` pour mettre à jour le contenu de `rows` de l'objet `proprietaire`. Les méthodes `tables` et `setTables` proviennent d'une constante qui utilise la fonction `useState` qui est une variable d'état qui permet de retourner un tableau avec 2 éléments, qui est utilisé dans ce code « `const [tables, setTables] = useState(ProprietaireTableData)` », `ProprietaireTableData` est importé via « `import {ProprietaireTableData} from "Components/Values/ProprietaireData/ProprietaireTableData"` » décrite auparavant.

```
//Ouvre le modal pour une création
1 usage  ➤ Meysson17Alexis
const showProprietaireModal = () => {
  setProprietaire( value: null)
  setOpenModal( value: 'createProprietaire')
}

//ferme le modal
2 usages  ➤ Meysson17Alexis
const hideProprietaireModal = () => {
  setOpenModal( value: '')
}

//Ouvre le modal pour un update et set les valeurs a update
1 usage  ➤ Meysson17Alexis
const updateProprietaire = (record) => {
  setOpenModal( value: 'updateProprietaire')
  setProprietaire(record)
}

<ProprietaireModal
  visible={includes( collection: ['createProprietaire', 'updateProprietaire'], openedModal)}
  onCancel={hideProprietaireModal}
  title={
    openedModal === 'updateProprietaire'
      ? I18n.t( key: 'Mettre a jour un Proprietaire')
      : I18n.t( key: 'Ajouter un Proprietaire')
  }
  onOk={handleSuccess}
  proprietaires={proprietaire}
/>
```

Fig.31 Extrait `ProprietaireScreen.js` pour accéder à la modal

L'importation de 'actions' depuis le fichier de ressource 'ProprietairesRessource' est effectuée en utilisant la syntaxe `import { actions as ProprietaireActions } from 'Resources/ProprietairesRessource'` qui sont décrites auparavant. Ces méthodes sont utilisées dans le fichier `ProprietaireScreen.js` pour interagir avec l'API et gérer les données des propriétaires.

Dans ces extraits de code, j'utilise le composant ProprietaireModal pour afficher une modale, en utilisant différents propriétés. La propriété "visible" permet de contrôler l'affichage de la modale en fonction de la valeur de ce booléen. La propriété "onCancel" est une fonction qui sera appelée lorsque l'utilisateur ferme la modale, par exemple en cliquant sur le bouton "Annuler". La propriété "title" permet de définir le titre de la modale en fonction de l'action demandée, soit la création d'un nouveau propriétaire soit la mise à jour d'un propriétaire existant. La propriété "onOk" est une fonction qui sera appelée lorsque l'utilisateur soumettra le formulaire de la modale, par exemple en cliquant sur le bouton "Enregistrer". Enfin, la propriété "proprietaires" permet de passer les données du propriétaire à afficher dans la modale.

La fonction "showProprietaireModal" permet d'ouvrir la modale pour la création d'un nouveau propriétaire. Elle utilise les fonctions "setProprietaire" et "setOpenedModal" pour mettre à jour les états de la propriété et de la modale, en passant "null" à la propriété pour effacer les données du propriétaire précédent et en passant le nom de la modale "createProprietaire" à "setOpenedModal".

```
/*
 * update d'un proprietaires en base de données
 */
1 usage  ➤ Meysson17Alexis *
const updateProprietaires = (data) => {
  actions
    .updateProprietaireProprietaires({
      id: proprietaires.idproprietaire,
      code_proprietaire: data.code_proprietaire,
      libelle: data.libelle,
    })
    .then(() => {
      message.success(I18n.t( key: 'success.proprietaires.update'))
      onOk()
    })
    .catch(() => {
      message.error(I18n.t( key: 'errors.proprietaires.update'))
    })
}
```

```
/*
 * creation d'un proprietaires en base de données
 */
1 usage  ➤ Meysson17Alexis *
const createProprietaires = (data) => {
  console.log(data)
  actions
    .createProprietaires(data)
    .then(() => {
      message.success(I18n.t( key: 'success.proprietaires.create'))
      onOk()
    })
    .catch(() => {
      message.error(I18n.t( key: 'errors.proprietaires.deletee'))
    })
}
```

Fig.32 Extrait Modal ProprietaireModal.js

La fonction `updateProprietaire` permet d'ouvrir la modal pour la mise à jour d'un propriétaire et de mettre à jour les valeurs du propriétaire à partir de l'enregistrement (record) fourni en paramètre. Elle utilise la méthode `setOpenedModal` pour ouvrir la modal avec la valeur 'updateProprietaire', puis utilise `setProprietaire` pour définir les valeurs du propriétaire à mettre à jour avec les valeurs de l'enregistrement fourni.

J'ai développé cette fonction `createProprietaire` pour créer un propriétaire dans la BDD, elle prend en paramètre les données. Ces données sont envoyées à la méthode `createProprietaires` de l'objet `actions`, qui est importé depuis le fichier `ProprietairesRessource`. Si la création réussit, un message de succès est affiché et la fonction `onOk` est appelée. Si la création échoue, un message d'erreur est affiché.

Pour une mise à jour des informations d'un propriétaire existant, j'utilise la fonction `updateProprietaires` en utilisant `actions` de `ProprietaireActions`. Cette fonction prend un paramètre « data » qui contient les informations comme l'id du propriétaire, son libelle et son code pour faire la mise à jour du propriétaire. J'emploie la fonction `updateProprietaireProprietaires` qui permet d'utiliser la méthode `PATCH` qui peut mettre à jour certains champs comme par exemple libelle ou le code propriétaire. Si la mise à jour réussit, un message de succès est affiché et la fonction "onOk" est appelée. Si la mise à jour échoue, un message d'erreur est affiché.

```

{ /* Nom */
<Form.Item
  name='code_proprietaire'
  label={I18n.t( key: 'pages.proprietaire.table.code')}
  rules={[
    {
      required: true,
      message: I18n.t( key: 'fields.proprietaire.requiredMessageCode'),
    },
    {
      max: 10,
      message: I18n.t( key: 'fields.proprietaire.patternLong'),
    },
    {
      pattern: /^[a-zA-Z0-9&$/\-\s.]+$/,
      message: I18n.t( key: 'fields.proprietaire.patternCode'),
    },
  ]}
>
  <Input placeholder={I18n.t( key: 'fields.proprietaire.placeholderCode')}/>
</Form.Item>

```

Fig. 33 Extrait Formulaire Modal ProprietaireModal.js

Dans le formulaire de création ou de mise à jour d'un propriétaire, j'utilise une bibliothèque Ant Design en utilisant les composants fournis pour `Form.Item`, `Input`.

J'utilise la propriété `rules` qui contient une liste de règles de validation qui sont appliquées dans ce formulaire lors de la soumission, c'est 3 règles pour le champ code propriétaire sont :

- Le `required` pour indiquer que le champ est obligatoire avec un message ;
- `Max` pour limiter la longueur du champ, par exemple 10 caractères
- Le `pattern` qui permet de restreindre les caractères autorisés lors de la saisie de l'utilisateur, par exemple des lettres majuscule/minuscule, des chiffres et certains symboles.

Le placeholder du champ de saisie, affiche un message de rappel pour aider l'utilisateur lors de sa saisie.

```
"proprietaire": {
  "fetch": "Erreur lors de la récupération des Propriétaires.",
  "create": "Erreur lors de la création des Propriétaires.",
  "update": "Erreur lors de la modification des Propriétaires.",
  "archive": "Erreur lors de l'archivage d'un ou des Propriétaires.",
  "archives": {
    "fetch": "Erreur lors de la récupération des Propriétaires archivés.",
    "restore": "Erreur lors de la restauration des Propriétaires."
  }
},
```

Fig. 34 Extrait I18n sur propriétaire

Dans cet extrait de code, j'utilise I18n pour afficher les messages d'erreur en cas de récupération de données échouée ou de création de propriétaire échouée. Les messages d'erreur sont écrits dans la langue de l'utilisateur grâce à la bibliothèque de traduction I18n.

- Le choix des colonnes des Engins (Annexe)

Dans l'annexe, la figure 43 met en évidence la version de PHP et présente également les différentes options permettant de choisir les colonnes.

Les figures 44 et 46 décrivent un processus permettant à l'utilisateur de sélectionner les colonnes souhaitées à l'aide d'un Select Multiple. Ce dernier s'affiche progressivement au fur et à mesure des sélections effectuées par l'utilisateur. Une fois que l'utilisateur a validé sa sélection de colonnes, il devient possible de rechercher le contenu spécifique des colonnes choisies.

La fonction `handleColumnSelection` permet de gérer la sélection des colonnes dans un tableau et de mettre à jour les colonnes affichées dans un tableau en fonction de la sélection de l'utilisateur.

La fonction `getEnginsData` est utilisée pour récupérer les données des engins en fonction des colonnes sélectionnées, met à jour les tables filtrées avec les nouvelles

données et gère les cas où aucune colonne n'est sélectionnée en utilisant les colonnes par défaut.

La fonction `handleValidateSelection` permet de valider les colonnes choisies et d'envoyer à la fonction `getEnginData`.

La composante `MultiSelectTags` qui affiche une liste déroulante qui permet à l'utilisateur de sélectionner plusieurs colonnes à l'aide de tags. Lorsque la sélection change, la fonction `onChange` est appelée avec les colonnes sélectionnées en tant que paramètre.

La section du code qui permet d'afficher le composant `MultiSelectTags` pour la sélection des colonnes, un bouton pour valider la sélection, et le composant `TableLayout` pour afficher un tableau avec les colonnes et les données filtrées en fonction de la sélection des colonnes.

- *Présentation du jeu d'essai élaboré par le candidat de la fonctionnalité la plus représentative*

J'utilise Swagger pour tester l'API RestFul, c'est une plateforme open-source.

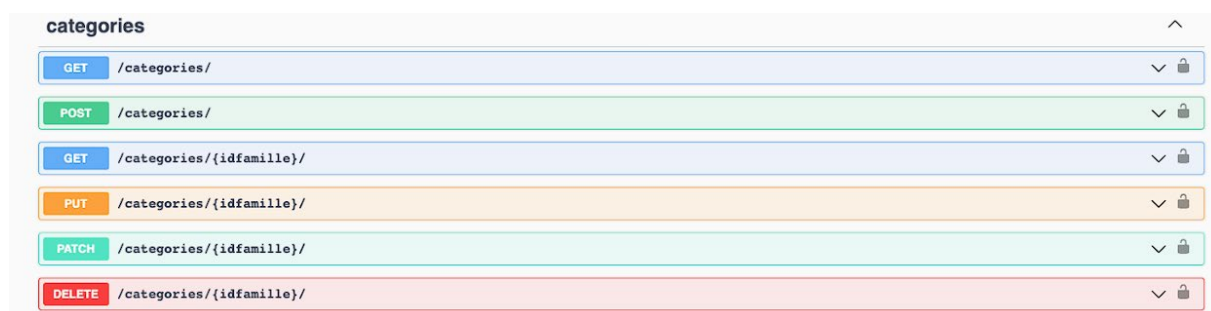


Fig. 35 Extrait Swagger sur Catégorie

Swagger permet d'attribuer une couleur spécifique à chaque méthode de l'API pour faciliter leur identification. Ainsi, la méthode GET est généralement associée à la couleur bleue, le POST à la couleur verte, le PATCH à la couleur vert clair, le PUT à la couleur orange et le DELETE à la couleur rouge.

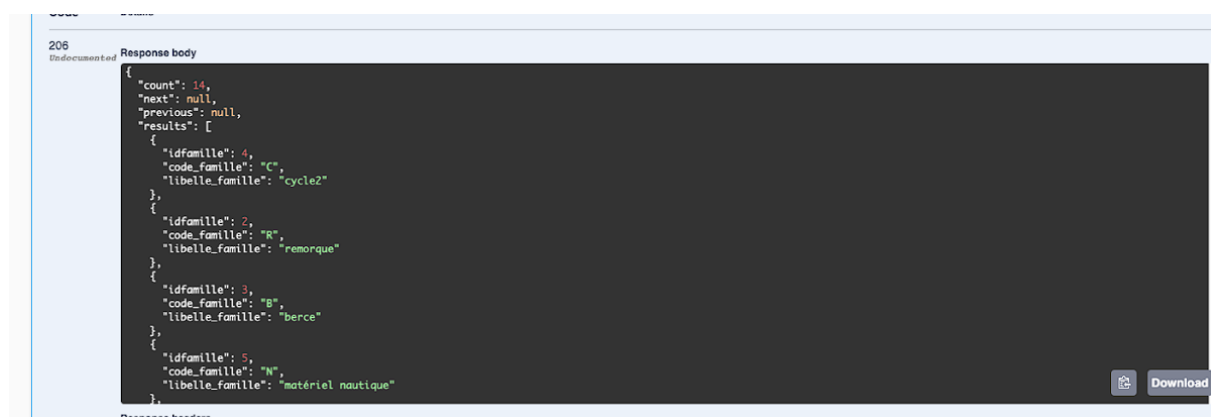


Fig. 36 Méthode GET de Swagger pour l'API catégorie

La méthode GET permet de récupérer une liste de catégories enregistrées dans la base de données. Le code de réponse HTTP 206 peut être utilisé pour indiquer que seule une partie de la ressource demandée a été renvoyée avec succès.

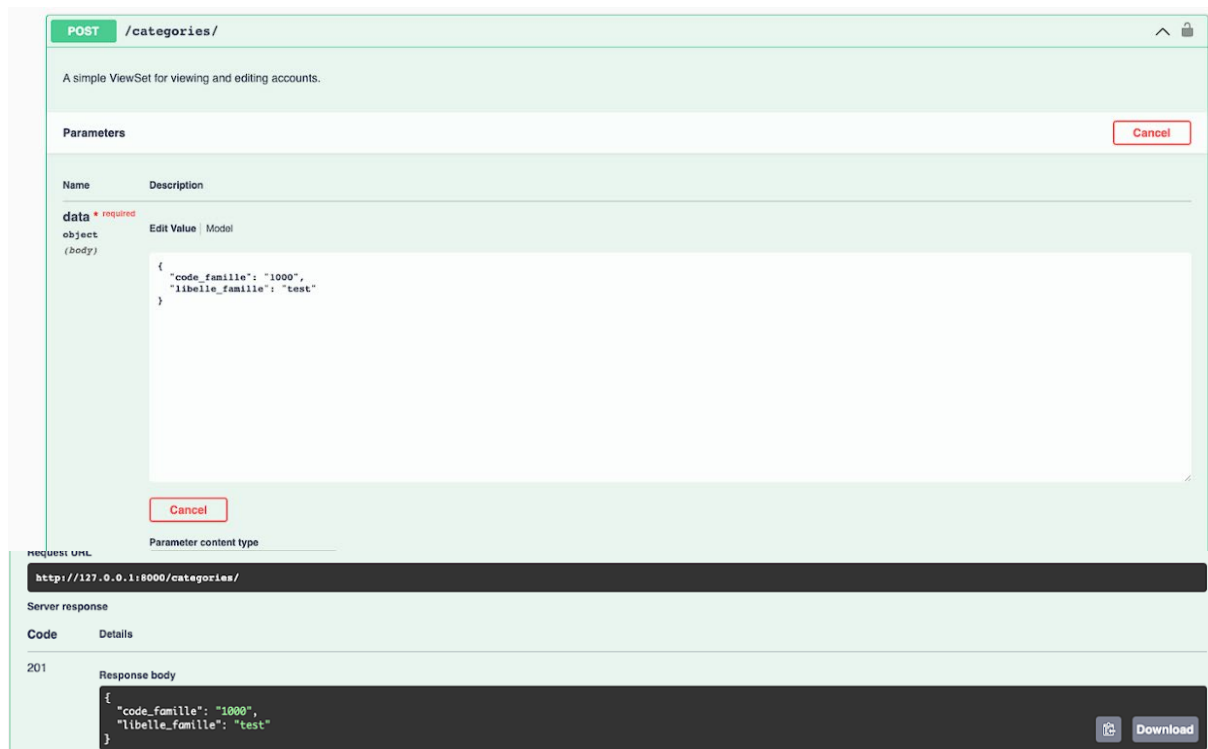
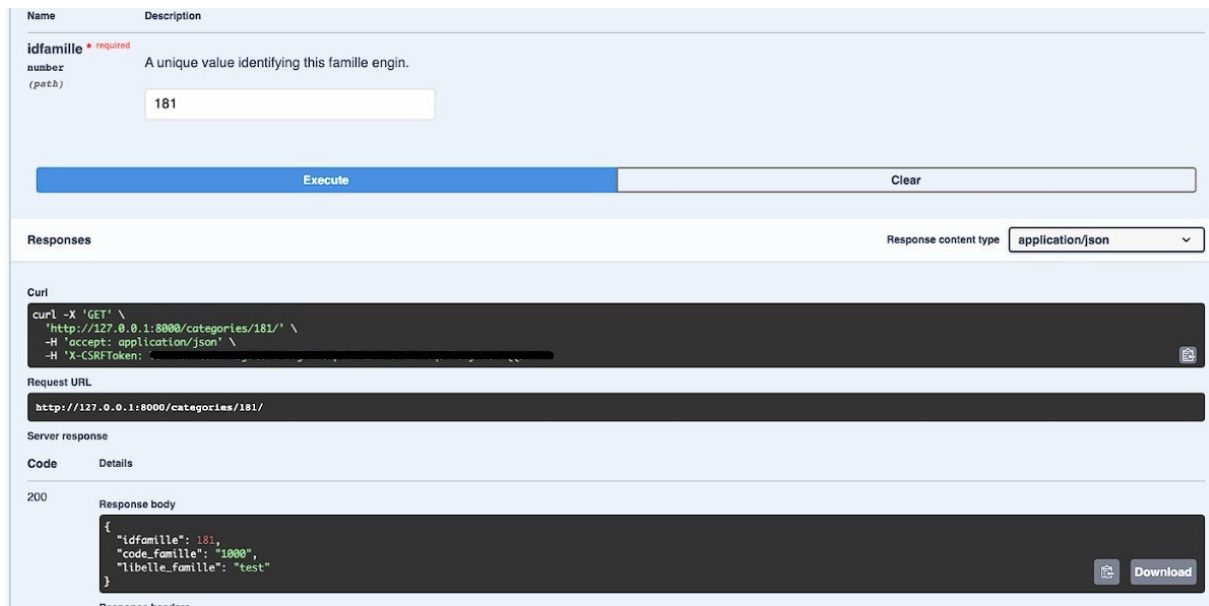


Fig. 37 Méthode POST de Swagger pour l'API categorie

La méthode POST permet d'envoyer des données au serveur qui sont soit pour créer une nouvelle ressource, soit pour ajouter une nouvelle entrée. Si la requête est effectuée avec succès, le code de réponse HTTP 201 est renvoyé, indiquant que la création de la nouvelle donnée a été réalisée avec succès.

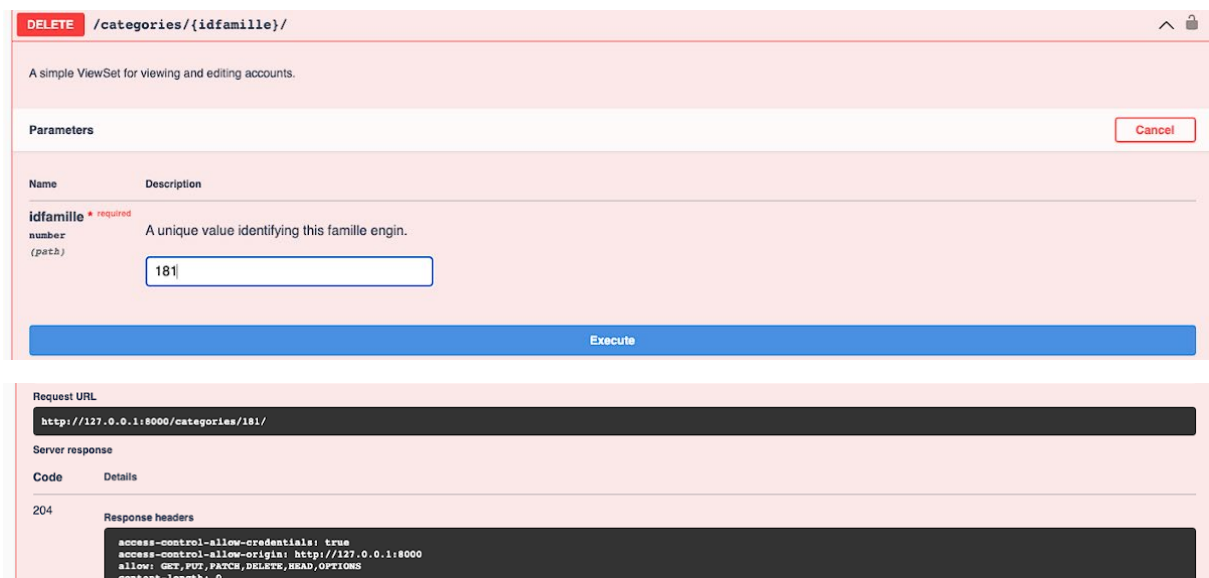
En revanche, si la requête échoue, un code d'erreur sera renvoyé, tel que le code d'erreur 4xx ou 5xx, qui indique que la requête n'a pas été traitée avec succès. Dans le cas où un champ est obligatoire qui n'a pas été renseigné, un message d'erreur sera affiché pour indiquer quel champ est manquant, par exemple "require is the fields" pour indiquer que le champ est requis.



The image shows the Swagger UI interface for a GET request. The top section is titled 'Name' and 'Description'. The parameter 'idfamille' is marked as 'required' and has a description 'A unique value identifying this famille engin.'. Below this, there is a text input field containing the value '181'. There are 'Execute' and 'Clear' buttons. The 'Responses' section shows the 'Response content type' as 'application/json'. Below this, the 'Curl' section displays the command: `curl -X 'GET' \ 'http://127.0.0.1:8000/categories/181/' \ -H 'accept: application/json' \ -H 'X-CSRFToken: ...'`. The 'Request URL' section shows 'http://127.0.0.1:8000/categories/181/'. The 'Server response' section shows a 'Code' of '200' and a 'Response body' containing a JSON object: `{ "idfamille": 181, "code_famille": "1000", "libelle_famille": "test" }`. There is a 'Download' button next to the response body.

Fig. 38 Méthode GET avec id de Swagger pour l'API catégorie

La méthode GET avec Id permet de récupérer les informations d'une catégorie précise en utilisant son identifiant. Cela permet de récupérer son code_famille et son libelle_famille.



The image shows the Swagger UI interface for a DELETE request. The top section is titled 'DELETE' and '/categories/{idfamille}/'. Below this, there is a description 'A simple ViewSet for viewing and editing accounts.'. The 'Parameters' section shows the parameter 'idfamille' marked as 'required' with the description 'A unique value identifying this famille engin.'. There is a text input field containing the value '181'. There is a 'Cancel' button. The 'Execute' button is at the bottom. The 'Request URL' section shows 'http://127.0.0.1:8000/categories/181/'. The 'Server response' section shows a 'Code' of '204' and 'Response headers' containing: `access-control-allow-credentials: true, access-control-allow-origin: http://127.0.0.1:8000, allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS, content-length: 0`.

Fig. 39 Méthode DELETE avec id de Swagger pour l'API categorie

La méthode DELETE permet de supprimer les informations d'une catégorie en utilisant son identifiant dont le code de réponse 204 qui sera renvoyé pour dire que la suppression a été un succès.

- *Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité*

Le facteur la plus importantes de la sécurité c'est la gestion des informations d'identifications des utilisateurs. Pour cela, les mots de passe des utilisateurs sont stockés de manière sécurisée en utilisant les algorithmes de hachage et de salage pour renforcer la sécurité sans pour autant de ne jamais stocker les données sensibles dans des cookies et/ou URL.

Les différentes attaques qui peuvent être utilisé sont les Injections SQL, les XSS, les CSRF, etc....

Django propose des mesures de sécurité variée pour prévenir efficacement sur chaque attaque.

Les requêtes Django sont immunisées contre l'injection SQL car leurs requêtes sont construites à l'aide de paramètres de requête. Le code SQL de la requête est défini séparément de ses paramètres. Étant donné qu'ils peuvent être lancés par l'utilisateur et donc dangereux, le pilote de base de données sous-jacent s'assure qu'ils sont supprimés.

Les attaques XSS permettent à un attaquant d'injecter des scripts côté client dans les navigateurs des utilisateurs.

L'idée générale est de stocker les scripts malveillants dans une base de données où ils peuvent être récupérés et montrés à d'autres utilisateurs, ou même amener les utilisateurs à cliquer sur un lien qui provoque l'exécution du JavaScript de l'attaquant dans le navigateur des utilisateurs. Cependant, les attaques XSS peuvent provenir de n'importe quelle source de données non fiable, comme les cookies ou les services Web, si les données ne sont pas suffisamment filtrées avant d'être ajoutées à la page.

Les attaques CSRF permettent à un attaquant d'effectuer des actions en utilisant les informations d'identification d'un autre utilisateur à son insu.

Django fournit une protection intégrée contre la plupart des attaques CSRF si vous l'activez et l'utilisez correctement. Mais comme toute technologie de défense, elle a des limites. Par exemple, le module CSRF peut être désactivé globalement ou pour des vues spécifiques. D'autres restrictions s'appliquent si l'application contient des sous-domaines qui ne sont pas sous le contrôle.

Pour limiter l'accès des utilisateurs aux ressources, Django utilise des mécanismes d'authentification et d'autorisation. Ces mécanismes permettent des restreindre les actions des utilisateurs, en fonction de leurs droits d'accès.

Chaque mesure de protection permet de sécuriser l'application et de protéger les utilisateurs contre les vulnérabilités potentiel, les fuites de données sensibles, de compromettre la base de données et d'autre formes de cybercriminalité. Ces mesures sont essentielles pour garantir la vie de l'application et la confiance des utilisateurs.

« Source utilisé pour la veille : <https://docs.djangoproject.com/fr/4.2/topics/security/> et <https://owasp.org/Top10/fr/> »

- *Description d'une situation de travail ayant nécessité une recherche et effectuée par le candidat durant le projet*

Au début de ce projet, je n'avais aucune expérience avec le Framework Django Rest Framework ou la bibliothèque REACTJS. Ma première étape a donc été de parcourir la documentation en ligne pour comprendre leur fonctionnement et comment les utiliser. Ensuite, étant donné que l'application web de SIGEE existait déjà sous PHP, j'ai consacré du temps pour explorer son architecture et ses fichiers, tout en testant les fonctions utilisées.

Pour réaliser ce projet, j'ai dû effectuer de nombreuses recherches, allant de la création de l'API Restful et de la connexion à la base de données Oracle avec Django Rest Framework, à la compréhension du fonctionnement d'Ant Design, qui est une bibliothèque de composants d'interface utilisateur.

Pour expliquer ma méthodologie, je vais prendre comme exemple l'enregistrement de la date dans la base de données avec un format de date spécifique. J'ai donc cherché une bibliothèque qui permettrait de formater les dates pour qu'elles correspondent aux conditions requises. En utilisant des mots clés tels que "format date ReactJS ant design library yyyy-mm-dd", j'ai consulté de nombreuses sources et testé les solutions proposées. La seule qui a fonctionné est la bibliothèque moment.js (<https://momentjs.com/>), qui permet de définir la valeur par défaut du composant DatePicker d'Ant Design et d'utiliser le format souhaité pour enregistrer la date dans la base de données au format anglais.

J'ai donc installé moment.js en utilisant la commande "yarn add moment" et je l'ai importé dans l'application avec les lignes de code suivantes :

```
import moment from 'moment';  
  
import 'moment/locale/zh-cn';  
  
import locale from 'antd/es/locale/en_US';
```

Ensuite, j'ai utilisé la fonction DatePicker de Ant Design avec la propriété defaultValue pour utiliser moment(montage) comme valeur par défaut et le format "yyyy-MM-DD" pour enregistrer la date dans la base de données.

J'ai effectué aussi plusieurs recherches sur le ModelSerializers, qui permet de simplifier la sérialisation et la désérialisation des modèles de données, et faciliter les opérations de transfert et de manipulation de données. (Annexe Fig. 48)

BILAN ET CONCLUSION

- *Bilan et perspectives du (ou des) projet(s) présenté(s)*

Une fois que le service Active Directory sera activé, l'application SIGEE ne sera plus connectée à Internet. Par conséquent, j'importerai toutes les bibliothèques utilisées avant de l'activer car cette application sera utilisée uniquement en intranet. Après le déploiement, il y aura une période de maintenance pour maintenir le code et la sécurité de l'application à jour, ainsi que pour ajouter de nouvelles fonctionnalités et la faire évoluer.

- *Conclusion*

Pendant mon alternance, j'ai pu mettre en pratique les connaissances théoriques que j'ai acquises lors de ma formation à l'ENI et j'ai été confronté au monde professionnel.

Cette expérience a été très enrichissante et m'a permis de comprendre toutes les facettes du métier de développeur. J'ai apprécié le fait d'être pleinement intégrée dans la vie de l'entreprise et d'avoir participé aux diverses réunions et échanges, même ceux liés à d'autres projets. À la fin de mon alternance, l'entreprise m'a proposé un contrat de travail et que j'accepterai avec plaisir. J'ai également été très reconnaissante pour les commentaires constructifs que j'ai reçus de mon maître d'apprentissage.

Cette expérience m'a permis de répondre aux questions que j'avais sur la réalité du métier de Concepteur Développeur d'Application. En accomplissant les missions qui m'ont été confiées, j'ai pu acquérir de nouvelles compétences et découvrir de nouvelles technologies très réputées dans le domaine du web. Je suis entrée dans cette formation avec une grande motivation pour apprendre à coder et j'en ressors passionnée. Je tiens à remercier toutes les personnes qui m'ont aidé à en arriver là.

Annexes

Elément	Application	Description	Taches	Commentaire	Droits users	Sprint
Gestion des propriétaires d'engins	Back Office	En tant qu'administrateur, je peux visualiser et de gérer les propriétaires d'engins	[FRONT] Créer la vue liste des propriétaires	1 tableau avec 3 colonnes (code_propriétaire, Libelle_propriétaire, action) + 1 bouton de création + barre de recherche	Admin	2
			[API] Créer le service de récupération des propriétaires			
			[FRONT] Créer la vue de création des propriétaires avec le formulaire associé			
			[API] Créer un service d'ajout des propriétaires			
			[FRONT] Créer la vue de modification et de suppression des propriétaires avec le formulaire associé	Mettre un bouton modifier pour chaque ligne, fait apparaitre une modal avec le formulaire mettre un bouton supprimer pour chaque ligne, demande de confirmation avant suppression		
			[API] Créer un service de modification et de suppression des propriétaires			

Fig. 40 Extrait du Backlog de l'application SIGEE



Fig. 41 Planning Sprint Jira

Grâce au système de planification Jira, il est possible de suivre l'avancement de l'application SIGEE depuis la phase d'analyse jusqu'à la maintenance en passant par les différents sprints. Cette fonctionnalité permet d'avoir une visualisation claire et précise de l'avancement du projet et de la planification des tâches.

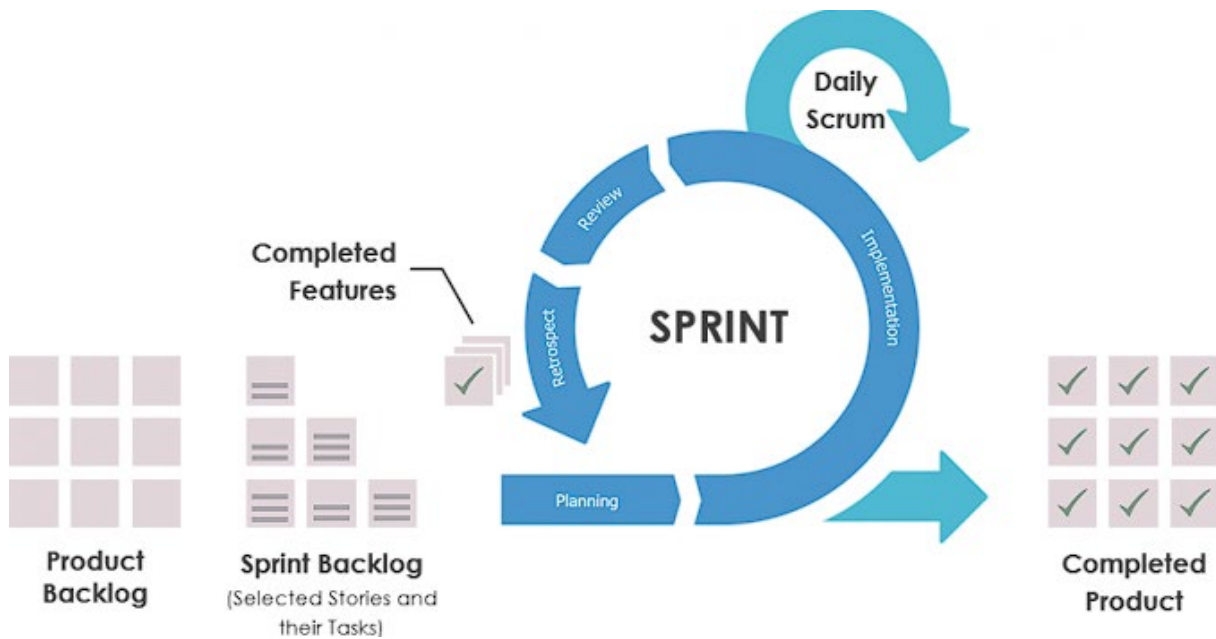


Fig. 42 Méthode Agile

Sélection des colonnes des Engines

◇ Affichage des colonnes du tableau

CLASSIFICATION ☐ GAMME ☐ MARQUE ☐ APPCOMM ☐ EQUIPEUR ☒ TYPE TECH ☒ IMMAT ☒ PHY ☐ CARROSSERIE ☐ D_MISE_CIRCULATION ☐ ANNEE ☐ PROPRIETAIRE ☐ CATEGORIE ☐
 SORTIE INVENTAIRE VILLE ☐ ETAT ☐ FVADMINISTRATIVE ☐ SORTIE SERVICE ☐ NNO ☒ TYPE OPS ☐ TYPE SERV ☐ ADM ☐ COMPTEUR ☐ DUREE VIE THEOR ☐ POSITION ☐
 N°SERIE ☐ ENTREE SERVICE ☐ CAT.AFFEC ☐ STATUT OPS ☐ STATUT TECH ☐ COMPTEUR CORRIGE ☐ HORAMETRE ☐ N°FOLIO ☐ N°DOSSIER ☐ N°INVENTAIRE VILLE ☐ OBSERVATIONS ☐
 LIBELLE ☐ NUMERO ☐ AGE ENGIN ☐ ATTRIBUTION ☐ RADIO ☐ ATTELAGE ☐ ENTREE REPARATION ☐ SORTIE REPARATION ☐ RAISON REPARATION ☐

de: au:

◇ Paramètres supplémentaires

Age du parc :

Afficher 25 éléments

Affichage de l'élément 1 à 25 sur 1,020 éléments

N°	EQUIPEUR	TYPE TECH	IMMATRICULATION	NNO
N°	EQUIPEUR	TYPE TECH	IMMATRICULATION	NNO
1	DURISOTTI	VTUM	B281 ZY 13	HN0040124
2	DURISOTTI	VTUM	B281 ZY 13	HN0040124
3	---	VL	AF-486-SG	HN0100020
4	DURISOTTI	VLTC	CH-892-MY	hn0130081
5	DURISOTTI	VLTC	CH-892-MY	hn0130083
6	METIS	VRM	FM-144-AN	HN0130083

Fig. 43 Choix des colonnes en PHP ancienne version

Immat. x Type Tech x |

D_Mise_Circulation

App_Comm

Gamme

Marque

Etat

Radio

Attelage

Compteur

Age du parc: 7

Recherche

+ Créer un engin

Affichage de l'élément 1 à 10 sur x éléments

Numero	Immatriculation	Type TECH	D_MISE_CIRCULATION
1	B281 ZY 13	VTUM	2004-02-27
2	AF-486-SG	VL	2021-02-02
3	CH-892-MY	VLTC	2021-02-02
4	CH-937-MY	VLTC	2021-02-02
5	CM-138-AN	CBPC	2021-02-02
6	CM-676-MY	CCF4	2021-02-02
7	CM-478-MY	CCF4	2021-02-01

Afficher 10 éléments

Fig. 44 Maquette

Immatriculation x Carrosserie x Classification x |

Mots clés de recherche

10 éléments affichés

Immatriculation	Carrosserie	Classification
B255 ZS 13	ZR 17	CDT
B281 ZY 13	TMDEPOL	SPE
B3	P 01	--
B340 VE 13	V5	--
B40 PV 13	???	--
B406357	P 01	--

Immatriculation x Carrosserie x Classification x |

Immatriculation ✓

Type Tech

Carrosserie ✓

Date 1ere mise en circulation

Classification ✓

Gamme

Marque

Appellation Commerciale

B3	P 01	--
B340 VE 13	V5	--
B40 PV 13	???	--
B406357	P 01	--

Immatriculation x Carrosserie x Classification x Gamme x

Mots clés de recherche

10 éléments affichés

<input type="checkbox"/>	Immatriculation	Carrosserie	Classification	Gamme
<input type="checkbox"/>	8255 ZS 13	2R 17	CDT	2R
<input type="checkbox"/>	8281 ZY 13	TMDEPOL	SPE	VL
<input type="checkbox"/>	83	P 01	--	VL
<input type="checkbox"/>	8340 VE 13	VS	--	VL
<input type="checkbox"/>	840 PV 13	???	--	VL

Fig. 45 Choix des colonnes en ReactJS

```
const handleColumnSelection = (selectedColumns) => {
  setSelectedColumns(selectedColumns);
  const updatedColumns = selectedColumns.map((column) => ({
    ...EnginTableData.engins.columns[column],
  }));
  setFilteredTables( value: () => ({
    engins: {
      pagination: {
        position: 'both',
        showSizeChanger: true,
      },
      rowKey: 'id',
      columns: updatedColumns,
    },
  }));
};
```

```

const getEnginsData = () => {
  setFetching( value: true);
  // Vérifier si les colonnes sont sélectionnées
  if (selectedColumns.length > 0) {
    actions
      .getEngins(null, {
        query: {
          limit: 10,
        },
      })
      .then(({body}) => {
        var finalData = {
          'count': body.count,
          'next': body.next,
          'previous': body.previous,
          'results': body.results
        }
        setFilteredTables( value: (prevTables : {engins: {columns: any[], rows: any[]}} ) => ({
          ...prevTables,
          engins: {
            ...prevTables.engins,
            rows: finalData,
          },
        }));
        setFetching( value: false);
      })
      .catch(() => {
        message.error(I18n.t( key: 'errors.engin.fetch'));
        setFetching( value: false);
      });
  } else {
    // Utiliser toutes les colonnes par défaut
    setFilteredTables( value: {
      engins: {
        columns: Object.keys(EnginTableData.engins.columns),
        rows: [],
      },
    });
    setFetching( value: false);
  }
}

```

```

const MultiSelectTags = ({onChange}) => {
  1 usage  ± Meysson17Alexis
  const handleColumnSelection = (selected) => {
    setSelectedColumns(selected);
    onChange(selected);
  };

  return (
    <Select
      mode="multiple"
      style={{width: '75%', margin: '0px 15px 15px 0px'}}
      placeholder="Sélectionnez les colonnes"
      optionLabelProp="label"
      value={selectedColumns}
      onChange={handleColumnSelection}
    >
      {Object.keys(tables.engins.columns).map((key : string ) => (
        <Select.Option key={key} value={key} label={tables.engins.columns[key].title}>
          {tables.engins.columns[key].title}
        </Select.Option>
      ))}
    </Select>
  );
};

```

```
const handleValidateSelection = () => {
  // Effectuez ici les actions souhaitées lorsque l'utilisateur clique sur le bouton "Valider"
  // Permet d'appeler une fonction pour enregistrer les colonnes sélectionnées dans votre base de données
  console.log('Colonnes sélectionnées validées :', selectedColumns);
  getEnginsData(selectedColumns); // Recherche des tables correspondantes à la sélection validée
};

<MultiSelectTags onChange={handleColumnSelection}/>
<Button onClick={handleValidateSelection}>Valider</Button>
<TableLayout
  buttons={buttons}
  loading={fetching}
  tables={{
    engins: {
      pagination: {
        position: 'both',
        showSizeChanger: true,
      },
      rowKey: 'id',
      columns: filteredTables.engins.columns || [],
      rows: filteredTables.engins.rows || []
    }
  }}
  updateSelectedData={setSelectedEngins}
  allowSelection={true}
/>
```

Fig. 46 Code du choix colonnes

Avancement













A FAIRE	EN COURS	TERMINÉ
Gestion des utilisateurs et Authentification  SIG-26 AM	Gestion des Engins  SIG-49	Gestion des gammes d'engins  SIG-59
[FRONT] Construire la vue 'Login' avec les formulaires nécessaires  SIG-27 AM	[FRONT] Créer la vue liste engins + bouton recherche + age du parc  SIG-50	[FRONT] Créer la vue liste des classifications  SIG-60
[API] Créer le service de login  SIG-28 AM	[API] Créer un service de récupération des engins  SIG-51	[API] Créer le service de récupération des classifications  SIG-61
[FRONT] Ajouter le bouton 'Mot de passe oublié' et le formulaire nécessaire  SIG-29	[FRONT] Créer la vue création d'un engin avec le formulaire associé  SIG-54	[FRONT] Créer la vue de création des gammes avec le formulaire associé  SIG-62

Fig. 47 Extrait Avancement

ModelSerializer

Serializers

- Declaring Serializers
- Serializing objects
- Deserializing objects
- Saving instances
- Validation
- Accessing the initial data and instance
- Partial updates
- Dealing with nested objects
- Writable nested representations
- Dealing with multiple objects
- Including extra context
- ModelSerializer**
- Inspecting a ModelSerializer
- Specifying which fields to include
- Specifying nested serialization
- Specifying fields explicitly
- Specifying read only fields
- Additional keyword arguments
- Relational fields
- Customizing field mappings
- HyperlinkedModelSerializer**
- Absolute and relative URLs
- How hyperlinked views are determined
- Changing the URL field name
- ListSerializer**
- allow_empty
- max_length
- min_length
- Customizing ListSerializer behavior
- BaseSerializer**
- Don't inherit BaseSerializer classes

ModelSerializer

Often you'll want serializer classes that map closely to Django model definitions.

The `ModelSerializer` class provides a shortcut that lets you automatically create a `Serializer` class with fields that correspond to the Model fields.

The `ModelSerializer` class is the same as a regular `Serializer` class, except that

- It will automatically generate a set of fields for you, based on the model.
- It will automatically generate validators for the serializer, such as `unique_together` validators.
- It includes simple default implementations of `.create()` and `.update()`.

Declaring a `ModelSerializer` looks like this:

```
class AccountSerializer(serializers.ModelSerializer):
    class Meta:
        model = Account
        fields = ['id', 'account_name', 'users', 'created']
```

By default, all the model fields on the class will be mapped to a corresponding serializer fields.

Any relationships such as foreign keys on the model will be mapped to `PrimaryKeyRelatedField`. Reverse relationships are not included by default unless explicitly included as specified in the [serializer relations](#) documentation.

Inspecting a ModelSerializer

Serializer classes generate helpful verbose representation strings, that allow you to fully inspect the state of their fields. This is particularly useful when working with `ModelSerializers` where you want to determine what set of fields and validators are being automatically created for you.

To do so, open the Django shell, using `python manage.py shell`, then import the serializer class, instantiate it, and print the object representation...

```
>>> from myapp.serializers import AccountSerializer
>>> serializer = AccountSerializer()
>>> print(repr(serializer))
AccountSerializer():
  id = IntegerField(label='ID', read_only=True)
  name = CharField(allow_blank=True, max_length=100, required=False)
  owner = PrimaryKeyRelatedField(queryset=User.objects.all())
```

Specifying which fields to include

Fig. 48 Extrait article ModelSerializer

- Webographie

- <https://docs.djangoproject.com/fr/4.2/topics/security/> => Sécurité avec Django
- [https://www.cnil.fr/fr/definition/interface-de-programmation-dapplication-api#:~:text=Une%20API%20\(application%20programming%20interface,des%20donn%C3%A9es%20et%20des%20fonctionnalit%C3%A9s.](https://www.cnil.fr/fr/definition/interface-de-programmation-dapplication-api#:~:text=Une%20API%20(application%20programming%20interface,des%20donn%C3%A9es%20et%20des%20fonctionnalit%C3%A9s.) => API
- <https://www.kaspersky.fr/resource-center/definitions/what-is-a-cross-site-scripting-attack> => XSS
- <https://www.kaspersky.fr/resource-center/definitions/sql-injection> => injection SQL
- <https://legacy.reactjs.org/docs/hooks-effect.html> => useEffect()
- <https://ant.design/> => ant Design
- <https://swagger.io/> => Swagger
- <https://owasp.org/Top10/fr/> => Open Web Application Security (OWASP) pour sécurité supplémentaire
- <https://momentjs.com/> => moment.js format date
- <https://www.oracle.com/fr/database/technologies/instant-client.html> => Oracle Instant-client
- <https://www.django-rest-framework.org/api-guide/serializers/#modelserializer> => ModelSerializer