Année Universitaire : 2024-2025
Licence MIASHS
UGA UFR SHS

# Documentation suite du projet de développement de l'application de gestion des absences de la licence MIASHS

Réalisé par :
Léna BOUCQ-KIEFFEL
Roxane DUBUS
Carole MITTON

Supervisé par :
Benoît LEMAIRE
Jérôme DAVID

# Résumé

Cette documentation vise à présenter le développement de l'application de gestion des absences de la licence MIASHS dans le but d'une reprise du projet pour finaliser la conception de l'application. En effet, même si l'application est utilisable, il reste à résoudre quelques problèmes et des améliorations peuvent être apportées.
Ce livrable vise donc à faciliter la reprise du projet.
Dans un premier temps, les missions relevées par le groupe pour ce projet ainsi que le contenu et les fonctionnalités de l'application seront présentés.
Dans un second temps, le résultat du travail fourni par le groupe et les outils utilisés seront abordés. Cette partie est divisée en 5 sous-parties présentant les différents aspects de la conception de l'application web : la base de données, le front-end, le back-end, ADE et également Github.
Ensuite, les problèmes que l'application soulèvent et des possibles solutions seront traités.
Pour finir, ce seront des propositions d'améliorations graphiques et fonctionnelles qui sont présentées.

# Sommaire

# 1. Introduction

## 1.1 Contexte

Ce document vise à présenter le travail réalisé sur le projet de développement d'une application web de gestion des absences pour la licence MIASHS durant l'année universitaire 2024-2025. Ce projet a été effectué dans le cadre de l'UE projet sur les deux semestres. Jérome DAVID et Benoît LEMAIRE ont supervisé ce projet.

## 1.2. Présentation de la mission

Ce projet vise à développer une plateforme pour les enseignant.e.s et les gestionnaires de scolarité de la licence MIASHS ayant pour objectif de faire l'appel et de consulter les absences des étudiant.e.s. Cette application web répond à une problématique rencontrée par les enseignant.e.s et gestionnaires en MIASHS qui n'ont aucun outil pratique afin de partager et gérer les absences au sein de la licence. L'application serait un moyen de faciliter ces tâches. Elle a donc besoin de répondre à différentes fonctionnalités qui ont été réfléchies avant sa programmation. Les étudiants qui ont travaillé sur le projet en 2023-2024 avaient décidé de faire trois profils : les étudiant.e.s, les enseignant.e.s et les gestionnaires de scolarité. Or, le profil étudiant nécessite un moyen d'identification, comme les identifiants Agalan par exemple. Étant trop compliquée à mettre en place, l'idée a été abandonnée par notre groupe.
Les fonctionnalités de l'application web ont alors été uniquement pensées pour un profil enseignant et un profil gestionnaire de scolarité sans identification. Pour les enseignant.e.s, il est important qu'ils/elles puissent facilement faire l'appel et accéder aux créneaux de leur matière avec les étudiant.e.s du groupe correspondant à ce créneau. Après une entrevue avec l'une des gestionnaires de scolarité, il a été décidé qu'il était nécessaire de pouvoir accéder aux récapitulatifs des absences des étudiant.e.s par promotion et par semestre et de modifier des groupes ou les informations d'étudiant.e.s. Il faudrait également qu'elle puisse télécharger un fichier avec un bilan du nombre total d'absences par étudiant.e.
Concernant la base de données, une partie de sa structure a été conservée comme elle avait été conçue dans le projet de l'année passée. Quelques modifications ont été apportées. Le schéma a donc été complété et modifié selon la nouvelle vision de l'application.
Pour ce projet, des missions ont été définies pour les deux semestres. Lors du semestre 5, l'interface de l'application web et la structure de la base de données ont été terminées et les notions de développement web apprises, afin de pouvoir suffisamment avancer le semestre suivant. Au semestre 6, le groupe s'est concentré sur la liaison de la base de données avec l'interface grâce au back-end et sur l'alimentation de la base de données.

# 2. Travail réalisé

Pour résumer, l'application remplit tout de même les fonctionnalités qui étaient prévues pour ce projet. En effet, l'appel peut être réalisé étant donné que les créneaux sont récupérés depuis ADE et que les absences sont enregistrées dans la base de données. De plus, l'insertion des données des étudiant.e.s et des groupes dans la base est possible en

important des fichiers .csv. Ainsi, la base de données peut être initialisée, mais également modifiée au cours de l'année, car une fonctionnalité de modification des informations des étudiant.e.s et de modification des groupes a été implémentée. De même, les récapitulatifs des absences des étudiant.e.s sont consultables et téléchargeables en fichiers .csv.
Cependant, l'appel ne peut être effectué qu'à la condition qu'un.e étudiant.e assiste au créneau associé à son groupe de TD/TP. Dans le cas contraire, l'absence qui sera indiquée pour son créneau sera enregistrée dans la base de données et considérera qu'il/elle était absent.e. Aussi, les données qui nous sont fournies par ADE ne concernent que les données des emplois du temps de la licence de l'année précédente, cela implique de devoir rajouter un filtrage s'il y a connexion avec l'ADE complet. En effet, sur ADE, les données de toutes les licences de la composante SHS, voire davantage, seront présentes, ainsi, il faudra ne sélectionner que les données de la licence MIASHS. Enfin, les questions d'homologation et de sécurité pour un déploiement futur de l'application n'ont pas été abordées. Tout le code et la documentation du projet sont accessibles sur le lien du GitHub (Annexe 1). L'IDE VSCode (Visual Studio Code) a été utilisé pour développer ce projet.

## 2.1. Base de données

La base de données (Annexe 2) est composée de 8 tables.
**Table student** : représente les étudiant.e.s avec leur numéro étudiant et leur nom.
**Table inscription** : table de liaison, permet de faire le lien entre les groupes et les étudiant.e.s.
**Table group** : représente les groupes auxquels peuvent être inscrit.e.s les étudiant.e.s (COG/ECO, promotion,...) avec leur nom. Les groupes sont rattachés à un semestre.
**Table semester** : représente les différents semestres d'une licence avec leur nom.
**Table course_material** : représente les matières avec leur nom. Les matières sont rattachées à un semestre.
**Table session_type** : représente les types de séance avec leur nom. Chaque session_type est rattachée à un course_matérial. Ainsi cette table permet de faire la différence entre les créneaux qui sont des CM ou des TD par exemple.
**Table slot** : représente les créneaux de cours avec leur date. Chaque slot est rattaché à un session_type et à un groupe.
**Table presence** : représente les absences des étudiant.e.s. Seules les absences sont retenues dans la base de données. Il s'agit d'une table de liaison entre les étudiant.e.s et les créneaux. Cette table n'enregistre que les absences des étudiant.e.s, si une ligne existe dans cette table alors c'est une absence.

Les données stockées dans la BDD proviennent de trois sources différentes :
Les tables en vert dans le schéma (Annexe 3) sont remplies grâce aux données extraites d'ADE (emplois du temps UGA). Ainsi, ADE permet de remplir les informations qui touchent aux structures des cours qui sont données dans la licence (matière, type de séance,...).
Les tables en bleu dans le schéma sont remplies grâce aux données fournies par les gestionnaires de scolarité. A la suite des inscriptions administrative et pédagogique, Geneviève Gaude, l'une des gestionnaires de scolarité, récupère des fichiers .csv qui permettent de remplir la base de données. Elle a accès aux listes des étudiant.e.s inscrit.e.s par semestres ainsi qu'à des listes des étudiant.e.s qui sont inscrit.e.s dans les différents groupes possibles (ECO/COG, groupe 1/2/3,...). Grâce à ses fichiers, en début de semestres les gestionnaires de scolarité peuvent renseigner les données des étudiant.e.s de

l'année ainsi que de créer les différents groupes en renseignant les étudiant.e.s qui y sont inscrit.e.s.
Enfin, les tables en jaunes dans le schéma sont remplies grâce aux professeur.e.s lors de l'appel. Après avoir choisi le créneau auquel ils/elles font cours, ils/elles peuvent alors choisir le groupe qui est concerné ce qui permet de faire la liaison entre les créneaux et les groupes. La table de présence est remplie à partir de la page d'appel qu'il peuvent alors remplir.

## 2.2. Front-end

Le code du front-end se trouve dans le dossier GestionAbsenceFront.

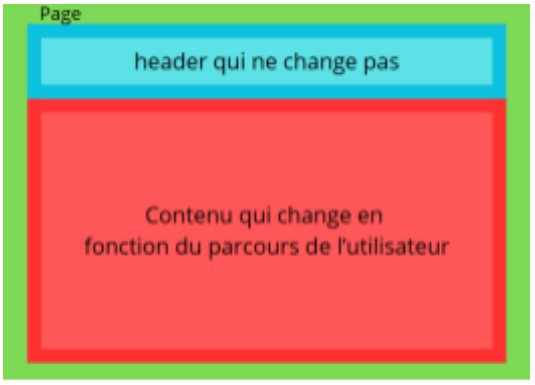

Figure 1. Schéma d'une SPA

L'application web est une Single Page Application ou SPA. Comme son nom l'indique, c'est une application qui n'a qu'une seule page. Lors de la navigation entre les différentes parties de l'interface, la page ne change pas. C'est ce qui y est affiché qui change. Dans le cas du projet, comme dans beaucoup d'autres applications, il y a une partie de l'affichage qui ne bouge pas, la barre de navigation. C'est le header, c'est-à-dire l'entête de la page.

Le framework Vue donne la possibilité de créer des composants, ce sont des éléments customisés que l'on peut ensuite réutiliser dans le code. Les différents éléments des maquettes ont donc été étudiés pour trouver des composants qui seront réutilisés dans plusieurs pages ou des composants plus complexes à coder à part pour rendre le code plus clair. Il existe parmi les composants une distinction entre les composants de présentation qui ne servent que pour l'interface et les composants conteneurs qui permettent de gérer et d'afficher les données que le back-end transmet au front-end.
Pour naviguer entre les différents composants à afficher, il y a un routeur qui permet de définir des chemins et de renvoyer à d'autres parties de l'interface en fonction des liens du routeur que l'on sélectionne sur l'application. Il y a un fichier route.js qui contient la définition de tous les chemins pour se déplacer dans les différentes parties de l'interface. C'est le routeur qui va indiquer quelle partie du code afficher à l'endroit où se trouve la balise RouterView. Cette balise se trouve dans le fichier App.vue qui est la base de l'interface, c'est aussi ici que se trouve l'appel du composant de la barre de navigation qui elle ne bougera jamais. Lorsque dans une partie de l'interface, un élément, qui est une balise RouterLink, est sélectionné, la balise RouterView va alors afficher le contenu auquel mène le chemin (path) du RouterLink. Il est possible de créer des chemins dynamiques.

Le code est structuré en suivant le principe de colocation. Si un composant est utilisé dans plusieurs pages, alors il se trouve dans le fichier "shared". Par contre, si un composant n'est utilisé que dans une seule page, alors on le retrouvera dans le dossier de cette page. Un avantage de cette architecture est qu'elle permet de savoir, lorsque l'on veut modifier un composant, si ces changements vont affecter une ou plusieurs pages.

Le groupe s'est aussi mis d'accord en amont pour suivre différentes normes d'écriture de code. Premièrement, tout est écrit en anglais pour garder la cohérence entre le langage de programmation et les noms donnés aux variables, méthodes... Secondement en HTML et CSS les noms des classes et autres suivent la norme kebab-case, c'est-à-dire que tous les mots sont écrits en minuscule et séparés par des tirets.

Afin de mieux visualiser le fonctionnement du front-end, des fichiers JavaScript Object Notation (JSON) ont été créés dans un dossier public (ils ne sont plus disponibles, car ils ont été utilisés uniquement pour les tests). Les JSON sont un format texte qui est compréhensible par une machine et simplement lisible pour les humain.e.s. Ces fichiers ont été remplis avec des données comme les informations des étudiant.e.s (nom, prénom, mail,...). Cela permet d'afficher des données dans les différentes parties de l'interface, afin de pouvoir les tester.

L'application vise à être utilisée par deux types d'utilisateurs différents : les enseignant.e.s et les gestionnaires de scolarité. Ils/Elles ont tous et toutes accès aux mêmes fonctionnalités, mais n'utilisent pas l'application pour les mêmes raisons.

Une fois que le back-end a été développé, la liaison du back-end avec le front-end était possible pour récupérer les données de la BDD et ne plus utiliser les JSON. Pour faire appel au backend, ce sont les endpoints qui sont utilisés. La conception des endpoint est expliquée dans la partie suivante.
Dans le dossier shared, un dossier fetchers a été ajouté pour contenir tous les fichiers .js contenant les fonctions asynchrones et exportables qui font appels aux endpoints. Ainsi, un fichier .js est créé pour chaque entité de la base de données, c'est-à-dire, pour chaque table. Les fonctions sont ensuite importées dans les fichiers .vue des pages dont on en a besoin. Par exemple, sur la page d'accueil où les professeur.e.s sélectionnent leur créneau, pour afficher tous les créneaux correspondant à la date sélectionnée, c'est la fonction getSlots(date) du fichier slots.js qui est importée et utilisée.

## 2.3. Back-end

Le code du back-end se trouve dans le fichier GestionAbsenceBack. Les commandes pour initialiser le back-end sont dans le read.me.
NestJS est le framework back-end qui a été choisi pour ce projet. Il est très complet et aujourd'hui très utilisé pour le développement de back-end. Il utilise par défaut le langage TypeScript (ts) mais peut aussi être utilisé en JavaScript (js). Dans ce projet, le langage ts a été utilisé. Un des désavantages de NestJS souvent souligné, c'est qu'il est assez lourd dans sa syntaxe puisque assez contraint. Cependant, c'est un avantage dans le développement à plusieurs, en étant forcé de tou.te.s suivre les mêmes normes, il est alors plus simple de comprendre ce que les autres font. Il offre aussi un grand panel de fonctionnalités, faisant de lui un framework très complet. D'où notre choix.
Ce framework permet donc de développer une **API REST**. Les **API** (**Applications Programming Interface**) sont des interfaces qui permettent la communication entre deux entités. Elles permettent de spécifier quelles méthodes peuvent être appelées, avec quels paramètres et quelles seront leurs réponses. Dans le cas des **API REST** (**REpresentational State Transfer**), c'est ce qui est fait, avec des requêtes **HTTP** grâce à des **endpoints**. Un

**endpoint**, c'est l'extrémité d'un canal de communication, c'est par là que l'on va pouvoir adresser une requête et recevoir la réponse. Les requêtes **HTTP** sont composées de plusieurs éléments : l'adresse du serveur à qui l'on s'adresse, la méthode que l'on va utiliser et le contenu de la requête. Il existe différentes méthodes mais les quatre plus utilisées suivent l'appellation **CRUD** (**Create, Read, Update, Delete**) et sont nommées dans les requêtes **HTTP** comme suit dans l'ordre **POST, GET, PUT, DELETE**. Ce sont les opérations les plus couramment utilisées dans la gestion de données. En développant le back-end, l'objectif est donc de créer la logique derrière chacun des **endpoints** nécessaires au projet et de les rendre accessibles via des requêtes **HTTP**.
Au niveau du développement du back-end, pour chaque table de la base de données, trois fichiers ts ont été codés : un **controller**, un **service** et un **module**.
Le **controller** va permettre de rendre accessibles les **endpoints** en spécifiant les méthodes grâce aux décorateurs (exemple : **@Get**) ainsi que les paramètres de la requête. Il va ensuite pouvoir faire suivre la requête au **service** et renverra la réponse au client une fois qu'il l'aura reçue du **service**. Il s'agit donc de gérer la communication entre le client et le back-end.
Le **service**, quant à lui, permet d'effectuer les opérations demandées, il permet de mettre en place la logique derrière les méthodes appelées pour ensuite renvoyer au **controller** la réponse. Il s'agit donc de gérer les opérations à effectuer sur la base de données.
Le **module** quant à lui permet d'organiser le tout en spécifiant les entités qui vont pouvoir travailler ensemble (exemple : studentController et studentService vont travailler ensemble).
Pour chacune des tables de la base de données, ces trois entités ont été créées et placées dans un dossier qui porte le nom de la table afin de développer des **endpoints** en fonction de nos besoins. Par exemple, pour les étudiant.e.s, l'**endpoint Get** avec comme paramètre un identifiant qui permet de récupérer les informations d'un.e étudiant.e en fonction de cet identifiant a été créé. De plus, il existe un fichier csv qui permet de gérer la réception des fichiers en .csv pour l'intégration des données grâce à la bibliothèque "csv-parser".
Enfin, dans certains cas, des **DTOs** (**Data Transfer Object**) ont été créés ; ils permettent de transporter des données entre deux processus et de spécifier le type de données qui est attendu. On les retrouve dans certains cas, comme les méthodes **POST**, entre les **controllers** et les **services**.
La liste de tous les endpoints nécessaire pour le développement de l'application web sont explicitées dans le document "tableau des endpoints". Ce tableau contient les endpoints avec leur méthode, URL, paramètre, body, la réponse renvoyée ainsi qu'une description de ce que fait l'endpoint.

Afin de simplifier la communication avec la base de données, qui est en SQLite dans ce projet, un **ORM** (**Object-Relational Mapping**) est utilisé. Les **ORM** permettent de communiquer avec des bases de données en passant par des objets. Elles permettent ainsi de ne pas avoir besoin de formuler de requête en SQL et de continuer à utiliser le langage de programmation orientée objet choisi. L'**ORM** permet donc de traduire les requêtes qui sont en TypeScript en SQL pour les transmettre à la base de données. Celui qui a été choisi est Prisma. Cet **ORM** offre trois outils principaux : **Prisma Schema**, **Prisma Migrate**, **Prisma Client**.
**Prisma Schema** est un fichier avec l'extension .prisma qui permet de décrire notre base de données, les différentes tables qui la composent ainsi que les relations qui existent entre elles.

**Prisma Migrate** permet de créer et de modifier la base de données en fonction des changements que l'on peut faire dans schema.prisma. Dans notre cas, la base de données n'est pas déposée sur GitHub, elle reste en local pour éviter que les données qui sont ajoutées pour tester soient écrasées par les push des unes et des autres. Par conséquent, cette fonctionnalité est très utile car elle nous permet de toujours avoir la même structure de base de données sans avoir besoin de la changer manuellement.
**Prisma Client**, quant à lui, joue un rôle très important, c'est lui qui va générer les méthodes qui vont permettre de requêter la base de données sans avoir besoin de passer en SQL. On retrouve l'appel à ces méthodes dans les fichiers **service**. Par exemple, on peut appeler la méthode findMany() définie par **Prisma Client** qui va nous renvoyer toutes les lignes d'une table de la base de données. Il existe un grand nombre de méthodes qui sont auto-générées par **Prisma Client**.

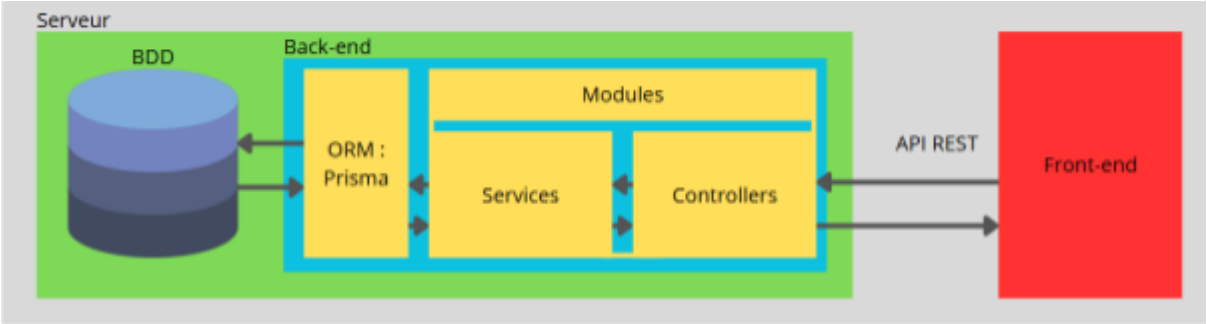


Figure 13. Schéma récapitulatif de l'application web

La liste de tous les endpoints qui sont utilisés dans l'interface est détaillée dans le fichier "Tableau des endpoints" disponible dans la documentation. De plus une fois le serveur lancé, il est possible de retrouver la document de l'API à l'URL suivant : "http://localhost:3000/api". Il existe des endpoints qui ne sont pas utilisés dans le front-end, généralement des endpoints simples qui servaient pour faire des tests au début du développement.

## 2.4. ADE

Comme dit précédemment, les données fournies par ADE dans notre projet sont celles de l'année 2023-2024, avec le nom de professeur.e.s anonymisé.e.s. Ce ne sont pas les données complètes, le projet web n'est pas à jour avec le vrai ADE.
Il faudra alors se contraindre à des normes et protocoles spécifiques définis par les responsables techniques du projet, notamment en ce qui concerne la conformité aux règles d'accès etc. Les personnes à contacter pour avoir l'api et la documentation d'ADE sont Amandine Bourgey (amandine.bourgey@univ-grenoble-alpes.fr), Lysiane Bais (lysiane.bais@univ-grenoble-alpes.fr),
Catherine Parent (catherine.parent@univ-grenoble-alpes.fr)
et Jean-François Terpan (jean-francois.terpan@grenet.fr).
Pour se servir de l'api web;
Tous les liens doivent être mis dans un navigateur (Google Chrome, Safari, Firefox…).
1) Le lien de connexion est fourni par les représentant.e.s ADE. En mettant le lien de connexion dans le navigateur, cela va renvoyer le numéro de session qui sera utile pour récupérer les données.
2) Il faut indiquer le numéro de projet reçu par les représentant.e.s d'ADE dans le lien de

récupération de données.
Ce lien ressemble à cela :
lienADEsessionId=NUMEROSESSIONfunction=setProject&projectId=NUMEROPROJET
Avant de pouvoir commencer à récupérer les données, il faut “activer” le projet. Pour activer le projet, il faut écrire cela :
lienADEsessionId=NUMEROSESSIONfunction=setProject&projectId=NUMEROPROJET
NB : la récupération et l’activation du projet se fait dans le code du projet automatiquement à chaque récupération de données.
3) Pour récupérer les données, il faut changer la function=setProject.
Ceux utilisés dans le projet sont getEvents et getActivities.
getEvents renvoie les activity_id qui sont des id associés aux cours. Il existe date=MM/DD/YYYY, qui renvoie seulement les données de ce jour précis.
par exemple :
lienADEsessionId=NUMEROSESSIONfunction=getEvents&date=MM/DD/YYYY
getActivities renvoie des fichiers.
Voici comment sont organisés les fichiers :
année de promo L1/L2/L3 -> S1/S2 -> UE -> Cours -> CM/TD/TP
Dans les fichiers des cours, il y a l'activity id qui est associé au créneau.
Les données sont déjà récupérées dans le code, avec une recherche récursive dans tous les fichiers.
Attention, si il y a écrit rw dans l’URL d’ADE, ils ne seront accessibles seulement sur eduroam. Demander une solution pour cela aux représentant.e.s ADE.

## 2.5 GitHub

Le répertoire GitHub contient le code du front-end dans le dossier GestionAbsenceFront et le code du back-end dans le dossier GestionAbsenceBack. De plus, il contient aussi la documentation du projet dans le dossier documentation.
Enfin les commandes pour initialiser le projet sont dans le read.me, elles sont à effectuer dans le terminal VSCode (ou autre en fonction de l’IDE choisi). Il faut démarrer en premier le serveur (GestionAbsenceBack) puis l’interface (GestionAbsenceFront). Il est possible d'ouvrir simultanément deux terminaux pour faire les commandes nécessaires.
Le code a été commenté pour faciliter la reprise du projet.

# 3. Problème à résoudre

Il reste un problème de l’ordre du développement pour rendre le projet viable et un problème de l’ordre plus de l'organisation.
Le problème de l’ordre du développement touche au fait qu’il est fréquent que des étudiant.e.s ne se rendent pas à leur créneau mais à un autre créneau du même cours. Dans le travail qui a été réalisé, l’appel ne prend pas en compte cette problématique. Un.e étudiant.e ne peut être présent.e qu'à son créneau de cours, même s'il/elle se présente à un autre il/elle ne pourra pas être enregistré.e comme présent.e.
Une solution pour résoudre ce problème repose sur deux choses ; la première est de retenir dans la base de données aussi les informations des présences et pas simplement des absences en ajoutant un booléen dans la table des présences pour indiquer si l’étudiant.e est présent.e ou absent.e. La seconde est de faire ressortir dans le nom des créneaux, pour

les cours qui ont plusieurs répétitions (ex : TD) le numéro de séance (ex : TD1 statistique 3 NumSéance 1). Ainsi lors de l'enregistrement de l'appel fait par un.e professeur.e il sera possible dans la base de données de rechercher les présences et absences associées aux créneaux de la même matière et du même numéro de séance pour correctement renseigner les présences des étudiant.e.s peu importe que ce soit sur le créneau de leur groupe ou non. Il faut aussi ajouter dans la page d'appel, la possibilité de sélectionner la présence des étudiant.e.s qui ne sont pas dans ce groupe mais dans le même semestre. Par contre cette solution implique de retenir significativement plus d'informations dans la base de données.
Enfin le problème de l'ordre de l'organisation vient du fait que les données d'ADE soient soumises à des réglementation RGPD, il n'est donc pas possible d'accéder aux données sans que l'application ne respecte certaines contraintes que l'on ne connaît pas. De plus, dans la continuité de cette problématique, le déploiement de l'application reste en suspens quant à la faisabilité de l'intégration de l'application au système informatique de l'UGA. Durant le projet nous ne nous sommes pas posées la question en détail mais les responsables d'ADE nous ont rapidement expliqué qu'il fallait répondre à certaines normes.

# 4. Améliorations

## 4.1 Graphique

Gestion de la vidange de la base de données : Sur la page ManagementStudent, il y a deux boutons. L'un pour refresh la base de données avec les données et l'autre pour vider la base de données pour ce qui concerne les étudiant.e.s, les groupes, les inscriptions, les présences et les créneaux. Il est possible de faire ces deux actions avec un seul bouton qui permettrait de vider l'intégralité de la base de données avant d'y injecter les données d'ADE. Ce serait plus simple pour les gestionnaires de scolarité de n'avoir qu'un seul bouton de remise à jour de la base de données au début d'un nouveau semestre.

## 4.2 Fonctionnalité

Création et remplissage automatique du groupe Promotion : Sur la page ManagementStudent, faire en sorte que lorsqu'un fichier .csv renseignant les étudiant.e.s inscrit.e.s à un semestre est envoyé, créer le groupe "Promotion" associé à ce semestre et y inscrire les étudiant.e.s contenu.e.s dans le fichier .csv. (Les endpoints nécessaires à cette amélioration existent déjà)

Ajout d'un bouton de suppression d'étudiant.e.s : Dans la page ModificationStudent, ajouter un bouton permettant de supprimer de la base de données l'étudiant.e en question. Le bouton ne doit apparaître que dans le cas où l'on veut modifier un.e étudiant.e pas dans le cas où on va en ajouter un.e. (Les endpoints nécessaires à cette amélioration existent déjà)

Ajout de la possibilité de supprimer un groupe ou de modifier le nom : Dans l'application, il n'existe pas de moyens de supprimer ou de modifier le nom d'un groupe en cas de mauvaise saisie au moment de sa création. Cette amélioration viserait donc à donner ses fonctionnalités.

## 4.3 Base de données

Éliminer la redondance des données de la table session_type : Les données de cette table sont redondantes (plusieurs fois TD, TP, etc.). Pour supprimer ces redondances, il faudrait renommer la table session_type par session et ajouter une nouvelle table session_type qui ne contiendrait que les noms des types des séances possibles (TD, TP, CM, etc.). Enfin, dans la table nouvellement appelée session on y retrouverait deux clés étrangères, une pour l'id des matières comme c'est le cas actuellement et une pour l'id du type de session (la nouvelle table). La table session pourra être liée au créneaux en fonction de ces deux clés étrangères.

# 5. Annexe

## 5.1 Lien GitHub

https://github.com/CaroleLeCrac/Projet-Info-2024.git

## 5.2 Lien des documentations des outils

Vue.js : https://vuejs.org/
NestJS : https://nestjs.com/
Prisma NestJS : https://docs.nestjs.com/recipes/prisma

## 5.3 Schéma de la base de données