COMPTE RENDU

Rapport d'Alternance

Engagement de non-plagiat

Je, soussigné(e), Nathan Badoual.....

déclare être pleinement conscient(e) que le plagiat de documents ou d'une partie d'un document publiés sur toutes formes de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour écrire ce rapport ou mémoire.

Nom-Prénom : Badoual Nathan.....

Cet engagement de non-plagiat doit être inséré en première page de tous les rapports, dossiers, mémoires.

Université d'Angers

Informatique

Deuxième année d'alternance
Au Département du Maine-et-Loire

BADOUAL Nathan

Période d'alternance du 29 Aout 2022 au 31 Août 2024

Tuteur pédagogique: LARDEUX Frédéric
Tuteur professionnel: PITHON Denis

Année Universitaire 2023 – 2024

Remerciements

Je tenais tout d'abord à remercier M. Denis PITHON qui m'a suivi tout au long de cette alternance mais également mes collègues de bureaux M. Styve JAUMOTTE, M. Axel BAUDIN, M. Jérôme LARDEUX et M. Sébastien COGNAC qui ont permis de passer ces années bien entouré et avec une bonne ambiance.

Je tiens à remercier M. Gérard PHILIPPE, notre chef de service pour son accueil et sa fantaisie.

Je remercierai également M. Matéo GRIMAUD pour sa présence et son amitié.

Je tenais enfin à remercier M. Frédéric LARDEUX mon tuteur pédagogique pour son intérêt et sa disponibilité durant cette période.

Table des matières

1 - Introduction

Durant mon Master, j'ai pu effectuer une alternance au sein du Département de Maine-et-Loire sur le site du Département au 14 boulevard Lavoisier. Elle a débuté le 29 Août 2022 et a une durée de deux ans.

J'ai plus spécifiquement collaboré avec Denis PITHON, mon tuteur d'alternance dans le Département Logistique et Systèmes d'Informations dans l'unité de Système de Production dont il est le responsable.

Ce rapport décrit les différentes missions qu'il m'a été donné de faire, dans le but de me former et de prendre en main les outils informatiques qui m'ont été utiles par la suite.

Initialement, mon sujet d'alternance était de développer un interface Web pour gérer un bus de données qui s'occupe d'échanger de manière fiable des données entre différents serveurs, qu'ils soient internes ou externes. Mais en raison d'un changement de priorité au sein du service et avec l'accord de l'université, mon sujet d'alternance a changé.

Dès lors, mon projet d'alternance a été de reprendre en main et d'interfacer un outil de gestion de droits de bases de données développé en interne, nommé Privgres, avec une API REST.

Dans ce rapport, je commencerai par présenter brièvement l'entreprise dans laquelle j'ai travaillé puis j'expliquerai les premières missions qui avaient pour but de me former. J'expliquerai l'utilité de l'application sur laquelle je travaille et comment j'ai pu la prendre en main. Ensuite, j'expliquerai sa mise en place et la façon dont elle fonctionne, et comment j'ai pu l'interfacer.

Pour finir, je résumerai le travail de mes deux années au sein de l'entreprise et je montrerai les compétences et les connaissances que j'ai pu acquérir, tant bien sur le plan personnel que professionnel.

2 - Présentation de l'entreprise

2.1 - Département de Maine-et-Loire

J'ai réalisé mon alternance au sein de la direction logistique et système d'information (DLSI) du département de Maine-et-Loire et plus précisément dans le service Systèmes, Intervention et Architecture.

Le département est une collectivité territoriale composée de plus de 2800 agents territoriaux, qui appliquent les décisions prises par 42 élus. Ils assurent la mise en œuvre des politiques votées par les conseillers élus du département.

Il y a trois grandes directions, la première, direction des territoires, celle-ci regroupe différentes directions :

- La direction des routes départementales
- La direction de l'éducation de la jeunesse et des sports
- La direction ingénierie territoriale et environnement
- La direction de la culture et du patrimoine
- La direction des archives départementales
- La direction insertion
- La direction habitat et logement

La deuxième direction, organisation et ressource, regroupe d'autres directions :

- La direction de l'assemblée et de l'administration générale,
- La direction du patrimoine immobilier,
- La direction des ressources humaines,
- La direction stratégie, organisation et projets
- La direction logistique et systèmes d'information, celle où j'effectue mon alternance

La direction développement social et solidarité regroupe :

- La mission prévention de la perte d'autonomie
- La maison départementale d'autonomie
- La direction de l'offre d'accueil pour l'autonomie,
- La direction de l'action sociale territoriale
- La direction de l'enfance et de la famille

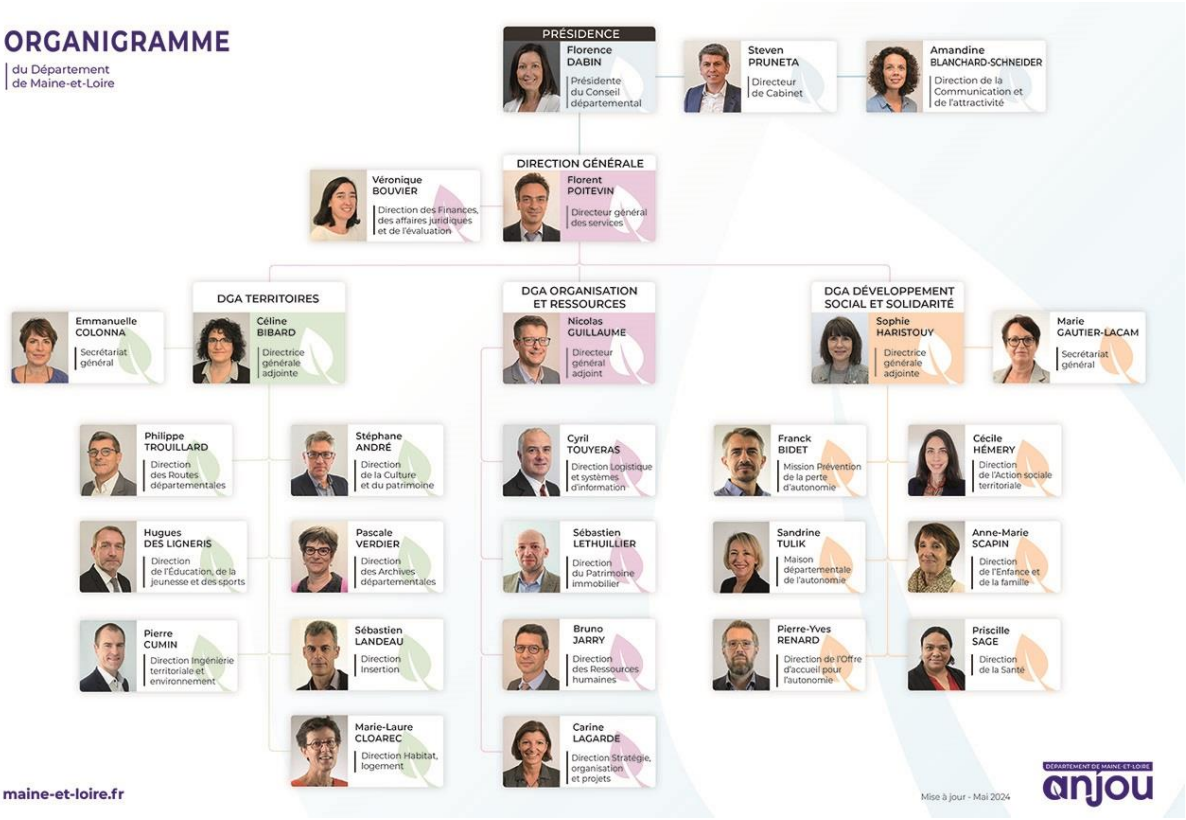

Figure 1 : Organigramme des différentes directions

2.2 - Direction Logistique et Systèmes d'Information

La DLSI est composée de six services, le service usages numériques et internet, le service projets logiciels, le service réseau, sécurité et télécom, le service logistique, le service relations usagers et celui où je réalise mon alternance, le service système, intervention et architecture.

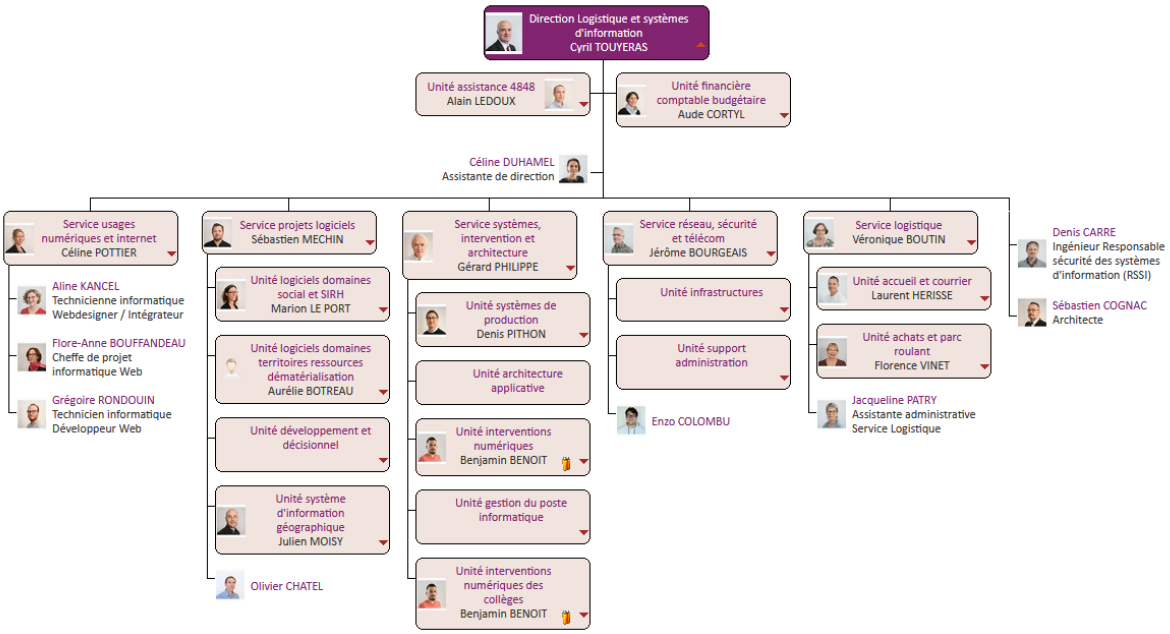

Figure 2 : Organigramme de la Direction Logistique et Systèmes d'Information

Le responsable de ce service est M. Gérard Philippe, il encadre 5 unités qui sont celle de l'architecture applicative, celle des interventions numériques, l'unité gestion du poste informatique, celle des interventions numériques des collèges et enfin, celle où je travaille, l'unité systèmes de production dont le responsable est Denis PITHON qui est également mon tuteur d'alternance. Cette unité a pour but d'administrer les serveurs du département.

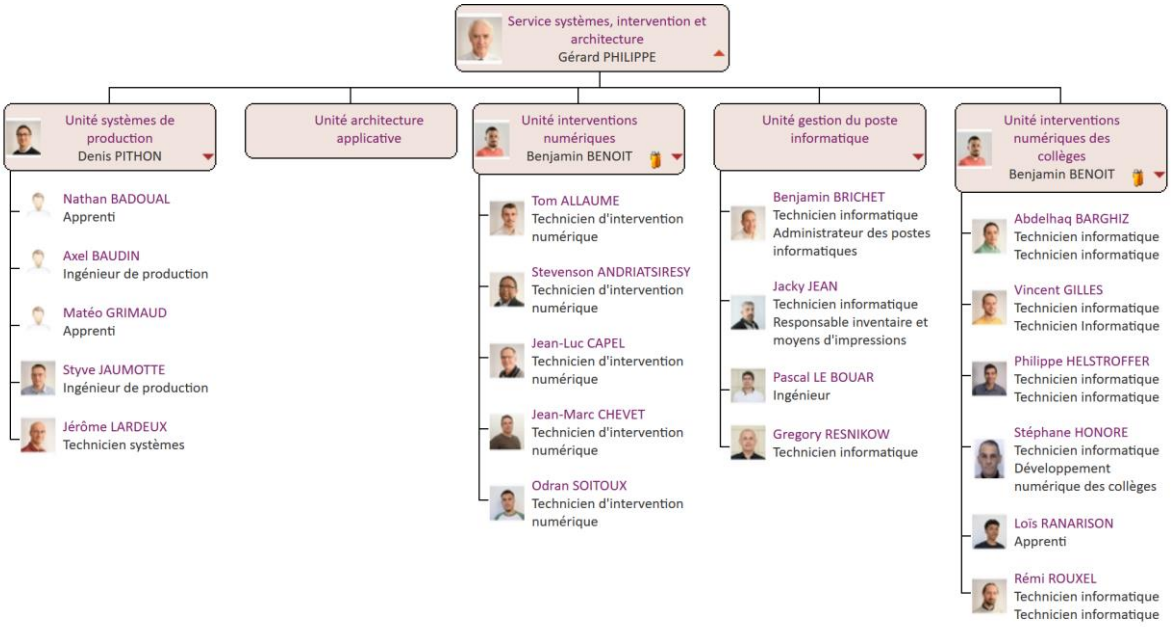

Figure 3 : Organigramme du Service Systèmes, Intervention et Architecture

3 – Mes missions

3.1 – Apprentissage

Durant ma première année d'alternance, j'avais en premier lieu été chargé de créer une interface web pour un bus de données permettant l'acheminement de données entre différentes applications. Pour me familiariser avec Linux et les outils associés, j'ai donc suivi des didacticiels et effectué diverses missions de prise en main de services tels qu'Apache, MariaDB et Postfix. J'ai également revu les principales commandes Linux, comme la navigation dans le système de fichiers, la gestion des permissions, la manipulation de fichiers et de répertoires, etc.

Pour ma première mission, j'ai réalisé un script shell qui transmet un fichier par mail lorsqu'il apparaît dans un répertoire donné, puis le supprime de ce répertoire. Cette tâche m'a permis de mettre en place et configurer le client mail de Linux et d'utiliser Inotify, un mécanisme de Linux fournissant des notifications concernant les fichiers. J'ai également transformé la commande en deamon pour qu'elle s'exécute en arrière-plan. Cette mission m'a permis de comprendre comment fonctionne la surveillance de répertoires et l'envoi automatique de mails en utilisant des scripts shell.

D'autres missions de prise en main m'ont été confiées, comme monter un disque dur virtuel de grande taille (500 Po) pour prendre en main certaines commandes utilisées dans l'unité. Cette tâche m'a permis d'apprendre à manipuler les disques durs virtuels, à les partitionner et à les monter dans le système de fichiers Linux.

Pour faciliter la prise en main de mon futur projet, on m'a demandé de me former sur Poetry, un outil open source de gestion des dépendances, des packagings et des environnements virtuels en Python. J'ai également dû me familiariser avec GitLab, un outil de versionning et de travail collaboratif. J'ai ainsi développé un projet basique en Python utilisant Poetry et l'ai importé sur le GitLab de l'entreprise. Ce projet permettait de générer des mots de passe pour un site donné et de les enregistrer dans un fichier pour chaque nom de site et nom d'utilisateur donné. J'ai également appris à utiliser des bibliothèques externes en Python, comme la bibliothèque "secrets" pour générer des mots de passe sécurisés.

Ensuite, j'ai appris à utiliser Flask, un framework open-source en Python pour le développement web, et j'ai créé un site web pour mon projet de génération de mots de passe. J'ai également appris à utiliser des templates et des bases de données avec Flask. J'ai utilisé le langage de template Jinja2 pour créer des modèles de pages web dynamiques, et j'ai utilisé la bibliothèque SQLAlchemy pour interagir avec une base de données SQLite. J'ai ainsi pu stocker les mots de passe générés dans une base de données et les afficher sur le site web.

3.2 – Changement de mission

Ma mission principale s'est alors réorientée vers l'interfaçage d'une application avec une API REST permettant de lire et modifier un fichier YAML présent dans un serveur, en suivant la logique du logiciel Privgres, un logiciel interne utilisable en ligne de commande et seulement présent sur un poste. J'ai étudié les API REST et leur fonctionnement, comme les méthodes HTTP (GET, POST, PUT et DELETE principalement), les codes de statut HTTP, les formats de données (JSON, YAML, etc.) et leur façon générale de fonctionnement (stateless…). J'ai également appris à utiliser la bibliothèque Requests en Python pour interagir avec une API REST. J'ai créé plusieurs versions d'interface web pour faciliter la gestion des droits des utilisateurs sur les bases de données.

J'ai alors travaillé sur la prise en main de l'outil Privgres, développé par Denis PITHON, servant à gérer les droits sur une base de données PostgreSQL de manière plus efficace et sécurisée. Privgres utilise un fichier YAML définissant les droits sur une base de données afin de modéliser des commandes SQL. J'ai réalisé des tests et vérifié l'attribution de droits pour éviter des problèmes d'accès. J'ai également appris à utiliser PostgreSQL, un système de gestion de base de données relationnelle, et à interagir avec une base de données PostgreSQL en utilisant des commandes SQL.

3.3 - Organisation de l'API REST

Pour la mise en place de l'API REST, j'ai utilisé le framework Flask en Python, qui est un micro-framework léger et facile à utiliser, permettant entre autre de créer des API REST. J'ai également prévu d'utiliser la bibliothèque Psycopg pour interagir avec la base de données PostgreSQL, qui sera utilisée pour certaines données de l'API.

J'avais initialement prévu d'organiser l'API REST autour des ressources suivantes :

- Les utilisateurs : il s'agit des personnes qui utiliseront l'API pour gérer leurs droits sur les bases de données. Les utilisateurs pourront se connecter et se déconnecter de l'API, et gérer leurs informations personnelles et certaines autres en fonction des droits accordés.
- Les bases de données : il s'agit des bases de données PostgreSQL auxquelles les utilisateurs qui en ont le droit auront accès. Les utilisateurs pourront créer, lire, mettre à jour et supprimer des bases de données.
- Les droits : il s'agit des autorisations que les utilisateurs ont sur les bases de données. Les utilisateurs pourront donner le droit de modification de leurs bases de données à d'autres utilisateurs.

J'ai prévu d'utiliser les méthodes HTTP suivantes pour interagir avec les ressources :

- GET : pour récupérer des informations sur une ressource.
- POST : pour créer une nouvelle ressource.
- PUT : pour mettre à jour une ressource existante ou la créer.
- DELETE : pour supprimer une ressource existante.

J'ai également prévu d'utiliser des codes de statut HTTP pour indiquer le résultat des requêtes effectuées sur les ressources. Par exemple, un code de statut 200 indiquera que la requête a réussi, tandis qu'un code de statut 404 indiquera que la ressource demandée n'a pas été trouvée.

3.4 - Préparation

Pour me préparer à la mise en place de l'API REST, il m'a été demandé d'effectuer plusieurs entraînements :

- J'ai suivi des tutoriels en ligne sur la création d'API REST avec Flask.
- J'ai créé une API REST basique pour gérer une liste d'identifiants. Cette API me permettait de créer, lire, mettre à jour et supprimer des tâches, et d'utiliser des codes de statut HTTP pour indiquer le résultat des requêtes.
- J'ai également créé une application web simple permettant l'interfaçage de l'API REST afin d'afficher la liste des tâches et permettre aux utilisateurs d'ajouter, de modifier et de supprimer des identifiants.

Ces entraînements m'ont permis de me familiariser avec les concepts de base de la création d'API REST, d'apprendre à utiliser Flask pour créer des API, et de tester la fonctionnalité et la fiabilité de mes API. Je suis maintenant prêt à mettre en place une API REST plus complexe pour gérer les droits des utilisateurs sur les bases de données PostgreSQL.

Voici une liste plus détaillée des fonctionnalités que je prévoyais de mettre en place dans l'API REST à la fin des entrainements :

- Authentification et autorisation : je prévois d'utiliser le protocole OAuth2 pour authentifier et autoriser les utilisateurs à accéder aux ressources de l'API. Les utilisateurs devront s'authentifier en utilisant leur nom d'utilisateur et leur mot de passe, et les autorisations seront gérées en utilisant des jetons d'accès.
- Gestion des utilisateurs : les utilisateurs pourront créer leur compte, se connecter et se déconnecter de l'API, et gérer leurs informations personnelles. Ils ne pourront pas réinitialiser leur mot de passe en cas d'oubli

étant donné que la gestion des mots de passes se fera sur l'AD de l'entreprise.

- Gestion des bases de données : les utilisateurs pourront créer, lire, mettre à jour et supprimer des bases de données.
- Gestion des droits : les utilisateurs pourront créer, lire, mettre à jour et supprimer des droits sur les bases de données. Ils pourront également gérer les rôles et les autorisations associés à chaque droit.
- Gestion des erreurs : je prévois de gérer les erreurs en utilisant des codes de statut HTTP appropriés et en renvoyant des messages d'erreur détaillés aux utilisateurs.

Ainsi, sur ma seconde année d'alternance, je devais créer un cahier des charges pour la mise en place d'une API REST permettant d'accéder et gérer les droits sur les bases de données auxquelles nous avons accès. Cette documentation me permettrai de développer plus facilement en suivant les normes paramétrées. Je devais également mettre en place une authentification sécurisée et prévoir une façon d'utiliser l'API, à la manière de la commande curl sur Linux avec une application simplifiée en ligne de commandes. Il m'a aussi été demandé de créer une interface web accessible à tous les utilisateurs, en suivant la charte graphique et certaines normes graphiques du web si j'en avais le temps.

4 – Deuxième année

4.1 – API REST

Durant cette seconde année d'alternance, j'ai commencé par reprendre et revoir l'organisation de l'application. J'ai ainsi pu mettre en place une documentation sur l'application Privgres que je compte développer. Cette documentation disponible en Annexe 2 permettra une prise en main simple de l'API dans le cas de la création d'une application l'utilisant par exemple. Elle m'a également permis de structurer mon API afin de pouvoir programmer par la suite les différentes fonctionnalités avec leurs paramètres respectifs, leurs retours et leurs erreurs au besoin.

Suite à différents échanges j'ai ainsi pu la finaliser et la baser sur deux points principaux, la gestion des droits sur l'applications et la gestion des droits des bases de données PostgreSQL.

Cette API REST est programmée en Python à l'aide des framework Flask pour le développement web, JWT pour la création et la gestion des jetons d'authentification, de Psycopg2 pour l'interfaçage entre les bases de données et Python et de ConfigParser pour la lecture du fichier de configuration.

Gestion application

Le projet de l'application Privgres est partie de l'idée de respecter le principe de moindre privilège qui est je le rappelle un principe de sécurité où seuls les droits nécessaires pour effectuer ce qu'il a à faire sont accordés à l'utilisateur, c'est-à-dire de faire en sorte que seuls les utilisateurs qui en ont l'utilité, peuvent modifier les droits nécessaires d'un utilisateur ou une base de données spécifique. Ainsi, il nous a semblé nécessaire d'appliquer ce même principe sur l'API en développement, c'est pour cela que nous avons opté pour une application avec différents droits pour chaque utilisateur donné.

À l'origine, l'application gérait les droits via différents fichiers YAML, mais cela pouvait poser des problèmes en cas de modifications parallèles. Deux utilisateurs pouvaient accéder à la même donnée, effectuer les modifications et une fois envoyé à l'API, seul la dernière envoyée sera retenue.

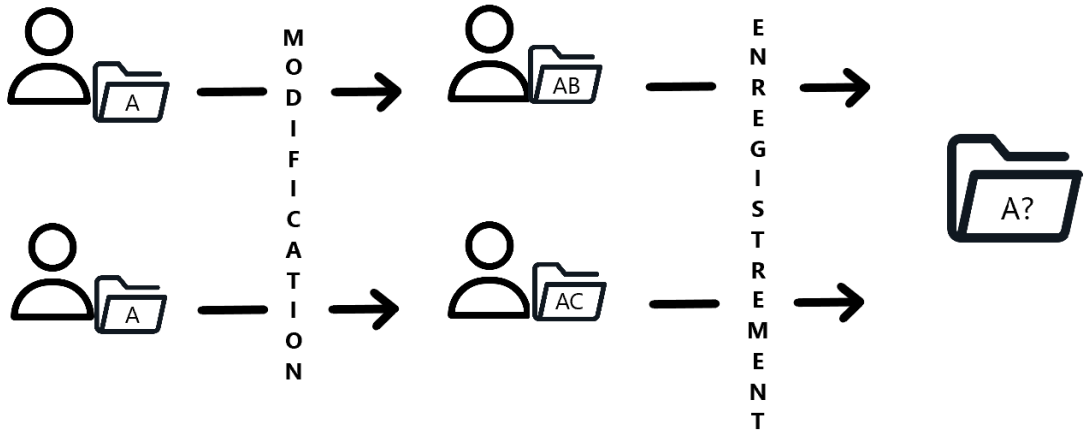

Figure 4 : Exemple d'une double modification d'une base de données provoquant un résultat aléatoire

Pour résoudre ce problème, l'API possède désormais sa propre base de données pour gérer les utilisateurs et les bases de données associées. Cette base est gérée en Python en utilisant la bibliothèque Psycopg, qui est la plus populaire pour interfacer PostgreSQL et Python.

Connexion

Tout d'abord, la connexion est la partie initiale de l'API. Rien ne peut être fait sans s'être authentifié auprès de l'application. Afin de créer un compte, une première connexion est nécessaire. A des fins d'organisation, il fallait départager quels utilisateurs devraient pouvoir utiliser l'application, et qui parmi ceux-là pourraient en gérer les droits.

Pour faire cela, lors de l'identification, nous allons utiliser une connexion par l'Active Directory (AD) de l'entreprise. Ceci permet de ne stocker aucuns identifiants et de seulement les transférer pour vérifier qu'ils sont valides. Nous allons commencer par nous connecter au serveur de l'Active Directory en paramétrant une connexion en LDAPS avec le protocole de sécurité de l'entreprise. Nous devons utiliser la connexion en LDAPS et pas seulement en LDAP car pour respecter les normes de sécurité, on doit protéger les identifiants de la personne qui se connecte. Une fois la connexion confirmée, on va vérifier que l'utilisateur appartient au groupe nécessaire à la connexion à l'application. En procédant ainsi par étape, il est très facile de gérer les différentes erreurs de connexion, comme des mauvais identifiants ou encore le fait que l'utilisateur n'appartienne pas au groupe nécessaire à la connexion à l'application. Une fois l'utilisateur connecté à l'API, elle va vérifier s'il a déjà un compte sur sa base de données et si non, va lui en créer un avec les droits de base, donc la possibilité de créer des bases de données.

Chaque utilisateur connecté sur un appareil aura un jeton d'authentification associé. Un jeton d'authentification est une suite de caractère permettant à l'application, lorsqu'elle le décrypte de la bonne façon, de savoir qui est l'utilisateur et si son temps d'expiration n'est pas dépassé. La date d'expiration de ce jeton sera mise à jour à chaque commande effectuée par l'utilisateur, ainsi le jeton changera à chaque fois également. On appelle ce genre de gestion de jeton, une expiration glissante, permettant de réduire la durée d'expiration en comparaison à une expiration absolue qui consiste à créer un seul jeton d'authentification lors de la connexion mais qui deviendra forcément invalide après une certaine durée. Pour créer ce jeton, j'ai utilisé la bibliothèque « jwt » de Python.

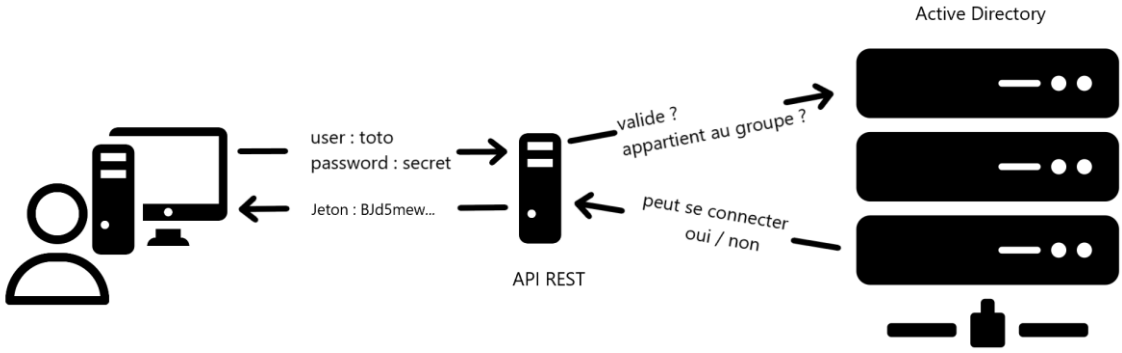

Figure 5 : Schéma de la connexion de l'API REST utilisant l'Active Directory

Rôle « user »

Une fois la connexion effectuée, l'application va donc commencer par chercher le rôle de l'utilisateur connecté. Si l'utilisateur se connecte pour la première fois, l'application va lui créer un champ avec son nom d'utilisateur et un rôle basique, celui de « user ». Si l'utilisateur s'est déjà connecté et qu'il n'a aucun droit spécifique, il aura également ce rôle.

Un utilisateur ayant ce rôle aura la possibilité de créer sa propre base de données et de la modifier également. Il sera ainsi le propriétaire, au sens applicatif, de cette base et ainsi pourra donner les droits de modification associés à n'importe quel utilisateur. Il aura également la possibilité de révoquer ce droit sur les bases qu'il possède.

Comme nous venons de le voir, un utilisateur pourra ainsi également modifier n'importe quelle base si le propriétaire de la base lui en a donné les droits.

En résumé, un utilisateur peut créer des bases de données, et modifier seulement celles sur lesquelles il a les droits.

Rôle « admin »

Nous venons de voir le rôle « user » et, afin de pouvoir superviser tous ces utilisateurs classiques, il y aura également le rôle « admin » qui permettra l'administration des droits et des bases de l'application.

Cet administrateur aura les mêmes droits de base qu'un utilisateur avec le rôle « user » mais en plus aura la possibilité de modifier n'importe quelle base de données sans avoir reçu les droits de son propriétaire. Il pourra également révoquer ces droits.

Un administrateur aura la possibilité de rajouter un utilisateur en tant qu'administrateur ou de rétrograder un administrateur au rôle d'utilisateur.

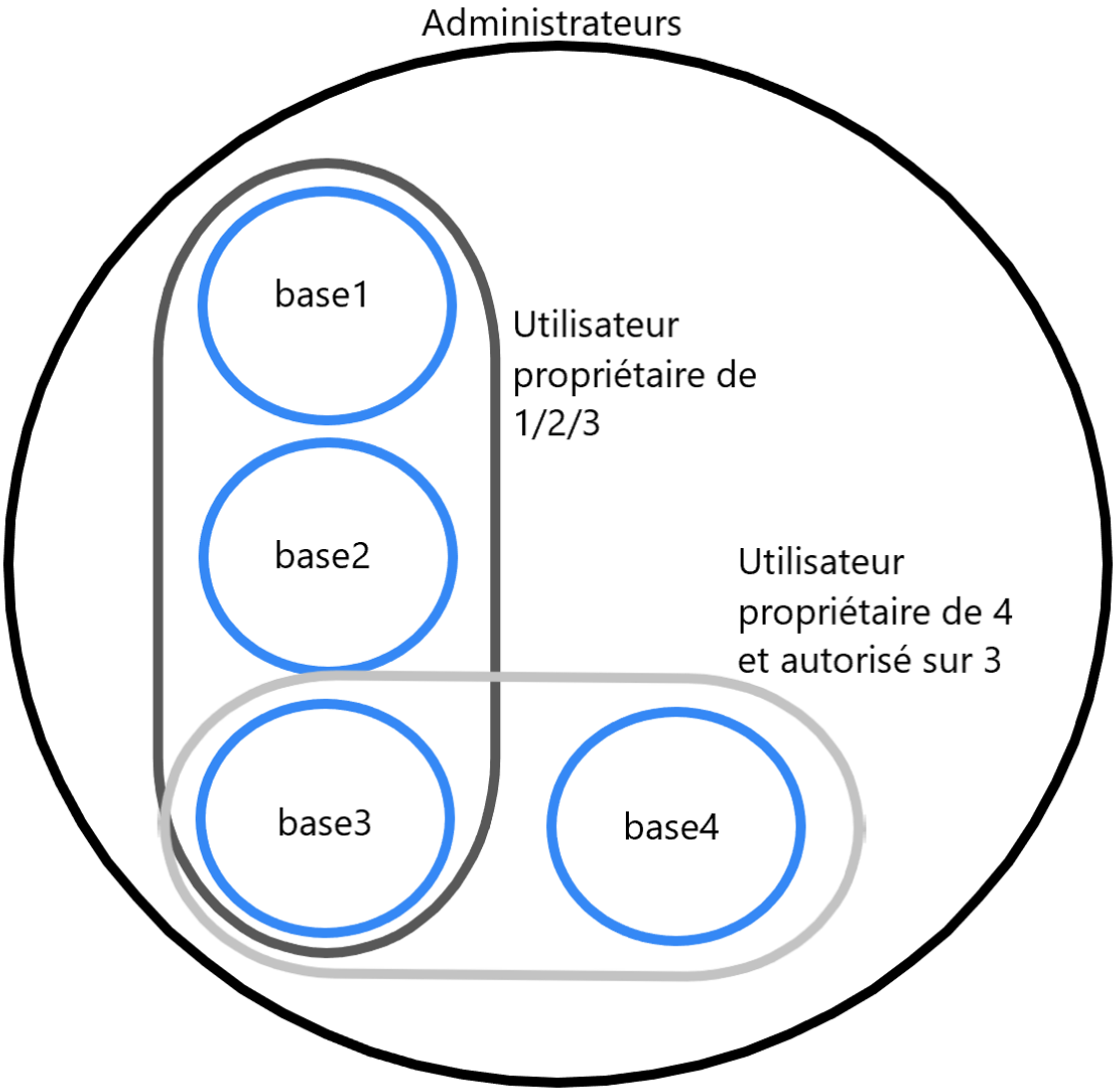

Figure 6 : Schémas d'exemples de droits sur les bases

Gestions bases

Une fois les utilisateurs partitionnés et la modification des bases organisée, nous avons pu mettre en place la gestion des bases de données. L'application va garder les droits sur les bases de données sous forme de fichiers YAML avec différentes parties dans chaque fichier. La documentation va décrire la façon dont pourront être modifié les fichiers YAML.

Format des fichiers

Chaque fichier yaml sera composé de 3 ou 4 parties. La partie « base », la partie « schemas », puis soit la partie « applications », soit la partie « users », ou les deux.

- La partie « base » : elle permettra de décrire les fondations de cette base de données, c'est-à-dire le nom de la base et si nécessaire l'administrateur interne à la base. S'il n'est pas précisé, il sera généré automatiquement.
- La partie « schemas » : elle décrit les différents schémas et les utilisateurs ayant les droits sur ces schémas.
- La partie « applications » ou « users » : au moins l'une des deux parties doit être présente étant donné qu'elles décrivent la création d'utilisateur, soit pour la connexion d'utilisateurs, soit pour celle d'applications, il faut au minimum un utilisateur pouvant se connecter c'est pour cela qu'il faut au moins une de ces deux parties.
    - La partie « applications » permettra de décrire des utilisateurs utilisés par des applications qui auront soit des droits de lecture, soit d'écriture sur tous les schémas directement.
    - Enfin, la partie « users » permettra de décrire des utilisateurs qui auront des droits plus précis que les comptes applicatifs. On aura par exemple leur moyen de connexion, s'ils sont interne à la base de données, auquel cas un mot de passe sera généré s'il n'est pas précisé, ou alors il sera associé à un compte utilisateur de l'entreprise où il pourra se connecter avec ses identifiants, dans ce cas son moyen d'authentification sera appelé « ldap ». On pourra préciser le rôle de l'utilisateur, il pourra être administrateur (« admin »), auquel cas il aura les droits pour gérer la base de données et ses utilisateurs, ou bien modificateur global (« global-cruder »), dans ce cas-là il aura le droit de modification sur toutes les bases, ou encore lecteur global (« global-reader »), où il aura un droit de lecture sur toutes les bases, sans avoir la possibilité de les modifier, ou enfin il sera un utilisateur classique si rien n'est précisé, il n'aura pas de base spécifique, mais on pourra lui en fournir plus précisément directement sur les schémas.

Voici un exemple d'un fichier sous le format YAML :

```yaml
base:
  name: _42

applications:
  pandora:
  arrakis:
    mode: crud

users:
  albator:
    global-reader: true
  c.flam:
    global-cruder: true
  a.skywalker:
    admin: true
    auth: ldap
  popeye:
    auth: ldap
  zorro: { auth: ldap }
  c.macleod: { auth: internal }
  goldorak: { auth: internal , password: 123abc}

schemas:
  mercure:
  venus:
  terre:
    read: [goldorak, c.macleod]
  mars:
    crud: [zorro, c.macleod]
  jupiter:
  saturne:
  uranus:
  neptune:
```

Figure 7 : Exemple de document descriptif des droits d'une base de données

Communication

Chaque élément de l'API peut être récupéré, modifié ou supprimé via des méthodes HTTP. Un tableau représentant la liste des différentes méthodes disponibles pour chaque élément est disponible plus bas en *Figure 8*. Cependant, certaines informations quant aux données récupérables ou modifiables ne sont pas représentées sur ce schéma. Dans cette section, nous allons expliquer partie par partie les méthodes utilisables.

Applicatif

Les éléments de gestions d'applications comme la connexion ou les différents type d'accords de droits sont accessibles via la méthode POST.

Pour la connexion, les identifiants sont envoyés en précisant les paramètres « username » et « password ». En cas d'authentification réussie, une réponse avec un élément « token » est renvoyée. Ce jeton a une durée de vie limitée définie dans le fichier de configuration et il doit être précisé pour les requêtes suivantes en guise d'identifiant.

Pour accorder ou révoquer des droits d'accès à une base de données, des informations sont précisées dans le corps de la requête, telles que le nom de l'utilisateur et la base concernée. Le paramètre « token » doit également être fourni pour permettre à l'API de vérifier les droits de l'utilisateur. En cas de requête réussie, la réponse contient le nouveau jeton créé. Si l'authentification échoue ou si des paramètres sont manquants, la réponse renvoie l'erreur associée.

Pour accorder ou révoquer des droits d'utilisateur, le paramètre « token » doit être fourni pour vérifier que l'utilisateur a les droits d'administrateur. En cas de requête réussie, la réponse contient le nouveau jeton créé. Si l'authentification échoue ou si des paramètres sont manquants, la réponse renvoie l'erreur associée.

Le statut est accessible via la méthode GET et permet de récupérer différentes informations sur la connexion. Le paramètre « token » doit être fourni dans la requête. La réponse contient le rôle de l'utilisateur, ses bases accessibles et celles dont il est propriétaire, ainsi que le jeton d'authentification mis à jour.

Bases

Les méthodes HTTP sont utilisées pour contacter différents éléments ou parties de la représentation en format YAML de la base de données.

Pour connaître les bases accessibles, deux choix s'offrent à nous : utiliser l'endpoint « status » vu précédemment ou la méthode GET sur l'URI « /bases ». Le paramètre « token » doit être fourni. Cet endpoint permet également la création de nouvelles bases en utilisant la méthode POST et en fournissant des données en format YAML à importer dans la nouvelle base de données. En cas de réussite, la réponse contient le nouveau jeton créé. En cas d'erreur, la réponse renvoie l'erreur associée.

Pour récupérer une base spécifique, une requête avec la méthode GET peut être effectuée sur un endpoint de ce type « /bases/<base> ». Le paramètre « Accept » peut être utilisé pour demander différents formats de réponse : YAML, JSON ou « text/x-sql » pour obtenir le code PostgreSQL correspondant à la création de la base. Le paramètre « token » doit être fourni. La méthode DELETE peut également être utilisée pour supprimer la base spécifiée. Dans ce cas, la réponse contient le jeton mis à jour ou l'erreur associée.

On va pouvoir également récupérer les informations de bases du fichier YAML comme le nom du propriétaire de la base avec la méthode GET sur l'endpoint « /bases/<base>/base ». Le paramètre « Accept » peut être utilisé pour demander différents formats de réponse : YAML ou JSON. Le paramètre « token » doit être fourni.

Il est possible de récupérer la liste des applications, des utilisateurs ou des schémas, une requête avec la méthode GET peut être effectuée sur un endpoint de ce type dans

le cas de la liste des applications, mais qui peut être adaptée en fonction de la liste souhaitée : « /bases/<base>/apps ». Le paramètre « Accept » peut être utilisé pour demander différents formats de réponse : YAML ou JSON. Le paramètre « token » doit être fourni. La méthode POST peut également être utilisée pour ajouter un élément à la base spécifiée. Dans ce cas, la réponse contient le jeton mis à jour ou l'erreur associée.

Afin de récupérer un élément des applications, des utilisateurs ou des schémas, une requête avec la méthode GET peut être effectuée sur un endpoint de ce type dans le cas d'une application spécifique, mais qui peut être adaptée en fonction de la liste souhaitée : « /bases/<base>/apps/<app> ». Le paramètre « Accept » peut être utilisé pour demander différents formats de réponse : YAML ou JSON ou « text/x-sql » pour obtenir le code PostgreSQL correspondant à la création de l'élément. Le paramètre « token » doit être fourni, et dans ce cas, la réponse contient l'information demandée, mais également le jeton mis à jour ou l'erreur associée. La méthode PUT peut être utilisée pour ajouter ou modifier un élément à l'endpoint fourni, dans ce cas, la réponse contient le jeton mis à jour ou l'erreur associée. La méthode DELETE peut également être utilisée pour supprimer un élément à la base spécifiée. Dans ce cas également, la réponse contient le jeton mis à jour ou l'erreur associée.

Le tableau suivant représente les différentes URI accessibles avec leurs méthodes disponibles en fonction de ce qui est nécessaire :

	GET	PUT	POST	DELETE
/bases				
/bases/<base>				
/bases/<base>/base				
/bases/<base>/apps				
/bases/<base/apps/<app>				
/bases/<base/users				
/bases/<base/users/<user>				
/bases/<base>/schemas				
/bases/<base/schemas/<schema>				

/login				

/status				
/grant_admin				
/revoke_admin				
/grant				
/revoke				

4.2 – Interfaçage

L'API REST documentée et développée, une grande partie du travail était effectué. Cependant elle peut être compliquée à utiliser pour des personnes qui n'ont pas l'habitude de programmer, ou d'utiliser en brut des APIs. C'est pour cela qu'on m'a également demandé d'interfacer cette API une interface en ligne de commande (CLI) afin de pouvoir effectuer des requêtes rapidement ou automatiser certains scripts plus facilement.

Command Line Interface

Afin de créer cette interface en Python, j'ai opté sur Click, une bibliothèque qui pourrait facilement être reprise par Denis Pithon et qui permet une mise en place intuitive de ce genre d'interface. Un tutoriel pour l'installation a été fait pour simplifier son utilisation. Après initialisation, cette interface est utilisable avec la commande `*privgres*` suivi des commandes ou des paramètres nécessaires que nous verrons plus tard.

Connexion

Chaque requête de l'API nécessitait une connexion initiale afin d'utiliser un jeton d'authentification par la suite. Etant donné que le jeton pouvait expirer, j'ai opté pour l'utilisation d'une interface qui demandait un jeton avant chaque requête au serveur, cela n'utilisait pas les nouveaux jetons créés après chaque requête, mais cela permettait de n'avoir aucun jeton qui arrive à expiration.

Les jetons relancés après chacune des requêtes pourront avoir une utilité dans une autre interface qui lancerai plusieurs requêtes à la suite.

Pour simplifier ce système de connexion avant chaque requête et pour éviter de taper sans cesse ses identifiants, j'ai mis en place un système de connexion automatique à partir d'une configuration. L'utilisateur peut configurer un environnement nommé, il peut ainsi choisir l'URL de l'API à envoyer les requêtes, le nom d'utilisateur et le mot de passe à utiliser. Ces données seront stockées dans un fichier de configuration et pour des mesures de sécurité, les mots de passe ne seront pas stockées en clair, mais chiffrés.

A chaque requête, il sera possible de choisir son environnement si besoin, sinon la CLI utilisera l'environnement nommé « DEFAULT » automatiquement. De la même façon, le lien de l'API ne sera pas forcément connu de tous, donc s'il n'est pas précisé dans la configuration de l'environnement, ce sera celui que j'ai défini par défaut.

Commandes

La connexion étant géré automatiquement dès lors qu'un environnement a été créé, j'ai pu commencer par les commandes permettant la gestion de l'API. En utilisant la bibliothèque Click, on peut ainsi aisément paramétrer des commandes. La commande la plus simple était de demander le statut de connexion, permettant la récupération des informations basiques de l'utilisateur, tel que son rôle, les bases de données dont il est propriétaire et les bases de données qu'on lui a permis de manager. Cette commande ne nécessite aucunes autres informations que celles donnés lors de la connexion et peut se lancer de cette façon :

```
$privgres status
```

Qui remontera un objet json avec les informations précisées précédemment.

Click permet également de fournir des paramètres à la commande, ainsi si je veux donner des droits à un utilisateur sur une base que j'ai créée, je peux le faire de cette manière grâce à la commande :

```
$privgres grant-right -u toto -b nomDeLaBase
```

J'ai ainsi pu fournir le nom de l'utilisateur (toto) à qui je voulais fournir les droits, et sur quelle base de données (nomDeLaBase). Ici nous avons les paramètres écrit en court, mais cette commande est la même que précédemment :

```
$privgres grant-right --user toto --base nomDeLaBase
```

J'ai également paramétrer afin de faire en sorte que si un paramètre n'est pas donnée, il sera demandé par la suite à l'utilisateur, de cette façon, si on lance la commande sans paramètres, on obtient ceci :

```
$privgres grant-right
Base: nomDeLaBase
User: toto
```

Avec les paramètres « base » et « user » à remplir à la main.

Ainsi, la bibliothèque Click m'a permis de mettre en place l'accord et la révocation de droits sur une base de données, mais également l'accord ou la révocation de droits d'administrateur de l'API.

Une fois le coté gestion de l'API fait, il fallait mettre en place la modification des bases de données. Les droits se gérant directement sur l'API, la CLI n'avait besoin que d'envoyer les données à l'API qui allait faire son travail directement et gérer les erreurs si cela était nécessaire.

Les commandes utilisées suivaient le principe de fonctionnement de l'API, sauf pour les méthodes GET. En effet dans le cas où l'utilisateur voulait récupérer les commandes PostgreSQL associées à la base, à un utilisateur, à une application ou à un schéma, j'ai trouvé plus approprié que la commande s'appelle « generate » et non pas « get » qui est la commande associée à la récupération en format YAML ou JSON des données. Ainsi on a donc les commandes « generate-full », « generate-app », « generate-user » et « generate-schema » pour la génération en PostgreSQL des différentes parties, et ces mêmes parties en YAML ou JSON avec « get » à la place de « generate » et en plus « get-list-apps », « get-list-users » ou encore « get-list-schemas ».

Pour les méthodes PUT, j'ai utilisé la nomenclature « replace » qui me semblait plus compréhensible, ce qui a permis d'avoir les commandes « replace-app », « replace-user », « replace-schema »

5-Conclusion

D'un point de vue professionnel, ces années d'alternance m'ont beaucoup appris, cela m'a permis de m'enrichir sur le fonctionnement d'une équipe, d'une entreprise à si grande échelle, la façon dont tout s'organise, les différents aspects intrinsèques au sein d'une organisation de la sorte. J'ai également pu voir que d'autres entreprises sont organisées différemment après des échanges avec certains collègues.

D'un point de vue personnel, ma première année m'a permis de me préparer à mon futur projet, j'ai pu monter en compétences sur Linux et en Python grâce à ces différentes missions et des prises en main. J'ai appris beaucoup sur la mise en place, les bonnes pratiques, les fonctionnalités et bien d'autres encore.
Ma seconde année d'alternance m'a permis de mettre en place concrètement tout ce que j'avais appris, en commençant par la création d'une documentation facilement compréhensible de l'API qui conviendrait bien aux utilisateurs, de la mise en place de cette API et de tests effectués. De la documentation de l'application en ligne de commande qui interrogerait cette API, par quels moyens, avec quels outils, sa création mais également ses tests. J'ai ainsi pu faire un tutoriel pour que n'importe quel utilisateur qui le voudrait puisse l'utiliser de sa configuration à son utilisation.
Grâce à cette alternance, j'ai pu énormément développer mes compétences en gestion de projet, en développement Python, mais également en tests et en communication technique.

J'ai pu rencontrer des difficultés, notamment sur l'organisation de l'API, la façon dont je devais la mettre en place, mais j'ai pu les surmonter à force d'échanges avec mes collègues et de recherche. J'ai également eu du mal à l'installation de cette API sur un serveur car ce n'est pas quelque chose que j'ai pu voir, mais je suis reconnaissant d'avoir pu apprendre cette partie applicative importante.

Tout ceci m'a permis de mettre en place une solution logicielle complète, me permettant ainsi d'achever le travail demandé qui, je l'espère, saura faciliter l'utilisation de cette application.

Cette expérience m'a donné envie de poursuivre une carrière dans le développement d'application et je suis persuadé que les compétences que j'ai acquises pendant mon alternance me seront utiles pour atteindre mes objectifs professionnels.

6-Annexes

Privgres

Nathan Badoual

25 juillet 2024

1 Présentation

Privgres est une application polyvalente qui permet de gérer les droits sur des bases de données PostgreSQL de manière indirecte. Actuellement, cette application est disponible en ligne de commande (CLI) pour répondre aux besoins des différents utilisateurs (https://gitlab.cg49.fr/n.badoual/privgres-cli.git). Une interface Web intuitive pourrait être envisagée dans de futures versions.

Contrairement aux outils de gestion des droits traditionnels, Privgres ne modifie pas directement les bases de données. Au lieu de cela, il gère des fichiers YAML qui correspondent à une représentation des droits de bases de données. Ces fichiers décrivent les utilisateurs, les applications et leurs rôles sur les schémas pour les bases de données PostgreSQL.

L'un de ses principaux atouts est qu'il se base sur une API REST. Cela signifie que l'application utilise des protocoles Web standardisés pour communiquer avec les fichiers YAML, ce qui permet une grande flexibilité et une intégration facile avec d'autres systèmes.

Au sein de Privgres, la gestion des droits est effectuée de manière à ce que seuls les accès autorisés aux bases soient permis.

Avec Privgres, les administrateurs de bases de données peuvent gérer les droits aux différents utilisateurs de manière simple et intuitive.

Une fois les rajouts apportées aux fichiers YAML, Privgres peut en générer les commandes PostgreSQL. Ces commandes peuvent ensuite être exécutées manuellement pour mettre à jour les bases de données.

En somme, Privgres est un outil complet et facile à utiliser pour la gestion indirecte des droits sur les bases de données PostgreSQL qui peut vous aider à simplifier et à sécuriser la gestion des accès à vos données.

Table des matières

2 /login

2.1 Méthode : Post

2.1.1 Description

Ce point de terminaison permet de se connecter à l'application en envoyant les identifiants de l'utilisateur. En cas de succès, il renvoie un jeton d'authentification (token) qui doit être utilisé pour les requêtes ultérieures nécessitant une authentification. Il est préconisé, surtout pour la connexion, de passer en https et non en http afin que le mot de passe soit directement crypté dans le réseau. Conditions : Avoir le rôle suivant dans l'AD "GGS-APP-Privgres"

2.1.2 Paramètres

— Corps de la requête : contient deux paramètres, un « username » qui contient le nom de l'utilisateur et « password » qui contient le mot de passe de l'utilisateur.

2.1.3 Réponse

— Code de statut 200 : la connexion a réussi. Le corps de la Réponse contient un objet JSON avec le jeton d'authentification.
— Corps de la Réponse : Jeton d'authentification.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : les identifiants sont incorrects.
— Erreur 403 Forbidden : l'utilisateur n'est pas dans le groupe "GGS-APP-Privgres" dans l'Active Directory.
— Erreur 500 Internal Server Error : l'application n'arrive pas à contacter le serveur LDAP.

2.1.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
POST /login HTTP/1.1
                # body
{
    "username": "n.badoual",
    "password": "mypassword"
}
```

2.1.5 Exemple de réponse réussie

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                # headers
Content-Type: application/json
                # body
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

2.1.6 Exemple de réponse erronée

```
                # The HTTP code response and text
HTTP/1.1 401 Unauthorized
                # headers
Content-Type: application/json
                # body
{
    "error": "Connection expired",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

3 /grant_admin

3.1 Méthode : Post

3.1.1 Description

Ce point de terminaison permet d'accorder des droits d'administrateur à un utilisateur donné. Conditions : être administrateur.

3.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « user » (obligatoire) : l'utilisateur a qui les droits seront accordés.

3.1.3 Réponse

— Code de statut 200 : la transmission de droit est effectuée.
— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur demandeur n'a pas les droits ou l'utilisateur est déjà administrateur.

3.1.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
POST /grant_admin HTTP/1.1
                # headers
Authorization : Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                # body
{
    "user" : "nameNewAdmin"
}
```

3.1.5 Exemple de réponse réussie

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                #body
{
    "token": "eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb..."
}
```

3.1.6 Exemple de réponse erronée

```
                # The HTTP code response and text
HTTP/1.1 403 Forbidden
                #body
{
    "error": "user already have the right",
    "token": eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb..."
}
```

4 /revoke_admin

4.1 Méthode : Post

4.1.1 Description

Ce point de terminaison permet d'enlever des droits d'administrateur à un administrateur. Conditions : être administrateur

4.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « user » (obligatoire) : l'utilisateur a qui les droits d'administration seront enlevés.

4.1.3 Réponse

— Code de statut 200 : la suppression de droit est effectuée.
— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur demandeur n'a pas les droits ou l'utilisateur ciblé n'as déjà pas les droits d'administrateur.

4.1.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
POST /revoke_admin HTTP/1.1
                # headers
Authorization : Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                # body
{
    "user" : "nameRemovedAdmin"
}
```

4.1.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

5 /grant

5.1 Méthode : Post

5.1.1 Description

Ce point de terminaison permet d'accorder les droits de manager une base à un utilisateur. Conditions : avoir créé la base ou être administrateur

5.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « base » (obligatoire) : la base sur lesquels les droits seront accordés.
— « user » (obligatoire) : l'utilisateur a qui les droits seront accordés.

5.1.3 Réponse

— Code de statut 200 : la transmission de droit est effectuée.
— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur demandeur n'a pas les droits ou l'utilisateur a déjà les droits sur cette base

5.1.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
POST /grant HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                # body
{
    "base" : "baseName",
    "user" : "userName"
}
```

5.1.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

6 /revoke

6.1 Méthode : Post

6.1.1 Description

Ce point de terminaison permet d'enlever les droits de manager une base à un utilisateur. Conditions : avoir créé la base ou être administrateur

6.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « base » (obligatoire) : la base sur lesquels les droits seront enlevés.
— « user » (obligatoire) : l'utilisateur a qui les droits seront enlevés.

6.1.3 Réponse

— Code de statut 200 : la suppression de droit est effectuée.
— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur demandeur n'a pas les droits ou l'utilisateur n'a pas les droits sur cette base.

6.1.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
POST /revoke HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                # body
{
    "base" : "baseName",
    "user" : "userName"
}
```

6.1.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

7 /status

7.1 Méthode : Get

7.1.1 Description

Ce point de terminaison permet de récupérer les informations sur le niveau de droits de l'utilisateur actuellement connecté.

7.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".

7.1.3 Réponse

— Code de statut 200 : les informations sur le niveau de droits ont été récupérées avec succès. Le corps de la Réponse contient un objet JSON ou YAML avec les informations.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Corps de la Réponse : Les données avec : le rôle de l'utilisateur (« admin » pour gérer la totalité des bases et « user » pour gérer les bases sur lesquelles il a les droits seulement), les bases auxquelles il a accès et les bases qu'il a créé et qu'il gère, Ainsi que les bases verrouillée, et par quel utilisateur. Il contient également le jeton d'authentification nouvellement créé avec la date d'expiration à jour.

7.1.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
GET /status HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

7.1.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                # headers
Content-Type: application/json
                # body
{
    "data": {
        "role": "user"
        "bases_available" :
            ["base1", "base2", "base3"]
        "bases_owned" :
            ["base1"]
    },
    "token": "eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb..."
}
```

8 /bases

8.1 Méthode : Get

8.1.1 Description

Renvoie la liste de toutes les bases accessibles ou non.

8.1.2 Paramètres

— « Authentication » (obligatoire) : le jeton d'authentification obtenu lors de la connexion.
— « Accept » : format des données renvoyées en 'application/json' ou 'application/x-yaml'. Valeur par défaut : 'application/x-yaml'.

8.1.3 Réponse

— Code de statut 200 : les informations sur le niveau de droits ont été récupérées avec succès. Le corps de la réponse contient les informations demandées.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Corps de la Réponse : liste des différentes bases présentes et le jeton d'authentification nouvellement créé avec la date d'expiration à jour.

8.1.4 Exemple de requête

```
              # The HTTP Method, Path, HTTP Version
GET /bases HTTP/1.1
              # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Accept: application/json
```

8.1.5 Exemple de réponse

```
              # The HTTP code response and text
HTTP/1.1 200 OK
              # headers
Content-Type: application/json
              # body
{
    "data": {
        "bases": {"base1", "base2"}
    },
    "token": "eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb..."
}
```

8.2 Méthode : Post

8.2.1 Description

Ajoute une nouvelle base. Pour ce faire, il faut certaines données à minima telles que : - Le nom de la base (précisé dans « base : nom : nom-de-la-base ») - Un schéma (précisé dans « schemas : nom-du-schéma ») - Un utilisateur ou une application (respectivement précisé dans « users : nom-de-l-utilisateur » ou « applications : nom-de-l-application »)

8.2.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— Corps de la requête : les données de la base en yaml.

8.2.3 Réponse

— Code de statut 201 : la base a bien été créée.
— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : la base existe déjà.

8.2.4 Exemple de requête

```
            # The HTTP Method, Path, HTTP Version
POST /bases HTTP/1.1
            # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
            # body
base:
    name: base1
    owner:
        name: proprietaire
        password: motDePasse?!?
applications:
    app1:
        mode: crud
    app2:
        mode: read
    app3:
users:
    user1:
        global-cruder: true
    user2:
        global-reader: true
    user3: { admin: true, auth: ldap }
    user4: { auth: internal }
    user5: { auth: internal }
schemas:
    schema1:
    schema2:
        crud: [user4]
    schema3:
        crud: [user4]
        read: [user5]
```

8.2.5 Exemple de réponse

```
            # The HTTP code response and text
HTTP/1.1 201 Created
            # headers
Location: /bases/base1
            # body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

9 /bases/<base>

9.1 Méthode : Get

9.1.1 Description

Renvoie les informations de la base dans le format demandé.

9.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « Accept » : format des données de la base que l'on souhaite recevoir. Peut être en 'application/json' ou 'application/x-yaml'. Il est possible de préciser également la valeur 'text/x-sql' afin de récuperer les commandes Postgresql correspondant à la création de la base. Valeur par défaut : 'application/x-yaml'.

9.1.3 Réponse

— Code de statut 200 OK : Les informations sur la base spécifiée ont été récupérées avec succès.
— Corps de la Réponse : informations demandée sur la base spécifiée et le jeton d'authentification nouvellement créé avec la date d'expiration à jour..
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base n'existe pas.

9.1.4 Exemple de requête

```
              # The HTTP Method, Path, HTTP Version
GET /bases/base1 HTTP/1.1
              # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

9.1.5 Exemple de réponse

```
              # The HTTP code response and text
HTTP/1.1 200 OK
              # headers
Content-Type: application/x-yaml
              # body
data:
    base:
        name: base1
        owner:
            name: proprietaire
            password: motDePasse?!?
    applications:
        app1:
            mode: crud
        app2:
            mode: read
        app3:
    users:
        user1:
            global-cruder: true
        user2:
            global-reader: true
        user3: { admin: true, auth: ldap }
        user4: { auth: internal }
        user5: { auth: internal }
    schemas:
```

```
        schema1:
        schema2:
            crud: [user4]
        schema3:
            crud: [user4]
            read: [user5]
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

9.2 Méthode : Delete

9.2.1 Description

Supprime la base.

9.2.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".

9.2.3 Réponse

— Code de statut 200 OK : le référentiel de la base de donnée a été supprimé.
— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base n'existe pas.

9.2.4 Exemple de requête

```
            # The HTTP Method, Path, HTTP Version
DELETE /bases/base1 HTTP/1.1
            # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

9.2.5 Exemple de réponse

```
            # The HTTP code response and text
HTTP/1.1 200 OK
            #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

10 /bases/<base>/base

10.1 Méthode : Get

10.1.1 Description

Renvoie les informations telles que le nom de la base et le propriétaire de la base.

10.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « Accept » : format des données de la base en 'application/json' ou 'application/x-yaml'. Il est possible de préciser également la valeur 'text/x-sql' afin de récuperer les commandes Postgresql correspondant à la création de la base. Valeur par défaut : 'application/x-yaml'.

10.1.3 Réponse

— Code de statut 200 OK : Les informations sur la base spécifiée ont été modifiées avec succès.
— Corps de la réponse : informations sur la base spécifiée et le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base n'existe pas.

10.1.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
GET /bases/base1/base HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

10.1.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                # headers
Content-Type: application/x-yaml
                # body
data:
    name: base1
    owner:
        name: base1_owner
        password: a4fJ3Mec1O9msp
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

11 /bases/<base>/apps

11.1 Méthode : Get

11.1.1 Description

Renvoie la liste des applications de la base.

11.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « Accept » : format des données des applications en 'application/json' ou 'application/x-yaml'. Il est possible de préciser également la valeur 'text/x-sql' afin de récuperer les commandes Postgresql correspondant à la création des applications. Valeur par défaut : 'application/x-yaml'.

11.1.3 Réponse

— Code de statut 200 OK : Les informations sur la base spécifiée ont été récupérées avec succès. Le corps de la réponse contient un objet avec les informations.
— Corps de la réponse : informations sur la base spécifiée.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base n'existe pas ou il n'y a pas d'applications.

11.1.4 Exemple de requête

```
              # The HTTP Method, Path, HTTP Version
GET /bases/base1/apps HTTP/1.1
              # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

11.1.5 Exemple de réponse

```
              # The HTTP code response and text
HTTP/1.1 200 OK
              # headers
Content-Type: application/json
              # body
data:
    [app1, app2, app3]
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

11.2 Méthode : Post

11.2.1 Description

Ajoute une nouvelle application à la base.

11.2.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— Corps de la requête : les informations en yaml de l'application à ajouter.

11.2.3 Réponse

— Code de statut 201 Created : Les informations sur la base spécifiée ont été modifiées avec succès.

— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits ou l'application existe déjà.
— Erreur 404 Not Found : la base n'existe pas.

11.2.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
POST /bases/base1/apps HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                # body
app4:
    mode: crud
```

11.2.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 201 Created
                # headers
Location: /bases/base1/apps/app4
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

12 /bases/<base>/apps/<app>

12.1 Méthode : Get

12.1.1 Description

Renvoie les données de l'application de la base.

12.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « Accept » : format des données de l'application en 'application/json' ou 'application/x-yaml'. Il est possible de préciser également la valeur 'text/x-sql' afin de récuperer les commandes Postgresql correspondant à la création de l'application. Valeur par défaut : 'application/x-yaml'.

12.1.3 Réponse

— Code de statut 200 OK : Les informations sur la base spécifiée ont été récupérées avec succès.
— Corps de la réponse : informations sur la base spécifiée et le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits
— Erreur 404 Not Found : la base ou l'application n'existe pas.

12.1.4 Exemple de requête

```
              # The HTTP Method, Path, HTTP Version
GET /bases/base1/apps/app4 HTTP/1.1
              # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

12.1.5 Exemple de réponse

```
              # The HTTP code response and text
HTTP/1.1 200 OK
              # headers
Content-Type: application/x-yaml
              # body
data:
   mode: crud
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

12.2 Méthode : Put

12.2.1 Description

Crée l'application de la base à l'URI demandé. Si une application a déjà ce nom, la remplace

12.2.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— Corps de la requête : les informations en yaml de l'application à créer ou à modifier.

12.2.3 Réponse

- Code de statut 200 OK : Les informations sur la base spécifiée ont été modifiées avec succès.
- Code de stat 201 Created : Les informations sur la base spécifiée ont été créées avec succès.
- Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
- Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
- Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
- Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
- Erreur 404 Not Found : la base n'existe pas.

12.2.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
PUT /bases/base1/apps/app4 HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                # body
mode: read
```

12.2.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

12.3 Méthode : Delete

12.3.1 Description

Supprime l'application de la base.

12.3.2 Paramètres

- « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".

12.3.3 Réponse

- Code de statut 200 OK : l'application de la base de donnée a été supprimée.
- Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
- Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
- Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
- Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
- Erreur 404 Not Found : la base ou l'application n'existe pas.

12.3.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
DELETE /bases/base1/apps/app4 HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

12.3.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

13 /bases/<base>/users

13.1 Méthode : Get

13.1.1 Description

Renvoie la liste des utilisateurs de la base.

13.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « Accept » : format des données des utilisateurs en 'application/json' ou 'application/x-yaml'. Il est possible de préciser également la valeur 'text/x-sql' afin de récuperer les commandes Postgresql correspondant à la création des utilisateurs. Valeur par défaut : 'application/x-yaml'.

13.1.3 Réponse

— Code de statut 200 OK : Les informations sur la base spécifiée ont été récupérées avec succès. Le corps de la réponse contient un objet avec les informations
— Corps de la réponse : informations sur les utilisateurs et le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base n'existe pas ou il n'y a aucun utilisateurs.

13.1.4 Exemple de requête

```
              # The HTTP Method, Path, HTTP VersionVersion
GET /bases/base1/users HTTP/1.1
              # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

13.1.5 Exemple de réponse

```
              # The HTTP code response and text
HTTP/1.1 200 OK
              # headers
Content-Type: application/x-yaml
              # body
data:
    ["user1", "user2"]
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

13.2 Méthode : Post

13.2.1 Description

Ajoute un nouvel utilisateur à la base.

13.2.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— Corps de la requête : les informations en yaml de l'utilisateur à ajouter.

13.2.3 Réponse

- Code de statut 201 Created : Les informations sur la base spécifiée ont été créées avec succès.
- Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
- Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
- Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
- Erreur 403 Forbidden : l'utilisateur n'a pas les droits ou l'utilisateur existe déjà.
- Erreur 404 Not Found : la base n'existe pas.

13.2.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
POST /bases/base1/users HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                # body
user6: {admin: true, auth: ldap}
```

13.2.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 201 Created
                # headers
Location: /bases/base1/users/user11
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

14 /bases/<base>/users/<user>

14.1 Méthode : Get

14.1.1 Description

Renvoie les données de l'utilisateur de la base.

14.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « Accept » : format des données de la base en 'application/json' ou 'application/x-yaml'. Il est possible de préciser également la valeur 'text/x-sql' afin de récuperer les commandes Postgresql correspondant à la création de la base. Valeur par défaut : 'application/x-yaml'.

14.1.3 Réponse

— Code de statut 200 OK : Les informations sur la base spécifiée ont été récupérées avec succès.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base ou l'utilisateur n'existe pas.
— Corps de la réponse : informations sur l'utilisateur de la base spécifiée et le jeton d'authentification nouvellement créé avec la date d'expiration à jour.

14.1.4 Exemple de requête

```
              # The HTTP Method, Path, HTTP Version
GET /bases/base1/users/user1 HTTP/1.1
              # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

14.1.5 Exemple de réponse

```
              # The HTTP code response and text
HTTP/1.1 200 OK
              # headers
Content-Type: application/x-yaml
              # body
data:
    global-cruder: true
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

14.2 Méthode : Put

14.2.1 Description

Crée l'utilisateur de la base à l'URI demandé. Si un utilisateur a déjà ce nom, le remplace

14.2.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— Corps de la requête : les informations en yaml de l'utilisateur à modifier.

14.2.3 Réponse

— Code de statut 200 OK : Les informations sur la base spécifiée ont été modifiées avec succès.
— Code de statut 201 Created : Les informations sur la base spécifiée ont été créées avec succès.

— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base n'existe pas.

14.2.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
PUT /bases/base1/users/user1 HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                # body
global-reader: true
```

14.2.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

14.3 Méthode : Delete

14.3.1 Description

Supprime l'utilisateur de la base.

14.3.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".

14.3.3 Réponse

— Code de statut 200 OK : l'utilisateur de la base de donnée a été supprimé. Les commandes retournées sont à exécuter.
— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base ou l'utilisateur n'existe pas.

14.3.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
DELETE /bases/base1/users/user1 HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

14.3.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

15 /bases/<base>/schemas

15.1 Méthode : Get

15.1.1 Description

Renvoie la liste des schémas de la base.

15.1.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— « Accept » : format des données des schémas en 'application/json' ou 'application/x-yaml'. Il est possible de préciser également la valeur 'text/x-sql' afin de récuperer les commandes Postgresql correspondant à la création des schémas. Valeur par défaut : 'application/x-yaml'.

15.1.3 Réponse

— Code de statut 200 OK : Les informations sur les schémas ont été récupérées avec succès. Le corps de la réponse contient un objet avec les informations.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base n'existe pas.
— Corps de la réponse : informations sur les schémas et le jeton d'authentification nouvellement créé avec la date d'expiration à jour.

15.1.4 Exemple de requête

```
              # The HTTP Method, Path, HTTP Version
GET /bases/base1/schemas HTTP/1.1
              # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

15.1.5 Exemple de réponse

```
              # The HTTP code response and text
HTTP/1.1 200 OK
              # headers
Content-Type: application/x-yaml
              # body
data:
    [schema1, schema2, schema3]
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

15.2 Méthode : Post

15.2.1 Description

Ajoute un nouveau schéma à la base.

15.2.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— Corps de la requête : les informations en yaml du schéma à ajouter.

15.2.3 Réponse

— Code de statut 201 Created : Le schéma a bien été ajouté.

— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits ou le schéma existe déjà.
— Erreur 404 Not Found : la base n'existe pas.

15.2.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
POST /bases/base1/schemas HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
                # body
schema4:
   read: [user1]
```

15.2.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 201 Created
                # headers
Location: /bases/base1/schemas/schema4
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

16 bases/<base>/schemas/<schema>

16.1 Méthode : Get

16.1.1 Description

Renvoie les données du schéma de la base.

16.1.2 Paramètres

— « Authorization » : le jeton d'authentification obtenu lors de la connexion.
— « Accept » : format des données de la base en 'application/json' ou 'application/x-yaml'. Il est possible de préciser également la valeur 'text/x-sql' afin de récuperer les commandes Postgresql correspondant à la création de la base. Valeur par défaut : 'application/x-yaml'.

16.1.3 Réponse

— Code de statut 200 OK : Les informations sur le schéma spécifié ont été récupérées avec succès.
— Code de stat 201 Created : Les informations sur la base spécifiée ont été créées avec succès.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base ou le schéma n'existe pas.
— Corps de la réponse : informations sur le schéma spécifié et le jeton d'authentification nouvellement créé avec la date d'expiration à jour.

16.1.4 Exemple de requête

```
                # The HTTP Method, Path, HTTP Version
GET /bases/base1/schemas/schema4 HTTP/1.1
                # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

16.1.5 Exemple de réponse

```
                # The HTTP code response and text
HTTP/1.1 200 OK
                # headers
Content-Type: application/x-yaml
                # body
data:
    read: [user1]
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

16.2 Méthode : Put

16.2.1 Description

Crée le schéma de la base à l'URI demandé. Si un schéma a déjà ce nom, le remplace

16.2.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".
— Corps de la requête : les informations en yaml du schéma à créer ou modifier.

16.2.3 Réponse

— Code de statut 200 OK : Les informations sur le schéma spécifié ont été modifiées avec succès.

— Code de statut 201 Created : Les informations sur le schéma spécifié ont été créées avec succès.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base n'existe pas.
— Corps de la réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.

16.2.4 Exemple de requête

```
              # The HTTP Method, Path, HTTP Version
PUT /bases/base1/schemas/schema4
              # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
              # body
read: [user2]
```

16.2.5 Exemple de réponse

```
              # The HTTP code response and text
HTTP/1.1 200 OK
              #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

16.3 Méthode : Delete

16.3.1 Description

Supprime le schéma de la base.

16.3.2 Paramètres

— « Authorization » (obligatoire) : le dernier jeton d'authentification obtenu précédé de la mention "Bearer".

16.3.3 Réponse

— Code de statut 200 OK : Les informations sur le schéma de la base de donnée ont été supprimé. Les commandes retournées sont à exécuter.
— Corps de la Réponse : Le jeton d'authentification nouvellement créé avec la date d'expiration à jour.
— Erreur 400 Bad Request : des paramètres sont manquants ou invalides dans la requête.
— Erreur 401 Unauthorized : le jeton d'authentification est incorrect ou a expiré.
— Erreur 403 Forbidden : l'utilisateur n'a pas les droits.
— Erreur 404 Not Found : la base ou le schéma n'existe pas.

16.3.4 Exemple de requête

```
              # The HTTP Method, Path, HTTP Version
DELETE /bases/base1/schemas/schema4
              # headers
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

16.3.5 Exemple de réponse

```
              # The HTTP code response and text
HTTP/1.1 200 OK
```

```
                #body
token: eyJhbGciOiJIUfjDo58sbTJ8Nqg3csGJ60Xb...
```

28

17 Tableau récapitulatif

	GET	PUT	POST	DELETE
/bases				
/bases/<base>				
/bases/<base>/base				
/bases/<base>/apps				
/bases/<base/apps/<app>				
/bases/<base/users				
/bases/<base/users/<user>				
/bases/<base>/schemas				
/bases/<base/schemas/<schema>				

/login				

/status				
/grant_admin				
/revoke_admin				
/grant				
/revoke				

FIGURE 1 – Tableau récapitulatif.

18 Diagramme

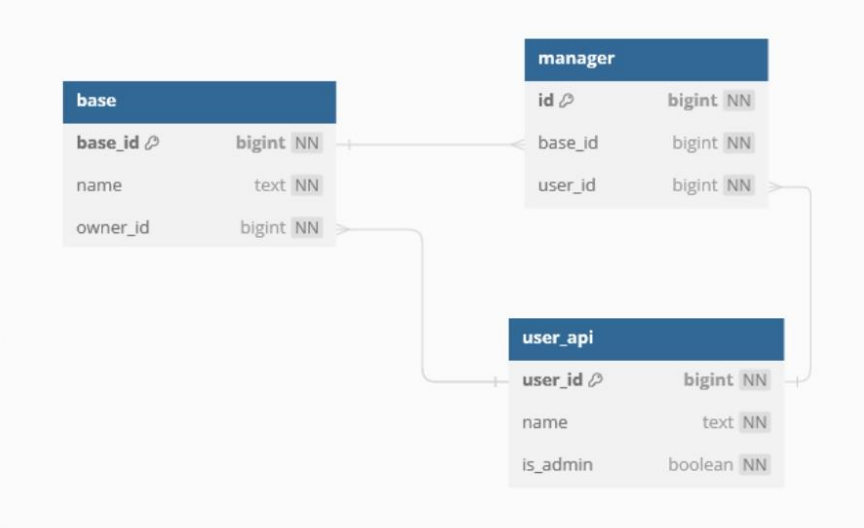

FIGURE 2 – Représentation des droits associés à la gestion de bases.

DÉPARTEMENT DE MAINE-ET-LOIRE

anjou

CS 94104 - 49 941 ANGERS CEDEX 9