

Brevet de technicien supérieur C.I.EL

option Informatique et réseaux

Rapport de projet

Système d'information - Pôle Info



Elias GAUTHIER

BTS CIEL
Session 2025

<i>Étudiants</i>	<i>Enseignants</i>
Elias GAUTHIER	Dominique SUTTER
Ethan CLEMENT	Alain TRINQUET
Lucas GUILLOTEAU	

Sommaire

Introduction.....	3
Contexte.....	3
Problématique.....	3
Objectifs du projet.....	4
Procédé de conception.....	5
Conception préliminaire.....	7
Analyse du cahier des charges.....	7
Besoins fonctionnels et non fonctionnels.....	7
Diagramme des exigences.....	9
Diagramme de cas d'utilisation.....	11
Aperçu.....	12
Liste du matériel.....	13
Architecture du système.....	13
Diagramme de déploiement.....	14
Méthodologie.....	16
Outils et technologies utilisées.....	16
GitHub.....	16
commit.....	17
pull.....	17
push.....	17
fetch.....	18
diff.....	18
GitHub Desktop.....	18
Ansible.....	18
FastAPI.....	19
PHP.....	19
Chaîne de développement.....	20
Intégration continue.....	20
Répertoire GitHub.....	21
Configuration service.....	23
Configuration ansible.....	26
Répartition des tâches.....	30
Diagramme prévisionnel.....	31
Évaluation du matériel.....	33
Objectif.....	33
Environnement de test.....	33
Procédures et critères de test.....	33
Utilitaires utilisés.....	34
Htop.....	34
Description.....	34
Installation.....	35
vcgencmd.....	35

Description.....	35
Installation.....	35
Résultats.....	36
Conception détaillée.....	37
Développement et mise en œuvre.....	37
Base de données.....	37
Application Programming Interface (Elias GAUTHIER).....	41
Introduction aux APIs.....	41
Architecture de l'API.....	43
Point d'entrée.....	45
Création utilisateur.....	48
Authentification.....	52
Vérification jeton.....	58
Réservations.....	61
POST.....	62
GET.....	69
DELETE.....	74
PUT.....	75
Interface utilisateur (Elias GAUTHIER).....	77
Mise en page.....	77
Structure.....	78
Technologies utilisées.....	78
Fichier principal.....	79
Suppression de réservation.....	84
Ajout de réservation.....	86
Modification de réservation.....	87
Rendu final.....	90
Tests et validation.....	91
Test unitaires.....	91
Recettes.....	93
Conclusion.....	96
Difficultés rencontrées.....	96
Acquis et bénéfices.....	96
Conclusion générale.....	96
Annexe.....	97
Glossaire.....	97
Documentation d'usage de l'API.....	98

Introduction

Contexte

Au sein de tout établissement scolaire moderne, on retrouve un Environnement Numérique de Travail. L'ENT est une plateforme en ligne qui centralise un ensemble de services numériques à destination des membres de la communauté éducative : élèves, enseignants, personnels administratifs et parents. Il permet un accès distant et sécurisé à des ressources pédagogiques, à la messagerie interne, aux plannings, aux notes, ou encore aux documents administratifs. En France, les ENT sont largement déployés depuis la fin des années 2000 dans les établissements scolaires, dans le cadre d'un partenariat entre l'État et les collectivités territoriales.

Véritable **bureau virtuel**, l'ENT facilite la communication, le suivi scolaire et la gestion de la vie de l'établissement, tout en s'inscrivant dans une stratégie de **numérisation de l'éducation** visant à améliorer la réussite des élèves.

Problématique

Au sein de notre établissement, le lycée Pilote Innovant International, il est nécessaire de consulter son ENT pour visualiser les cours et activités à venir. Seulement une problématique est récurrente : la facilité d'accès aux informations et le manque d'informations essentielles.



Une problématique peut alors être posée :

Comment améliorer la diffusion des informations relatives aux activités et cours scolaires afin de fluidifier l'organisation et renforcer la communication entre les acteurs pédagogiques ?

Objectifs du projet

Notre mission vise à créer un système d'affichage d'informations en temps réel au niveau de la zone d'accueil et à l'entrée des salles utilisées pour les activités et les cours. Des interfaces web permettront également la visualisation des réservations.

L'objectif est de permettre aux étudiants de connaître immédiatement les horaires et les salles pour chacune de leurs activités.

Parmi les activités, on souligne particulièrement les activités de soutien et d'approfondissement (B.A.S.¹) et les projets en équipe transdisciplinaire et inter-niveaux (A.C.F.²). A travers le développement d'une extension pratique à l'ENT, nous respectons ainsi des critères liés au **Digital Workspace**.

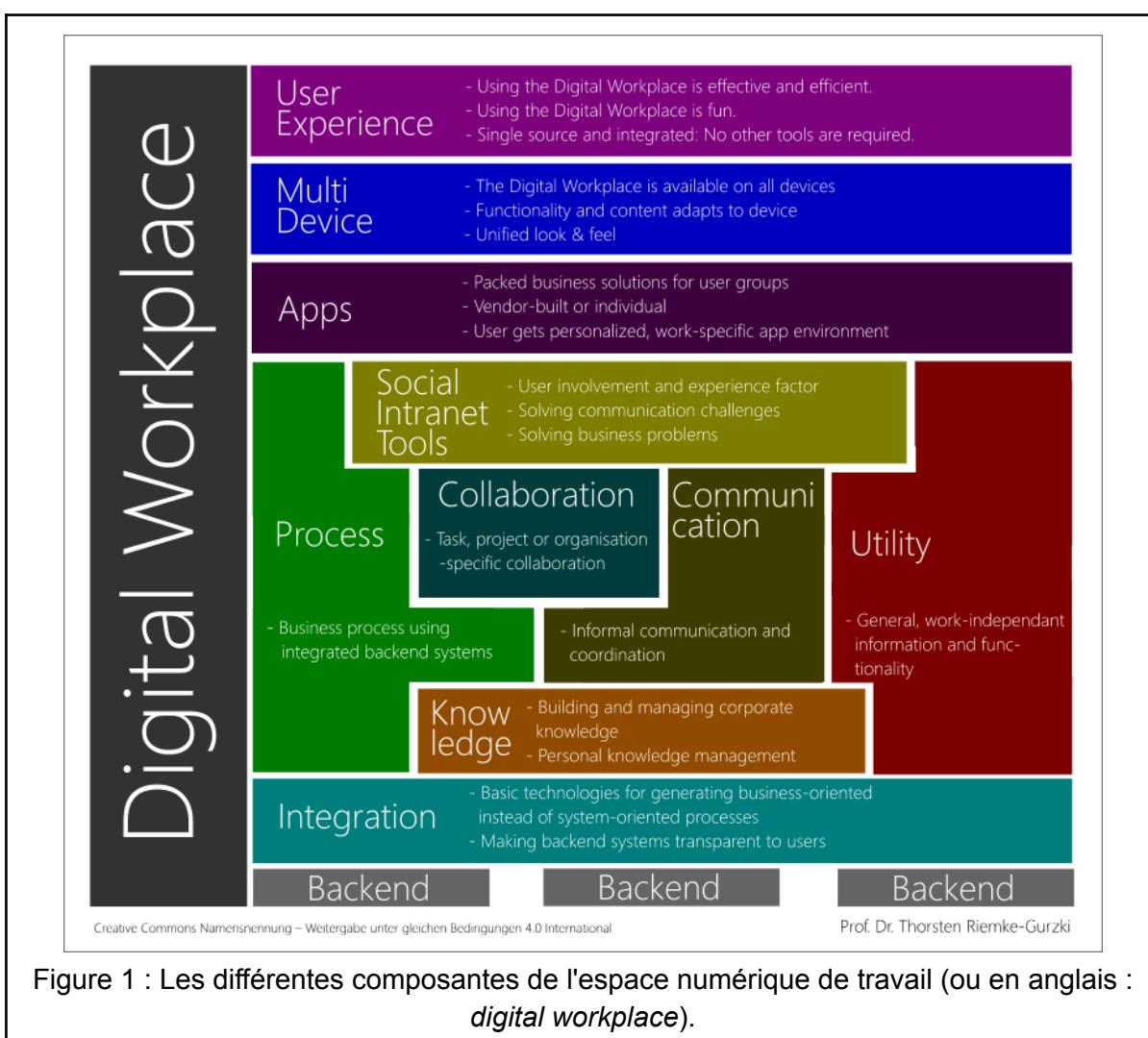


Figure 1 : Les différentes composantes de l'espace numérique de travail (ou en anglais : *digital workplace*).

¹ Besoin Autonomie Suivi

² Activité complémentaire de formation

User Experience (Expérience utilisateur)	Multi Device	Apps	Social intranet Tools	Process
Améliorer l'accès à l'information pour les étudiants et enseignants, de manière simple, rapide et agréable.	Interface web consultable sur PC et application mobile.	Interface web personnalisée par type d'utilisateur (étudiant, enseignant, administrateur), avec des fonctionnalités ciblées.	Favoriser la communication interne, la coordination des activités pédagogiques, et la réduction des silos d'information.	Numérisation de processus métiers internes (réservations de salles et des plannings)

Collaboration	Communication	Utility	Integration & Backend
Aider à coordonner les membres, à planifier les séances et à les faire collaborer plus efficacement grâce à une meilleure visibilité.	L'affichage en temps réel et l'interface web sont des outils de communication directe, qui permettent d'éviter les erreurs, retards ou absences dues à un manque d'information.	L'information affichée est générale mais essentielle au bon déroulement de la vie scolaire : horaires, salles etc. Cela relève de la fonction utilitaire du Digital Workplace.	Interfaçage obligatoire avec une base de donnée et une API

Procédé de conception

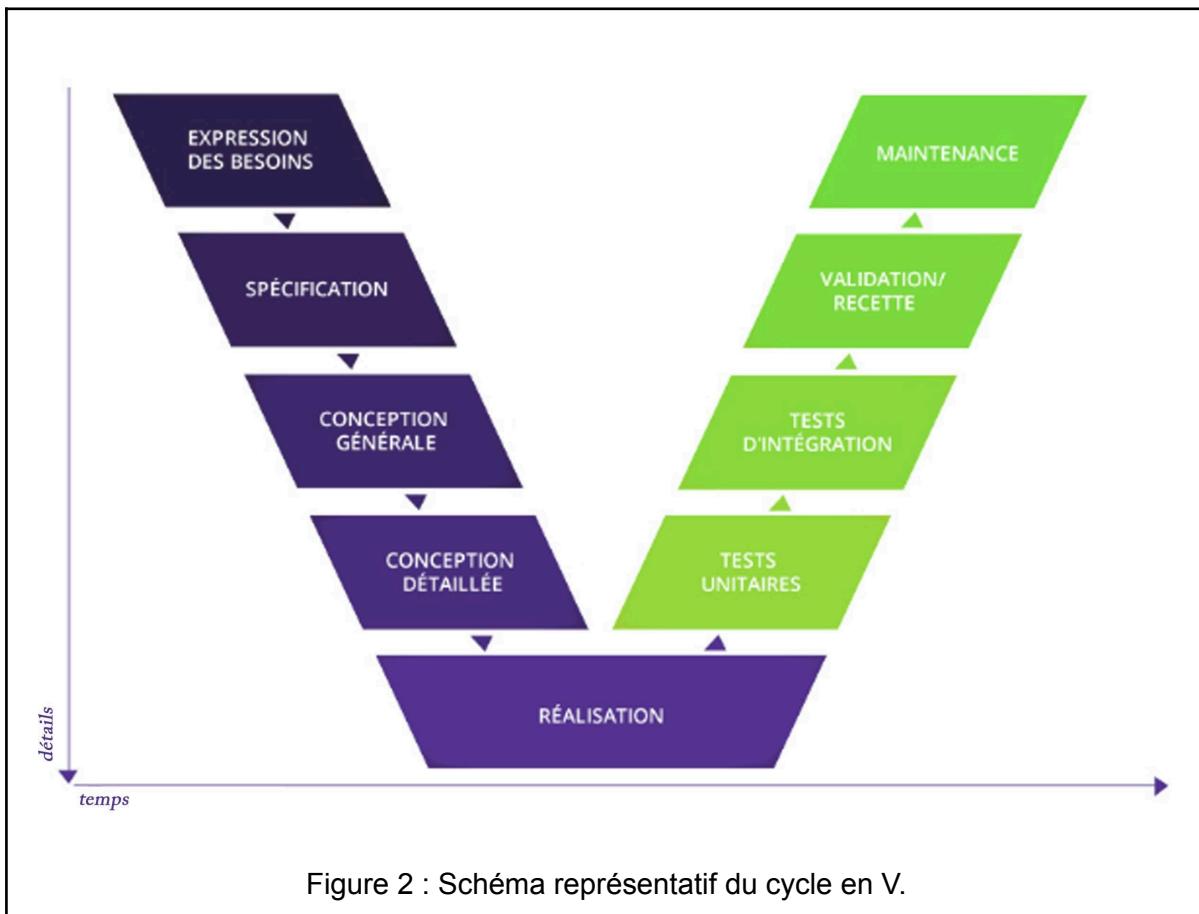
Afin de mener à bien notre projet à travers chacune de ces composantes nous effectuerons d'abord une conception préliminaire liée à l'analyse du cahier des charges, diagrammes UML, répartitions des tâches etc avant de réaliser la conception détaillée.

La conception détaillée contiendra toutes les réalisations concrètes nécessaires à l'aboutissement du projet.

Enfin, des tests unitaires et des validations seront effectués afin de pouvoir fournir un prototype fonctionnel et répondant aux exigences attendues.

En d'autres termes, nous utiliserons ce que l'on appelle une méthodologie de cycle **en V**. Cette méthodologie est réputée et reconnue pour sa fiabilité et sa performance depuis les années 1970.

Dans un projet d'une telle envergure il est inutile d'utiliser une méthodologie **Agile** lourde de type "Scrum" qui serait une perte de temps significative. **Cependant, toute l'intégration sera continue au fur et à mesure de la réalisation.** (Voir [Chaîne de développement](#))



Expression du besoin : Définir ce que l'utilisateur final attend du système.

Spécifications fonctionnelles : Traduire les besoins en fonctionnalités logicielles observables, définir l'architecture technique du système.

Conception détaillée : Décrire les structure de données, API, etc

Réalisation : Développement concret du système informatique.

Test/validation : Vérifier que le produit fini corresponds aux attentes

Recette : Validation client

Suite → [Conception préliminaire](#)

Conception préliminaire

Analyse du cahier des charges

Auteurs de cette section :

- Elias GAUTHIER

Correspond à : **Expression des besoins** (Voir [Cycle en V](#))

La première étape de la mise en place lors de notre projet fut d'analyser et de comprendre clairement les besoins et les exigences demandées. Nous avons donc réalisé une analyse du cahier des charges en examinant scrupuleusement les besoins fonctionnels/non fonctionnels, en rédigeant un diagramme des exigences ainsi qu'un diagramme de cas d'utilisation.

Besoins fonctionnels et non fonctionnels

Une exigence fonctionnelle (ou besoin fonctionnel) décrit ce que le système doit faire. Elle exprime une fonction, une action ou un service attendu du système, généralement exprimé du point de vue de l'utilisateur final.

Ces exigences peuvent être déterminées depuis le document de présentation du projet qui nous a été fourni par les professeurs :

1. Gestion des comptes utilisateurs

- Consulter, créer, modifier, supprimer les comptes des utilisateurs autorisés à accéder à l'interface de gestion.

2. Gestion des informations à afficher

- Consulter, créer, modifier, supprimer les informations associées aux différents écrans d'affichage (horaire, contenu, etc.).
- Afficher, aux horaires prévus, les contenus définis par l'utilisateur sur chaque écran.

3. Synthèse des informations pour l'affichage général

- Pour la zone d'accueil, récapituler et afficher les informations concernant l'ensemble des salles utilisées en BTS.

4. Maintenance de la base de données

- Outils pour supprimer les informations obsolètes.
- Outils pour la sauvegarde de la base de données

5. Interface utilisateur

- Fournir une interface (web en PHP) pour la gestion de toutes les informations et matériels d'affichage.

Ensuite, les exigences non-fonctionnelles décrivent comment le système doit être ou les contraintes à respecter. Elle concerne les qualités, les performances, l'environnement, la sécurité, la compatibilité, etc. Ce ne sont pas des fonctions, mais des propriétés ou des restrictions.

1. Technologies et outils imposés

- Développement PHP avec utilisation de MariaDB (base de données).
- Interface web hébergée sur un Raspberry Pi dédié.

2. Sécurité et gestion accès

- Restriction de l'accès à l'interface d'administration via des comptes utilisateurs gérés.
- Analyse et mise en place des mécanismes de restriction d'accès.

3. Interopérabilité

- Capacité à fonctionner à la fois sur des écrans standards et sur des panneaux lumineux à une ligne.

4. Simplicité d'utilisation

- Interface utilisateur (administrateur et utilisateur final) intuitive et simple d'utilisation pour faciliter la gestion et la consultation.

Diagramme des exigences

Nous pouvons représenter de manière synthétique ces exigences sur un diagramme des exigences "REQ" appartenant à la famille SysML (Systems Modeling Language), qui est une extension d'UML adaptée à l'ingénierie système .

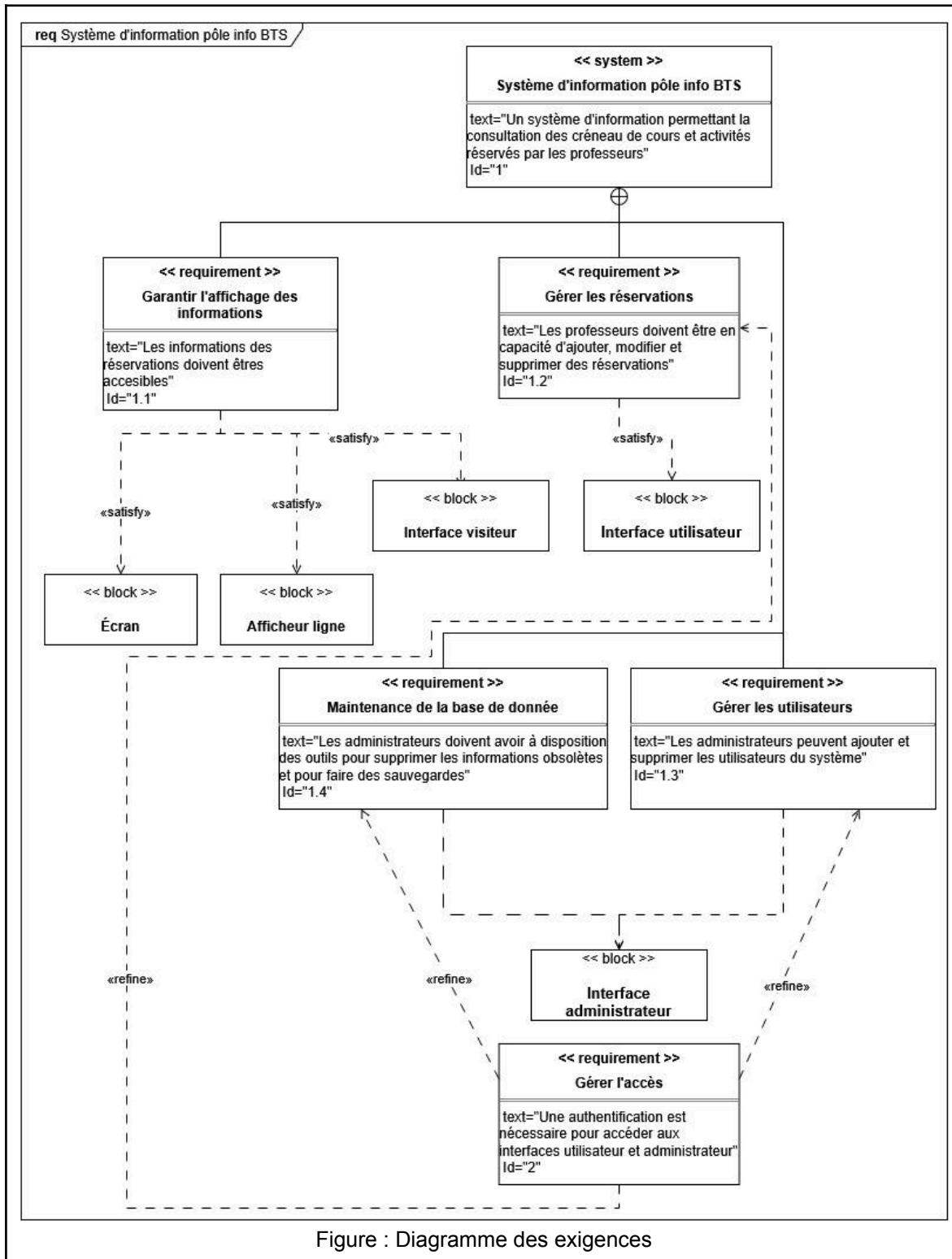


Figure : Diagramme des exigences

Un diagramme d'exigences permet de représenter visuellement les exigences d'un système et de les organiser de manière hiérarchique. Il montre également les relations entre éléments/composants du système.

On distingue dans notre cas 3 types de composants différents :

<< system >>	<< requirement >>	<< block >>
Composant du système de qui découle les exigences. On va donc décrire les exigences qui sont associées au système d'information pôle Info BTS	Il indique qu'un élément du diagramme représente une exigence que le système doit satisfaire. Il possède systématiquement un ID ainsi qu'un texte descriptif.	Un bloc peut modéliser n'importe quel élément structurel d'un système, qu'il s'agisse d'un composant matériel, logiciel, d'une personne, d'une installation, ou même d'un concept. Dans notre cas, ces blocs représentent différentes parties structurelles du système d'information qui réalisent ou satisfont les exigences définies.

Nous avons également 3 types de relations différentes :

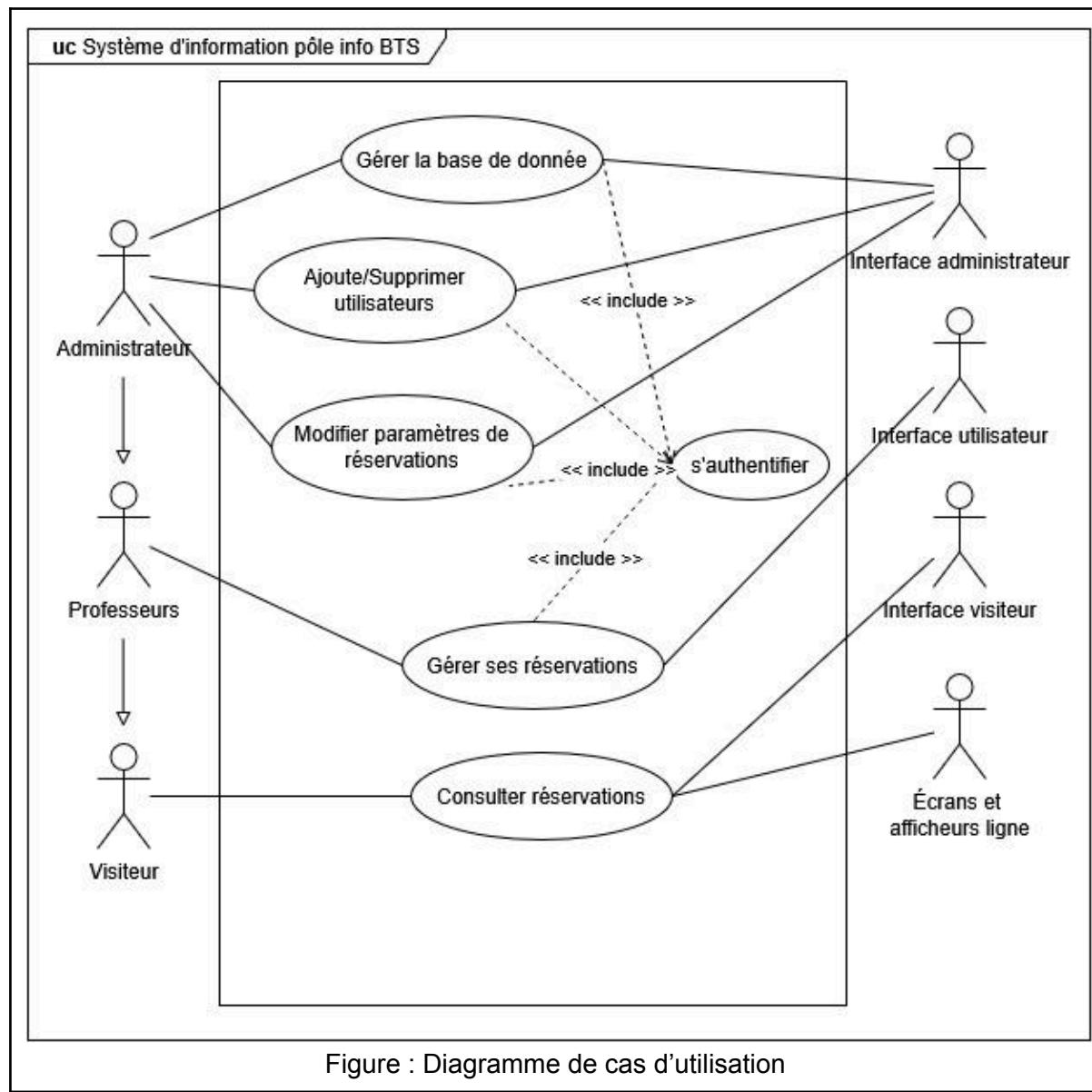
Composition	<< satisfy >>	<< refine >>
Représente la décomposition hiérarchique des exigences, une exigence parent peut contenir plusieurs sous-exigences.	Indique qu'un élément de conception (comme un block) satisfait une exigence	Précise ou affine une exigence avec des informations supplémentaires

Dans notre cas, **Garantir l'affichage des informations**, **Gérer les réservations**, **Maintenance de la base de donnée** et **Gérer les utilisateurs** sont des exigences << requirement >> qui doivent être satisfaites par le système.

Elles sont satisfaites par les << block >> correspondants avec la relation << satisfy >>.

Diagramme de cas d'utilisation

En langage UML, les diagrammes de cas d'utilisation modélisent le comportement d'un système et permettent de capturer les exigences du système. Les diagrammes de cas d'utilisation décrivent les fonctions générales et la portée d'un système. Ces diagrammes identifient également les interactions entre le système et ses acteurs. Les cas d'utilisation et les acteurs dans les diagrammes de cas d'utilisation décrivent ce que le système fait et comment les acteurs l'utilisent, mais ne montrent pas comment le système fonctionne en interne. On se sert ici d'un diagramme de cas d'utilisation pour représenter les différents acteurs et actions de notre projet.

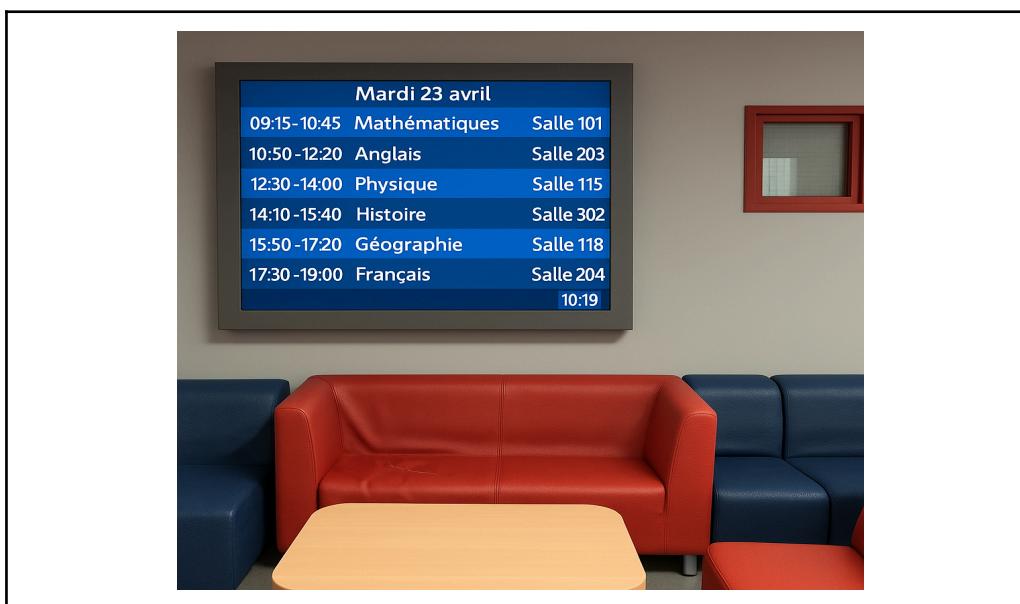


On retrouve trois acteurs principaux : l'administrateur, le professeur et le visiteur ainsi que trois acteurs secondaires : Interface administrateur, Interface utilisateur, interface visiteur et écran et afficheurs lignes.

Il est nécessaire de s'authentifier pour certaines tâches, la relation <<include>> permet de représenter cela.

Aperçu

Afin de visualiser à quoi ressemblerait l'affichage physique, nous avons demandé à une intelligence artificielle d'intégrer une image d'afficheur ligne au-dessus d'une porte ainsi qu'un écran d'affichage type gare au mur. Cela permet de projeter les attendus finaux du projet.



Liste du matériel

Dans le cadre de la réalisation de ce projet, divers équipements matériels ont été mobilisés. Cette section présente la liste complète des ressources matérielles utilisées, accompagnée d'une description technique, de leur rôle dans le projet et des raisons de leur sélection.

Quantité	Description	Utilisation
>2	Raspberry Pi 3	Exécuter logiciel C++ contrôlant les afficheurs lignes
1	Raspberry Pi 3	Serveur
>1	Écran	Affichage principal
>1	Afficheur ligne LED	Affichage d'informations synthétique

Notre projet ne demande pas de grandes ressources matérielles, les éléments les plus importants étant les Raspberry Pi et les afficheurs lignes. Les descriptions techniques des composants sont inscrites dans la conception détaillée.

Architecture du système

Auteurs de cette section :

- Elias GAUTHIER

Correspond au Cycle en V : **Spécifications**

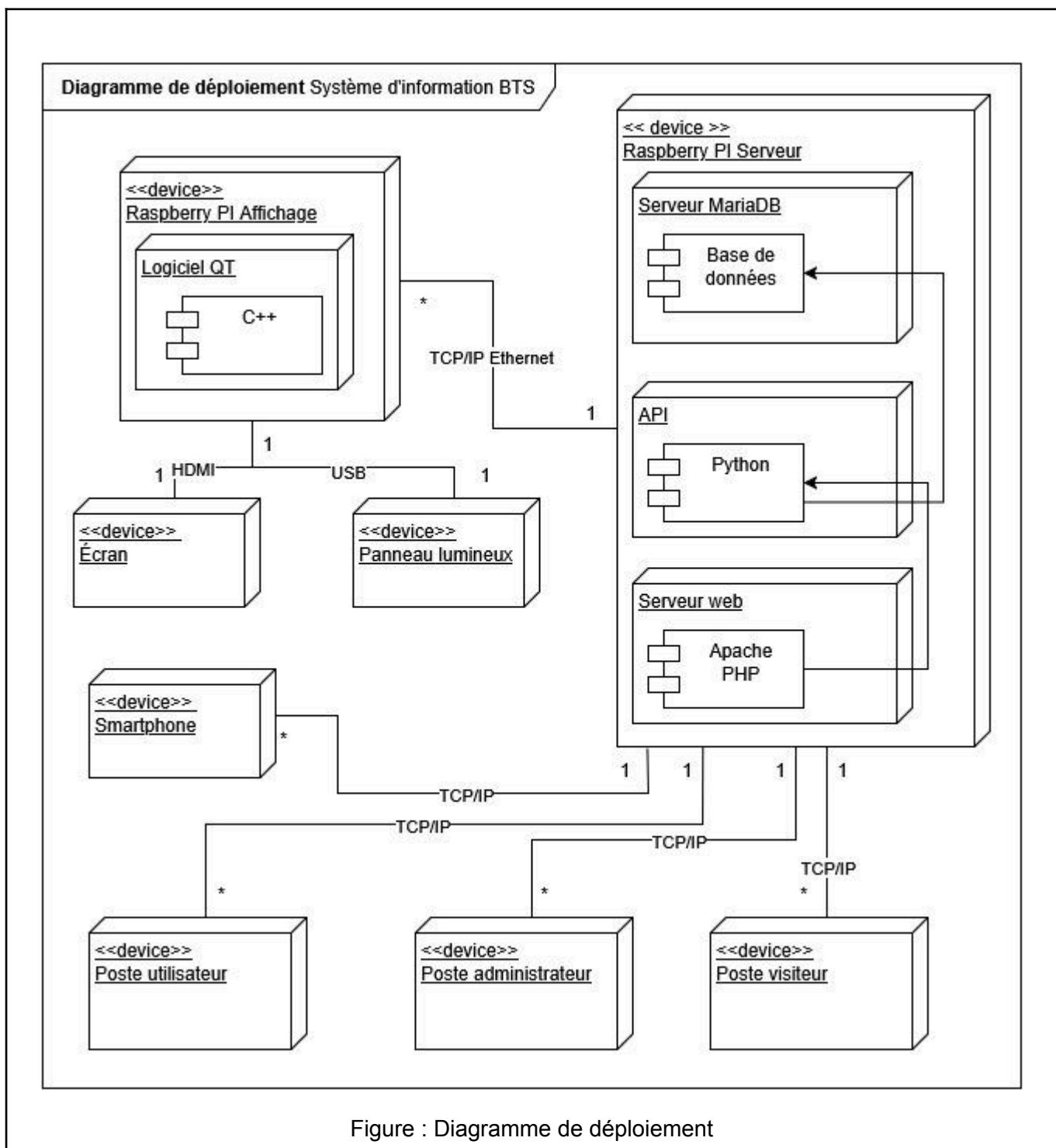
Une fois le matériel listé et les exigences du système définies, il est préférable de réaliser une description de l'architecture finale du système. Cette section contiendra ainsi l'ensemble des schémas fonctionnels nécessaires à l'explication de la structure du système.

Initialement ce projet incluait uniquement un développement des interfaces web en PHP et la programmation C++ des afficheurs physique. Nous avons décidé d'aller plus loin en intégrant une **API** et une **application mobile** afin de s'inscrire dans le développement logiciel moderne.

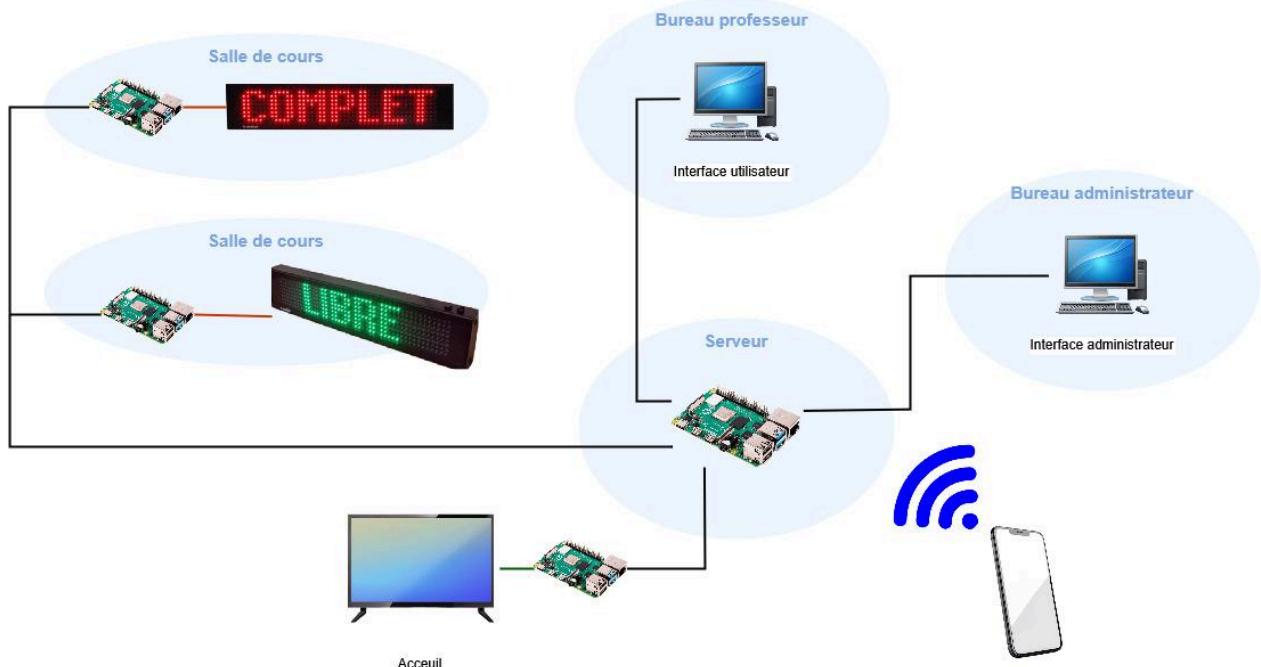
L'API joue un rôle intermédiaire entre les interfaces web et la base de données, elle possède de nombreux avantages qui seront explicités dans la conception détaillée. (Voir [Application Programming Interface \(Elias GAUTHIER\)](#))

Diagramme de déploiement

Dans le contexte du langage de modélisation unifié (UML), un diagramme de déploiement fait partie de la catégorie des diagrammes structurels, car il décrit un aspect du système même. On utilise donc un diagramme de déploiement pour décrire notre système d'information Pôle Info BTS.



On peut également représenter le système encore plus simplement de cette manière avec un schéma non conventionnel.



Note : Ici le Raspberry n'a pas 4 interfaces, ce schéma n'est pas un schéma technique et est uniquement à but représentatif.

Méthodologie

Auteurs de cette section :

- Elias GAUTHIER
- Lucas GUILLOTEAU
- Ethan CLEMENT

Correspond à : **Conception générale** (Voir [Cycle en V](#))

Dans cette section, nous présentons la démarche méthodologique adoptée tout au long de la réalisation du projet. Elle vise à expliciter l'organisation, les outils et les méthodes choisis afin d'assurer une gestion efficace et structurée des différentes phases de développement.

Outils et technologies utilisées

Les outils numériques et technologies utilisés au sein de notre projet sont divers et variés et représentent des éléments clés pour l'aboutissement du projet. Nous devons choisir des solutions technologiques pertinentes et efficaces que nous maîtrisons (pour la plupart).

Vous trouverez ainsi ici la liste des outils numériques utilisés pour répondre correctement au cahier des charges étudié précédemment.

GitHub

Avant toute chose, il nous paraissait évident de devoir partager tout le code source de notre projet de manière collaborative. Si chacun avait une version différente du code source en local, nous aurions perdu beaucoup de temps à devoir mettre à jour manuellement les différentes versions etc.



GitHub de l'entreprise GitHub, Inc. est un service d'hébergement et de gestion de développements de logiciels utilisant le logiciel de gestion des versions Git.

GitHub est compatible avec la commande git créé par Linus Torvalds, connu notamment pour avoir créé le noyau Linux en 1991 à l'âge de 21 ans. Git utilise un système de connexion pair à pair. Le code informatique d'un projet peut-être hébergée soit sur le ou les ordinateurs de chaque contributeur du projet soit sur un serveur dédié (ici en l'occurrence les serveurs de GitHub)

GitHub nous sert à héberger notre projet (notre code source), et nous permet également de surveiller minutieusement les moindres changements (via les "commits*" et les "diffs*" permettant de voir simplement les ajouts et les retraits sous forme de code couleur : vert pour les ajouts et rouge pour les retraits).

Tout un ensemble de commandes sont associées à Git (et donc à Github), nous utiliserons donc principalement les commit, diff, push, pull et fetch.

commit

Un commit est un enregistrement d'un ensemble de modifications. Si un code est différent de celui présent sur les serveurs de GitHub, on peut faire un commit pour mettre à jour le code avec les dernières modifications.

On peut également donner un nom à notre commit, afin de décrire manuellement ce qu'on a voulu modifier.

git-commit - Record changes to the repository

```
git commit [-a | --interactive | --patch] [-s] [-v] [-u[<mode>]] [--amend]
[--dry-run] [(-c | -C | --squash) <commit> | --fixup [(amend|reword):]<commit>]
[-F <file> | -m <msg>] [--reset-author] [--allow-empty]
[--allow-empty-message] [--no-verify] [-e] [--author=<author>]
[--date=<date>] [--cleanup=<mode>] [--[no-]status] [-i | -o]
[--pathspec-from-file=<file> [--pathspec-file-nul]] [((--trailer
<token>[=|:]<value>])...) [-S[<keyid>]] [--] [<pathspec>...]
```

pull

La commande git pull permet de récupérer les modifications du dépôt distant et de les fusionner avec le dépôt local.

git-pull - Rapatrier et intégrer un autre dépôt ou une branche locale

```
git pull [<options>] [<dépôt> [<spécification-de-référence>...]]
```

push

La commande git push permet de transmettre les modifications locales vers le dépôt distant sur GitHub.

git-push - Met à jour les références distantes ainsi que les objets associés

```
git push [--all | --branches | --mirror | --tags] [--follow-tags] [--atomic] [-n
| --dry-run] [--receive-pack=<git-receive-pack>]
[--repo=<dépôt>] [-f | --force] [-d | --delete] [--prune] [-q |
--quiet] [-v | --verbose]
[-u | --set-upstream] [-o <chaîne> | --push-option=<chaîne>]
[--[no-]signed|--signed=(true|false|if-asked)]
[--force-with-lease[=<nom-de-réf>[:<attendu>]] [--force-if-includes]]
[--no-verify] [<dépôt> [<spéc-de-réf>...]]
```

fetch

La commande git fetch permet de récupérer les nouvelles données du dépôt distant sans les fusionner automatiquement avec le dépôt local.

git-fetch - Télécharger les objets et références depuis un autre dépôt

```
git fetch [<options>] [<dépôt> [<spéc-de-réf>...]]  
git fetch [<options>] <groupe>  
git fetch --multiple [<options>] [(<dépôt> | <groupe>)...]  
git fetch --all [<options>]
```

diff

Un diff (abréviation de différence) est une comparaison entre deux versions d'un fichier. Pour GitHub le diff se compose d'éléments en vert et en rouge. Quand le code est en vert ça signifie qu'il est ajouté ou modifié, quand il est en rouge ça signifie qu'il a été retiré.

git-diff - Show changes between commits, commit and working tree, etc

```
git diff [<options>] [<commit> [--] [<path>...] git diff [<options>] --cached  
[--merge-base] [<commit>] [--] [<path>...] git diff [<options>] [--merge-base]  
<commit> [<commit>...] <commit> [--] [<path>...] git diff [<options>]  
<commit>...<commit> [--] [<path>...] git diff [<options>] <blob> <blob> git diff  
[<options>] --no-index [--] <path> <path>
```

GitHub Desktop

GitHub Desktop est le client PC de GitHub, il permet notamment de facilement gérer le projet. Il est principalement utilisé pour effectuer des commit et toute les opérations liées (pull, push, fetch, diff)

GitHub Desktop permet notamment de détecter le moindre changement de fichier, quand on modifie un fichier du projet GitHub Desktop le détecte permettant de faire un commit.

Ansible

Ansible est un outil open source d'automatisation informatique développé initialement par Michael DeHaan et maintenu par Red Hat. Il est principalement utilisé pour l'automatisation de la gestion de configuration, du déploiement d'applications, de l'orchestration de tâches et de la gestion d'infrastructure en tant que code (Infrastructure as Code – IaC).



Ansible fonctionne sans agent (agentless), ce qui signifie qu'il n'est pas nécessaire d'installer un logiciel spécifique sur les machines à gérer. Il exploite principalement SSH (pour les systèmes Unix/Linux) ou WinRM (pour les systèmes Windows) pour se connecter et exécuter des tâches sur des machines distantes. L'automatisation est décrite à travers des fichiers texte en **YAML** (appelés **playbooks**), ce qui rend la syntaxe lisible et facile à apprendre.

Ansible nous permettra le déploiement rapide du code source Github vers le serveur Raspberry Pi. Et ainsi consulter directement nos changements/modifications.

Python

Python est un langage de programmation interprété, multi paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions.



FastAPI



FastAPI est un framework web moderne pour Python, spécialement conçu pour créer des APIs rapides et robustes. Son adoption se justifie par plusieurs avantages :

- Performance élevée : FastAPI repose sur Starlette et Pydantic, et utilise la programmation asynchrone (async/await), ce qui permet de traiter plusieurs requêtes simultanément de façon efficace.
- Facilité de développement : sa syntaxe claire et ses fonctionnalités intégrées (gestion des routes, validation automatique des données, documentation Swagger automatique) accélèrent le développement.
- Documentation automatique : FastAPI génère automatiquement une documentation interactive via Swagger UI et ReDoc, ce qui facilite les tests et la compréhension de l'API par les développeurs et les utilisateurs.

PHP

PHP: Hypertext Preprocessor, plus connu sous son sigle PHP (sigle auto-référentiel), est un langage de programmation libre, principalement utilisé pour produire des pages Web dynamiques



via un serveur web, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. PHP est un langage impératif orienté objet.

PHP est un langage de script côté serveur, ce qui signifie qu'il permet de gérer la logique applicative directement sur le serveur, avant d'envoyer le contenu généré au navigateur de l'utilisateur. Cela offre plusieurs avantages :

- Sécurisation des traitements sensibles (accès à la base de données, authentification...)
- Génération dynamique des pages HTML
- Contrôle total sur les échanges entre le client et le serveur

Durant ces années de cours, PHP a été un langage enseigné et pratiqué dans plusieurs modules de développement web et d'applications. Ce choix permet de :

- Valoriser les compétences acquises en formation
- Garantir une meilleure maîtrise du langage et de ses bonnes pratiques
- Faciliter le travail en équipe et la relecture de code, car les membres du projet partagent cette base commune

PHP est parfaitement adapté pour interagir avec des API et des bases de données via des requêtes HTTP et SQL. Dans le cadre de ce projet :

- PHP pourra effectuer des requêtes vers des API externes (REST, JSON) pour récupérer ou envoyer des données
- Gérer la connexion et les opérations sur le serveur de base de données (MySQL, PostgreSQL...) via des extensions comme PDO ou MySQLi
- Centraliser la logique serveur et les échanges de données, assurant ainsi une communication fluide entre le front-end et le back-end

Chaîne de développement

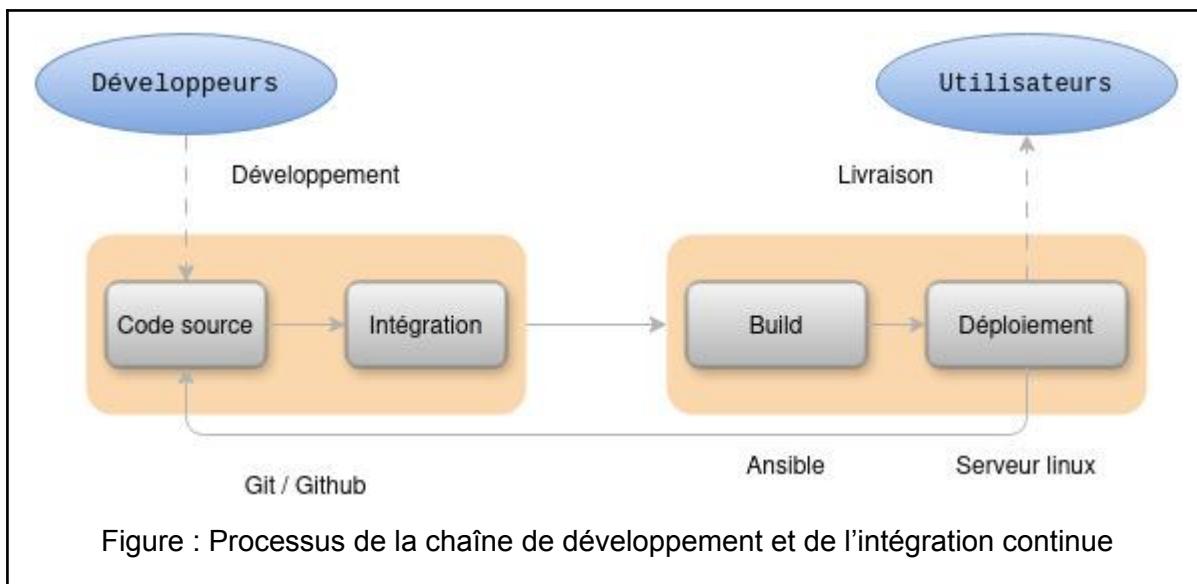
La chaîne de développement rassemble l'ensemble des outils et des étapes nécessaires à la conception, à la validation et à la mise en production de notre projet et plus précisément de notre API. Elle permet d'automatiser et de standardiser les tâches de développement.

En effet, toutes nos productions logicielles seront directement déployées sur le serveur Raspberry Pi. Même pour le développement, rien ne sera exécuté en local, c'est ce que l'on appelle de l'intégration continue.

Intégration continue

Définition : L'intégration continue est une méthode de développement de logiciel DevOps par laquelle les développeurs intègrent régulièrement leurs modifications de code à un référentiel centralisé, suite à quoi des opérations de création et de test sont automatiquement exécutées.

Comme expliqué précédemment l'usage de Github pour héberger le code et de Ansible pour déployer un service à distance sera utilisé. Ainsi, à chaque nouvelle modification sur le code, tous les collaborateurs fusionnent leur code automatiquement et ce dernier est déployé sur le serveur.

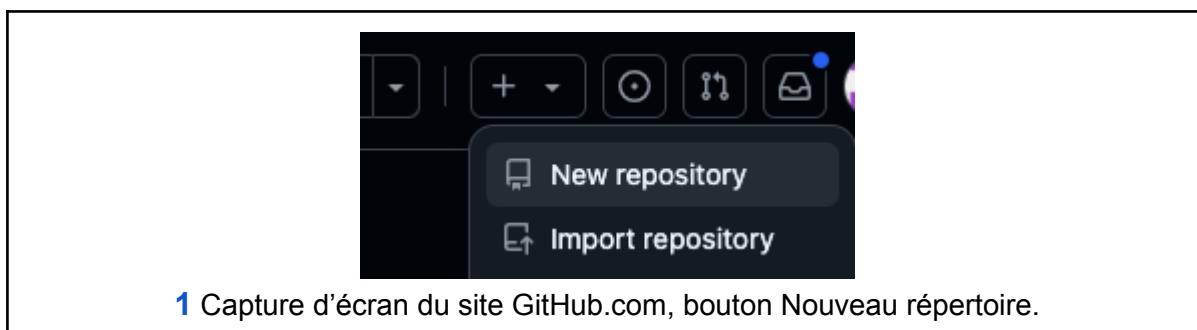


Répertoire GitHub

Le répertoire GitHub rassemble l'ensemble des fichiers de développement du projet : le site, les applications Qt et l'application mobile.

Toutes nos productions logicielles sont hébergées sur les serveurs de GitHub via notre répertoire

Nous créons donc un répertoire GitHub pour le projet, avec un nom personnalisé puis on invite les membres du projets via leurs nom d'utilisateurs GitHub et enfin nous “clonons” le répertoire à l'aide de GitHub Desktop afin de maintenir le projet.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

Owner * Repository name *

/
PoleInfo_rapport is available.

Great repository names are short and memorable. Need inspiration? How about [musical-potato](#) ?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file
This is where you can write a long description for your project. [Learn more about READMEs](#)

2 Capture d'écran du site GitHub.com, page de création de répertoire



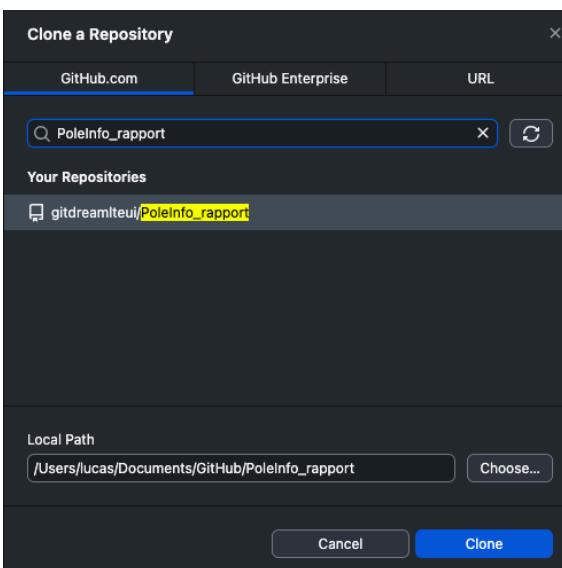
Add collaborators to this repository

Search for people using their GitHub username or email address.

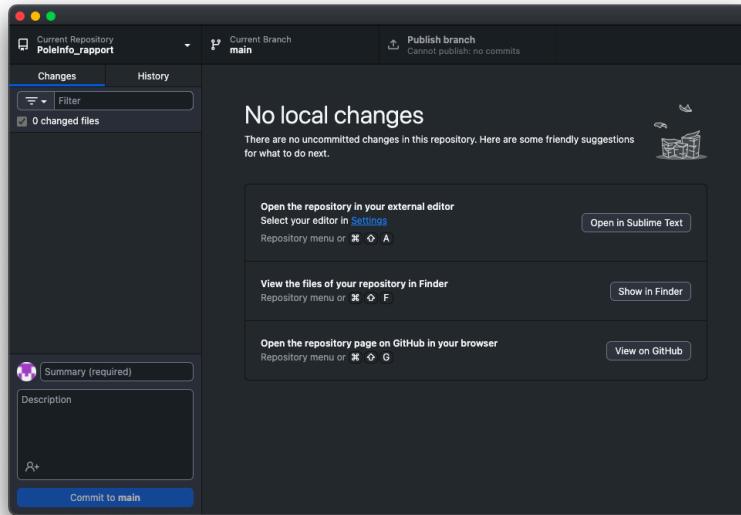
Invite collaborators

3 Capture d'écran du site GitHub.com, option d'ajout de collaborateurs*

*Les membres du projet ont un accès en écriture au répertoire du projet une fois devenus collaborateurs sur GitHub



4 Capture d'écran de GitHub Desktop, clonage d'un répertoire



5 Capture d'écran de GitHub Desktop, page d'un répertoire

Ainsi, le répertoire GitHub est prêt à faire feu à héberger les premières lignes de code et permettre à Ansible de récupérer le code source.

Configuration service

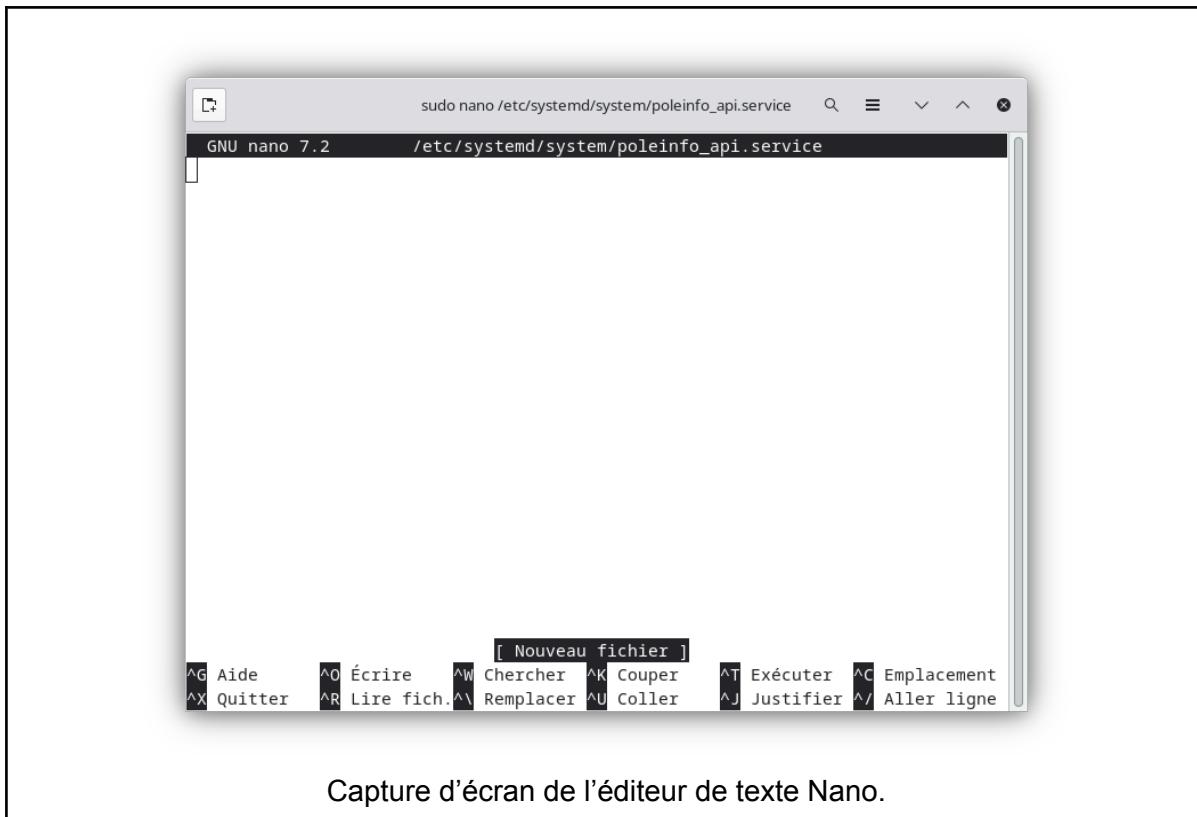
Afin d'utiliser correctement le Raspberry en tant que serveur il était obligatoire de configurer l'API en tant que **service systemd** du serveur.

systemd est le gestionnaire de services et de démarrage pour les systèmes Linux, utilisé par défaut sur la plupart des distributions notamment *Raspberry Pi OS* basé sur *Debian*. Il se lance automatiquement lors de la première étape de démarrage du système et prend en charge l'activation et la gestion des services. En tant que premier processus du système (PID 1), systemd contrôle le fonctionnement des services et reste actif jusqu'à l'arrêt du système.

Premièrement on créer un fichier service :

```
$ sudo nano /etc/systemd/system/poleinfo_api.service
```

On se retrouve ensuite sur l'utilitaire **nano** qui nous permet d'éditer le fichier.



Capture d'écran de l'éditeur de texte Nano.

On y écrit ensuite notre fichier de service avec toutes les informations nécessaires.

```
[Unit]
Description=PoleInfo API
After=network.target

[Service]
User=user
WorkingDirectory=/var/www/html/PoleInfo/poleinfo_api
ExecStart=/var/www/html/PoleInfo/poleinfo_api/venv/bin/python main.py
Restart=always

[Install]
WantedBy=multi-user.target
```

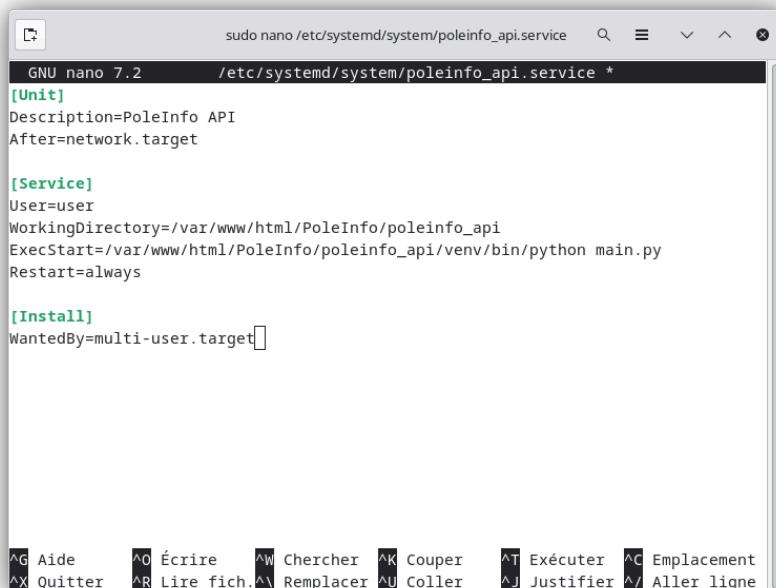
La ligne `After=network.target` indique au systemd de démarrer ce service uniquement après l'initialisation du réseau.

`WorkingDirectory=/var/www/html/PoleInfo/poleinfo_api` indique le répertoire dans lequel le processus sera exécuté (notamment où se trouve le `main.py`)

ExecStart=/var/www/html/PoleInfo/poleinfo_api/venv/bin/python main.py → correspond à l'instruction d'exécution de l'API. On exécute main.py avec un interpréteur virtuel Python venv.

Restart=always → Si le processus plante alors systemd essaiera de le redémarrer automatiquement.

WantedBy=multi-user.target → Le service sera activé au démarrage du système (niveau multi-utilisateur, sans interface graphique).



```
sudo nano /etc/systemd/system/poleinfo_api.service
[Unit]
Description=PoleInfo API
After=network.target

[Service]
User=user
WorkingDirectory=/var/www/html/PoleInfo/poleinfo_api
ExecStart=/var/www/html/PoleInfo/poleinfo_api/venv/bin/python main.py
Restart=always

[Install]
WantedBy=multi-user.target
```

On sauvegarde enfin avec CTRL+X et O.

On active ensuite le processus manuellement la première fois :

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable poleinfo_api
$ sudo systemctl start poleinfo_api
```

On peut vérifier avec :

```
$ sudo systemctl status poleinfo_api
```

Configuration ansible

Une fois le service configuré et le répertoire Github créer il ne reste plus qu'à installer et configurer Ansible.

Comme expliqué précédemment, Ansible possède ce que l'on appelle des inventaires et des playbooks.

Inventaire : fichier listant les machines cibles (au format INI, YAML ou dynamique).

Playbook : fichier YAML décrivant les tâches à effectuer.

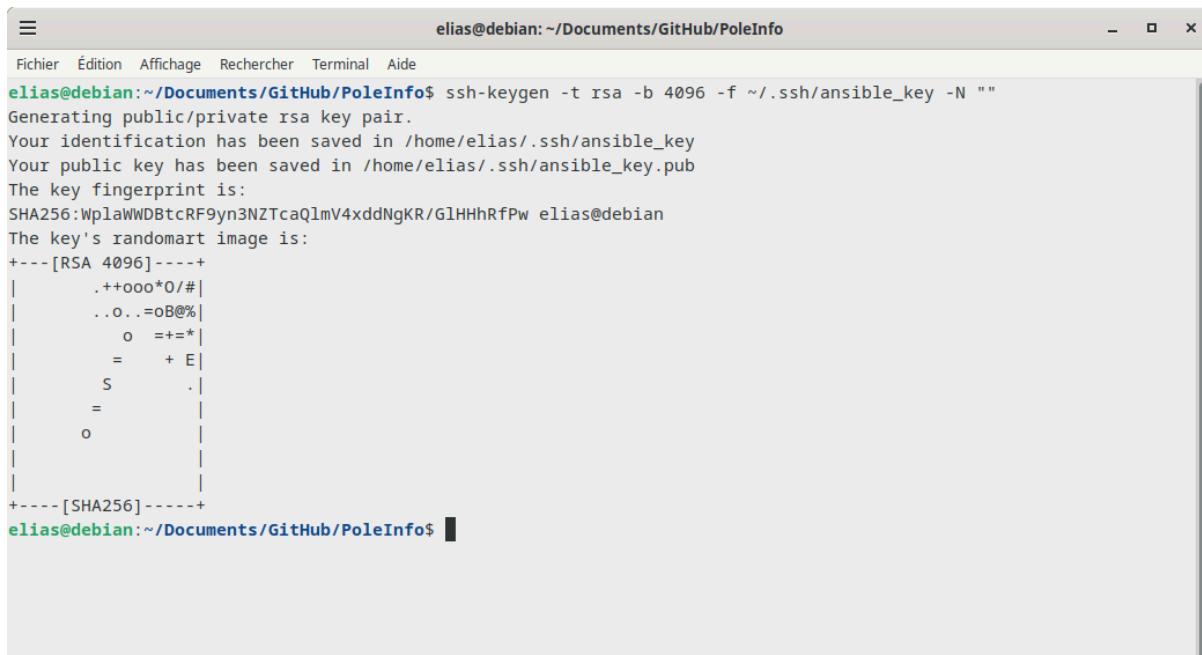
Ansible utilise ssh afin de se connecter aux machines cibles, il est donc nécessaire d'activer et de créer des clés ssh depuis le serveur.

Installation et activation de ssh

```
sudo apt install openssh-server
sudo systemctl enable --now ssh
```

Créer la clé :

```
ssh-keygen -t rsa -b 4096 -f ~/.ssh/ansible_key -N ""
```



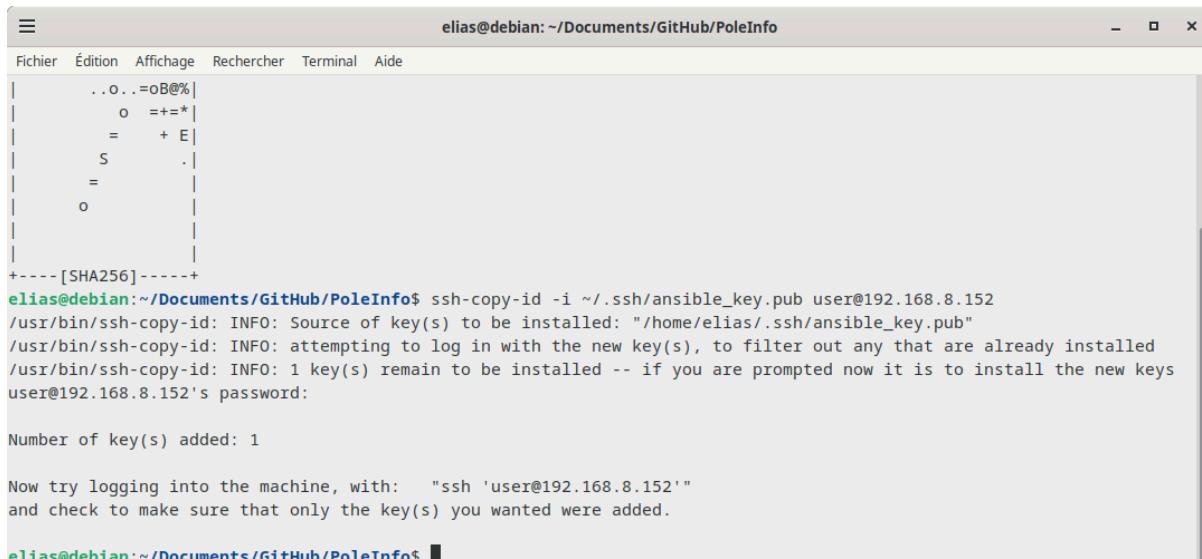
The screenshot shows a terminal window titled "elias@debian: ~/Documents/GitHub/PoleInfo". The user has run the command "ssh-keygen -t rsa -b 4096 -f ~/.ssh/ansible_key -N """. The terminal displays the following output:

```
Fichier Édition Affichage Rechercher Terminal Aide
elias@debian:~/Documents/GitHub/PoleInfo$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/ansible_key -N ""
Generating public/private rsa key pair.
Your identification has been saved in /home/elias/.ssh/ansible_key
Your public key has been saved in /home/elias/.ssh/ansible_key.pub
The key fingerprint is:
SHA256:WplaWWDBtcRF9yn3NZTcaQlmV4xddNgKR/GlHHhRfPw elias@debian
The key's randomart image is:
+---[RSA 4096]----+
| .++ooo*0/#|
| ..o..=oB@%|
| o  ==+=*|
| =    + E|
| S      .|
| =        |
| o        |
|           |
+---[SHA256]----+
elias@debian:~/Documents/GitHub/PoleInfo$
```

Donner la clé publique au Raspberry

```
ssh-copy-id -i ~/.ssh/ansible_key.pub <USER>@192.168.X.X
```

en remplaçant par le compte du raspberry pi ainsi que son adresse IP.



```

elias@debian: ~/Documents/GitHub/PoleInfo
Fichier Édition Affichage Rechercher Terminal Aide
| ..o..=oB@%|
| o   +=*|
| =   + E|
| S   .|
| =   |
| o   |
|     |
|     |
+---[SHA256]---+
elias@debian:~/Documents/GitHub/PoleInfo$ ssh-copy-id -i ~/.ssh/ansible_key.pub user@192.168.8.152
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/elias/.ssh/ansible_key.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
user@192.168.8.152's password:

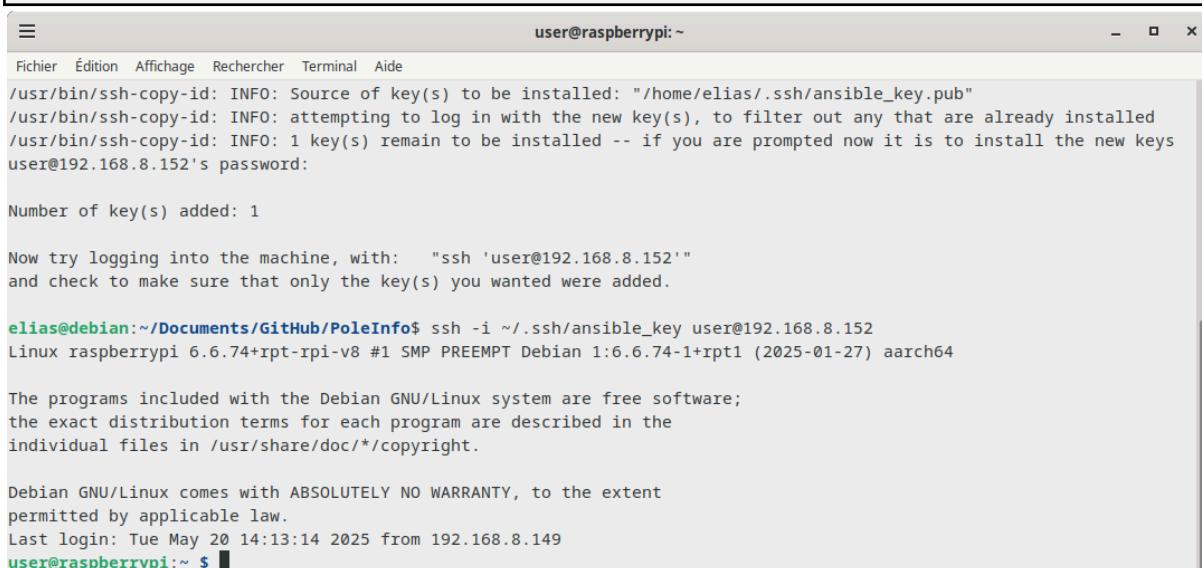
Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'user@192.168.8.152'"
and check to make sure that only the key(s) you wanted were added.

elias@debian:~/Documents/GitHub/PoleInfo$ 
```

Tester la connexion :

```
ssh -i ~/.ssh/ansible_key <USER>@192.168.X.X
```



```

user@raspberrypi: ~
Fichier Édition Affichage Rechercher Terminal Aide
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/elias/.ssh/ansible_key.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
user@192.168.8.152's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'user@192.168.8.152'"
and check to make sure that only the key(s) you wanted were added.

elias@debian:~/Documents/GitHub/PoleInfo$ ssh -i ~/.ssh/ansible_key user@192.168.8.152
Linux raspberrypi 6.6.74+rpi-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.74-1+rpt1 (2025-01-27) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue May 20 14:13:14 2025 from 192.168.8.149
user@raspberrypi:~ $ 
```

Enfin on peut créer nos fichiers :

inventory.ini

The screenshot shows a terminal window titled "nano inventory.ini". The file content is:

```
GNU nano 7.2                               inventory.ini
[api_poleinfo]
192.168.8.152 ansible_user=user ansible_ssh_private_key_file=~/ssh/ansible_key
```

The status bar at the bottom indicates "[Lecture de 2 lignes]".

deploy.yml (playbook)

The screenshot shows a terminal window titled "nano deploy.yml". The file content is:

```
GNU nano 7.2                               deploy.yml
- name: Déployer l'API sur le PC distant
  hosts: api_poleinfo
  tasks:
    - name: Récupérer le dernier code depuis GitHub
      ansible.builtin.git:
        repo: "git@github.com:gitdreamteui/PoleInfo.git"
        dest: "/var/www/html/PoleInfo"
        version: main
        accept_hostkey: yes
        force: yes

    - name: Redémarrer l'API
      ansible.builtin.systemd:
        name: poleinfo_api
        state: restarted
        enabled: yes
        become: yes
```

The status bar at the bottom indicates "[Lecture de 18 lignes]".

On définit 2 tâches, la première étant la récupération du code source depuis Github vers notre répertoire de travail et le second le redémarrage de l'API avec le nouveau code source.

A chaque modification du code et après chaque commit, le développeur devra exécuter la commande `ansible-playbook -i inventory.ini deploy.yml`



The screenshot shows a terminal window titled "elias@debian: ~/Documents/GitHub/PoleInfo". The terminal displays the output of an Ansible playbook run:

```
elias@debian:~/Documents/GitHub/PoleInfo$ ansible-playbook -i inventory.ini deploy.yml

PLAY [Déployer l'API sur le PC distant] ****
TASK [Gathering Facts] ****
ok: [192.168.8.152]

TASK [Récupérer le dernier code depuis GitHub] ****
changed: [192.168.8.152]

TASK [Redémarrer l'API] ****
changed: [192.168.8.152]

PLAY RECAP ****
192.168.8.152 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

elias@debian:~/Documents/GitHub/PoleInfo$
```

Ansible récupèrera donc le nouveau code du Github et le déploiera sur le serveur.

Répartition des tâches

Lucas GUILLOTEAU

- Installation et Configuration du matériel (Raspberry Pi, Afficheur Ligne)
- Développement logiciel C++
- Développement de l'application Mobile

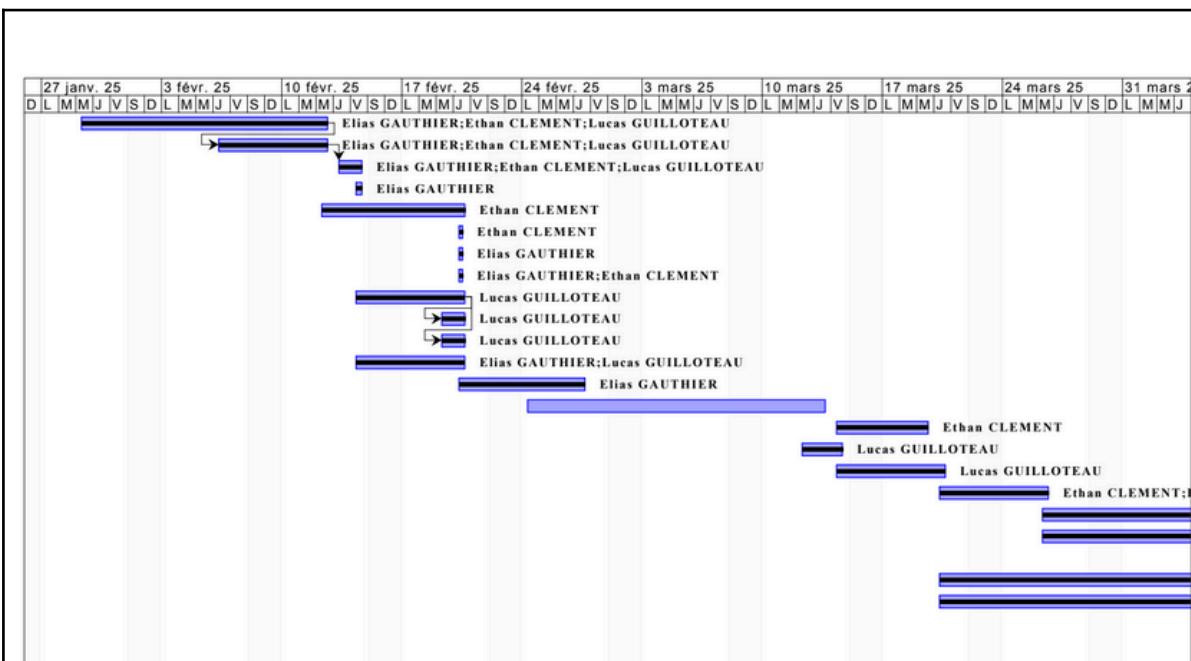
Elias GAUTHIER

- Création et Gestion de l'API
 - Crédit de tokens d'authentification
 - Chiffrement de mots de passe avec la base de données
 - Créations des endpoints liant méthodes HTTP et Requêtes SQL
- Développement de l'interface utilisateur
 - Ajout de réservations
 - Suppression de réservations
 - Modification de réservations
- Gestion de l'intégration continue et de la chaîne de développement
 - Automatisation du déploiement du code du GitHub sur le serveur
- Développement de l'interface visiteur
 - Frontend

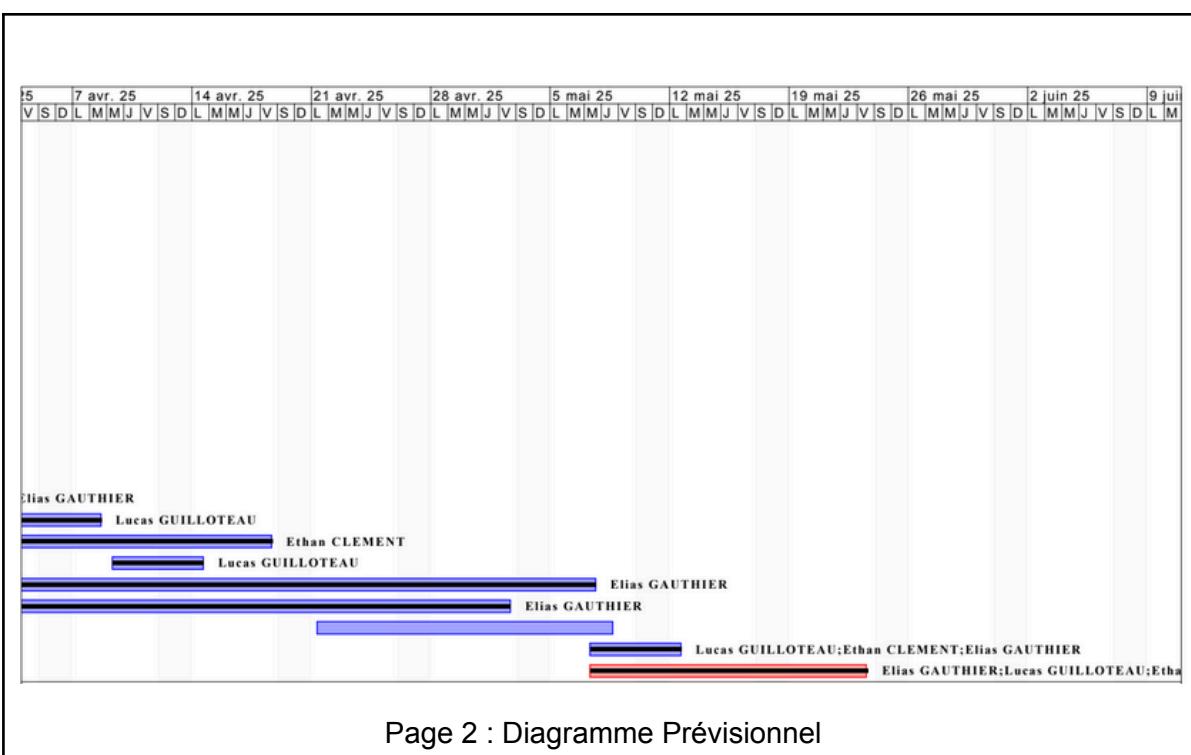
Ethan CLEMENT

- Crédit/Gestion de la base de données
 - Construction intégrale de la base de données
- Développement de l'interface Administrateur
 - Ajout d'utilisateurs
 - Suppression d'utilisateurs
 - Gestion de tous les paramètres de réservations
 - Gestion de sauvegardes de la base de données
- Développement de l'interface Visiteur
 - Affichage des réservations par ordre croissant

Diagramme prévisionnel



Page 1 : Diagramme Prévisionnel



Page 2 : Diagramme Prévisionnel

Note : On remarque ici une période rouge correspondant à la rédaction du rapport qui fut trop tardive.

	Nom	Durée	Début	Fin	Noms des ressources
1	Introduction et lecture	11 jours	29/01/2025 08:00	12/02/2025 17:00	Elias GAUTHIER;Ethan CLEMENT;Lucas GUILLOTEAU
2	Analyse du cahier des charges	5 jours	06/02/2025 08:00	12/02/2025 17:00	Elias GAUTHIER;Ethan CLEMENT;Lucas GUILLOTEAU
3	Répartition des tâches	2 jours	13/02/2025 08:00	14/02/2025 17:00	Elias GAUTHIER;Ethan CLEMENT;Lucas GUILLOTEAU
4	Mise en place environnement de projet	1 jour	14/02/2025 08:00	14/02/2025 17:00	Elias GAUTHIER
5	BDD et schéma relationnel	7 jours	12/02/2025 08:00	20/02/2025 17:00	Ethan CLEMENT
6	Diagramme de déploiement	0,625 jours	20/02/2025 08:00	20/02/2025 14:00	Ethan CLEMENT
7	Diagramme prévisionnel	0,625 jour...	20/02/2025 08:00	20/02/2025 14:00	Elias GAUTHIER
8	Diagramme de cas d'utilisations	0,625 jours	20/02/2025 08:00	20/02/2025 14:00	Elias GAUTHIER;Ethan CLEMENT
9	Installation des Raspberry Pi	5 jours	14/02/2025 08:00	20/02/2025 17:00	Lucas GUILLOTEAU
10	Installation PhpMyAdmin	2 jours	19/02/2025 08:00	20/02/2025 17:00	Lucas GUILLOTEAU
11	Installation serveur LAMPP	2 jours	19/02/2025 08:00	20/02/2025 17:00	Lucas GUILLOTEAU
12	Mise en oeuvre chaîne de développement	5 jours	14/02/2025 08:00	20/02/2025 17:00	Elias GAUTHIER;Lucas GUILLOTEAU
13	Début conception API	6 jours	20/02/2025 08:00	27/02/2025 17:00	Elias GAUTHIER
14	Vacances février	14 jours?	22/02/2025 08:00	13/03/2025 17:00	
15	Début conception page visiteur	4 jours	14/03/2025 08:00	19/03/2025 17:00	Ethan CLEMENT
16	Début application C++ d'affichage	3 jours	12/03/2025 08:00	14/03/2025 17:00	Lucas GUILLOTEAU
17	Prise en main afficheur ligne	5 jours	14/03/2025 08:00	20/03/2025 17:00	Lucas GUILLOTEAU
18	Refonte page visiteur	5 jours	20/03/2025 08:00	26/03/2025 17:00	Ethan CLEMENT;Elias GAUTHIER
19	Conception application C++ afficheur...	10 jours	26/03/2025 08:00	08/04/2025 17:00	Lucas GUILLOTEAU
20	Conception page admin	18 jours	26/03/2025 08:00	18/04/2025 17:00	Ethan CLEMENT
21	Conception application mobile	4 jours	09/04/2025 08:00	14/04/2025 17:00	Lucas GUILLOTEAU
22	Conception page user	35 jours	20/03/2025 08:00	07/05/2025 17:00	Elias GAUTHIER
23	Conception API	32 jours	20/03/2025 08:00	02/05/2025 17:00	Elias GAUTHIER
24	Vacances paques	14 jours	19/04/2025 07:00	08/05/2025 17:00	
25	Validation et test	4 jours	07/05/2025 08:00	12/05/2025 17:00	Elias GAUTHIER;Ethan CLEMENT;Elias GAUTHIER
26	Rédaction rapport	12,75 jours	07/05/2025 08:00	23/05/2025 15:00	Elias GAUTHIER;Lucas GUILLOTEAU;Ethan CLEMENT

Page 3 : Échéancier des tâches à réaliser

Évaluation du matériel

Auteurs de cette section :

- Elias GAUTHIER

Correspond au Cycle en V : **Conception générale**

Avant de commencer toute réalisation et développement, il est nécessaire de réaliser une évaluation de performance sur le matériel utilisé. En particulier sur le Raspberry Pi 3 Model B qui occupera le rôle de serveur.

Pour rappel, ce dernier assurera l'hébergement du **serveur web**, de la **base de données** et de l'**API**.

Objectif

Vérifier que le Raspberry Pi est capable d'héberger et d'exécuter de manière stable et performante les services requis, en respectant les critères de performance, de sécurité et de fiabilité. Le Raspberry Pi doit aussi pouvoir supporter normalement le déploiement et l'intégration continue.

Environnement de test

Raspberry Pi 3 Model B Rev 1.2		
Réseau	Hotspot wifi	Réseau ethernet
OS	Debian GNU/Linux 12 bookworm - aarch64 (dernière version <input checked="" type="checkbox"/>)	
Kernel	6.6.74+rpt-rpi-v8	
CPU	1.2GHz Quad-Core ARM Cortex-A53	
RAM	1 Go LPDDR2 (900 MHz)	
Connectivité	Ethernet: 10/100 Mbps	Wi-Fi: 802.11n (2.4 GHz)

Spécifications logicielles et matérielles du Raspberry Pi 3 Model B

Procédures et critères de test

Lorsque les services requis seront en exécutions sans interaction particulière avec ces derniers, cet état sera considéré comme charge *normale*. Nous considérons l'état en charge élevée lors d'un trafic continu sur le serveur avec plusieurs requêtes par minutes à l'API et clients sur le serveur web. Cet état sera évalué lors d'une seconde phase de l'évaluation post-conception lorsque tout sera opérationnel.

Nous effectuerons également des tests en charge très élevée avec des outils de benchmark pour voir comment le Raspberry réagirait face à de grosses charges.

Le Raspberry sera donc évalué sur les exigences suivantes :

- Charge CPU < 70% en charge normale
- Charge CPU < 90 % en charge élevée et très élevée
- Température CPU < 60° en charge normale
- Température CPU < 70° en charge élevée
- Température CPU < 80° en charge très élevée
- Utilisation RAM < 70% en charge normale
- Utilisation RAM < 80% en charge élevée
- Latence réseau < 10ms

Au-delà de 70~80 degrés en utilisation constante, il est possible de remarquer des baisses de performances significatives et cela peut engendrer une détérioration de la carte sur le long terme.

Utilitaires utilisés

Htop

Description

htop est un outil logiciel à utiliser en ligne de commande qui permet de superviser les ressources sur un serveur. Il est à l'origine développé exclusivement comme gestionnaire de tâches Linux, sous licence GNU, et remplit le même rôle que la commande **top** permettant ainsi aux processus actifs d'être affichés sur les systèmes UNIX.

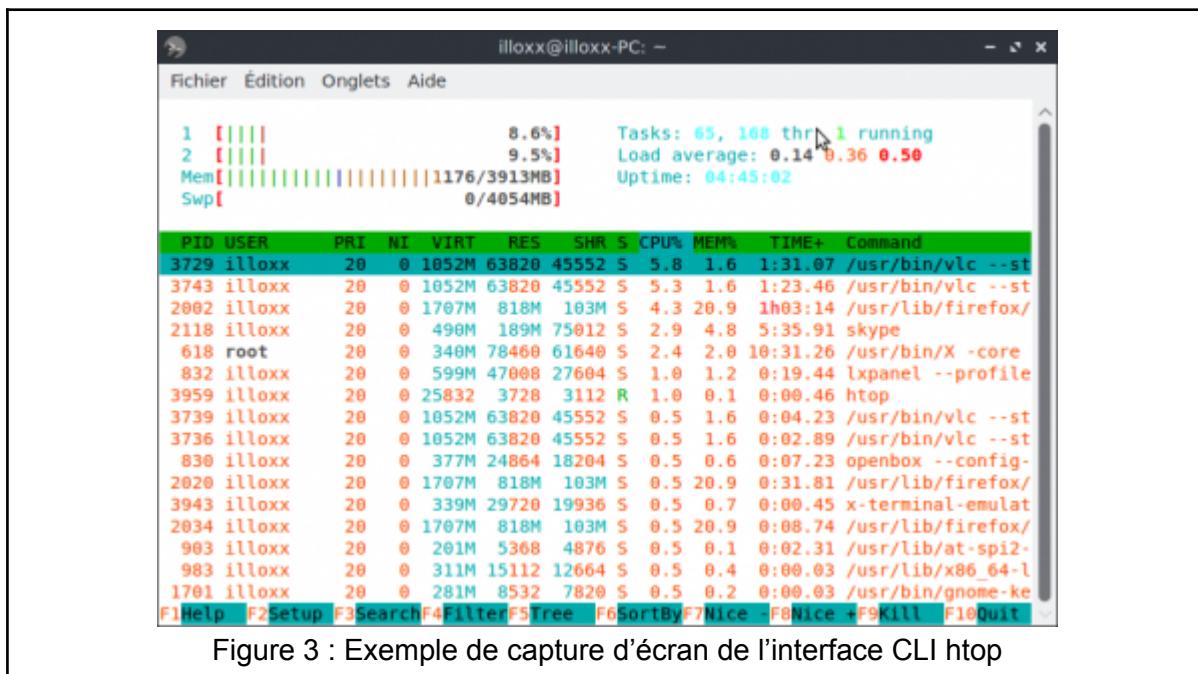


Figure 3 : Exemple de capture d'écran de l'interface CLI htop

On remarque 3 parties distinctes sur l'interface : **l'en-tête, la zone principale, et le pied de page.** Dans notre cas, la zone qui nous sera le plus intéressante sera l'en-tête qui contient les utilisations des ressources des coeurs du processeurs et de la mémoire RAM.

Installation

Pour installer `htop` sur notre serveur on utilisera les commandes suivante :

```
$ wget https://hisham.hm/htop/releases/2.0.1/htop-2.0.1.tar.gz
$ tar -xvf htop-2.0.1.tar.gz
$ cd htop-2.0.1/
$ ./configure
$ make #make install
```

`vcgencmd`

Description

`vcgencmd` est une commande spécifique aux Raspberry Pi permettant d'accéder à des informations système et matérielles. Elle est utilisée pour interroger l'état du GPU, de la température, du voltage, de la fréquence et d'autres paramètres internes.

Cette commande est souvent utilisée pour le diagnostic ou le monitoring de l'appareil. Elle nécessite généralement des privilèges d'administration pour fonctionner correctement.

```
chewett@bunker-master:~$ vcgencmd measure_volts core
volt=1.2000V
chewett@bunker-master:~$ vcgencmd measure_temp
temp=42.2'C
chewett@bunker-master:~$ vcgencmd get_mem cpu
cpu=0M
chewett@bunker-master:~$ vcgencmd get_mem gpu
gpu=64M
chewett@bunker-master:~$ vcgencmd get_mem arm
arm=448M
chewett@bunker-master:~$ vcgencmd measure_temp
temp=41.7'C
chewett@bunker-master:~$ vcgencmd measure_volts core
volt=1.2000V
chewett@bunker-master:~$ vcgencmd version
Apr 27 2017 17:19:34
Copyright (c) 2012 Broadcom
version 17af5814bb19dbb7c70cccd2c845b80a160943811 (clean) (release)
```

Figure 4 : Exemple de capture d'écran des différentes options de l'utilitaire `vcgencmd`

On utilisera principalement l'argument `measure_temp` afin de relever la température du CPU du serveur lors des tests.

Installation

Pour installer `vcgencmd` sur notre serveur on utilisera les commandes suivante :

```
$ sudo apt update && sudo apt upgrade
$ sudo apt install libraspberrypi-bin
```

Résultats

Charge normale avec Apache, phpmyadmin et une API rudimentaire

Usage de l'utilitaire *htop* avec la commande : *htop*
et de *vcgencmd* avec la commande : *vcgencmd measure_temp*

Charge CPU moyenne	8.1%
Utilisation RAM	300Mo (~33%)
Température	53.7°C

```

e[|          0.6%] Tasks: 88, 147 thr, 116 kthr; 1 running
1[          0.0%] Load average: 0.38 0.31 0.23
2[||| 3.9%] Uptime: 12:56:53
3[|       0.6%
Mem[||||| 300M/907M]
Swp[||||| 319M/512M]

```

```
user@raspberrypi:~ $ vcgencmd measure_temp
temp=53.7'C
```

Commentaire : En charge normale (quasiment au repos) sans sollicitations sur le serveur les résultats correspondent aux exigences que l'on a fixées sans aucun problème apparent.

Charge très élevée

CPU	Utilitaire	stress-ng
	Commande	stress-ng --cpu 2 --vm 2 --vm-bytes 256M --hdd 1 --timeout 60s --metrics
	Résultat	95% d'utilisation par cœur pendant une minute. température relevée : 80.1°C

Commentaire : En charge très élevée, les résultats sont à la limite des exigences posées précédemment. Avec une température de 80.1°C, les composants électroniques du Raspberry Pi 3 sont exposés à un potentiel endommagement bien qu'un tel état sera rarement prolongé. On peut ainsi déduire des résultats intermédiaires aux deux dernières évaluations lors d'une utilisation en charge élevée. La mise en place d'un refroidissement par dissipateur thermique sera peut-être envisagée au cours du développement.

Suite → [Conception détaillée](#)

Conception détaillée

Notes : Cette section ne contient que les tâches que j'ai personnellement réalisées. Toutes les autres réalisations sont documentées dans les rapports de [Ethan CLEMENT](#) et de [Lucas GUILLOTEAU](#). Seule la partie de la base de données est conservée afin de rendre plus compréhensible le développement de l'API qui demeure en étroite collaboration avec la BDD.

Développement et mise en œuvre

Base de données

Auteur(s) de cette section :

- Ethan CLEMENT

Correspond au Cycle en V : [Conception détaillée](#)

Une des réflexions et réalisations importantes de ce projet est la base de données. En effet la fonction principale du projet Pôle Info est de réserver des créneaux, une salle pour les étudiants et les afficher. Pour cela il faut enregistrer des informations ce qui implique ainsi une base de données.

Cette base de données doit comporter au minimum une table pour enregistrer les réservations. Les informations requises pour une réservation sont :

- La matière du cours
- La salle dans lequel le cours est réservé
- L'horaire du cours
- Le nom du professeur ayant réservé (celui faisant le cours)
- La date
- Des informations supplémentaires potentielles

Le nom et prénom du professeur réservant sont récupérés de son compte utilisateur. En toute logique, chaque professeur possède un compte pour se connecter et créer des réservations. Cela demande de créer une autre table pour y intégrer les informations de connexions.

Les réservations sont effectuées par des professeurs. Néanmoins, nous devons uniformiser les données pouvant être saisies, pour simplifier l'ordre et la saisie. Mais aussi pour éviter qu'une même information soit rentrée sous deux noms différents.

Exemple : Deux réservations ayant une matière saisie l'une "maths" et l'autre "Mathématiques".

L'uniformisation des données permet la simplification de traitement et aussi permet à l'administrateur de modifier le nom de certaines matières si elles viennent à changer. Cela vaut aussi pour le nom des classes ou des groupes d'élèves, le nom des salles et les créneaux du lycée.

En conséquence, il implique la création de plusieurs tables supplémentaires où la page de réservation cherche les informations à saisir.

Les tables supplémentaires à ajouter sont :

Une table matière contenant le nom des matières.

matiere		
Clé Primaire	id_matiere	int
	nom	varchar

Une table salle contenant le nom, la capacité, et le type de salle.

salle		
Clé Primaire	id_salle	int
	nom	varchar
	capacite	int
	type	varchar

Une table user contenant l'identifiant, le mot de passe, le nom, le prénom et le type de compte. Si il est simplement utilisateur ou administrateur.

user		
Clé Primaire	id_user	int
	login	varchar
	passwd	varchar
	type	bool
	nom	varchar
	prenom	varchar

Une table contenant tous les créneaux du lycée.

creneau		
Clé Primaire	id_creneau	int
	heure_debut	time

Une table contenant le nom des classes et des groupes.

classe_grp		
Clé Primaire	id_classe_grp	int
	nom	varchar

Naturellement les éléments suivants de la tables réservations deviennent des clé étrangères de leur table respective :

reservation		
Clé Primaire	id_reservation	int
Clé Étrangère	id_salle	int
Clé Étrangère	id_matiere	int
Clé Étrangère	id_creneau	int
Clé Étrangère	id_user	int
	duree	float
	date	date
	info	varchar

Vous pouvez remarquer qu'il n'y pas de champs pour les classes car il faut pouvoir ajouter plusieurs classes dans une même réservation. Dans le cas échéant une clé étrangère ne suffit pas qu'elle permet d'ajouter qu'une seule valeur de la table classe.

Il faut créer une table de jonction comportant une clé étrangère de la table réservation et une clé étrangère de la table classe. Ces deux mêmes clés étrangères formant la clé primaire de la table de jonction.

classe_reservation		
Clé Étrangère	id_reservation	int
Clé Étrangère	id_classe_grp	int

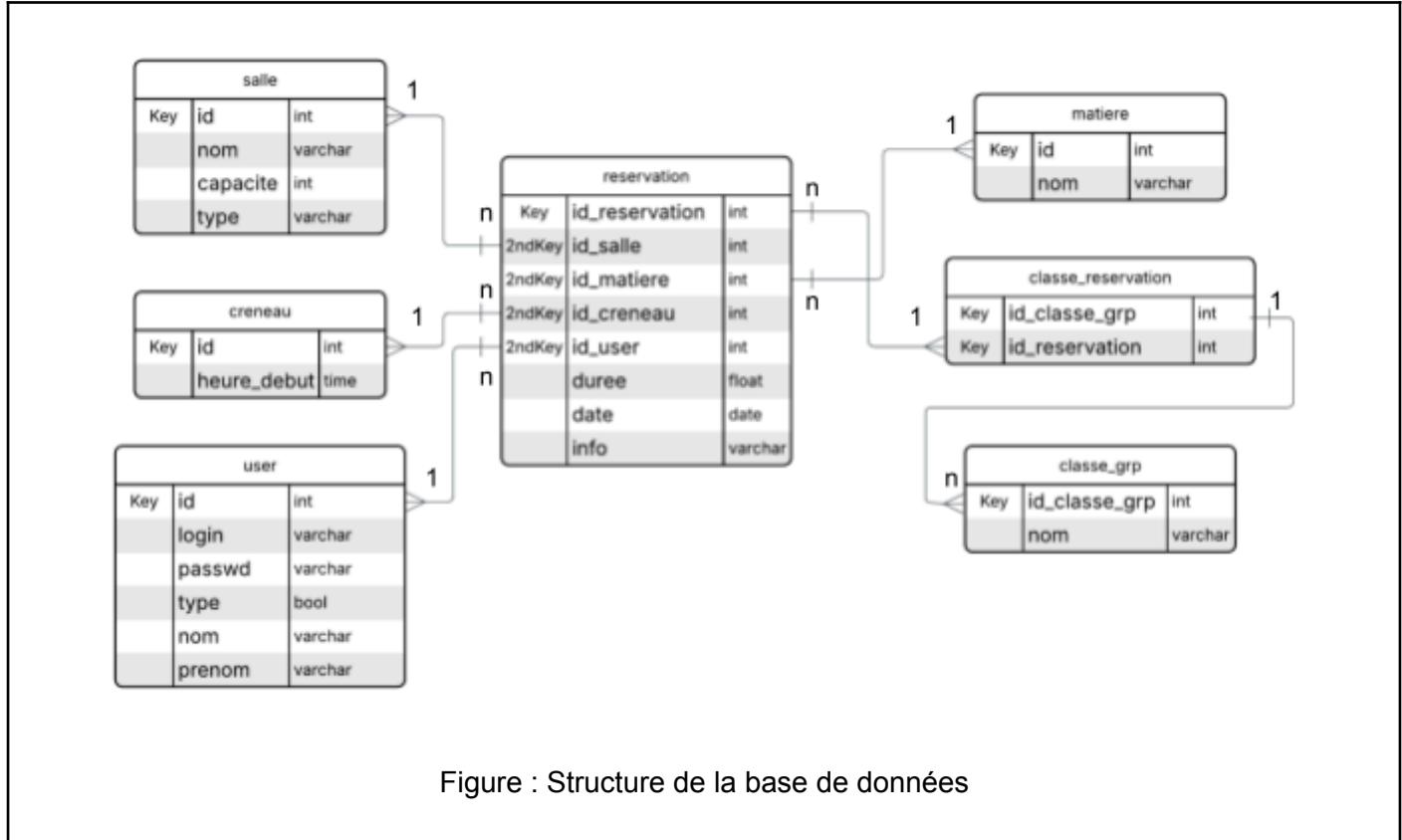


Figure : Structure de la base de données

Afin d'interagir avec ces données nous avons créé une API.

Application Programming Interface (Elias GAUTHIER)

Auteur(s) de cette section :

- Elias GAUTHIER

Correspond au Cycle en V : **Réalisation**

Introduction aux APIs

Une API (*application programming interface* ou « interface de programmation d'application ») est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

Introduite dès les années 1970 avec IBM, les API sont couramment utilisées dans l'univers informatique. Elles sont devenues l'un des piliers fondamentaux du développement logiciel moderne et on les retrouve quasiment partout chez les GAFAM.

Initialement, il n'était pas demandé de concevoir une API pour ce projet. En effet, le développement d'une API correctement construite pour être assez long et fastidieux mais lorsque tout est en place, elle facilite grandement le développement des applications et services liés au projet.

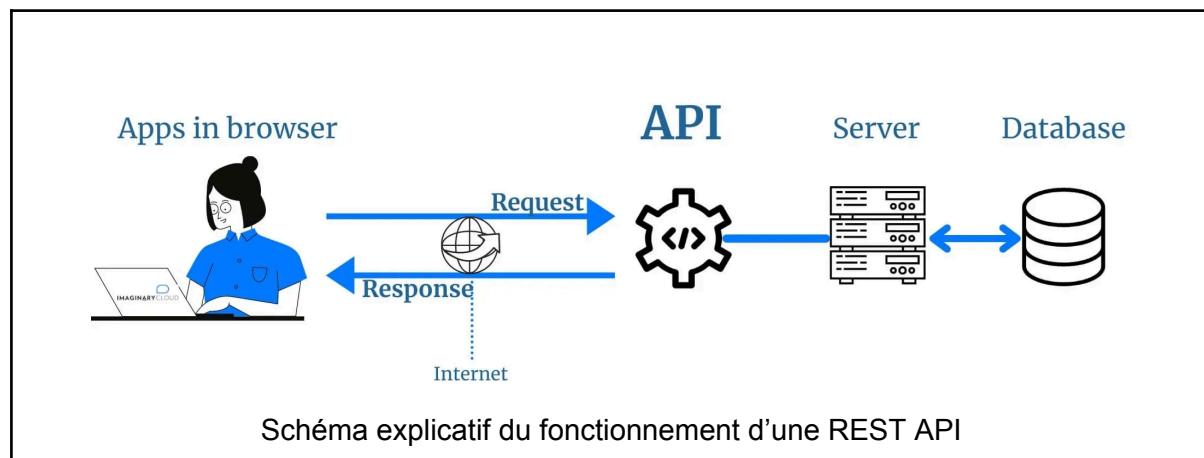
Il existe plusieurs types d'API. D'abord selon l'accessibilité :

1. API publique (Accessible à tout le monde, souvent via un portail développeur)
2. API privée (Utilisée uniquement en interne)
3. API partenaire

Puis en fonction de l'architecture :

1. API REST (Representational State Transfer)
2. API SOAP (Simple Object Access Protocol)
3. API GraphQL
4. API gRPC (Google Remote Procedure Call)

	REST	SOAP	GraphQL	gRPC
Avantages	<ul style="list-style-type: none"> - Simple à comprendre et à utiliser - verbes HTTP standards - Documentation facile - JSON ou XML 	<ul style="list-style-type: none"> - Très structuré et standardisé - Supporte les transactions, le contrôle d'erreur, les accusés de réception. 	<ul style="list-style-type: none"> - Requêtes très ciblées - Peu de requêtes 	<ul style="list-style-type: none"> - Très rapide et performant (HTTP/2 + Protobuf). - Faible consommation réseau (messages binaires)
Inconvénients	<ul style="list-style-type: none"> - Pas de standard strict - Implémentations variables - Peut vite devenir mal optimisée 	<ul style="list-style-type: none"> - Moins flexible que REST. - Peu adapté aux applications web modernes. 	<ul style="list-style-type: none"> - Peut devenir complexe à implémenter et à maintenir. - Requêtes complexes 	<ul style="list-style-type: none"> - Moins lisible - Outils de débogage/développement moins répandus - Prise en main très complexe



Comme vous l'aurez compris, mon API sera **privée** et elle servira de “passerelle” pour les données entre nos interfaces web PHP et la base de données MySQL présentée précédemment.

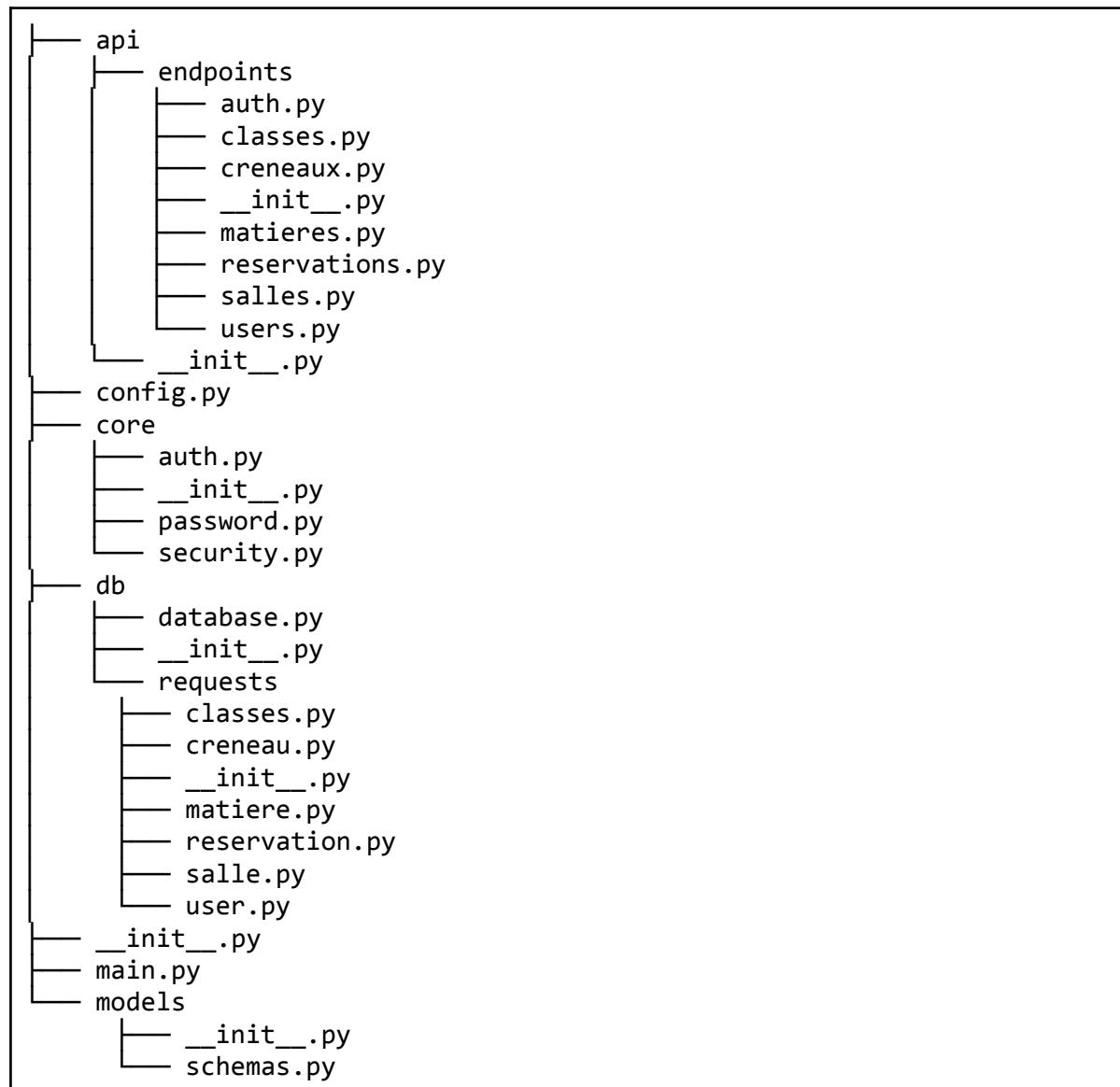
Après comparaison des différentes architectures j'ai choisi d'adopter une architecture **REST** de part sa simplicité et sa popularité. De plus, la documentation automatique est un atout permettant de gagner un temps considérable.

Une REST API utilise les méthodes HTTP classiques dont voici les principales : GET, POST, DELETE, PUT. Comme dans toute API nous possèderons des “routes” ou des “endpoints” (point de terminaison) et chaque route correspond à une ressource.

Voici les usages des différentes routes :

GET	Lire une ressource	GET /users/1 → récupère l'utilisateur 1
POST	Créer une ressource	POST /users + données utilisateur
PUT	Mettre à jour une ressource entière	PUT /users/1 + données complètes
DELETE	Supprimer une ressource	DELETE /users/1

Architecture de l'API



Comme vous pourrez le constater, j'ai construit mon API selon les SoC ("Separation of concerns" ou Séparation des préoccupations).

Définition : La séparation des préoccupations (ou séparation des responsabilités), traduction du concept d'informatique théorique *separation of concerns* (SoC) en anglais, est un principe de conception visant à segmenter un programme informatique en plusieurs parties, afin que chacune d'entre elles isole et gère un aspect précis de la problématique générale. C'est une bonne pratique largement reconnue et mise en œuvre dans l'ingénierie logicielle.

Source : [wikipedia.org](https://en.wikipedia.org/wiki/Separation_of_concerns)

/db	/core	/api/endpoints	/models
Contient le fichier de connexion à la base de données ainsi que chaque requête SQL par ressource.	Ensemble des opérations liées à l'authentification, hachage de mot de passe etc	Contient l'ensemble des routes.	Modèles Pydantic qui définissent la structure, les types et les validations des données attendues pour les accès, modifications etc des ressources.

Pour chaque endpoint, nous utiliserons des éléments de chaque répertoire.

A noter que l'on retrouve dans chaque répertoire un fichier `__init__.py` dont son objectif est simplement de rendre importable son répertoire courant en tant que module/bibliothèque Python.

L'API contient alors une route pour chaque ressource avec leurs méthodes :

- /reservations/ `GET POST DELETE PUT`
- /utilisateurs/ `GET POST DELETE`
- /matieres/ `GET POST DELETE`
- /creneaux/ `GET POST DELETE`
- /salle/ `GET POST DELETE`
- /classes/ `GET POST DELETE`
- /token/ `POST`
- /verify-token/ `GET`

Point d'entrée

A la genèse de l'API, le point d'entrée `main.py`. C'est ce fichier qui sera exécuté pour lancer l'API. La première chose à faire étant d'importer la bibliothèque FastAPI puis les routes.

```
"""
PoleInfo - API
-----
Auteur : Elias GAUTHIER
Date de création : 17/05/2025
Description : Point d'entrée de l'API de gestion des réservations de
salles Pôle Info
"""

from fastapi import FastAPI
From api.endpoints import auth, users, reservations, creneaux, salles,
matieres, classes
from datetime import datetime
import locale
import uvicorn

/main.py
```

- Importation de fastapi pour par la suite créer l'instance de l'application.
- Appel de chaque endpoint de notre API

Le module `datetime` est importé pour gérer les dates/heures et `locale` est importé pour gérer les paramètres régionaux.

Le module `uvicorn` est un serveur ASGI ultra performant conçu pour exécuter des applications web asynchrones (ce qui est le cas de notre API)

```
app = FastAPI(  
    title="PoleInfo API",  
    description=""  
    API de gestion des réservations de salles  
  
    Cette API permet la gestion complète des réservations de salles  
pour le Pôle Info.
```

Fonctionnalités principales

- Authentification des utilisateurs
- Gestion des utilisateurs (création, consultation, suppression)
- Gestion des réservations de salles
- Gestion des salles, créneaux, matières et classes

```
Développé par Elias GAUTHIER avec la co-conception d'Ethan CLEMENT  
"""",  
version="1.0.0",  
contact={  
    "name": "Pôle Info",  
    "email": "elias.gauthier@lp2i-poitiers.fr"  
}  
)
```

poleinfo_api/main.py

Cette partie crée l'instance principale de l'application FastAPI avec tous les paramètres utiles à la documentation automatique. Une fois notre instance *app* créée nous pouvons inclures nos **routes**.

```
# Inclusion des routeurs pour organiser les endpoints par domaine  
app.include_router(auth.router)  
app.include_router(users.router, prefix="/utilisateurs", tags=["utilisateurs"])  
app.include_router(reservations.router, prefix="/reservations",  
tags=["reservations"])  
app.include_router(creneaux.router, prefix="/creneaux", tags=["creneaux"])  
app.include_router(salles.router, prefix="/salles", tags=["salles"])  
app.include_router(matieres.router, prefix="/matieres", tags=["matieres"])  
app.include_router(classes.router, prefix="/classes", tags=["classes"])
```

poleinfo_api/main.py

On ajoute également notre point d'entrée à la racine (/) avec un message de bienvenue et l'heure :

```

@app.get("/", tags=["accueil"])
def read_root():
    """
    Point d'entrée principal de l'API qui affiche un message de bienvenue
    avec la date du jour au format français.
    """

    locale.setlocale(locale.LC_TIME, 'fr_FR.UTF-8')
    now = datetime.now()
    formatted_date = now.strftime("%d %B %Y")

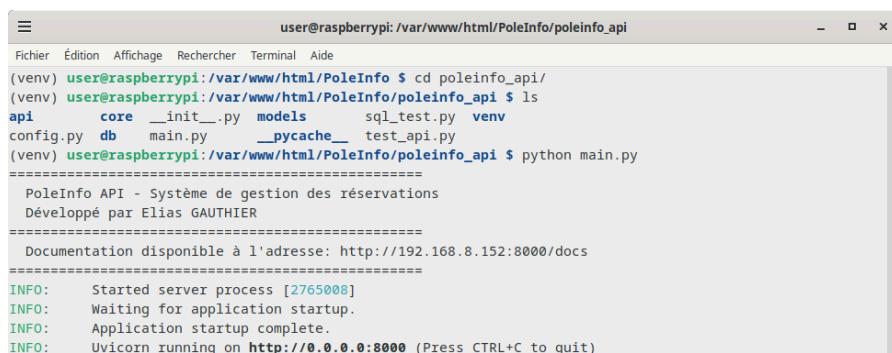
    return {
        "message": f"Bienvenue sur l'API PoleInfo, nous sommes le
{formatted_date}",
        "documentation": "/docs",
        "version": "1.0.0",
        "auteur": "Elias GAUTHIER"
    }
if __name__ == "__main__":
    # Message affiché au démarrage du serveur en mode autonome
    print("=" * 50)
    print(" PoleInfo API - Système de gestion des réservations")
    print(" Développé par Elias GAUTHIER")
    print("=" * 50)
    print(" Documentation disponible à l'adresse: http://ip:8000/docs")
    print("=" * 50)

    uvicorn.run(app, host="0.0.0.0", port=8000)

```

poleinfo_api/main.py

Puis on démarre le serveur pour qu'il écoute sur le port 8000.



The terminal window shows the following output:

```

user@raspberrypi:/var/www/html/PoleInfo/poleinfo_api
Fichier Édition Affichage Rechercher Terminal Aide
(venv) user@raspberrypi:/var/www/html/PoleInfo $ cd poleinfo_api/
(venv) user@raspberrypi:/var/www/html/PoleInfo/poleinfo_api $ ls
api      core __init__.py models      sql_test.py venv
config.py db      main.py  __pycache__ test_api.py
(venv) user@raspberrypi:/var/www/html/PoleInfo/poleinfo_api $ python main.py
=====
PoleInfo API - Système de gestion des réservations
Développé par Elias GAUTHIER
=====
Documentation disponible à l'adresse: http://192.168.8.152:8000/docs
=====
INFO:     Started server process [2765008]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)

```

Afin de décrire au mieux le fonctionnement de l'API, on expliquera le cheminement des opérations les plus importantes à savoir la **création d'un utilisateur**, **l'authentification** et **l'ajout/suppression/modification/consultation** de réservations. Effectivement, toutes les autres routes sont des tâches redondantes et similaires. Le code source sera cependant consultable dans les annexes.

Création utilisateur

La création d'un utilisateur est faite depuis la page administrateur et permet d'ajouter un utilisateur dans la base de données.

La création d'un utilisateur se passera donc via notre API depuis la route `/users/` et il sera nécessaire de posséder un compte administrateur.

192.168.xx.xx:8000/ <code>users</code>	
Ajoute un nouvel utilisateur dans le système.	
Méthode	POST
Données attendues	{ "login": "string", "type": 0, "nom": "string", "prenom": "string", "password": "string" }
Données renvoyées	{"message": "Utilisateur créé avec succès", "id": user_id}
Erreurs possibles	400 Bad Request : Un utilisateur avec le même login existe déjà. 403 Forbidden : Privilèges administrateurs requis.

En premier lieu dans notre programme on importe les modules et bibliothèques nécessaires:

```
from fastapi import APIRouter, HTTPException, status, Depends
from typing import List, Optional
from models.schemas import UserCreate, UserResponse, UserDelete
from db.requests.user import create_user, get_user_by_login,
get_all_users, delete_user_by_login
from core.security import verify_admin
```

On remarque les schémas de données attendus et de réponses, les fonctions de requêtes SQL ainsi que la fonction de vérification de compte administrateur.

On écrit ensuite le code de notre endpoint :

```
@router.post("/", response_model=dict)
def add_user(user: UserCreate, admin_id: int = Depends(verify_admin)):
    existing_user = get_user_by_login(user.login)
    if existing_user:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Un utilisateur avec ce login existe déjà"
        )

    user_id = create_user(
        login=user.login,
        password=user.password,
        type=user.type,
        nom=user.nom,
        prenom=user.prenom
    )

    return {"message": "Utilisateur créé avec succès", "id": user_id}
```

endpoints/users.py - add_user()

On met en dépendance **Depends** la fonction de vérification de compte administrateur `verify_admin` :

Fonction de vérification de compte administrateur

```
async def verify_admin(user_id: int = Depends(verify_token)):
    # Récupération des informations complètes de l'utilisateur
    user = get_user_by_id(user_id)

    # Vérification des droits administrateur
    if not user or user["type"] != ADMIN_TYPE:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Accès refusé: droits d'administrateur requis"
        )

    # Retour de l'identifiant utilisateur pour utilisation dans les routes
    # protégées
    return user_id
```

core/security.py - verify_admin()

Qui elle-même utilise en dépendance `verify_token` (décrite [ici](#)) qui renvoie le `user_id` de l'utilisateur concerné si son jeton est valide.

`verify_admin` vérifiera à son tour si l'utilisateur concerné est administrateur (si son type vaut 1). Si il n'est pas administrateur alors on lève une erreur HTTP 403 et on quitte le processus. Si il est administrateur on retourne `user_id` et on commence enfin l'ajout d'un utilisateur dans `endpoints/users.py`.

La première étape consiste à vérifier si l'utilisateur que l'on souhaite ajouter n'existe pas déjà. On utilise `get_user_by_login()` et on lève une erreur HTTP 400 si l'utilisateur existe. Ensuite on utilise `create_user()` en passant en argument toutes les informations du nouvel utilisateur fournies dans le corps de la requête dont voici le programme :

Fonction de création d'un utilisateur

```
def create_user(login, password, type, nom, prenom):
    """Créer un nouveau utilisateur"""
    hashed_password = hash_password(password)
    with get_db_cursor() as cursor:
        query = """
            INSERT INTO user (login, passwd, type, nom, prenom)
            VALUES (%s, %s, %s, %s, %s)
        """
        values = (login, hashed_password, type, nom, prenom)

        cursor.execute(query, values)

        cursor.execute("SELECT LAST_INSERT_ID() as id_user")
        result = cursor.fetchone()

        if result and 'id_user' in result:
            user_id = result['id_user']
            return user_id
        else:
            raise ValueError("Impossible de récupérer l'ID de
l'utilisateur créé")
```

db/user.py - `create_user()`

Pour des raisons évidentes de sécurité, nous n'insérons pas en clair le mot de passe du nouvel utilisateur. Pour ce faire on utilise le générateur de hash **bcrypt** :

Fonction de hachage de mot de passe

```
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def hash_password(password: str) -> str:
```

Fonction de hachage de mot de passe

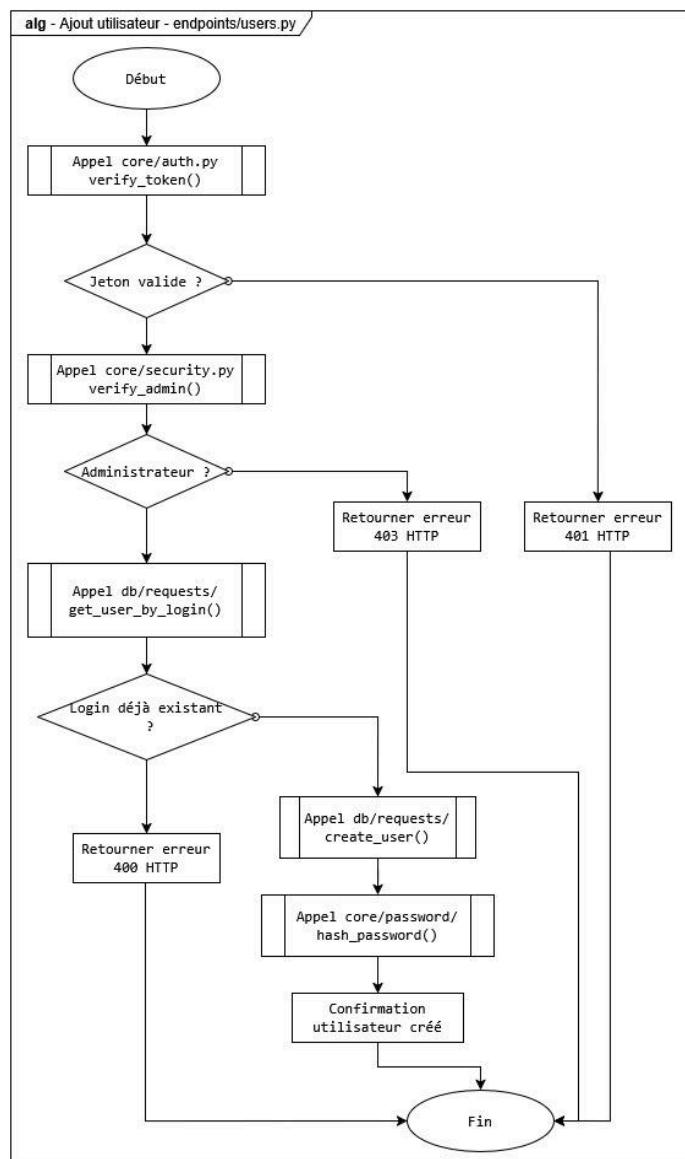
```
return pwd_context.hash(password)
```

core/password.py - hash_password()

Une fois le mot de passe haché, on insère dans notre table user toutes les informations du nouvel utilisateur et on renvoie son id en guise de confirmation.

De retour dans le programme de notre endpoint on retourne un code 200 avec le message confirmation.

On peut représenter tout le processus de création d'utilisateur avec cet algorithme :

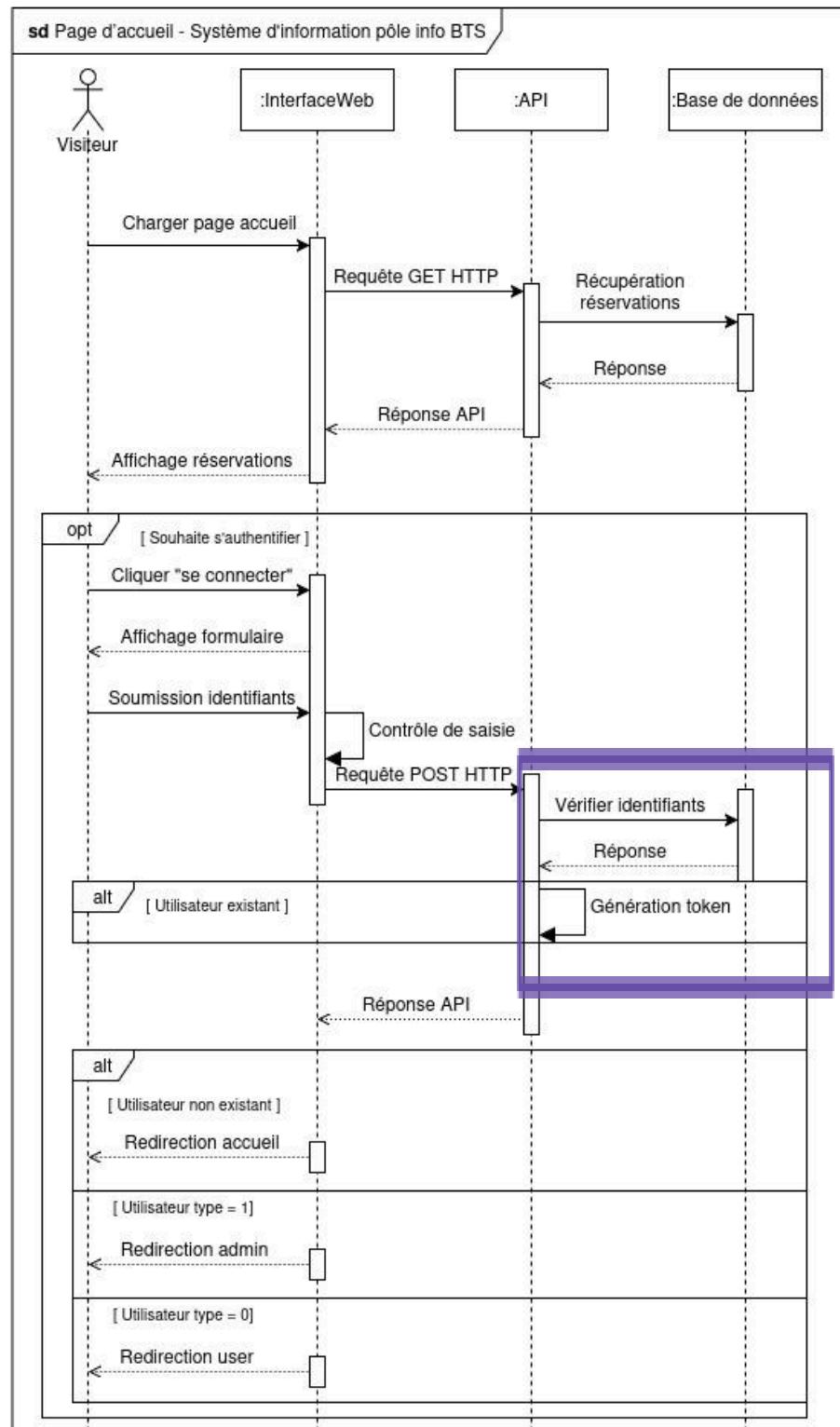


Authentification

Lorsqu'un visiteur souhaite s'authentifier via une page de login, une requête vers l'endpoint **/token** sera réalisée permettant ainsi la génération d'un jeton d'accès si l'utilisateur existe dans la base de données.

192.168.xx.xx:8000/<code>token</code>	
Authentifie un utilisateur avec ses identifiants (login et mot de passe) et retourne un token JWT à utiliser pour les requêtes futures protégées.	
Méthode	POST
Données attendues	{ " <code>username</code> " : str, " <code>password</code> ": str }
Données renvoyées	<code>access_token</code> : str <code>token_type</code> : int <code>user_type</code> : int <code>user_name</code> : str
Erreurs possibles	400 Bad Request : Identifiants incorrects. 422 Validation Error : Champs requis absents ou mal formatés.

On peut représenter ce processus avec ce diagramme de séquence:

**/token**

On s'intéresse donc au programme qui gère cette génération de token.

```
from fastapi import APIRouter, Depends, HTTPException, status
from fastapi.security import OAuth2PasswordRequestForm

from core.security import create_access_token
from core.auth import verify_token
from models.schemas import Token
from db.requests.user import authenticate_user

endpoints/auth.py
```

Dans les imports, on va chercher la **fonction permettant de créer un token d'accès** ainsi que le **schéma de modèle de réponse** et la **fonction d'authentification par requête SQL** d'un utilisateur. On retrouve également la fonction de **vérification du jeton**.

```
"""
API Pôle Info - Routes d'authentification
"""

router = APIRouter(tags=["authentification"])

@router.post("/token", response_model=Token)
def login(form_data: OAuth2PasswordRequestForm = Depends()):
    """Authentifie un utilisateur et génère un token JWT."""
    user = authenticate_user(form_data.username, form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Identifiants incorrects"
        )

    token = create_access_token(user["id user"])
    return {
        "access_token": token,
        "token_type": "bearer",
        "user_type": user["type"],
        "user_name": user["nom"],
        "user_login": user["login"]}
endpoints/auth.py - /token
```

- ① Définition de l'endpoint `/token` en appliquant le modèle de réponse Pydantic, c'est à dire que lors d'une requête vers cet endpoint, on récupérera précisément ces 5 valeurs.

```
class Token(BaseModel):
    access_token: str
    token_type: str
    user_type: int
    user_name: str
    user_login: str
```

models/schemas.py

- ② Nous utilisons `OAuth2PasswordRequestForm` pour recevoir les identifiants sous forme de formulaire (username/password).

- ③ Ensuite nous appelons la fonction `authenticate_user()` écrite dans `db/requests/users.py` qui utilise elle même `get_user_by_login()` ainsi que `verify_password()`

Authentification d'un utilisateur via une requête SQL

```
from db.database import get_db_cursor
from core.password import verify_password, hash_password
from typing import List, Dict, Any

def get_user_by_login(login):
    """Récupère un utilisateur par son login"""
    with get_db_cursor() as cursor:
        cursor.execute("SELECT * FROM user WHERE login = %s", (login,))
        user = cursor.fetchone()
    return user

def authenticate_user(login, password):
    """Authentifie un utilisateur en vérifiant son login et son mot de passe"""
    user = get_user_by_login(login)
    if not user:
        return None

    if not verify_password(password, user["passwd"]):
        return None

    return user
```

db/requests/users.py

On réalise une requête SELECT sur le login passé en paramètre et si la requête ne renvoie rien alors on renvoie None. On vérifie ensuite si le password passé également en paramètre est valide ou non avec la fonction `verify_password()`

Fonction vérification de mot de passe

```
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def hash_password(password: str) -> str:
    return pwd_context.hash(password)

def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)
```

core/password.py

Cette fonction compare le mot de passe fourni en texte brut avec le mot de passe haché stocké dans la base de données. Retourne `True` si les mots de passe correspondent, `False` sinon.

Pour hacher les mot de passe, j'ai fait le choix d'utiliser l'algorithme bcrypt qui est un algorithme robuste conçu spécialement pour le stockage sécurisé de mots de passe. On l'importe depuis la bibliothèque `passlib`.

De retour dans `authenticate_user()` si `user` est retourné alors on retourne toute les informations obtenues lors de la requête SQL SELECT sous forme de dictionnaire à savoir :

id_user
login
passwd
type
nom
prenom

- ④ Si `authenticate_user()` retourne `False` de part un login non existant ou un mot de passe erroné, on lève une exception HTTP avec le code de statut `400 Bad Request` et un message d'erreur "`Identifiants incorrects`". FastAPI transforme automatiquement cette exception en une réponse HTTP avec le code et le message spécifiés.

⑤ token = create_access_token(user["id_user"]) : Appelle la fonction externe `create_access_token()` pour générer un token JWT. Cette fonction prend l'identifiant unique de l'utilisateur (`id_user`) comme argument pour l'intégrer dans le token. Le token JWT est une chaîne de caractères encodée qui contient des informations sur l'utilisateur et qui peut être vérifiée ultérieurement pour autoriser l'accès à d'autres endpoints. ([Vérification jeton](#))

Fonction de création de jeton

```
def create_access_token(user_id: int):
    # Calcul de la date d'expiration du token
    expire = datetime.utcnow() +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    # Préparation du payload du token avec l'identifiant utilisateur et la
date d'expiration
    payload = {
        "sub": str(user_id),
        "exp": expire
    }
    # Encodage du token avec la clé secrète et l'algorithme configurés
    return jwt.encode(payload, SECRET_KEY, algorithm=ALGORITHM)
```

core/security.py

Fichier de configuration

```
SECRET_KEY = "*****"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60
```

/config.py

⑥ Enfin, on renvoie nos 5 valeurs définies par le modèle de réponse Pydantic :

```
"access_token": token,
"token_type": "bearer",
"user_type": user["type"],
"user_name": user["nom"],
"user_login": user["login"]}
```

Ainsi, le script PHP qui exécutera la requête `/token` obtiendra les informations suivantes et pourra les définir en tant que variables de sessions.

Vérification jeton

192.168.xx.xx:8000/verify-token	
Vérifie si le token JWT fourni dans l'en-tête d'autorisation est encore valide.	
Méthode	GET
Données attendues	Authorization: Bearer <access_token>
Données renvoyées	{ "message": "Token valide", "user_id": <ID utilisateur> }
Erreurs possibles	401 Unauthorized : Si le token est invalide, expiré, ou absent.

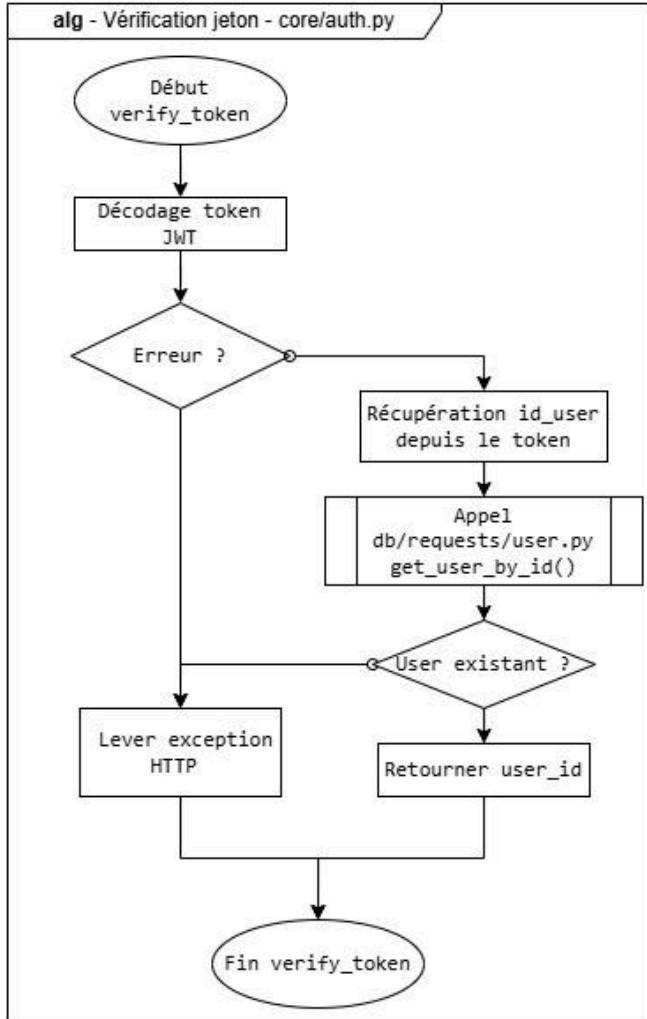
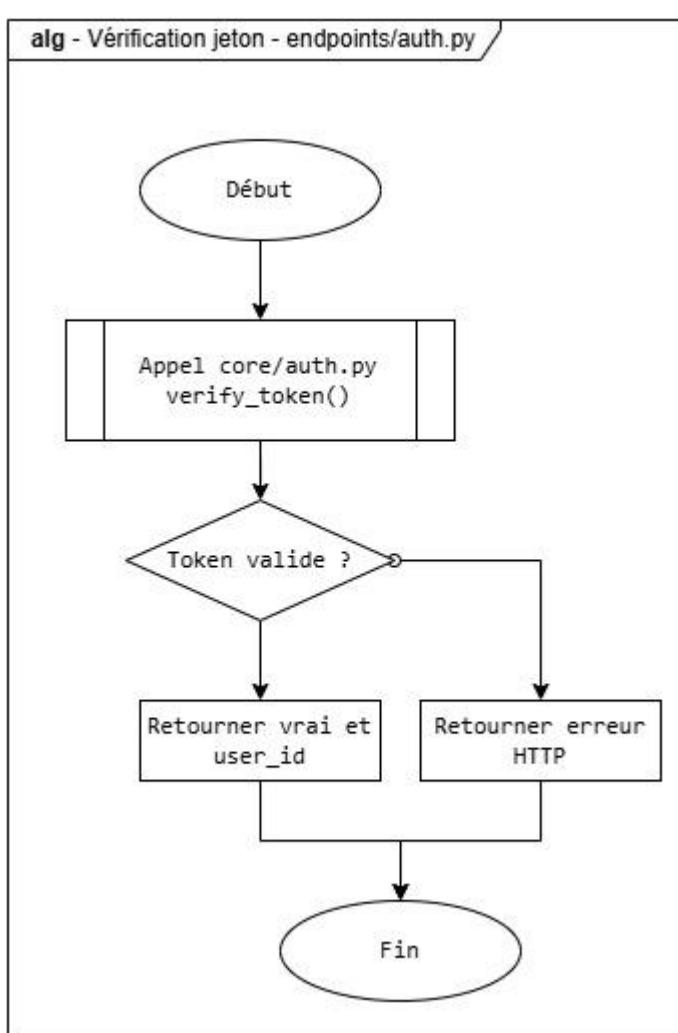
En plus de l'authentification, il est nécessaire d'implémenter une route permettant de vérifier la validité d'un jeton JWT afin de vérifier si un utilisateur est autorisé à accéder à une interface web ou si simplement son jeton n'a pas expiré

On rappelle le payload du jeton :

```
payload = {
    "sub": str(user_id),
    "exp": expire
}
```

Ce processus nécessitera de décoder le jeton de l'utilisateur puis de vérifier si son ID contenu dans son jeton est valide ou pas.

On peut représenter cette opération avec ces 2 algorigrammes :



```

@router.get("/verify-token")
def verify_token_endpoint(user_id: int = Depends(verify_token)):
    return {"valid": True, "user_id": user_id}
  
```

endpoints/auth.py - /verify-token

Ici on ajoute une dépendance pour la route `/verify-token` en utilisant **Depends** qui permet d'exécuter du code en amont de l'exécution de la fonction.

1. FastAPI appelle d'abord automatiquement la fonction `verify_token`
2. Si `verify_token` s'exécute sans erreur, elle renvoie un entier (l'ID de l'utilisateur)
3. Cet entier est automatiquement injecté en tant que paramètre `user_id` dans `verify_token_endpoint`
4. Si `verify_token` lance une exception (par exemple, si le token est invalide), l'exécution s'arrête et l'exception est renvoyée au client.

Fonction de vérification de jeton

```
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from jose import JWTError, jwt

from config import SECRET_KEY, ALGORITHM
from db.requests.user import get_user_by_id

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

def verify_token(token: str = Depends(oauth2_scheme)) -> int:
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Credentials invalides",
        headers={"WWW-Authenticate": "Bearer"},
    )

    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        user_id = int(payload.get("sub"))

        if not get_user_by_id(user_id):
            raise HTTPException(401)

    return user_id
    except (JWTError, ValueError, TypeError):
        raise credentials_exception
```

core/auth.py

Réservations

Une fois les routes de sécurité et d'authentification mise en place, nous devons créer les routes pour accéder aux ressources et notamment le plus important, les réservations.

On rappelle que la route `/reservations/` aura les méthodes **GET**, **POST**, **DELETE** & **PUT**

GET	POST	DELETE	PUT
Récupérer la liste des réservations ou récupérer une réservation particulière selon son auteur, son id ou sa salle.	Ajouter une réservation dans la base de données.	Supprimer une réservation dans la base de données.	Modifier une réservation dans la base de données.

Dans le fichier de l'endpoint on importe de la même manière que précédemment tous les modules nécessaires. On remarque cependant ici une grande quantité de fonctions de requêtes SQL permettant la manipulation des données de réservations au sein de la base de données.

```
from models.schemas import ReservationCreate, ReservationResponse,
ReservationDelete, ReservationUpdate
from core.auth import verify_token

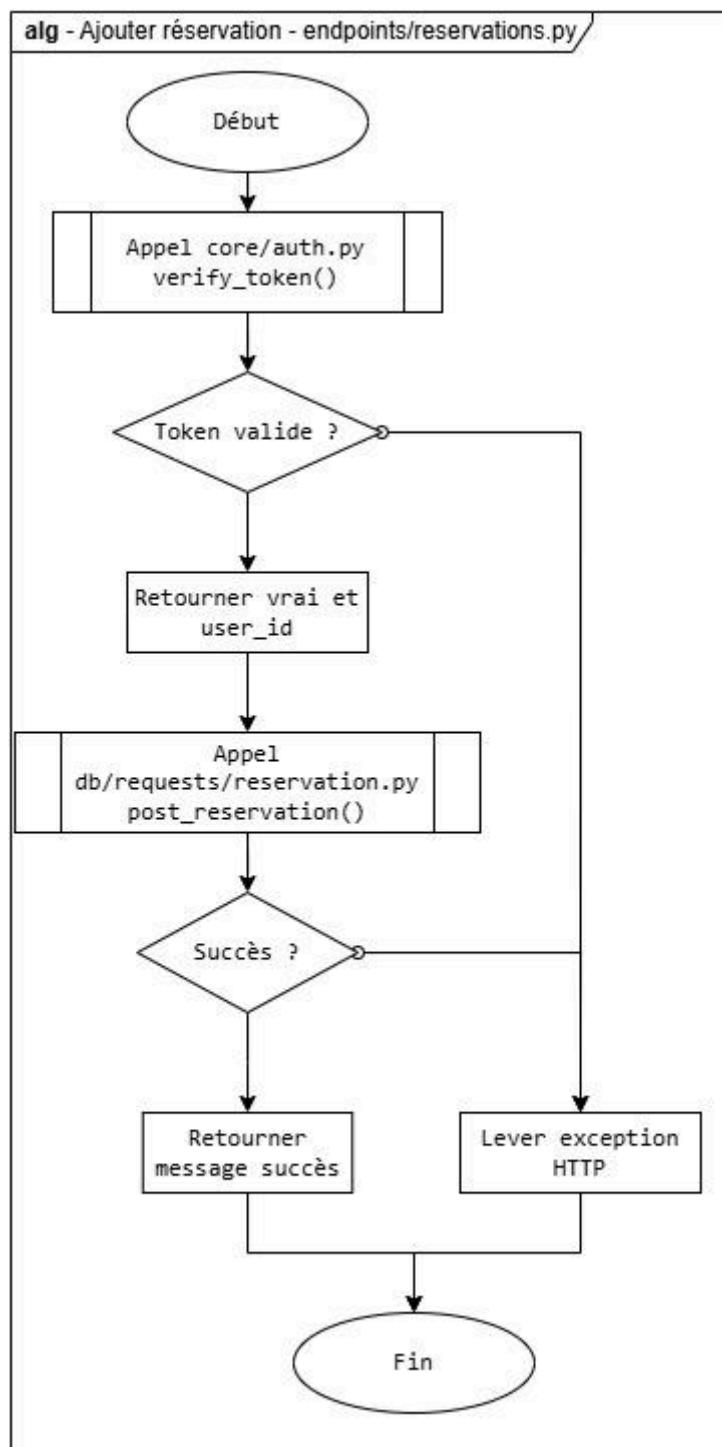
from fastapi import APIRouter, Depends, HTTPException, status, Query
from typing import List, Optional
from datetime import date

from db.requests.reservation import get_all_reservations,
get_reservations_by_salle_increase, get_reservations_by_salle,
post_reservation, get_reservations_by_prof_increase, remove_reservation,
remove_reservation_by_id, update_reservation, get_reservation_by_id
from db.requests.user import get_user_by_id

endpoints/reservations.py
```

POST

On représente le processus de création de réservation avec cet algorithme :



192.168.xx.xx:8000/reservations	
Créer une nouvelle réservation	
Méthode	POST
Données attendues	<p>Authorization: Bearer <access_token></p> <pre>{ "duree": float, "date": str (YYYY-MM-DD), "numero_salle": str, "nom_matiere": str, "heure_debut_creneau": str (HH:MM:SS), "login_user": str, "nom_classe": str, "info": str ou null }</pre>
Données renvoyées	<pre>{ "message": "Réservation créée avec succès", "id_reservation": <id> }</pre>
Erreurs possibles	Erreur 404 - Not Found Erreur 400 - Bad Request Erreur 401 - Unauthorized

Pour la méthode POST on définit le modèle de données suivant :

```
class ReservationCreate(BaseModel):
    duree: float
    date: date
    numero_salle: str
    nom_matiere: str
    heure_debut_creneau: str
    login_user: str
    nom_classe: str
    info: Optional[str] = None
```

models/schemas.py

On attendra ces 8 données dans le corps de la requête POST (ou 7 car les informations de la réservation sont optionnelles).

On retrouve ici le code qui constitue notre endpoint :

Endpoint de création de réservation

```
@router.post("/", response_model=dict)
def create_reservation(reservation: ReservationCreate, user_id: int = Depends	verify_token):
    """
    Crée une nouvelle réservation de salle.
    """

    user = get_user_by_id(user_id)
    username = user["login"]

    # Préparation des données pour la requête
    reservation_data = {
        "duree": reservation.duree,
        "date": reservation.date,
        "info": reservation.info or "",
        "numero_salle": reservation.numero_salle,
        "nom_matiere": reservation.nom_matiere,
        "heure_debut_creneau": reservation.heure_debut_creneau,
        "login_user": username,
        "nom_classe": reservation.nom_classe
    }

    # Appel à la fonction de réservation
    result = post_reservation(**reservation_data)

    # Gestion des réponses
    if result.get("status") == "success":
        return {
            "message": f"Réservation enregistrée par {username}.",
            "id": result.get("id_reservation")
        }

    # Gestion des erreurs
    error_mapping = {
        "error_reserv": "Cette salle est déjà réservée pour cet horaire",
        "error_overtime": "L'horaire ne peut pas dépasser 17h25"
    }

    error_message = result.get("message") or error_mapping.get(
        result.get("status"),
        "Erreur lors de la création de la réservation, veuillez consulter un administrateur."
    )
    raise HTTPException(status_code=400, detail=error_message)
```

endpoints/reservations.py

Lors de la définition de la fonction on y intègre comme paramètre le modèle de donnée défini auparavant ainsi que la dépendance verify_token vu précédemment.

Après avoir récupéré le login de l'auteur de la réservation on prépare le payload à envoyer à la fonction qui exécutera la requête SQL :

```
# Préparation des données pour la requête
reservation_data = {
    "duree": reservation.duree,
    "date": reservation.date,
    "info": reservation.info or "",
    "numero_salle": reservation.numero_salle,
```

```

    "nom_matiere": reservation.nom_matiere,
    "heure_debut_creneau": reservation.heure_debut_creneau,
    "login_user": username,
    "nom_classe": reservation.nom_classe
}

```

Nous allons donc décrire le fonctionnement de l'ajout d'une réservation dans la base de données avec ce payload. Le code étant très volumineux nous expliquerons partie par partie. La première étape étant de vérifier si la réservation ne dépasse pas 17h25 et si la salle n'est pas déjà réservée.

Vérifications avant ajout de réservation

```

def post_reservation(duree, date, info, numero_salle, nom_matiere, heure_debut_creneau, login_user,
nom_classe):
    with get_db_cursor() as cursor:
        # Vérifier si le créneau existe
        cursor.execute("SELECT TIME_TO_SEC(heure_debut) as seconds FROM creneau WHERE heure_debut = %s",
                      (heure_debut_creneau,))
        creneau_result = cursor.fetchone()

        if not creneau_result:
            return {"status": "error", "message": f"Créneau {heure_debut_creneau} non trouvé"}

        # Vérifier si la réservation dépasse 17h25
        debut_seconds = creneau_result['seconds']
        duree_seconds = duree * 3600
        fin_seconds = debut_seconds + duree_seconds
        limite_seconds = 17 * 3600 + 25 * 60 + 1000

        if fin_seconds > limite_seconds:
            return {
                "status": "error_time_limit",
                "message": f"La réservation se termine à {fin_seconds}, ce qui dépasse la limite de {limite_seconds}"
            }

        # Vérifier si la salle est déjà réservée
        query_check_salle = """
        SELECT id_reservation FROM reservation r
        JOIN salle s ON r.id_salle = s.id_salle
        JOIN creneau c ON r.id_creneau = c.id_creneau
        WHERE s.numero = %s AND r.date = %s AND (
            TIME_TO_SEC(c.heure_debut) <= TIME_TO_SEC(%s) AND
            TIME_TO_SEC(c.heure_debut) + (r.duree * 3600) > TIME_TO_SEC(%s))
        OR
            (TIME_TO_SEC(%s) < TIME_TO_SEC(c.heure_debut) + (r.duree * 3600) AND
            TIME_TO_SEC(%s) + %s > TIME_TO_SEC(c.heure_debut))
        )
        """
        cursor.execute(query_check_salle, (

```

Vérifications avant ajout de réservation

```

        numero_salle, date,
        heure_debut_creneau, heure_debut_creneau,
        heure_debut_creneau, heure_debut_creneau, duree_seconds
    ))
}

if cursor.fetchone():
    return {
        "status": "error_reserv",
        "message": f"La salle {numero_salle} est déjà occupée durant cette période"
}

```

db/requests/reservation.py

Reprendons étapes par étapes :

Récupération et vérification du créneau horaire

```

cursor.execute("SELECT TIME_TO_SEC(heure_debut) as seconds FROM creneau
WHERE heure_debut = %s",
              (heure_debut_creneau,))
creneau_result = cursor.fetchone()

if not creneau_result:
    return {"status": "error", "message": f"Créneau
{heure_debut_creneau} non trouvé"}

```

La requête convertit l'heure de début en secondes (depuis minuit) puis cherche si ce créneau existe dans la table creneau
Si le créneau n'existe pas, la fonction retourne une erreur.

Vérification de la limite horaire

```

debut_seconds = creneau_result['seconds']
duree_seconds = duree * 3600
fin_seconds = debut_seconds + duree_seconds
limite_seconds = 17 * 3600 + 25 * 60 + 1000

if fin_seconds > limite_seconds:
    return {
        "status": "error_time_limit",
        "message": f"La réservation se termine à {fin_seconds}, ce qui
dépasse la limite de {limite_seconds}"
    }

```

Calcule l'heure de fin de la réservation en ajoutant la durée à l'heure de début
Définit une limite maximale de 17h25.

Vérification de la limite horaire

Si l'heure de fin dépasse cette limite, retourne une erreur

Vérification de la disponibilité d'une salle

```
query_check_salle = """
SELECT id_reservation FROM reservation r
JOIN salle s ON r.id_salle = s.id_salle
JOIN creneau c ON r.id_creneau = c.id_creneau
WHERE s.numero = %s AND r.date = %s AND (
    (TIME_TO_SEC(c.heure_debut) <= TIME_TO_SEC(%s) AND
     TIME_TO_SEC(c.heure_debut) + (r.duree * 3600) > TIME_TO_SEC(%s))
    OR
    (TIME_TO_SEC(%s) < TIME_TO_SEC(c.heure_debut) + (r.duree * 3600)
AND
    TIME_TO_SEC(%s) + %s > TIME_TO_SEC(c.heure_debut)))
"""

```

Cette requête complexe vérifie s'il existe déjà des réservations pour la salle demandée à la date demandée qui chevauchent le créneau horaire souhaité. Elle gère deux cas :
 Si une réservation existante qui commence avant ou en même temps que la nouvelle réservation et qui finit après le début de la nouvelle réservation
 Si une réservation existante qui commence pendant la durée de la nouvelle réservation
 Si une telle réservation est trouvée, la fonction retourne une erreur indiquant que la salle est déjà occupée.

Cas 1 :

```
(TIME_TO_SEC(c.heure_debut) <= TIME_TO_SEC(%s) AND
 TIME_TO_SEC(c.heure_debut) + (r.duree * 3600) > TIME_TO_SEC(%s))
```

Le créneau existant commence avant ou en même temps que l'heure de début demandée, et se termine après celle-ci.

Cas 2 :

```
(TIME_TO_SEC(%s) < TIME_TO_SEC(c.heure_debut) + (r.duree * 3600) AND
 TIME_TO_SEC(%s) + %s > TIME_TO_SEC(c.heure_debut))
```

Le créneau demandé se termine après le début d'un créneau existant ET commence avant la fin de ce même créneau.

Une fois que cette requête est exécuté et si elle renvoie des conflits alors on renvoie le message d'erreur suivant et on sort de la fonction :

```
return {
    "status": "error_reserv",
    "message": f"La salle {numero_salle} est déjà
occupée durant cette période"
}
```

Par la suite on insère dans la base de donnée la réservation avec les id correspondants des informations fournies par l'utilisateur. (Les ID ont été récupérés au préalable via des SELECT dans chaque colonne)

```
cursor.execute("""
    INSERT INTO reservation (duree, date, info, id_salle,
    id_matiere, id_creneau, id_user)
    VALUES (%s, %s, %s, %s, %s, %s)
    """, (duree, date, info, id_salle, id_matiere, id_creneau, id_user))
```

Voici avec CURL un exemple de requête que l'on peut implémenter en PHP, C++, Kotlin etc permettant de créer une réservation :

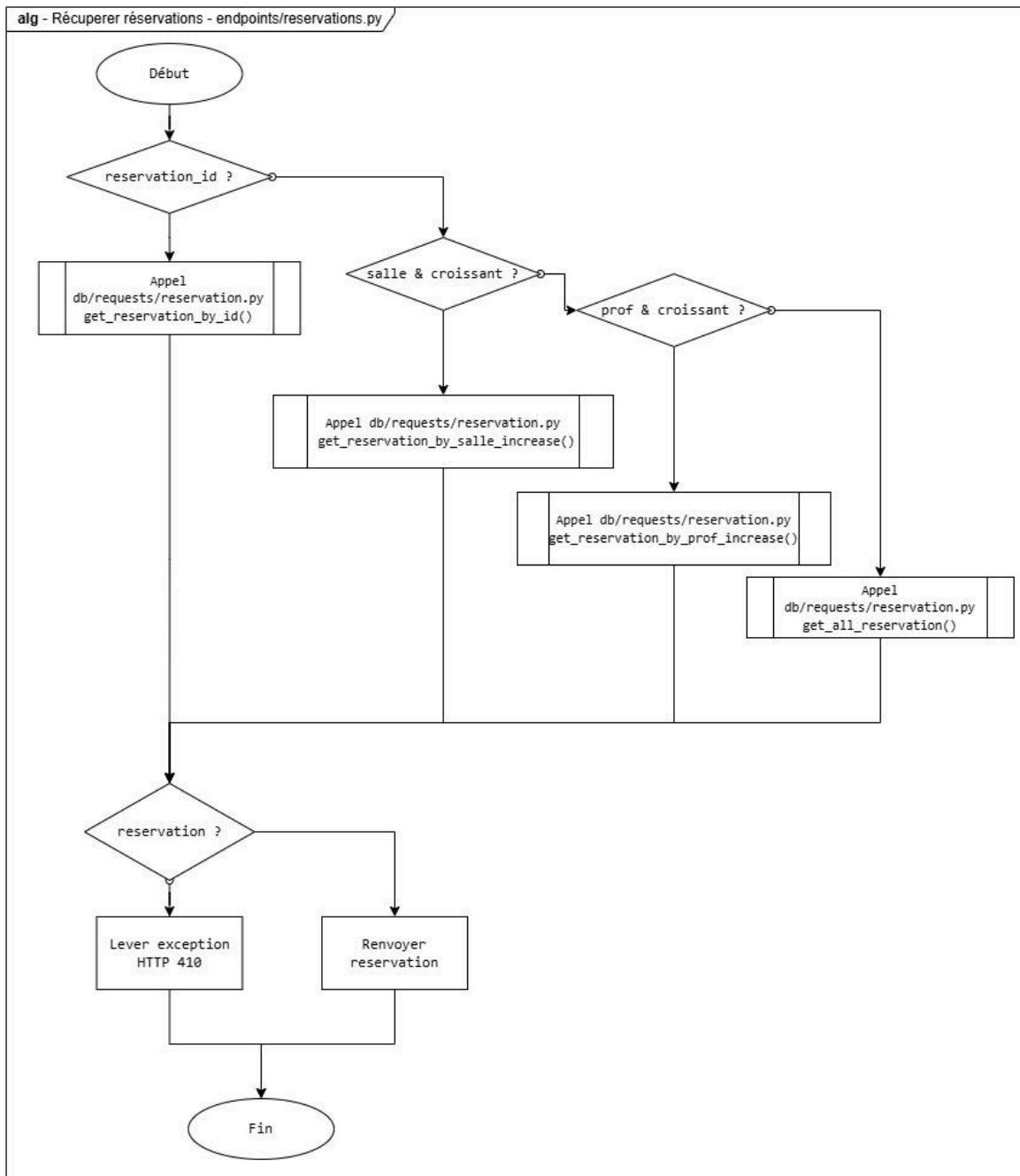
```
curl -X POST http://192.168.xx.xx/reservations \
-H "Authorization: Bearer <TOKEN_JWT>" \
-H "Content-Type: application/json" \
-d '{
    "duree": 0.84,
    "date": "2025-05-22",
    "numero_salle": "3W03",
    "nom_matiere": "Mathématiques",
    "heure_debut_creneau": "10:00:05",
    "login_user": "vmarignan",
    "nom_classe": "CIEL1",
    "info": "TP Python sur les fichiers"
}'
```

GET

192.168.xx.xx:8000/reservations	
Récupère la liste des réservations (avec filtres possibles)	
Méthode	GET
Paramètres	<p>salle (string, optionnel) : Filtrer par numéro de salle.</p> <p>croissant (boolean, optionnel) : Trier les résultats par date/heure croissante.</p> <p>prof (string, optionnel) : Nom du professeur.</p> <p>reservation_id (integer, optionnel) : Obtenir une réservation spécifique par son ID</p>
Données renvoyées	[{ "id_reservation": int, "duree": float, "date": str, "info": str, "numero_salle": str, "capacite_salle": int, "type_salle": str, "nom_matiere": str, "heure_debut": str, "nom_user": str, "prenom": str, "noms_classes": str }]
Erreurs possibles	Erreur 410 - Gone

Une requête GET sera exécutée sur cet endpoint lorsque l'on voudra consulter les réservations enregistrées et si l'on souhaite, par critères de tri.

On représente le processus de récupération de réservation avec cet algorithme :



Lors d'une requête GET, l'API renverra les données suivante.

```
class ReservationResponse(BaseModel):
    id_reservation: int
    duree: float
    date: date
    info: str
    numero_salle: str
    capacite_salle: int
    type_salle: str
    nom_matiere: str
    heure_debut: timedelta
    nom_user: str
    prenom: str
    noms_classes: str
```

models/schemas.py

La requête GET possède des paramètres optionnels permettant de filtrer les réservations qui nous seront renvoyées.

Voici le code de notre endpoint :

Endpoint de récupération des réservations

```
@router.get("/")
def get_reservations(
    salle: str = Query(None),
    croissant: bool = Query(None),
    prof: str = Query(None),
    reservation_id: int = Query(None)
):
    if reservation_id:
        reservation = get_reservation_by_id(reservation_id)
        if not reservation:
            raise HTTPException(status_code=status.HTTP_410_GONE, detail="Aucune réservation trouvée pour cet ID")
        return [reservation]

    if croissant and salle:
        reservations = get_reservations_by_salle_increase(salle)
    elif croissant and prof:
        reservations = get_reservations_by_prof_increase(prof)
    else:
        reservations = get_all_reservations()

    if not reservations:
        raise HTTPException(status_code=status.HTTP_410_GONE, detail="Aucune réservation trouvée")
    return reservations
```

① Définition des paramètres

② Tri par identifiant et envoie d'un message d'erreur si la réservation n'existe pas.

③ Trie par ordre croissant, salle, prof

Exemple de requête SQL dans db/requests/reservation.py pour obtenir les réservations dans l'ordre croissant pour une certaine salle :

Requête SQL de récupération des réservation

```
def get_reservations_by_salle_increase(numero_salle: str) -> List[Dict[str, Any]]:
    """Récupère toutes les réservations pour une salle spécifique par ordre
    croissant"""
    with get_db_cursor() as cursor:
        query = """
            SELECT
                r.id_reservation,
                r.duree,
                r.date,
                r.info,
                s.numero AS numero_salle,
                s.capacite AS capacite_salle,
                s.type AS type_salle,
                m.nom AS nom_matiere,
                c.heure_debut,
                u.nom AS nom_user,
                u.prenom,
                GROUP_CONCAT(cl.nom SEPARATOR ', ') AS noms_classes
            FROM reservation r
            LEFT JOIN salle s ON r.id_salle = s.id_salle
            LEFT JOIN matiere m ON r.id_matiere = m.id_matiere
            LEFT JOIN creneau c ON r.id_creneau = c.id_creneau
            LEFT JOIN user u ON r.id_user = u.id_user
            LEFT JOIN classe_reservation cr ON r.id_reservation = cr.id_reservation
            LEFT JOIN classe cl ON cr.id_classe_grp = cl.id_classe_grp
            WHERE s.numero = %s
            GROUP BY r.id_reservation, r.duree, r.date, r.info, s.numero, s.capacite,
            s.type, m.nom, c.heure_debut, u.nom, u.prenom
            ORDER BY r.date ASC, c.heure_debut ASC
        """
        cursor.execute(query, (numero_salle,))
        return cursor.fetchall()
```

Sélection des données

SELECT

```
r.id_reservation,          -- Identifiant de la réservation
r.duree,                  -- Durée de la réservation
r.date,                   -- Date de la réservation
r.info,                   -- Infos complémentaires
s.numero AS numero_salle, -- Numéro de la salle
```

Requête SQL de récupération des réservation

```

s.capacite AS capacite_salle,
s.type AS type_salle,      -- Type de salle
m.nom AS nom_matiere,     -- Nom de la matière
c.heure_debut,             -- Heure de début du créneau
u.nom AS nom_user,        -- Nom de l'utilisateur ayant réservé
u.prenom,
GROUP_CONCAT(cl.nom SEPARATOR ', ') AS noms_classes
                           -- Noms des classes concernées par la réservation,
concaténés

```

Jointures

```

FROM reservation r
LEFT JOIN salle s ON r.id_salle = s.id_salle
LEFT JOIN matiere m ON r.id_matiere = m.id_matiere
LEFT JOIN creneau c ON r.id_creneau = c.id_creneau
LEFT JOIN user u ON r.id_user = u.id_user
LEFT JOIN classe_reservation cr ON r.id_reservation = cr.id_reservation
LEFT JOIN classe cl ON cr.id_classe_grp = cl.id_classe_grp

```

Condition

```
WHERE s.numero = %s
```

(On ne garde que les réservations pour la salle spécifique, dont le numéro est passé en paramètre (%s).)

Tri :

```
ORDER BY r.date ASC, c.heure_debut ASC
```

Voici avec CURL un exemple de requête permettant de récupérer les réservations dans la salle 3W03 :

```
curl -X GET "http://192.168.xx.xx:8000/reservations?salle=3W03"
```

DELETE

192.168.xx.xx:8000/ reservations	
Supprime une réservation	
Méthode	DELETE
Donnée attendues	Authorization: Bearer <TOKEN_JWT> { "id_reservation": int }
Données renvoyées	{ "message": "Réservation supprimée avec succès" }
Erreurs possibles	422 - Unprocessable entity 400 - Bad request 401 - Unauthorized

Il est également possible de supprimer une réservation en combinant date, numéro de salle et heure de début mais nous n'expliquerons que la suppression par id dont voici la fonction de suppression :

Fonction de suppression de réservation

```
def remove_reservation_by_id(user_id: int, id_reservation: int):
    with get_db_cursor() as cursor:
        try:
            cursor.execute("""
                DELETE FROM reservation
                WHERE id_reservation = %s
            """, (id_reservation,))

            return {"status": "success", "message": "Réservation supprimée avec succès"}

        except Exception as e:
            return {"status": "error", "message": f"Erreur lors de la suppression: {str(e)}"}
```

Voici avec CURL un exemple de requête permettant de supprimer une réservations dont l'ID est 1 :

```
curl -X DELETE http://192.168.xx.xx:8000/reservations \
-H "Authorization: Bearer <TOKEN_JWT>" \
-H "Content-Type: application/json" \
-d '{
  "id_reservation": 1
}'
```

PUT

192.168.xx.xx:8000/reservations	
Mettre à jour les informations d'une réservation existante : date, créneau, salle, matière, classe, etc.	
Méthode	PUT
Donnée attendues	Identique au POST
Données renvoyées	{ "message": "Réservation mise à jour avec succès" }
Erreurs possibles	401 - Unauthorized 400 - Bad Request

Cette méthode est quasiment identique au POST nous ne détaillerons donc pas plus cette méthode.

Avec CURL :

```
curl -X PUT http://192.168.xx.xx:8000/reservations \
-H "Authorization: Bearer <TOKEN_JWT>" \
-H "Content-Type: application/json" \
-d '{
  "id_reservation": 2,
  "date": "2025-05-24",
  "heure_debut_creneau": "13:50:00",
  "duree": 0.84,
  "info": "Déplacé à cause d'un conseil de classe",
  "numero_salle": "B202",
  "nom_matiere": "Informatique",
  "nom_classe": "CIEL2"
}'
```

Avec Swagger qui est implémenté par défaut avec FastAPI on peut consulter la liste des endpoints disponibles. Ainsi, à la fin du projet l'on retrouve cela :

authentification ^

POST	/token	Login	▼
------	--------	-------	---

utilisateurs ^

GET	/utilisateurs/	Get Users	▼
POST	/utilisateurs/	Add User	▼
DELETE	/utilisateurs/	Delete Users	▼

reservations ^

POST	/reservations/	Create Reservation	▼
GET	/reservations/	Get Reservations	▼
DELETE	/reservations/	Delete Reservation	▼
PUT	/reservations/	Update Reservation Endpoint	▼

creneaux ^

GET	/creneaux/	Get Creneaux	▼
POST	/creneaux/	Add Creneau	▼
DELETE	/creneaux/	Delete Creneau	▼

salles ^

GET	/salles/	Get Salles	▼
POST	/salles/	Add Salle	▼
DELETE	/salles/	Delete Salle	▼

matieres ^

GET	/matieres/	Get Matieres	▼
POST	/matieres/	Add Matiere	▼
DELETE	/matieres/	Delete Matieres	▼

classes ^

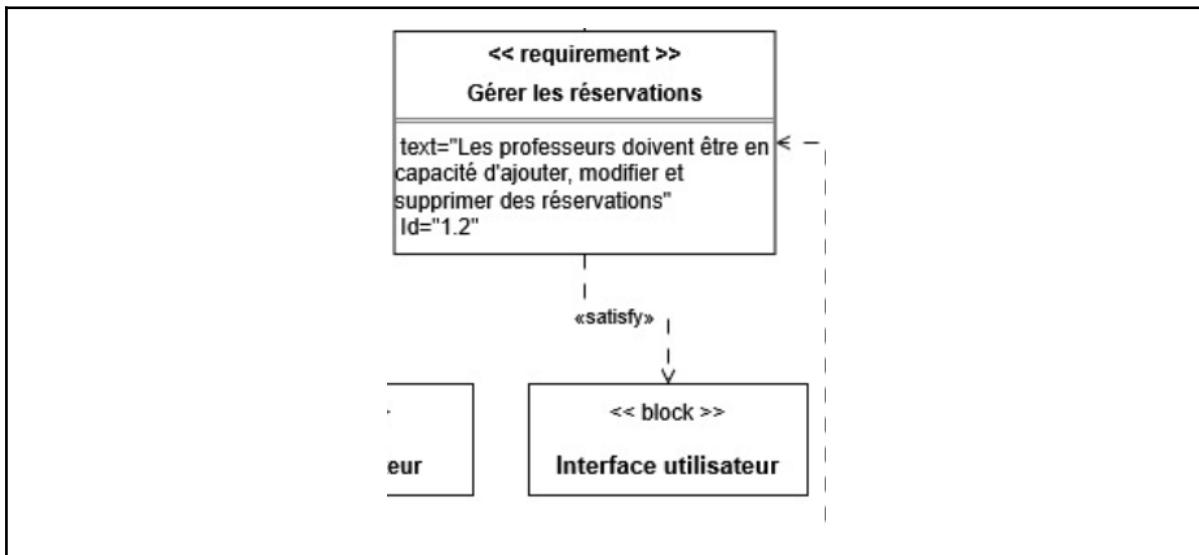
GET	/classes/	Get Classes	▼
POST	/classes/	Add Classe	▼
DELETE	/classes/	Delete Classes	▼

accueil ^

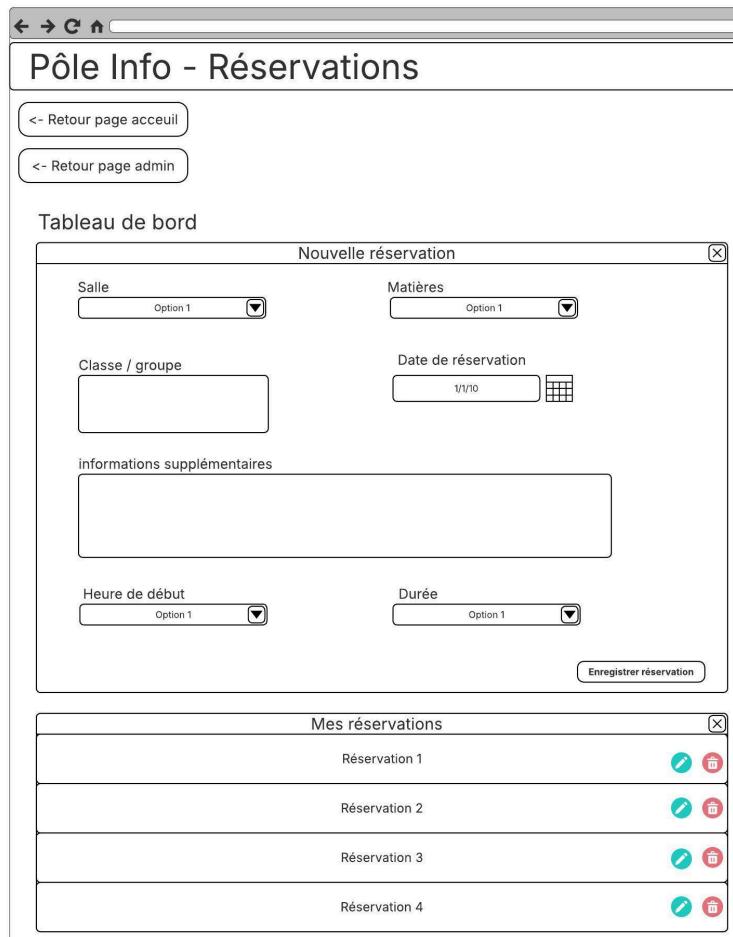
GET	/	Read Root	▼
-----	---	-----------	---

Interface utilisateur (Elias GAUTHIER)

L'interface utilisateur à pour objectif de permettre à un professeur de gérer ses réservations. Il pourra ajouter, modifier ou supprimer des réservations. Nous pouvons identifier facilement cela à travers l'exigence 1.2 du diagramme req.



Mise en page



The wireframe illustrates the user interface for managing reservations:

- Header:** Pôle Info - Réervations
- Navigation:** <- Retour page accueil, <- Retour page admin
- Tableau de bord:** A summary section.
- Nouvelle réservation:** A form for creating new reservations, containing fields for:
 - Salle (Room): Option 1
 - Matières (Subjects): Option 1
 - Classe / groupe (Class/group): Input field
 - Date de réservation (Reservation date): Date input field (1/1/10) with a calendar icon
 - Informations supplémentaires (Additional information): Text area
 - Heure de début (Start time): Option 1
 - Durée (Duration): Option 1
 A "Enregistrer réservation" (Save reservation) button is at the bottom.
- Mes réservations:** A list of four reservations, each with edit and delete icons:
 - Réservation 1
 - Réservation 2
 - Réservation 3
 - Réservation 4

La page possédera 2 grands menus, à savoir l'ajout de nouvelle réservation qui permettra d'enregistrer une nouvelle réservation selon les critères suivants correspondants à ce qu'attends l'API :

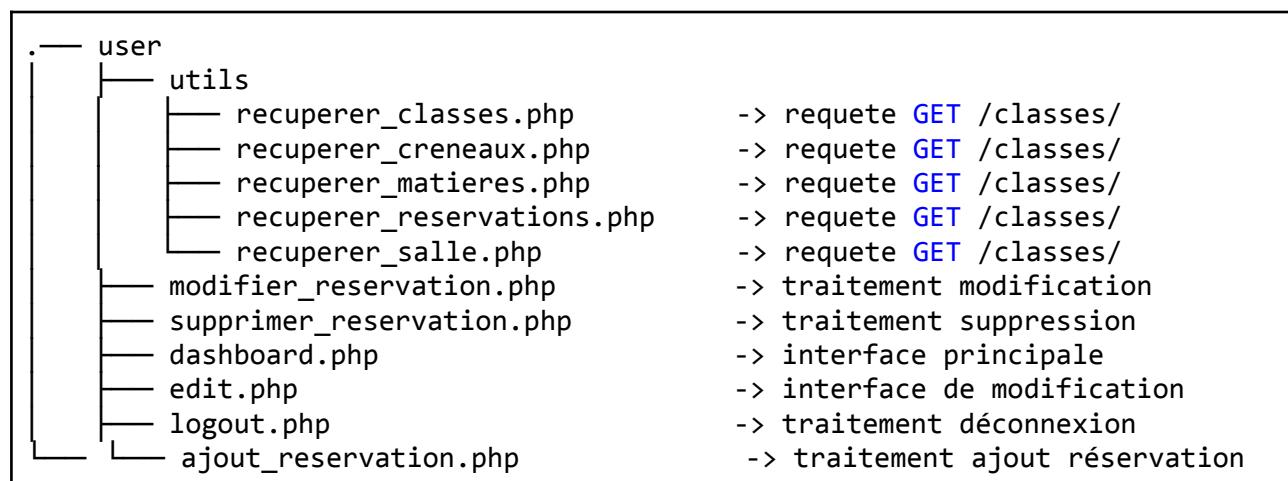
- Salle
- Matière
- Classe/groupe (avec possibilité de faire une sélection multiple)
- Date de réservation (avec calendrier)
- Zone de texte pour fournir une description
- Heure de début
- Durée

Chacun de ces champs sera rempli automatiquement à chaque chargement de la page en faisant des requêtes GET sur les endpoints des ressources souhaitées.

Le second menu consistera à l'affichage des réservations de l'utilisateur avec la possibilité de les modifier et supprimer.

Note : Si un administrateur se connecte sur la page user, on fera apparaître un bouton de redirection vers la page admin ainsi que la possibilité de supprimer toutes les réservations.

Structure



Technologies utilisées

Afin de concevoir une interface rapide et efficace nous avons utilisé comme pour l'intégralité le framework Tailwind CSS permettant de simplifier grandement le développement du style de la page.



Tailwind CSS est un framework utility-first CSS (feuilles de style en cascade) avec des classes prédéfinies que vous pouvez utiliser pour construire et concevoir des pages web directement dans votre balisage. Il permet d'écrire du CSS dans votre HTML sous la forme de classes prédéfinies.

Source : <https://kinsta.com/>

Voici un exemple d'usage de tailwind css comparé à une feuille de style classique :

HTML	CSS
<pre><div class="bg-blue-500 text-white p-4 rounded-lg"> Hello, Tailwind! </div></pre>	<pre>div { background-color: #3b82f6; color: white; padding: 1rem; border-radius: 0.5rem; }</pre>

Fichier principal

La première chose à faire dans notre dashboard.php est d'inclure nos fichiers de traitement ainsi qu'un fichier de configuration.

```
require_once __DIR__ . '/../config.php';
require_once 'utils/recuperer_creneaux.php';
require_once 'utils/recuperer_salles.php';
require_once 'utils/recuperer_matieres.php';
require_once 'utils/recuperer_classes.php';
require_once 'utils/recuperer_reservation.php';

dashboard.php
```

Le fichier de config.php contient l'ensemble des configurations d'adressage IP du serveur. Si l'on migre le serveur sur un autre réseau ou que l'on change son adresse IP, il n'y a qu'à changer l'adresse dans ce fichier de configuration pour reconfigurer l'intégralité du site.

Fichier de configuration

```
$config = [
    // Serveur d'API principal
    'api_server' => [
        'ip' => '192.168.8.152',
        'port' => '8000',
        'protocol' => 'http'
    ],
    // Serveur web
    'web_server' => [
        'ip' => '192.168.8.152',
        'port' => '80',
        'protocol' => 'http',
        'base_path' => '/'
    ],
    // Paramètres d'authentification
    'auth' => [
        'token_endpoint' => '/token',
        'grant_type' => 'password'
    ]
];
/***
 * Retourne l'URL complète du serveur API
 * @param string $endpoint Point de terminaison API (optionnel)
 * @return string URL complète
 */
function getApiUrl($endpoint = '') {
    global $config;
    $server = $config['api_server'];
    $base_url =
    "{$server['protocol']}://{$server['ip']}:{$server['port']}";
    return $base_url . $endpoint;
}
```

config.php

On vérifie ensuite si l'utilisateur est autorisé à accéder à cette interface web. On vérifie d'abord si il possède un jeton de session puis on vérifie ce jeton via l'endpoint /verify-token/ de l'API.

Processus de vérification de jeton

```
session_start();
if (!isset($_SESSION['token'])) {
    header("Location: " .
getWebUrl('interface_login.php?error=expired'));
    exit;
}

$token = $_SESSION['token'];
$username = $_SESSION['username'];
$login = $_SESSION['login'];
$type = $_SESSION['type_compte'];

$api_url_verify = getApiUrl('/verify-token');
$api_url_reservations = getApiUrl('/reservations/');

$ch = curl_init();
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
curl_setopt($ch, CURLOPT_URL, $api_url_verify);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, [
    "Authorization: Bearer $token",
]);
$response = curl_exec($ch);
$http_code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
$curl_error = curl_error($ch);
curl_close($ch);

if ($http_code != 200 || !$response) {
    error_log("Erreur de vérification du token : HTTP $http_code - "
$curl_error");
    session_destroy();
    header("Location: " .
getWebUrl('interface_login.php?error=expired'));
    exit;
}
```

dashboard.php

Avant toute chose, si l'utilisateur vient d'ajouter/supprimer/modifier une réservation on doit vérifier si un message doit être affiché, on écrit ainsi cela :

```
$success_message = "";
if (isset($_SESSION['info_message'])) {
    $success_message = $_SESSION['info_message'];
    unset($_SESSION['info_message']);
}
$error_message = "";
if (isset($_SESSION['error_message'])) {
    $error_message = $_SESSION['error_message'];
    unset($_SESSION['error_message']);
}
```

Enfin, on récupère la liste des réservations ainsi que les créneaux, salles, matières et classes :

```
$request_reservation = getApiUrl('/reservations/?croissant=true');
$response_reservation = file_get_contents($request_reservation);
$data = json_decode($response_reservation, true);

$creneaux = getCreneaux();
$salles = getSalles();
$matieres = getMatieres();
$classes = getClasses();
```

Si l'utilisateur connecté est un administrateur, il doit pouvoir gérer les réservations de toute le monde, ainsi l'on commence par filtrer la récupération des réservations avec getReservation qui nous permet de passer en argument le nom du professeur pour lequel on souhaite récupérer les réservations :

```
if ($type == 1) {
    $reservations = getReservations();
}
elseif ($type == 0) {
    $reservations = getReservations($username);
}
```

type 1 = administrateur

type 0 = professeur

On décrit ici le fonctionnement en amont de getReservations() en pseudo code :

```

DÉBUT getReservations(nomProf)
    reservations ← []
    url ← getApiUrl("/reservations/?croissant=true")

    SI nomProf ≠ VIDE ALORS
        url ← url + "&prof=" + urlencode(nomProf)

    reponseJson ← file_get_contents(url)
    donnees ← json_decode(reponseJson, VRAI)

    SI donnees EST_TABLEAU ALORS
        POUR item DANS donnees FAIRE
            reservation ← EXTRAIRE_ET_FORMATER(item)
            AJOUTER reservation À reservations
        FIN POUR

    RETOURNER reservations
FIN

```

```

function getReservations($nomProf = null) {
    $reservations = [];

    $get_reservation = getApiUrl('/reservations/?croissant=true');

    if (!empty($nomProf)) {
        $get_reservation .= "&prof=" . urlencode($nomProf);
    }

    $reponse_reservation = file_get_contents($get_reservation);
    $data_reservation = json_decode($reponse_reservation, true);

    if (is_array($data_reservation)) {
        foreach ($data_reservation as $item) {
            $reservation = [
                'id_reservation' => $item['id_reservation'],
                'duree' => $item['duree'],
                'date' => $item['date'],
                'info' => $item['info'],
                'numero_salle' => $item['numero_salle'],
                'capacite_salle' => $item['capacite_salle'],
                'type_salle' => $item['type_salle'],
                'nom_matiere' => $item['nom_matiere'],
                'heure_debut' => $item['heure_debut'],
            ];
            $reservations[] = $reservation;
        }
    }
}

```

```

        'nom_user' => $item['nom_user'],
        'prenom' => $item['prenom'],
        'noms_classes' => $item['noms_classes']
    ];
    $reservations[] = $reservation;
}
}

return $reservations;

```

recuperer_reservations.php

On affiche chaque réservation en suivant le même fonctionnement que pour la page visiteur avec des boutons pour modifier ou supprimer une réservation selon si le compte est administrateur avec une simple vérification à chaque itération de la boucle d'affichage :

```

<?php if ($type === 1): ?>
// Afficher bouton suppression
// Afficher bouton modification

```

Suppression de réservation

Lorsque l'utilisateur ou l'administrateur cliquera sur l'icône de suppression, on appellera notre fichier de traitement supprimer_reservation.php en passant en argument l'id de la réservation concernée.

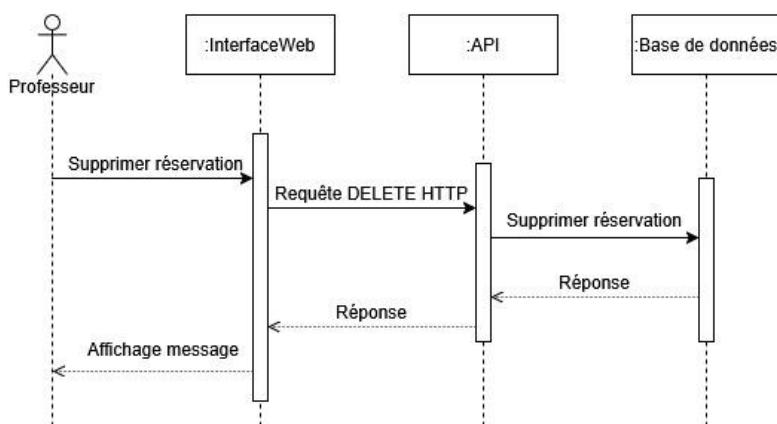
```

<a href="supprimer_reservation.php?id=<?php echo
$reservation['id_reservation']; ?>" class="text-red-600
hover:text-red-800" onclick="return confirm('Êtes-vous sûr de vouloir
supprimer cette réservation ?')">

</a>

```

dashboard.php



On décrit ensuite le processus de suppression de réservation :

1.

```
$reservation_id = intval($_GET['id']);
```

Récupérer l'ID de la réservation depuis le formulaire.

2.

```
$data = json_encode(['id_reservation' => $reservation_id]);
```

Création d'un objet JSON contenant l'ID de la réservation à supprimer. Format de donnée attendu par l'API (vu précédemment)

3.

```
$api_url = getApiUrl('/reservations/');
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $api_url);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");
curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, [
    'Content-Type: application/json',
    'Content-Length: ' . strlen($data),
    'Authorization: Bearer ' . $_SESSION['token']
]);
```

Construction de la requête CURL.

CURLOPT_URL : L'URL de l'endpoint /reservations/ API (construite via getApiUrl())

CURLOPT_CUSTOMREQUEST : Spécifie la méthode HTTP DELETE (au lieu de GET par défaut)

CURLOPT_POSTFIELDS : Les données JSON à envoyer dans le corps de la requête

CURLOPT_RETURNTRANSFER : Pour récupérer la réponse au lieu de l'afficher directement

CURLOPT_HTTPHEADER : En-têtes HTTP :

Content-Type : Indique que le contenu est du JSON

Content-Length : Taille des données envoyées

Authorization : Token d'authentification Bearer

4.

```
$response = curl_exec($ch);
$http_code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
curl_close($ch);
```

Exécution de la requête

Si la requête nous renvoie un code 200 alors on sait que la suppression de la réservation a été effective.

Ajout de réservation

Lorsque l'utilisateur remplit le formulaire d'ajout de réservation et clique sur enregistrer, un appel vers ajout_reservation.php est fait.

1.

```
$numero_salle = $_POST['salle'] ?? '';
$nom_matiere = $_POST['matiere'] ?? '';
$classes = isset($_POST['classe']) && is_array($_POST['classe']) ? 
$_POST['classe'] : [];
$date = $_POST['date_reserv'] ?? '';
$info = $_POST['message'] ?? '';
$heure_debut = $_POST['startTime'] ?? '';
$duree = number_format(floatval($_POST['duration'] ?? 0), 3, '.', '');
''');
```

Récupération des informations du formulaire

2.

```
$data = [
    "duree" => $duree,
    "date" => $date,
    "info" => $info,
    "numero_salle" => $numero_salle,
    "nom_matiere" => $nom_matiere,
    "heure_debut_creneau" => $heure_debut,
    "login_user" => $login_user,
    "nom_classe" => $nom_classe,
];
```

Préparation du payload des données pour l'API

3.

```
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $api_url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));
curl_setopt($ch, CURLOPT_HTTPHEADER, [
    "Authorization: Bearer $token",
    "Content-Type: application/json"
]);

$response = curl_exec($ch);
$http_code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
$curl_error = curl_error($ch);
curl_close($ch);
```

Paramétrage CURL et envoie de la requête.

4.

```

if ($http_code === 200) {
    $response_data = json_decode($response, true);
    $message = $response_data['message'] ?? "Réservation ajoutée
avec succès!";
    $_SESSION['info_message'] = $message;
    header("Location: " . getWebUrl('user/dashboard.php'));
    exit;
}

// Erreur
$error_message = "Erreur lors de la réservation";
if ($response) {
    $response_data = json_decode($response, true);
    if (isset($response_data['detail'])) {
        $error_message = $response_data['detail'];
    }
}

$_SESSION['error_message'] = $error_message;
header("Location: " . getWebUrl('user/dashboard.php'));
exit;

```

Récupération du message de retour (erreur ou succès) en tant que variable de session.

Par exemple, si l'ajout d'une réservation ne passe pas les vérifications imposées par l'API (heure limite = 17h25) alors on affichera le message de retour :



The screenshot shows a dark blue header bar with the text "Pôle Info - Réervations". On the right side of the header, there is a user profile icon with the name "Bonjour, Gauthier" and a "Se déconnecter" button. Below the header, the main content area has a light gray background. At the top left, it says "Aujourd'hui : 20/05/2025". At the top right, it says "Heure actuelle : 15:51:38". A red horizontal bar spans across the middle of the screen, containing the text "● La réservation se termine à 18:15, ce qui dépasse la limite de 17h25." In the bottom left corner of the main content area, there is a dark blue footer bar with the text "+ Nouvelle réservation".

Modification de réservation

La dernière fonctionnalité de cette interface est de permettre la modification de réservation. On retrouve quasiment le même procédé que précédemment mais ici, quand l'utilisateur cliquera sur le bouton de modification une autre page s'ouvrira avec un nouveau formulaire pré-rempli.

dashboard.php

1.

```
<a href="edit.php?id=<?php echo $reservation['id_reservation']; ?>">
```

Redirection vers le nouveau formulaire edit.php avec l'id de la réservation concernée

edit.php

1.

```
$api_url_reservation =
getApiUrl("/reservations/?reservation_id=$id_reservation");
```

Appel API pour récupérer les données existantes de la réservation pour l'ID donné.

2.

```
<select name="salle" id="salle" class="w-full px-3 py-2 border
border-gray-300 rounded-md focus:outline-none focus:ring-2
focus:ring-primary focus:border-transparent" required>
<option value="">--Sélectionnez une salle--</option>
<?php foreach ($salles as $salle): ?>
    <option value="<?php echo $salle['numero']; ?>" <?php echo
($salle['numero'] == $reservation['numero_salle']) ? 'selected' :
''; ?>>
        <?php echo $salle['numero']; ?>
    </option>
<?php endforeach; ?>
</select>
```

Pré-rempli automatiquement le formulaire avec les données récupérées

Un appel vers `modifier_reservation.php` est ensuite fait lorsque l'utilisateur enregistre les modifications et l'on construit le payload comme pour ajouter une réservation mais l'on configure cette fois CURL pour faire une requête PUT vers l'API.

```
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $api_url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PUT");
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($data));
curl_setopt($ch, CURLOPT_HTTPHEADER, [
    "Authorization: Bearer $token",
    "Content-Type: application/json"
]);
```

```
$response = curl_exec($ch);
$http_code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
$curl_error = curl_error($ch);
curl_close($ch);

if ($http_code === 200) {
    $response_data = json_decode($response, true);
    $message = $response_data['message'] ?? "Réervation modifiée avec
succès!";

    $_SESSION['info_message'] = $message;
    header("Location: " . getWebUrl('user/dashboard.php'));
    exit;
```

Rendu final

Pôle Info - Réservations

Bonjour gauthier

[Se déconnecter](#)[← Retour au menu principal](#)[← Interface d'administration](#)

Tableau de bord

Gérez vos réservations de salles et consultez le planning.

Aujourd'hui : 26/05/2025

Heure actuelle : 14:00:59

[**+ Nouvelle réservation**](#)

Salle

--Sélectionnez une salle--

Matière

--Sélectionnez une matière--

Classe/groupe

CIEL1
CIEL2
CIAP1
CIAP2

Date de réservation

jj / mm / aaaa



Informations sur le cours/activité (optionnel)

Détails supplémentaires sur le cours ou l'activité...

Heure de début

Sélectionnez une heure

Durée

Sélectionnez une durée

Enregistrer la réservation[**Réservations**](#)

DATE	HORAIRE	SALLE	MATIÈRE	CLASSE(S)	PROFESSEUR	ACTIONS
13/05/2025	16h35 (0.84h)	3W06 Salle des profs	Informatique	CIEL2	elias gauthier	
14/05/2025	15h45 (1.67h)	3W03 Cours	BAS	CIEL1	Ethan Clement	

Tests et validation

Auteur(s) de cette section :

- Elias GAUTHIER

Correspond au Cycle en V : **Test unitaire / Validation**

Test unitaires

Afin de vérifier efficacement et automatiquement le bon fonctionnement de l'API, j'ai écrit avec l'aide d'une intelligence artificielle un script bash testant l'accessibilité de chaque endpoint de mon API avec la méthode GET.

```
elias@DESKTOP-6GB2CFA:~/UnitTests$ ./tests3.sh

==== Tests de l'API PoleInfo (GET uniquement) ====
[URL de base: http://192.168.1.144:8000

==== Test de santé de l'API ====
✓ API accessible et répond
[Temps de réponse de la route racine: 0.005469s
✓ L'API retourne du JSON valide

==== Récupération du token d'authentification ====
✓ Token d'authentification récupéré

==== Test de vérification du token ====
✓ Vérification du token JWT (Status: 200)
[ Nombre d'éléments retournés: 2

==== Tests des utilisateurs ====
✓ Récupération des utilisateurs (sans auth) (Status: 200)
[ Nombre d'éléments retournés: 6
✓ Récupération des utilisateurs (avec auth) (Status: 200)
[ Nombre d'éléments retournés: 6

==== Tests des salles ====
✓ Récupération de la liste des salles (Status: 200)
[ Nombre d'éléments retournés: 5

==== Tests des créneaux ====
✓ Récupération de la liste des créneaux (Status: 200)
[ Nombre d'éléments retournés: 12

==== Tests des matières ====
✓ Récupération de la liste des matières (Status: 200)
[ Nombre d'éléments retournés: 10

==== Tests des classes ====
✓ Récupération de la liste des classes (Status: 200)
[ Nombre d'éléments retournés: 8
```

```
==== Tests des réservations ====
✓ Récupération de toutes les réservations (Status: 200)
[?] └ Nombre d'éléments retournés: 2
✓ Réservations en ordre croissant (Status: 200)
[?] └ Nombre d'éléments retournés: 2
✓ Réservations en ordre décroissant (Status: 200)
[?] └ Nombre d'éléments retournés: 2
✓ Réservations filtrées par salle 3C01 (Status: 200)
[?] └ Nombre d'éléments retournés: 2
✓ Réservations filtrées par salle 3W03 (Status: 200)
[?] └ Nombre d'éléments retournés: 2
✓ Réservations filtrées par professeur (Status: 200)
[?] └ Nombre d'éléments retournés: 2
✓ Réservation avec ID=22 (Status: 200)
[?] └ Nombre d'éléments retournés: 1
✓ Réservations salle 3W03 en ordre croissant (Status: 200)
[?] └ Nombre d'éléments retournés: 2
✓ Réservations prof gauthier en ordre décroissant (Status: 200)
[?] └ Nombre d'éléments retournés: 2

==== Tests divers ====
✓ Route racine de l'API (Status: 200)
[?] └ Nombre d'éléments retournés: 4
✓ Endpoint inexistant (doit retourner 404) (Status: 404)
[?] └ Nombre d'éléments retournés: 1
✓ Endpoint API invalide (doit retourner 404) (Status: 404)
[?] └ Nombre d'éléments retournés: 1
✓ GET sur endpoint POST uniquement (doit retourner 405) (Status: 405)
[?] └ Nombre d'éléments retournés: 1

==== Tests de performance et robustesse ====
✓ Paramètre croissant invalide (doit retourner 422) (Status: 422)
[?] └ Nombre d'éléments retournés: 1
✓ ID de réservation non numérique (doit retourner 422) (Status: 422)
[?] └ Nombre d'éléments retournés: 1
[?] Test de charge simple (20 requêtes simultanées)...
✓ Test de charge terminé en 0s
```

Tous les tests sont passés, on peut donc affirmer que notre API est fonctionnelle pour toutes les requêtes GET. Le reste des méthodes ont été testées manuellement.

Recettes

Afin de vérifier le bon fonctionnement des fonctionnalités, on peut rédiger des recettes fonctionnelles. Voici quelques exemples avec l'enregistrement d'une réservation et l'authentification. **Ces recettes devront être remplies en aval par le client.**

Recette fonctionnelle	Test d'enregistrement des réservations			
Objectif	Enregistrer une réservation dans le système.			
Éléments à tester	<ul style="list-style-type: none"> • Interface utilisateur • API 			
Pré-requis	<ul style="list-style-type: none"> • Compte utilisateur • Jeton valide 			
Scénario :				
ID	Démarche	Données	Comportement attendu	Validation
1	Remplir le formulaire et enregistrer la réservation	Salle Matière Classe/groupe Heure début Durée Date Informations	Un POST vers l'API est fait et ajoute la réservation dans la base de données. Ensuite un message de confirmation s'affiche.	OK/NOK
2	Remplir le formulaire en oubliant des champs et enregistrer la réservation	Salle Matière Classe/groupe Heure début Durée Date Informations	Un message au-dessus du champ manquant indique de remplir toutes les informations.	OK/NOK
3	Remplir le formulaire avec une durée dépassant l'heure limite et enregistrer la réservation	Salle Matière Classe/groupe Heure début Durée Date Informations	Un message indiquant que l'heure de fin de la réservation dépasse l'heure limite et annulation de l'enregistrement de la réservation.	OK/NOK
4	Remplir le formulaire avec des créneaux similaire à une autre réservation	Salle Matière Classe/groupe Heure début Durée Date Informations	Message indiquant un chevauchement de réservation et annulation de l'enregistrement de la réservation.	OK/NOK
Rapport de test	Testé par :		Le :	

Fonctionnalité	Conformité	Ergonomie
<input type="checkbox"/> Excellente <input type="checkbox"/> Bonne <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible	<input type="checkbox"/> Excellente <input type="checkbox"/> Bonne <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible	<input type="checkbox"/> Excellente <input type="checkbox"/> Bonne <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible
Commentaires : Fiches d'anomalies émises : Approbation :		

Recette fonctionnelle	Test d'authentification			
Objectif	S'authentifier dans le système			
Éléments à tester	<ul style="list-style-type: none"> • Interface login • Interface utilisateur • Interface administrateur • API 			
Pré-requis	<ul style="list-style-type: none"> • Compte utilisateur et/ou administrateur 			
Scénario :				
ID	Démarche	Données	Comportement attendu	Validation
1	Saisir login et mot de passe utilisateur	login password	Un token de session est généré et l'utilisateur est redirigé vers l'interface utilisateur.	OK/NOK
2	Saisir login et mot de passe administrateur	login password	Un token de session est généré et l'utilisateur est redirigé vers l'interface d'administration.	OK/NOK
3	Saisir mauvais login et/ou mauvais mot de passe	login password	L'API ne renvoie pas de token de session et l'utilisateur reste sur la page de login.	OK/NOK
4	Ne rien saisir		L'API ne renvoie pas de token de session et l'utilisateur reste sur la page de login.	OK/NOK
Rapport de test	Testé par :		Le :	

Fonctionnalité	Conformité	Ergonomie
<input type="checkbox"/> Excellente <input type="checkbox"/> Bonne <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible	<input type="checkbox"/> Excellente <input type="checkbox"/> Bonne <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible	<input type="checkbox"/> Excellente <input type="checkbox"/> Bonne <input type="checkbox"/> Moyenne <input type="checkbox"/> Faible
Commentaires : Fiches d'anomalies émises : Approbation :		

Nous pourrions rédiger des recettes pour la consultation de réservation, la déconnexion, l'ajout d'utilisateurs etc.

Conclusion

Difficultés rencontrées

Durant la réalisation de ce projet l'un des problèmes majeurs rencontrés fut la difficulté de coordination des tâches entre les membres du groupe. En effet certaines fonctionnalités nécessitent la dépendances d'autre fonctionnalités. Ainsi je devais parfois finir le développement d'une tâche pour qu'un membre de mon groupe puisse avancer notamment avec l'API qui constituait l'élément central du projet. L'intégration de technologies non étudiées en cours ont également été un facteur retardant sur la conception du projet se répercutant ainsi sur la rédaction tardive du rapport.

Acquis et bénéfices

En choisissant d'approfondir la conception du projet au-delà des exigences initiales, nous avons développé de nombreuses compétences techniques précieuses. Cette démarche nous a notamment permis d'acquérir une expertise en DevOps, particulièrement dans l'intégration et le déploiement continu, ainsi qu'en développement moderne à travers la création d'une API et d'une application mobile.

Pour ma part, ces compétences constituent une excellente préparation et facilitent ma transition vers la poursuite d'études en BUT Informatique, où ces technologies sont au cœur des enseignements de cette formation.

Conclusion générale

Ce projet de développement d'une solution de remplacement de l'ENT a pleinement répondu à sa vocation pédagogique. Au-delà de la réalisation technique, il nous a permis d'expérimenter les réalités du développement logiciel en équipe et d'acquérir des compétences directement valorisables dans notre future carrière professionnelle.

La problématique initiale d'amélioration de l'accès aux informations a trouvé une réponse technique complète, intégrant les contraintes de sécurité, d'ergonomie et de maintenance. Ce projet constitue ainsi une première expérience concrète de notre capacité à concevoir, développer et déployer une solution informatique répondant à un cahier des charges.

Le produit fini sera présenté lors de la démonstration à la fin de la soutenance.

Annexe

Glossaire

Terme	Définition
API (Application Programming Interface)	Interface logicielle permettant l'échange de données entre des applications. Le projet utilise une API REST développée avec FastAPI.
FastAPI	Framework web moderne pour Python, utilisé pour créer l'API. Il propose une documentation automatique, des performances élevées, et une gestion simple des routes.
PHP	Langage de script côté serveur utilisé pour le développement de l'interface web du projet.
GitHub	Plateforme d'hébergement de code utilisée pour la gestion du versionnage collaboratif via Git.
Ansible	Outil d'automatisation utilisé pour le déploiement du code source sur le serveur Raspberry Pi.
Raspberry Pi	Ordinateur monocarte utilisé comme serveur principal pour héberger l'API, la base de données, et les services web.
Systemd	Gestionnaire de services sous Linux permettant de démarrer, arrêter et surveiller des services système. Utilisé pour lancer l'API comme service.
Post, Get, Put, Delete	Méthodes HTTP utilisées par l'API pour créer, lire, modifier et supprimer des ressources.
Base de données MariaDB	Système de gestion de base de données relationnelle utilisé pour stocker les utilisateurs, réservations, matières, salles, etc.
Diagramme de cas d'utilisation	Représentation UML illustrant les interactions entre les utilisateurs et le système.
Diagramme de déploiement	Diagramme UML décrivant la disposition physique des composants du système sur l'infrastructure.
Cycle en V	Méthodologie de gestion de projet logiciel utilisée ici, avec des étapes séquentielles : besoin, spécification, conception, réalisation, tests.

Pôle Info	Nom du projet visant à améliorer la diffusion d'informations dans l'établissement scolaire.
Jeton d'authentification (Token) JWT	Clé numérique générée à la connexion pour sécuriser l'accès aux ressources API.
bcrypt	Algorithme de hachage utilisé pour sécuriser les mots de passe dans la base de données.
Htop	Outil CLI de surveillance des ressources système comme le CPU ou la RAM.
vcgencmd	Commande spécifique aux Raspberry Pi pour surveiller les paramètres matériels (température, fréquence, etc.).
Swagger UI	Interface de documentation interactive générée automatiquement par FastAPI pour tester les endpoints de l'API.

Documentation d'usage de l'API

Documentation API PoleInfo

Version 1.0.0

Développé par Elias GAUTHIER
Co-conception : Ethan CLEMENT
`elias.gauthier@lp2i-poitiers.fr`

26 mai 2025

Présentation

L'API PoleInfo est une interface de programmation dédiée à la gestion complète des réservations de salles pour le Pôle Info. Cette API offre un ensemble complet de fonctionnalités permettant l'administration des utilisateurs, la gestion des réservations, ainsi que la configuration des ressources (salles, créneaux, matières et classes).

Fonctionnalités principales

- Authentification sécurisée des utilisateurs avec génération de tokens JWT
- Gestion complète des utilisateurs (création, consultation, suppression)
- Système de réservation de salles avec gestion des conflits
- Administration des ressources : salles, créneaux horaires, matières et classes
- Contrôle d'accès basé sur les rôles utilisateurs

Table des matières

1 Authentification	4
1.1 POST /token - Connexion	4
1.1.1 Paramètres de requête	4
1.1.2 Réponses	4
1.2 GET /verify-token - Vérification du token	4
1.2.1 Réponses	4
2 Gestion des utilisateurs	4
2.1 GET /utilisateurs/ - Liste des utilisateurs	4
2.1.1 Réponses	5
2.2 POST /utilisateurs/ - Création d'utilisateur	5
2.2.1 Paramètres de requête	5
2.2.2 Réponses	5
2.3 DELETE /utilisateurs/ - Suppression d'utilisateur	5
2.3.1 Paramètres de requête	5
2.3.2 Réponses	5
3 Gestion des réservations	6
3.1 GET /réservations/ - Liste des réservations	6
3.1.1 Paramètres de requête (optionnels)	6
3.1.2 Réponses	6
3.2 POST /réservations/ - Création de réservation	6
3.2.1 Paramètres de requête	6
3.2.2 Réponses	6
3.3 PUT /réservations/ - Modification de réservation	6
3.3.1 Réponses	7
3.4 DELETE /réservations/ - Suppression de réservation	7
3.4.1 Paramètres de requête	7
3.4.2 Réponses	7
4 Gestion des créneaux	7
4.1 GET /creneaux/ - Liste des créneaux	7
4.1.1 Réponses	7
4.2 POST /creneaux/ - Création de créneau	7
4.2.1 Paramètres de requête	7
4.2.2 Réponses	8
4.3 DELETE /creneaux/ - Suppression de créneau	8
4.3.1 Réponses	8
5 Gestion des salles	8
5.1 GET /salles/ - Liste des salles	8
5.1.1 Réponses	8
5.2 POST /salles/ - Création de salle	8
5.2.1 Paramètres de requête	8
5.2.2 Réponses	9
5.3 DELETE /salles/ - Suppression de salle	9
5.3.1 Réponses	9

6	Gestion des matières	9
6.1	GET /matieres/ - Liste des matières	9
6.1.1	Réponses	9
6.2	POST /matieres/ - Création de matière	9
6.2.1	Paramètres de requête	9
6.2.2	Réponses	9
6.3	DELETE /matieres/ - Suppression de matière	10
6.3.1	Réponses	10
7	Gestion des classes	10
7.1	GET /classes/ - Liste des classes	10
7.1.1	Réponses	10
7.2	POST /classes/ - Création de classe	10
7.2.1	Paramètres de requête	10
7.2.2	Réponses	10
7.3	DELETE /classes/ - Suppression de classe	10
7.3.1	Réponses	11
8	Point d'entrée	11
8.1	GET / - Page d'accueil	11
8.1.1	Réponses	11
9	Sécurité et authentification	11
10	Codes d'erreur standards	11

1 Authentification

1.1 POST /token - Connexion

Description : Authentifie un utilisateur et génère un token JWT pour l'accès aux ressources protégées.

1.1.1 Paramètres de requête

Paramètre	Type	Description
username	string	Identifiant de connexion de l'utilisateur
password	string	Mot de passe de l'utilisateur
grant_type	string	Type d'autorisation (optionnel, doit être "password")
scope	string	Portée des autorisations (optionnel)

1.1.2 Réponses

Code	Description
200	Authentification réussie. Retourne le token JWT et les informations utilisateur
400	Identifiants incorrects
422	Erreur de validation des données d'entrée

Exemple de réponse (200) :

```
{
  "access_token": "jwt.token.value",
  "token_type": "bearer",
  "user_type": "admin",
  "user_name": "Elias Gauthier",
  "user_login": "egauthier"
}
```

1.2 GET /verify-token - Vérification du token

Description : Vérifie la validité d'un token JWT et retourne les informations associées.

Authentification requise : Oui

1.2.1 Réponses

Code	Description
200	Token valide. Retourne la validation et l'ID utilisateur
401	Token invalide ou expiré

2 Gestion des utilisateurs

2.1 GET /utilisateurs/ - Liste des utilisateurs

Description : Récupère la liste complète de tous les utilisateurs enregistrés dans le système.

2.1.1 Réponses

Code	Description
200	Liste des utilisateurs récupérée avec succès
404	Aucun utilisateur trouvé

2.2 POST /utilisateurs/ - Création d'utilisateur

Description : Ajoute un nouvel utilisateur au système. Cette opération nécessite des priviléges administrateur.

Authentification requise : Oui

2.2.1 Paramètres de requête

Paramètre	Type	Description
login	string	Identifiant unique de connexion
nom	string	Nom de famille de l'utilisateur
prenom	string	Prénom de l'utilisateur
password	string	Mot de passe de l'utilisateur
type	integer	Type d'utilisateur (niveau d'autorisation)

2.2.2 Réponses

Code	Description
200	Utilisateur créé avec succès
400	Un utilisateur avec ce login existe déjà
422	Erreur de validation des données

2.3 DELETE /utilisateurs/ - Suppression d'utilisateur

Description : Supprime un utilisateur existant du système. Cette opération nécessite des droits administrateur.

Authentification requise : Oui

2.3.1 Paramètres de requête

Paramètre	Type	Description
login	string	Identifiant de l'utilisateur à supprimer

2.3.2 Réponses

Code	Description
200	Utilisateur supprimé avec succès
404	Utilisateur non trouvé
422	Erreur de validation
500	Erreur lors de la suppression

3 Gestion des réservations

3.1 GET /reservations/ - Liste des réservations

Description : Récupère la liste des réservations avec possibilité de filtrage.

3.1.1 Paramètres de requête (optionnels)

Paramètre	Type	Description
salle	string	Filtre par numéro de salle
croissant	boolean	Ordre croissant des résultats
prof	string	Filtre par nom du professeur
reservation_id	integer	Récupère une réservation spécifique par ID

3.1.2 Réponses

Code	Description
200	Liste des réservations récupérée avec succès
410	Aucune réservation trouvée
422	Erreur de validation des paramètres

3.2 POST /reservations/ - Création de réservation

Description : Crée une nouvelle réservation de salle.

Authentification requise : Oui

3.2.1 Paramètres de requête

Paramètre	Type	Description
date	string (date)	Date de la réservation
duree	number	Durée de la réservation
numero_salle	string	Numéro de la salle à réserver
nom_matiere	string	Nom de la matière enseignée
heure_debut_creneau	string	Heure de début du créneau
login_user	string	Login de l'utilisateur faisant la réservation
nom_classe	string	Nom de la classe concernée
info	string	Informations complémentaires (optionnel)

3.2.2 Réponses

Code	Description
201	Réservation créée avec succès
400	Salle déjà réservée pour cet horaire
404	Utilisateur non trouvé
422	Erreur de validation des données

3.3 PUT /reservations/ - Modification de réservation

Description : Met à jour une réservation existante.

Authentification requise : Oui

3.3.1 Réponses

Code	Description
200	Réervation mise à jour avec succès
400	Erreur lors de la mise à jour
422	Erreur de validation

3.4 DELETE /reservations/ - Suppression de réservation

Description : Supprime une réservation existante.

Authentification requise : Oui

3.4.1 Paramètres de requête

Paramètre	Type	Description
id_reservation	integer	Identifiant de la réservation à supprimer

3.4.2 Réponses

Code	Description
200	Réervation supprimée avec succès
400	Erreur lors de la suppression
422	Paramètres insuffisants

4 Gestión des créneaux

4.1 GET /creneaux/ - Liste des créneaux

Description : Récupère la liste de tous les créneaux horaires disponibles.

4.1.1 Réponses

Code	Description
200	Liste des créneaux récupérée avec succès
404	Aucun créneau trouvé

4.2 POST /creneaux/ - Crédit de créneau

Description : Crée un nouveau créneau horaire. Cette opération nécessite des droits administrateur.

Authentification requise : Oui

4.2.1 Paramètres de requête

Paramètre	Type	Description
heure_debut	string	Heure de début du créneau

4.2.2 Réponses

Code	Description
201	Créneau créé avec succès
400	Créneau déjà existant
422	Erreur de validation

4.3 DELETE /creneaux/ - Suppression de créneau

Description : Supprime un créneau horaire existant. Cette opération nécessite des droits administrateur.

Authentification requise : Oui

4.3.1 Réponses

Code	Description
200	Créneau supprimé avec succès
404	Créneau non trouvé
422	Erreur de validation
500	Erreur serveur lors de la suppression

5 Gestion des salles

5.1 GET /salles/ - Liste des salles

Description : Récupère la liste de toutes les salles disponibles avec leurs caractéristiques.

5.1.1 Réponses

Code	Description
200	Liste des salles récupérée avec succès
404	Aucune salle trouvée

5.2 POST /salles/ - Création de salle

Description : Ajoute une nouvelle salle au système.

Authentification requise : Oui

5.2.1 Paramètres de requête

Paramètre	Type	Description
numero	string	Numéro unique de la salle
capacite	integer	Capacité d'accueil de la salle
type	string	Type de salle (Informatique, Labo, etc.)

5.2.2 Réponses

Code	Description
201	Salle créée avec succès
400	Une salle avec ce numéro existe déjà
422	Erreur de validation

5.3 DELETE /salles/ - Suppression de salle

Description : Supprime une salle existante du système.

Authentification requise : Oui

5.3.1 Réponses

Code	Description
200	Salle supprimée avec succès
404	Salle non trouvée
422	Erreur de validation
500	Erreur lors de la suppression

6 Gestion des matières

6.1 GET /matieres/ - Liste des matières

Description : Récupère la liste de toutes les matières d'enseignement.

6.1.1 Réponses

Code	Description
200	Liste des matières récupérée avec succès
404	Aucune matière trouvée

6.2 POST /matieres/ - Crédation de matière

Description : Crée une nouvelle matière d'enseignement. Cette opération nécessite des droits administrateur.

Authentification requise : Oui

6.2.1 Paramètres de requête

Paramètre	Type	Description
nom	string	Nom de la matière

6.2.2 Réponses

Code	Description
201	Matière créée avec succès
400	Matière déjà existante

422	Erreur de validation
-----	----------------------

6.3 DELETE /matieres/ - Suppression de matière

Description : Supprime une matière existante. Cette opération nécessite des droits administrateur.

Authentification requise : Oui

6.3.1 Réponses

Code	Description
200	Matière supprimée avec succès
404	Matière non trouvée
422	Erreur de validation
500	Erreur serveur lors de la suppression

7 Gestion des classes

7.1 GET /classes/ - Liste des classes

Description : Récupère la liste de toutes les classes.

7.1.1 Réponses

Code	Description
200	Liste des classes récupérée avec succès
404	Aucune classe trouvée

7.2 POST /classes/ - Création de classe

Description : Crée une nouvelle classe. Cette opération nécessite des droits administrateur.

Authentification requise : Oui

7.2.1 Paramètres de requête

Paramètre	Type	Description
nom	string	Nom de la classe

7.2.2 Réponses

Code	Description
201	Classe créée avec succès
400	Une classe avec ce nom existe déjà
422	Erreur de validation des données

7.3 DELETE /classes/ - Suppression de classe

Description : Supprime une classe existante. Cette opération nécessite des droits administrateur.

Authentification requise : Oui

7.3.1 Réponses

Code	Description
200	Classe supprimée avec succès
404	Classe non trouvée
422	Erreur de validation
500	Erreur serveur lors de la suppression

8 Point d'entrée

8.1 GET / - Page d'accueil

Description : Point d'entrée principal de l'API qui affiche un message de bienvenue avec la date du jour au format français.

8.1.1 Réponses

Code	Description
200	Message de bienvenue affiché avec succès

9 Sécurité et authentification

L'API utilise un système d'authentification basé sur des tokens JWT (JSON Web Tokens). Les endpoints protégés nécessitent l'inclusion du token dans l'en-tête Authorization de la requête HTTP sous la forme :

`Authorization: Bearer <token>`

Les opérations d'administration (création, modification, suppression des ressources) sont réservées aux utilisateurs ayant les privilèges administrateur appropriés.

10 Codes d'erreur standards

Code	Description
200	Succès - Opération réalisée avec succès
201	Créé - Ressource créée avec succès
400	Requête incorrecte - Données invalides ou conflit
401	Non autorisé - Authentification requise ou token invalide
404	Non trouvé - Ressource inexistante
410	Indisponible - Ressource temporairement indisponible
422	Entité non traitable - Erreur de validation des données
500	Erreur serveur - Erreur interne du serveur