

# Dossier Technique du projet « Accès campus » - Partie PSW (Poste Serveur Web)

## Table des matières

I – Présentation.....	2
1.1 - Contexte et objectifs .....	2
1.2 - Description du système .....	3
1.3 – Description de la partie personnelle .....	3
II – Conception .....	5
2.1 – Objectif .....	5
2.2 – Mise en place PSW .....	6
2.3 – Fonctionnement du site .....	6
2.4 – Réalisation .....	7
2.5 – Test.....	14
2.5.1 – Test unitaire .....	14
2.5.2 – Test d’intégration.....	15
III- Conclusion.....	17

# I – Présentation

## 1.1 - Contexte et objectifs

Le projet Campus Accès vise à améliorer la gestion et la sécurité des accès aux salles du Campus Saint Aubin La Salle. Grâce à un système automatisé basé sur des badges RFID, les étudiants et le personnel peuvent accéder aux salles de manière sécurisée tout en enregistrant leurs entrées et sorties. L'objectif est d'assurer un contrôle centralisé des accès, optimisé grâce au Poste Serveur Web (PSW).



Figure 1 Campus Saint Aubin La Salle

Ma tâche personnelle dans ce projet de grande envergure, est de permettre la visualisation pour les élèves ainsi que le personnel des absences, retards et des informations des différentes salles. (voir Figure 2).

Étudiant 4	Liste des fonctions assurées par l'étudiant	
EC <input type="checkbox"/> IR <input checked="" type="checkbox"/>	Poste Serveur Web : <ul style="list-style-type: none"> <li>• Visualiser les disponibilités des salles</li> <li>• Visualiser la présence des étudiants</li> <li>• Rechercher une salle libre pour une période donnée</li> <li>• Afficher les informations de présence dans une salle pour une période donnée</li> </ul>	Installation : <ul style="list-style-type: none"> <li>• Installer l'EDI utilisé.</li> <li>• Installer la maquette de test du système.</li> </ul> Mise en œuvre : <ul style="list-style-type: none"> <li>• Le serveur web, l'accès à la base de données, l'environnement de développement</li> </ul> Configuration : <ul style="list-style-type: none"> <li>• Infrastructure réseau externe.</li> </ul> Réalisation : <ul style="list-style-type: none"> <li>• Réalisation des fonctionnalités en charge.</li> </ul> Documentation : <ul style="list-style-type: none"> <li>• Prototypage des interfaces-utilisateur (Mockup)</li> <li>• Dossier décrivant les choix retenus.</li> <li>• Dossier décrivant le détail de chaque fonctionnalité développée.</li> <li>• Dossier de réalisation des différentes fonctionnalités en charge.</li> </ul>

Figure 2 Mission étudiant 4

## 1.2 - Description du système

Le Poste Serveur Web (PSW) est au cœur du système, il hébergera une base de données, une API ainsi qu'un site web. Ma partie est celle du site web, il permettra aux élèves, aux professeurs, à la vie scolaire de se connecter via leurs comptes personnels pour y voir leurs absences, retards, et les indications des différentes salles du lycée. La seule communication du site sera avec l'API à l'aide de requêtes GET et POST.

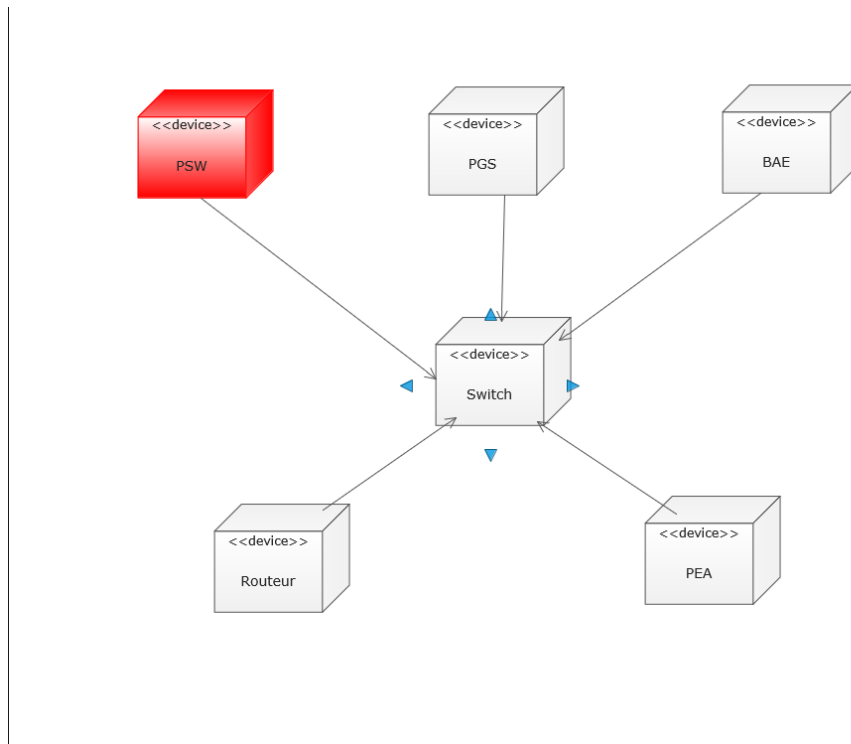


Figure 3 Diagramme de déploiement

## 1.3 – Description de la partie personnelle

A l'aide du diagramme des cas d'utilisations fourni dans le cahier des charges mes différentes tâches sont données.

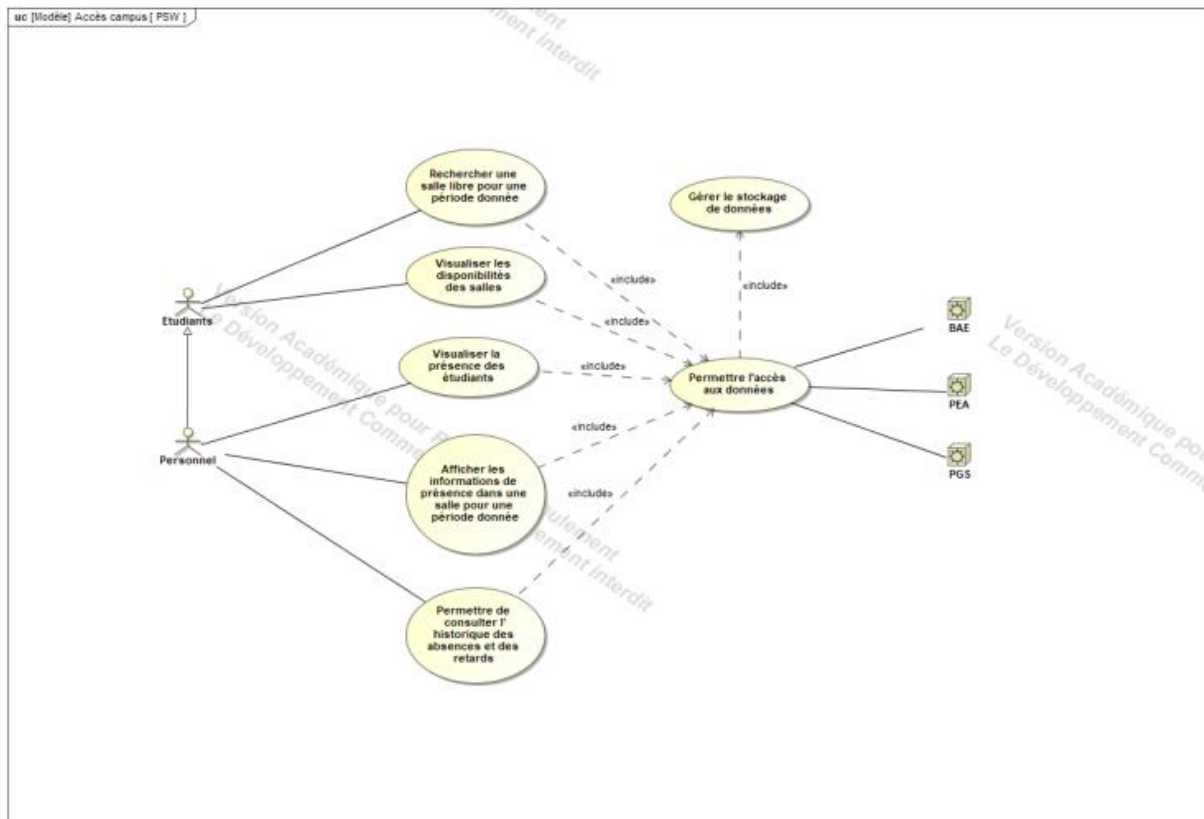


Figure 3 Diagramme des cas d'utilisations

Le PSW est divisé en 3 parties, BDD, API et site web, Lorick Fouquet s'occupant de la partie gestion de données, il nous donc a fallu travailler en binôme tout au long du projet. On a donc dès le début dû se mettre d'accord sur certains choix pour travailler sur le PSW tel que l'OS que nous allons utiliser qui est donc Debian 12. Nous avons fait le choix de travailler chacun sur une VM puis lorsque nous avons terminés une éventuelle tâche on assemblerait nos morceaux fonctionnels sur le serveur. Sur le PSW, nous avons d'abord installé différents paquets dont ufw comme pare feu qui nous a permis de fermer tous les ports inutilisés afin d'augmenter la sécurité sur le PSW, les ports ouverts sont écrits dans le fichier Installation.txt dans la partie PSW sur le GitHub du projet. En ce qui me concerne les paquets nodejs et npm ont été aussi installés pour le codage du site.

Afin de préciser les besoins pour le site PSW voici un diagramme de requiement :

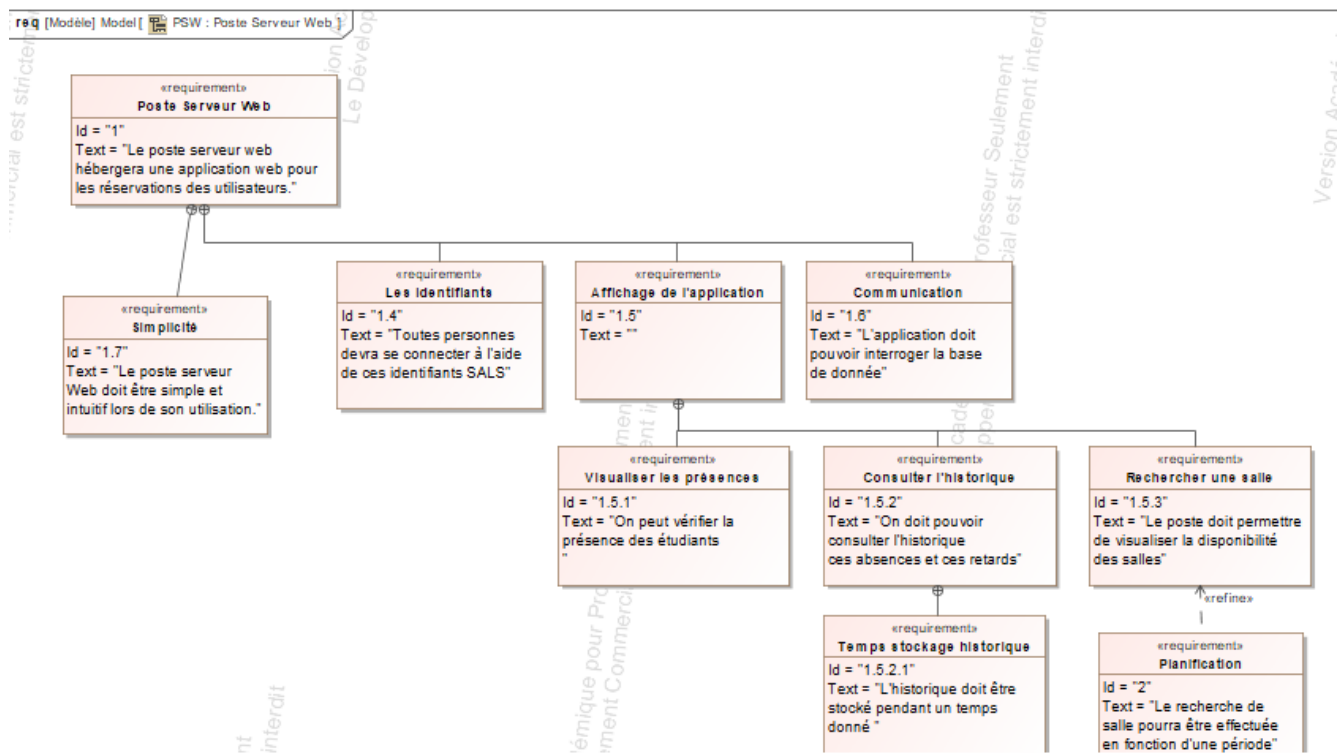


Figure 4 Diagramme de requiement

## II – Conception

### 2.1 – Objectif

L'objectif de ce premier incrément pour ma part sera :

- Créer la base du site web
- Permettre de consulter l'historique des absences et des retards.
- Préparer le serveur

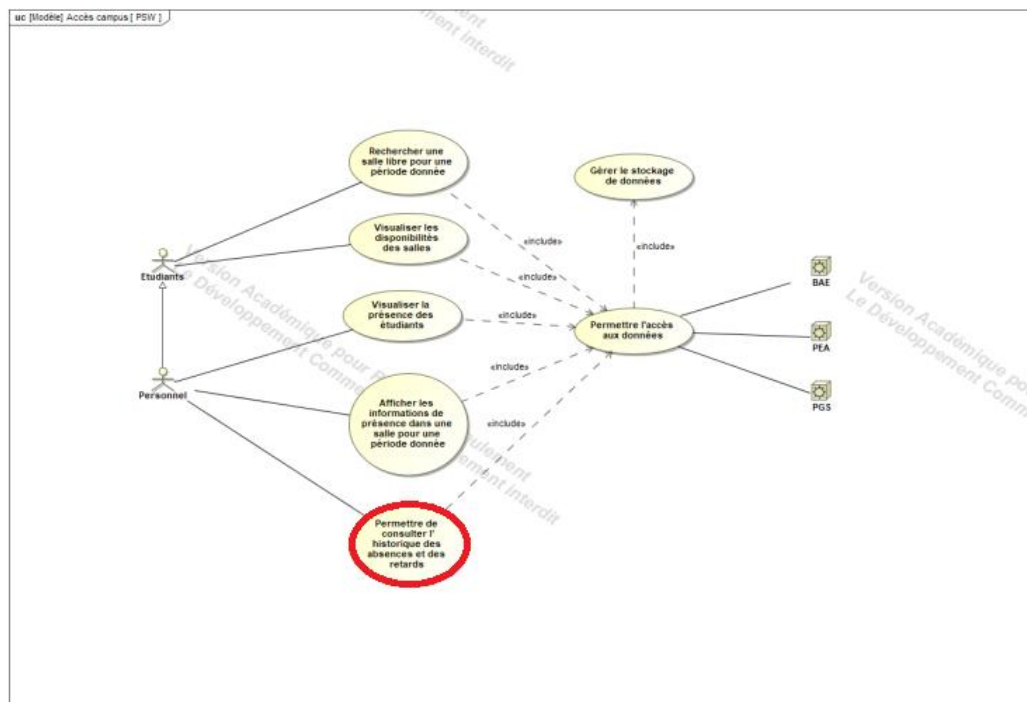


Figure 5 Diagramme des cas d'utilisations

## 2.2 – Mise en place PSW

Pour cette partie je ne suis pas seul à travailler dessus, je continue de partager mon travail avec Lorick Fouquet.

On a choisi de partir sur Debian 12 comme OS, pour sa stabilité. C'est plus adéquat que Ubuntu pour du faire du 24h sur 24. On installe ssh dès le début pour ensuite pouvoir travailler à distance dessus puis mettons en place le pare-feu ufw.

Nous avons installé micro comme éditeur de texte qui est plus simple et agréable d'utilisation que vim ou nano.

## 2.3 – Fonctionnement du site

Le site aura un fonctionnement simple, il devra être utilisable par les élèves, les professeurs, la vie scolaire, des personnes pas toujours habituées à utiliser l'informatique. Pour se faire, lors de l'utilisation du site il faudra obligatoirement se connecter à l'aide d'un compte personnel attribué à chacun. Une fois connecté, plusieurs possibilités seront possibles, elles dépendront de leurs « rôle » (professeur, élèves, administrateur...), par exemple les élèves pourront voir leurs propres absences et retards ainsi que les informations sur les salles tandis que le personnel auront accès à la liste des élèves complètes avec la possibilité de voir leurs absences et retards en plus d'avoir accès aux informations des différentes salles du lycée.

## 2.4 – Réalisation

De ce qui est de la réalisation, le site web ne s'est évidemment pas fait en une fois, plusieurs versions existent. Elles sont disponibles sur le GitHub afin de permettre un retour en arrière en cas de gros problèmes.

Le premier cas d'utilisation était de visualiser les absences et retards, c'est donc par là que j'ai commencé.

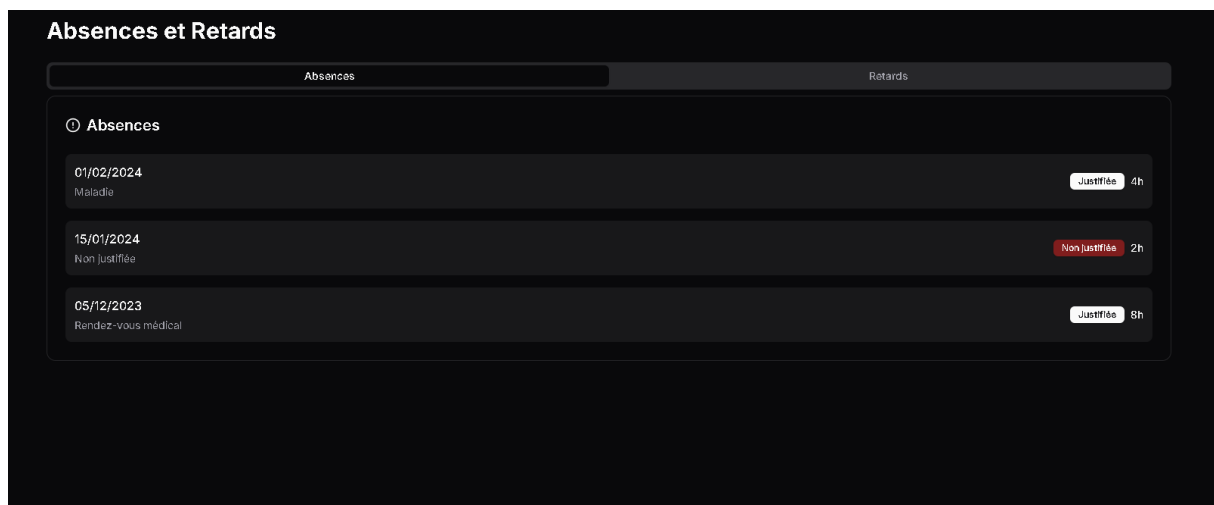


Figure 6 Illustration absences / retards

Cependant au fur et à mesure il y a eu de l'évolution dans le projet, et la « figure 6 » s'est transformé en « figure 7 », plus esthétique et plus agréable à l'œil pour l'utilisateur.

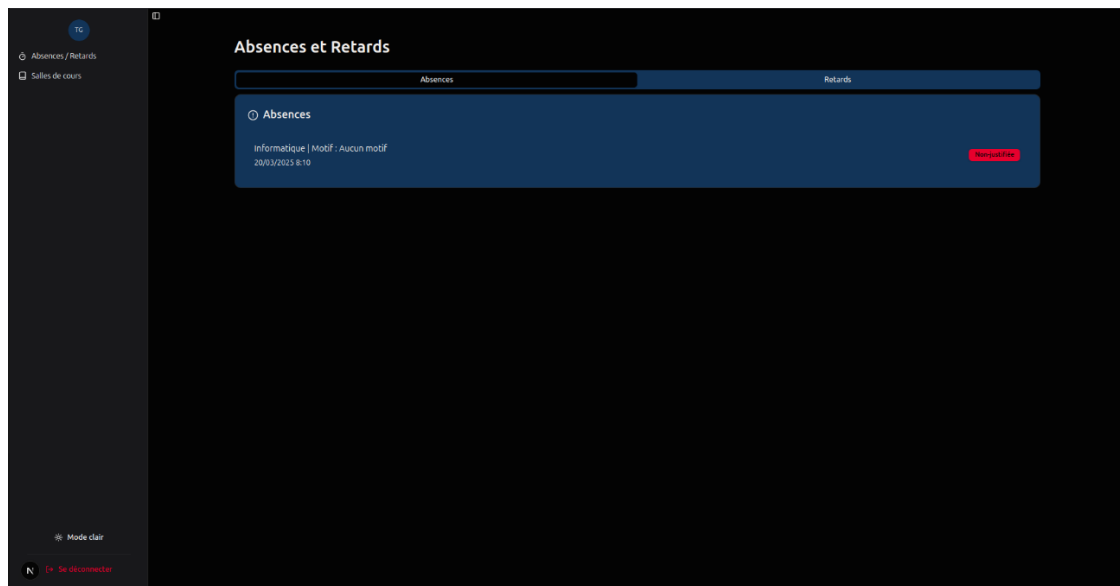


Figure 7 Illustration absences / retards

Pour récupérer les absences et retards j'utilise les routes données par Lorick Fouquet en utilisant la méthode GET dans mon code.

```
1 import { NextResponse } from "next/server";
2
3 const API_URL = "http://127.0.0.1:8000/psw";
4
5 export async function GET(request: Request, context: { params: { id: string } }) {
6   const { id } = await Promise.resolve(context.params);
7   //requete get pour récupérer les absences avec en param l'id de l'eleve
8   const res = await fetch(`${API_URL}/absence/${id}`, {
9     method: "GET",
10    headers: { "Accept": "application/json" },
11  });
12
13  //si aucune absence, renvoi tableau vide et pas d'erreur en console
14  if (res.status === 404) return NextResponse.json([], { status: 200 });
15
16  //si erreur, renvoie erreur
17  if (!res.ok) return NextResponse.error();
18
19  //sinon renvoi reponse
20  const data = await res.json();
21  return NextResponse.json(data);
22 }
```

Figure 8 Requête GET absence

Pour faire les requêtes d'absence et de retard il fallait d'abord avoir l'id de l'utilisateur qui voulait les voir, pour se faire la récupération de l'id se fait lorsque l'utilisateur se connecte dans la page



de login. Nous faisons une requête POST, suite à cela, la BDD renvoie les infos de l'utilisateur ainsi que son id.

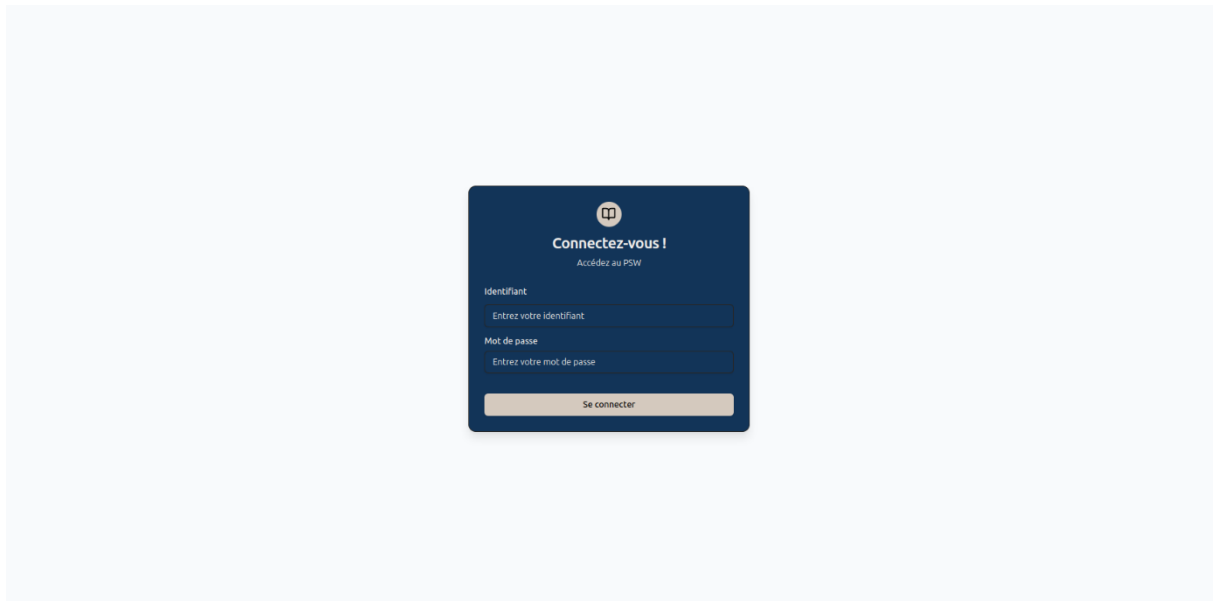


Figure 9 Page de login

```
1 import { NextResponse } from "next/server";
2 import { cookies } from "next/headers";
3
4 // URL de l'API pour l'authentification
5 const API_URL = "http://localhost:8000/psw/login/";
6
7 export async function POST(req: Request) {
8   try {
9     // Récupération des données envoyées par l'utilisateur
10    const { username, password } = await req.json();
11
12    // Création du body de la requête pour l'API
13    const body_req = {
14      identifiant: username,
15      mot_de_passe: password,
16    };
17
18    // Envoi de la requête à l'API pour vérifier les identifiants
19    const Response = await fetch(API_URL, {
20      method: "POST",
21      headers: { "Content-Type": "application/json" },
22      body: JSON.stringify(body_req),
23    });
24
25    // Vérification si erreur (mauvais identifiants)
26    if (!Response.ok) {
27      const responseBody = await Response.json();
28      return NextResponse.json(
29        { error: responseBody.error || "Identifiant ou mot de passe incorrect" },
30        { status: 401 }
31      );
32    }
33
34    // Récup données de la réponse
35    const data = await Response.json();
36
37    // Vérification si connexion non réussie
38    if (!data.success) {
39      return NextResponse.json(
40        { error: "Identifiant ou mot de passe incorrect" },
41        { status: 401 }
42      );
43    }
44
45    // Créer objet utilisateur avec ID et rôle
46    const user = {
47      id_utilisateur: data.id_utilisateur,
48      role: data.role,
49    };
50
51    // Création cookie pour stocker informations utilisateur
52    (await cookies()).set("user", JSON.stringify(user), {
53      httpOnly: true, // Empêche l'accès au cookie via JS côté client
54      sameSite: "strict", // Empêche l'envoi du cookie avec des requêtes tierces
55      maxAge: 3600, // Durée de validité du cookie (1 heure)
56    });
57
58    // Réponse si réussie
59    return NextResponse.json({ message: "Connexion réussie" });
60  } catch (error) {
61    // Gestion des erreurs serveur
62    console.error("Erreur serveur :", error);
63    return NextResponse.json({ error: "Erreur serveur" }, { status: 500 });
64  }
65 }
66
```

Dans le code, nous faisons une requête POST avec les identifiant personnel de l'utilisateur puis créons un cookie qui sera ensuite réutiliser pour accéder au site.

Figure 10 Code requête login

Bien sûr, le point de vue vie scolaire ou professeur n'est pas le même que celui des élèves. Le personnel à accès à la liste des élèves puis aux absences et retards de chacun d'entre eux.

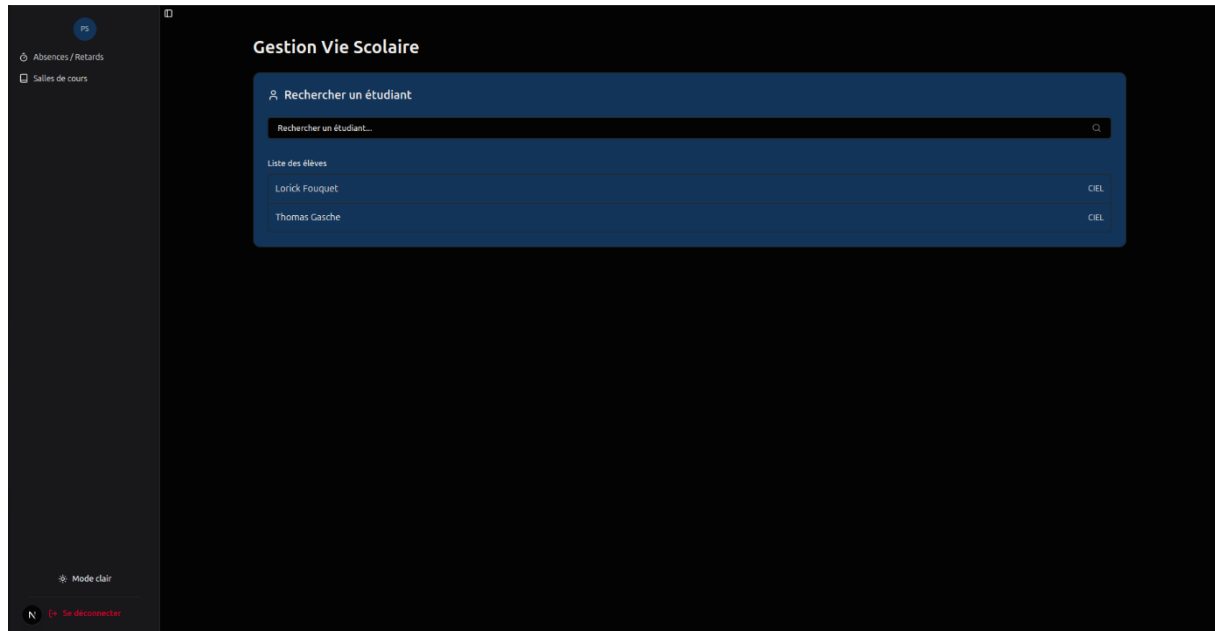


Figure 11 Absences / Retards vie scolaire

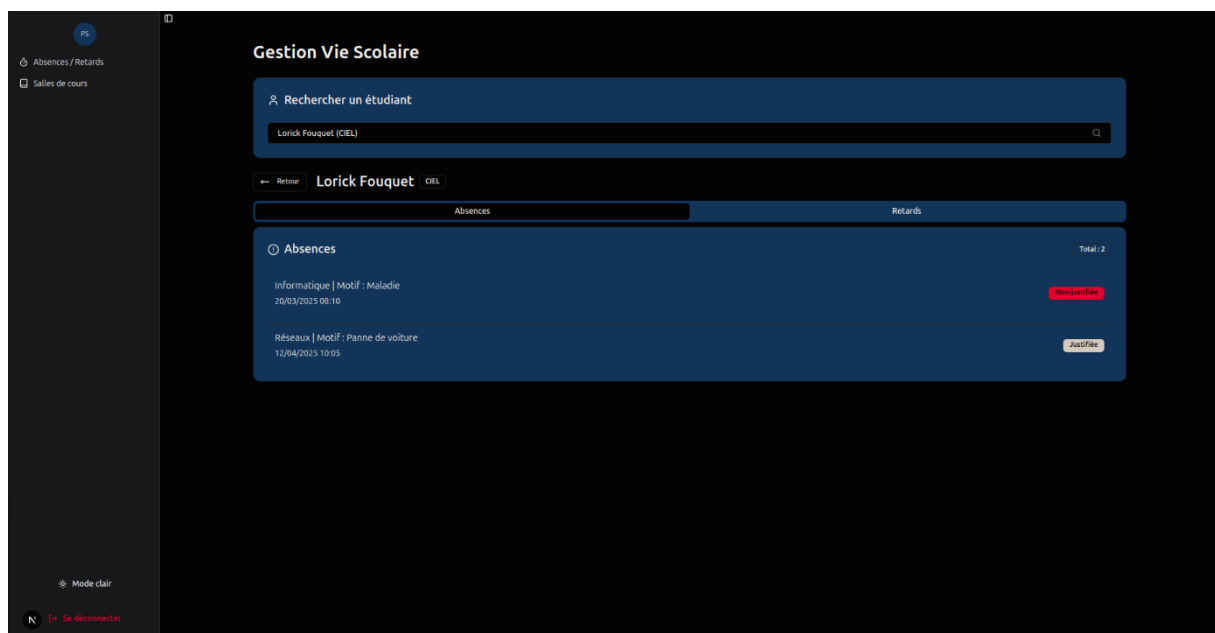
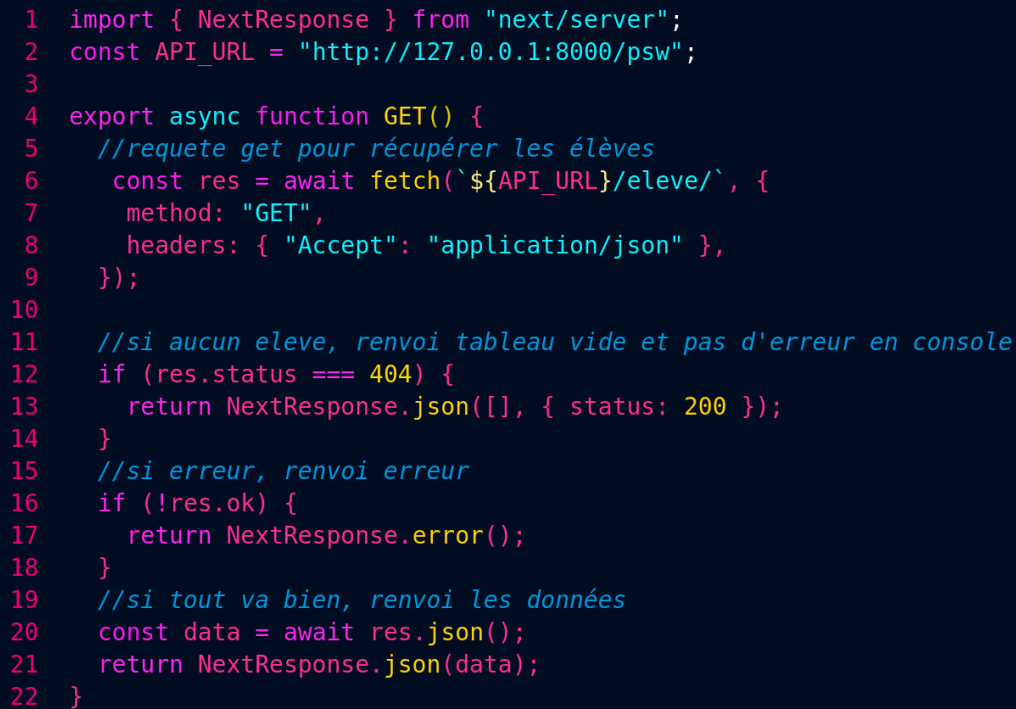


Figure 12 Absences / Retards d'un élève

Afin d'avoir la liste des élèves, Lorick Fouquet a créé une route pour me permettre une requête GET qui sert à afficher au personnel connecté d'afficher les élèves.



```
1 import { NextResponse } from "next/server";
2 const API_URL = "http://127.0.0.1:8000/psw";
3
4 export async function GET() {
5   //requete get pour récupérer les élèves
6   const res = await fetch(`${API_URL}/eleve/`, {
7     method: "GET",
8     headers: { "Accept": "application/json" },
9   });
10
11   //si aucun eleve, renvoi tableau vide et pas d'erreur en console
12   if (res.status === 404) {
13     return NextResponse.json([], { status: 200 });
14   }
15   //si erreur, renvoi erreur
16   if (!res.ok) {
17     return NextResponse.error();
18   }
19   //si tout va bien, renvoi les données
20   const data = await res.json();
21   return NextResponse.json(data);
22 }
```

Figure 13 Requête GET liste élève

Une fois la partie absences/retards terminées, il fallait s'occuper des salles et plus précisément des informations des salles. Nous devons voir le nombre d'élèves présents, si la salle est occupée ou non, si elle est réservée ou non et le cours actuel ou futur qui s'y déroule ou déroulera. Toutes ces informations seront visibles pour le personnel ainsi que les élèves. Voici en figure 14 et 15 l'affichage coté utilisateur des salles ainsi que des informations sur celles-ci.



Figure 14 Affichage des salles

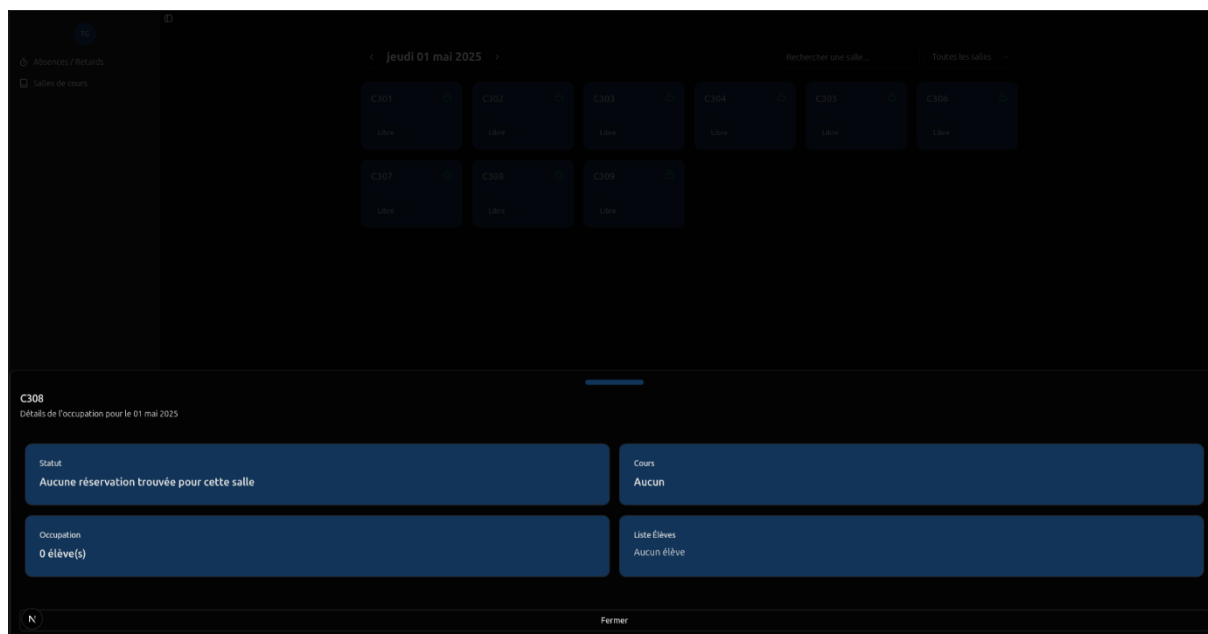


Figure 15 Affichage des informations de salles

```
1 import { NextResponse } from "next/server";
2 const API_URL = "http://127.0.0.1:8000";
3
4 export async function GET() {
5   //requete get pour récupérer les salles
6   const res = await fetch(`${API_URL}/salle/`, {
7     method: "GET",
8     headers: { Accept: "application/json" },
9   });
10
11   //si aucune salle, renvoi tableau vide et pas d'erreur en console
12   if (res.status === 404) {
13     return NextResponse.json([], { status: 200 });
14   }
15   //si erreur, renvoi erreur
16   if (!res.ok) {
17     return NextResponse.error();
18   }
19   //si tout va bien, renvoi les données
20   const data = await res.json();
21   return NextResponse.json(data);
22 }
23
```

Figure 16 Requête GET salle

```
1 import { NextResponse } from "next/server";
2 const API_URL = "http://127.0.0.1:8000/psw";
3
4 export async function GET(
5   request: Request,
6   { params }: { params: { id: string } }
7 ) {
8   const { id } = await Promise.resolve(params);
9   /// Requête GET pour récupérer les informations de la salle
10  const res = await fetch(`${API_URL}/salle/${id}/activite`, {
11    method: "GET",
12    headers: { Accept: "application/json" },
13  });
14
15  const data = await res.json();
16  // Si aucune activité trouvée, renvoie reponse de l'api
17  if (res.status === 404) {
18    return NextResponse.json(data, { status: 200 });
19  }
20
21  // Si erreur, renvoie erreur
22  if (!res.ok) {
23    return NextResponse.error();
24  }
25
26  // Si tout va bien, récup et renvoie données
27  return NextResponse.json(data);
28 }
29
```

Figure 17 Requête GET activité salle

En plus des cas d'utilisation terminés, un ajout non demandé a été réalisé : l'ajout d'un « mode clair/mode sombre ».

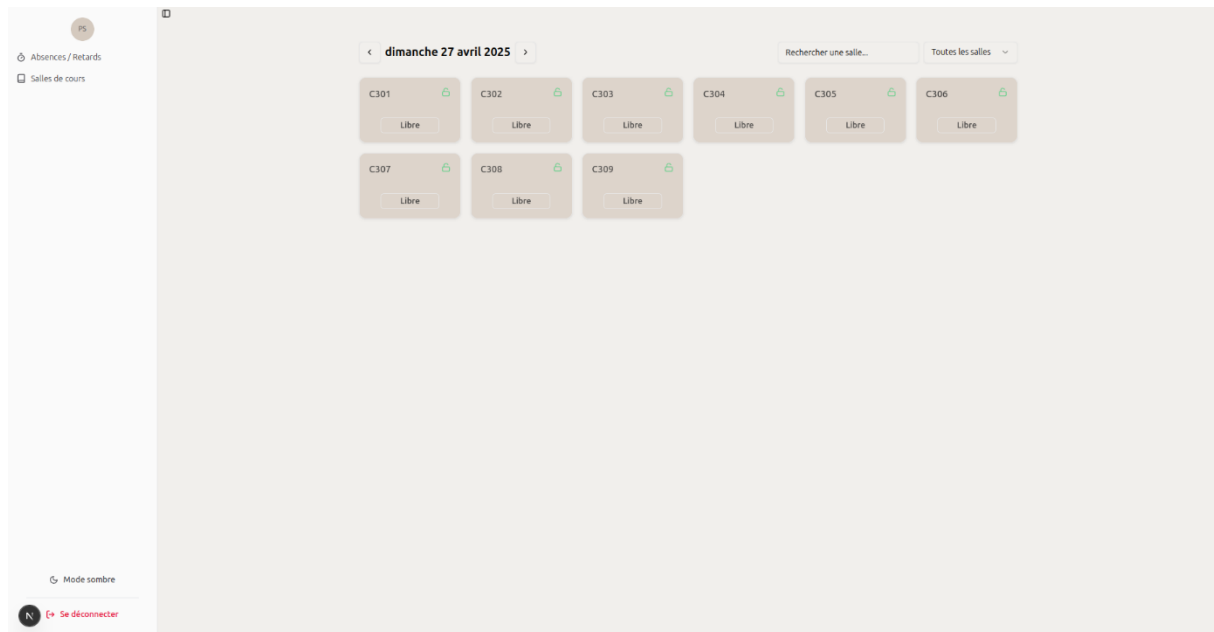


Figure 18 Mode clair mode sombre

## 2.5 – Test

### 2.5.1 – Test unitaire

Voici un plan de test unitaire, il permet de tester la fonction AbsenceRetardsPage sur la version brute du PSW.

#### Plan de test – Fonction AbsencesRetardsPage

##### 1 - Identification du test

**Nom** : Affichage élèves de ses absences et retards

**Numéro** : TU1

##### 2 - Référence du module testé

app/absences/eleve/page.tsx – Fonction AbsencesRetardsPage

##### 3 - Objectif du test

S'assurer qu'un élève peut voir ses absences et retards :

- Retour correct des absences et retards
- Affichage correct des absences et retards

#### 4 - Procédure du test

- **Initialisation :**
  - Un utilisateur
  - Des retards et absences créé en brut
- **Lancement :** clique sur Absences / Retards
- **Observation :** Affichage des absences et retards

#### 5 - Résultats attendus

N° Test	Condition	Résultat attendu	Statut attendu
1	Aucune absence ou retard	Aucune absence ou Aucun retard	Echec
2	Absence et retard présents	Liste des retards et absences (cours, horaire, durée, motif...)	Succès

#### 6 - Moyens à mettre en œuvre

- **Logiciels :** Oracle VirtualBox
- **Matériel :** Poste informatique

### 2.5.2 – Test d'intégration

Voici un plan de test unitaire, il permet de tester la fonction le login sur la version finale du PSW.

#### Plan de test – Login

##### 1 - Identification du test

**Nom :** Login utilisateur

**Numéro :** TI5

## 2 - Référence du module testé

/psw/login/

## 3 - Objectif du test

L'utilisateur se connecte à son compte personnel

- Envoie des identifiants utilisateurs
- Connecte l'utilisateur au PSW

## 4 - Procédure du test

- **Initialisation :**
  - Un utilisateur élève existe dans la BDD
- **Lancement :** L'utilisateur entre son identifiant dans la case « Identifiant », puis son mot de passe dans la case « Mot de passe » et pour finir clique sur « Se connecter »
- **Observation :** L'utilisateur se connecte au PSW

## 5 - Résultats attendus

N° Test	Condition	Résultat attendu	Statut attendu
1	Aucun compte utilisateur	Impossible de se connecter	Echec
2	Erreur dans les identifiants	Identifiant ou mot de passe incorrect	Echec
3	Identifiant correct	L'utilisateur se connecte au PSW	Succès

## 6 - Moyens à mettre en œuvre

- **Logiciels :** FastAPI, PostgreSQL, express, pm2, pgAdmin
- **Matériel :** Poste informatique connecté au réseau



### III- Conclusion

Pour finir, ce projet m'a permis de développer de nombreuses compétences essentielles, tant sur le plan technique que sur le plan humain. Travailler en équipe m'a appris l'importance de la communication, de l'adaptation aux autres et de l'entraide, des qualités indispensables à la réussite collective. Chaque membre ayant sa propre façon de travailler, il a été nécessaire de trouver un équilibre pour avancer ensemble dans la même direction.

Sur le plan technique, ce projet a été l'occasion pour moi de me familiariser avec le fonctionnement des API et des bases de données, qui constituaient le cœur du projet. J'ai également compris que la réalisation d'un projet ne se fait jamais en une seule tentative : rien n'est parfait du premier coup. Les améliorations, les ajustements et les évolutions font partie intégrante du processus de création. Cette prise de conscience s'est concrétisée à travers l'utilisation du versioning, qui permet de suivre et d'analyser l'évolution progressive du site PSW sur GitHub.

En conclusion, ce projet a été une expérience extrêmement enrichissante, tant sur le plan professionnel que personnel. Il m'a non seulement permis d'acquérir des compétences techniques solides, mais également de mieux comprendre l'importance de la collaboration, de la rigueur et de l'adaptabilité dans le monde de l'informatique.