## *Projet CA25*

# **PARTIE INDIVIDUELLE**

## ETUDIANT 1

# Gestion d’une demande d’autorisation d’accès

# PALAS Aykut

SOMMAIRE

[*Projet CA25* 1](#_Toc1)

[**PARTIE INDIVIDUELLE** 1](#_Toc2)

[**REMERCIEMENTS** 3](#_Toc6)

[**1. INTRODUCTION** 4](#_Toc7)

[**2. PRESENTATION DE MA PARTIE** 5](#_Toc8)

[2.1 Présentation globale de ma partie 5](#_Toc9)

[2.2 Recette 6](#_Toc10)

[2.2.1 Recette fonctionnelle 6](#_Toc11)

[2.2.2 Fonction de Contrainte 7](#_Toc12)

[2.3 Contraintes Matériel et Logiciel 7](#_Toc13)

[2.4.1 Utilité base de données 8](#_Toc14)

[2.4.2 Image de la table concerné sur la base de données. 9](#_Toc15)

[2.5.1 Page d’accueil du site WEB 10](#_Toc17)

[2.5.2 Formulaire 11](#_Toc18)

[2.5.3 Dashboard 12](#_Toc19)

[**3.Programmation Langage** 16](#_Toc20)

[3.1. HTML 16](#_Toc21)

[3.1.1 Zone de texte 16](#_Toc22)

[3.1.2 Les champs spécialisés 16](#_Toc23)

[3.1.3 Le bouton de soumission 17](#_Toc24)

[3.1.4 La balise <form> 17](#_Toc25)

[3.1.5 Les balises de structure et de style 17](#_Toc26)

[3.1.6 Conclusion 18](#_Toc27)

[3.2 CSS 18](#_Toc28)

[3.3 PHP 19](#_Toc30)

[3.3.1 Explication du PHP 19](#_Toc31)

[3.3.2 Utilisation du PHP 20](#_Toc34)

[3.3.3 Pages PHP 27](#_Toc49)

[**4. Envoi des mails** 29](#_Toc50)

[4.1. Composer 29](#_Toc51)

[4.2 PHP Mailer 30](#_Toc53)

[4.2.1 Installation 30](#_Toc54)

[4.2.2 Utilisation 30](#_Toc55)

[**5. Bonus** 34](#_Toc60)

[5.1 Inscription Admin 34](#_Toc61)

[5.2 Déconnexion 36](#_Toc65)

[**6. CONCLUSION** 36](#_Toc66)

# **REMERCIEMENTS**

Je tiens à remercier mes professeurs d’informatique pour m’avoir enseigné durant ces deux années de BTS Cybersécurité : M. Fournier, M. Battin ainsi que M. Maurin.  
Si j’ai pu autant apprendre dans ce domaine, c’est sans aucun doute grâce à la qualité de l’enseignement de ces professeurs, en particulier M. Fournier et M. Maurin, qui ont été très présents pour nous accompagner tout au long de ce projet.

Je souhaite également remercier mon équipe, avec qui j’ai partagé de très bons moments et appris à gérer un projet : Laurent Nhu, Hamza Aydogdu et enfin Raphaël Rodriguez.  
Nous nous sommes tous surpassés pour ce projet, et nous avons énormément coopéré afin d’en assurer la réussite.

## 

## **1. INTRODUCTION**

Dans le cadre du BTS CIEL – option A : Informatique et Réseaux – au lycée Charles Poncet, les étudiants doivent réaliser un projet technique pour l’épreuve E6. Ce projet permet de mettre en œuvre leurs compétences en réseaux, développement embarqué, sécurité et gestion de projet.

Le projet présenté dans ce rapport porte sur la mise en place d’un système de contrôle d’accès informatisé pour le local technique du bâtiment A du lycée. L’objectif principal est de remplacer un système de verrouillage mécanique par une solution plus moderne, sécurisée et automatisée, reposant sur l’utilisation de cartes NFC nominatives. Ce système doit permettre à un administrateur de gérer dynamiquement les autorisations d’accès, tout en assurant la traçabilité des entrées, le suivi des tentatives d’accès non autorisées, ainsi que la surveillance du local.

Ce rapport présentera l’ensemble des étapes du projet, ainsi que la répartition des rôles au sein de l’équipe. Je détaillerai ensuite plus spécifiquement la partie du projet dont j’ai eu la charge, correspondant aux tâches attribuées à l’étudiant 1.

Nous pouvons alors entamer le développement.

# **2. PRESENTATION DE MA PARTIE**

## 2.1 Présentation globale de ma partie

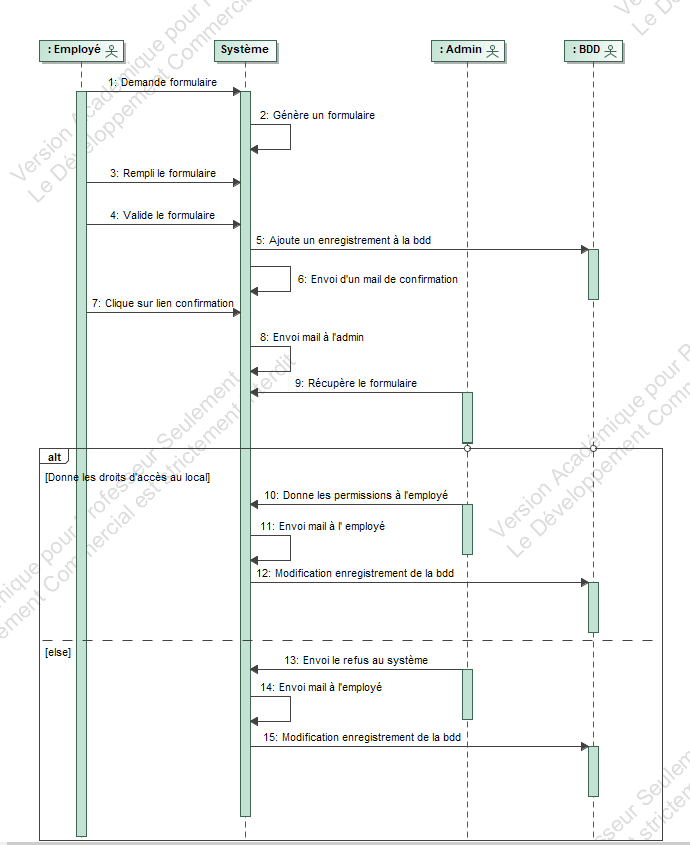
En tant qu’étudiant 1, j’ai été chargé de la gestion des demandes d’autorisation d’accès au local technique. Cela comprenait notamment la création d’un formulaire web sécurisé permettant aux utilisateurs de soumettre leurs informations, la mise en place d’un système de vérification d’email via l’envoi de liens de confirmation, ainsi que l’envoi automatique de notifications par email à l’administrateur lors de l’enregistrement d’une nouvelle demande.

J’ai également travaillé sur la sécurisation des échanges de données entre l’extérieur et le réseau du lycée, en intégrant des mécanismes de protection adaptés. Par ailleurs, j’ai assuré la configuration du réseau nécessaire au bon fonctionnement du système, en veillant au respect des règles de sécurité et de connectivité.

Enfin, j’ai rédigé la documentation associée, incluant le cahier de recettes pour cette partie du projet ainsi qu’une notice technique détaillant la configuration réseau et les mesures de sécurisation mises en place.

## 2.2 Recette

Voici le diagramme de séquence :



### 2.2.1 Recette fonctionnelle

La validation des fonctionnalités développées a été effectuée selon les étapes suivantes :

* Le formulaire de demande d’accès s’affiche correctement et permet la saisie des informations requises (identité, coordonnées, dates et motif de la demande).
* Une fois les champs renseignés, l’utilisateur peut soumettre le formulaire. À ce stade, aucune donnée n’est encore traitée tant que l’email n’a pas été validé.
* Un email contenant un lien de validation est alors envoyé à l’utilisateur.
* Lorsque l’utilisateur clique sur ce lien, la demande devient active, et un email est automatiquement envoyé à l’administrateur pour l’en informer.
* Si l’administrateur valide la demande dans un délai de 10 minutes, celle-ci est enregistrée comme « aboutie ». Dans le cas contraire, elle est automatiquement classée comme « non aboutie ».
* Des tests ont confirmé la sécurisation des échanges (formulaire, envois d’emails, accès à la base de données) ainsi que le respect des délais de validation définis.

### 2.2.2 Fonction de Contrainte

Certaines contraintes spécifiques ont été intégrées au système afin d’en garantir la fiabilité et la sécurité :

* Si l’adresse email renseignée par l’utilisateur est déjà présente dans la base de données, la demande sera automatiquement rejetée.
* L’administrateur ne reçoit la notification par email que si l’utilisateur a cliqué sur le lien de vérification envoyé à son adresse.
* Tant que l’utilisateur n’a pas validé sa demande via le lien reçu par email, celle-ci reste inaccessible pour l’administrateur. Elle ne devient visible et traitable qu’après cette action de confirmation.
* L’administrateur dispose d’un délai de 10 minutes pour valider ou refuser la demande une fois celle-ci activée. Passé ce délai, la demande est automatiquement classée comme « non aboutie ».

### 2.3 Contraintes Matériel et Logiciel

Pour assurer le bon fonctionnement du système, il a été nécessaire de disposer d’un serveur hébergeant à la fois un serveur web et une base de données.

La mise en place de la page de formulaire ainsi que l’envoi des emails ont nécessité l’utilisation de différents outils et langages :

* Frontend : l’interface utilisateur a été développée en HTML, permettant l’affichage et la saisie des informations nécessaires à la demande d’accès.
* Backend : les traitements côté serveur, les interactions avec la base de données via des requêtes SQL, ainsi que la gestion des différentes fonctionnalités ont été réalisés en PHP.
* Envoi d’emails : pour l’envoi automatique des mails (validation utilisateur, notification administrateur), j’ai utilisé la bibliothèque PHPMailer, installée via le gestionnaire de dépendances Composer sur le serveur.

### 2.4.1 Utilité base de données

Dans le cadre de ma partie du projet, la table principale utilisée dans la base de données est la table User. Elle contient tous les champs essentiels à la gestion des demandes d’accès, tels que :

* Prenom
* Nom
* Motif
* Fonction
* Tel
* Mail
* Mail\_verif
* ...et d'autres champs liés à la gestion des autorisations.

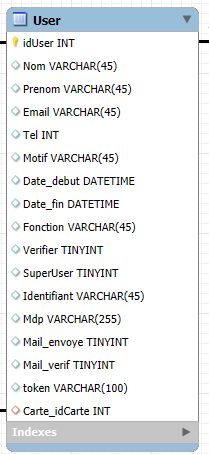
Le champ Mail\_verif joue un rôle clé : il permet de savoir si l’utilisateur a cliqué ou non sur le lien de validation envoyé par email. Tant que ce champ n’est pas confirmé, la demande n’est pas considérée comme active.

Les autres champs renseignent l’identité de l’utilisateur, le motif de la demande, sa fonction, ainsi que la date à laquelle il est autorisé à accéder au local technique.

Enfin, le champ SuperUser détermine si un utilisateur possède les droits d’administrateur. Seuls les utilisateurs identifiés comme SuperUser sont habilités à consulter et à traiter les demandes d’accès.

### 2.4.2 Image de la table concerné sur la base de données.

## 



Remarque : on distingue ici une clé étrangère nommée Carte\_idCarte. Il s’agit de la clé primaire de la table Carte, qui est utilisée pour associer chaque utilisateur autorisé à une carte NFC. Cette relation permet d’identifier quelle carte est attribuée à quel utilisateur..

On observe également une liaison avec la table Connect\_log\_admin, via une autre clé étrangère. Cette table est utilisée pour retracer l’historique des connexions des administrateurs. Ainsi, chaque administrateur est identifiable dans les logs de connexion grâce à son identifiant.

### 2.5.1 Page d’accueil du site WEB

Pour permettre l’accès au site depuis l’extérieur du lycée, un sous-domaine a été ouvert :  
ca25.charles-poncet.net

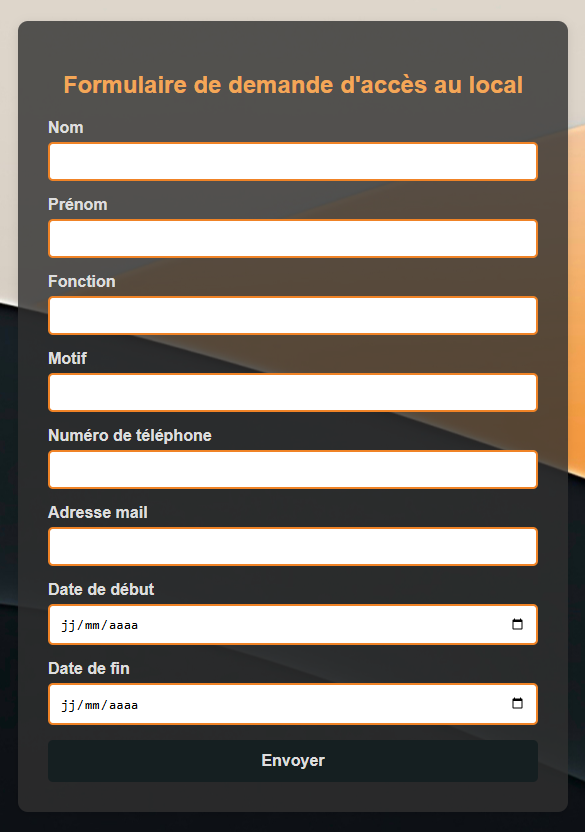
La page d’accueil est accessible via le port 8083.



Voici à quoi ressemble la page d’accueil. On peut également observer l’URL utilisée pour y accéder.

Le bouton qui nous intéresse ici est « Demander l’accès ». En cliquant dessus, on accède au formulaire.

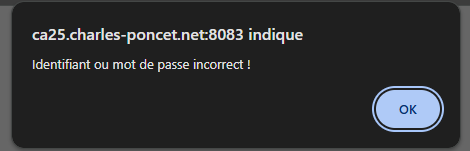
### 2.5.2 Formulaire



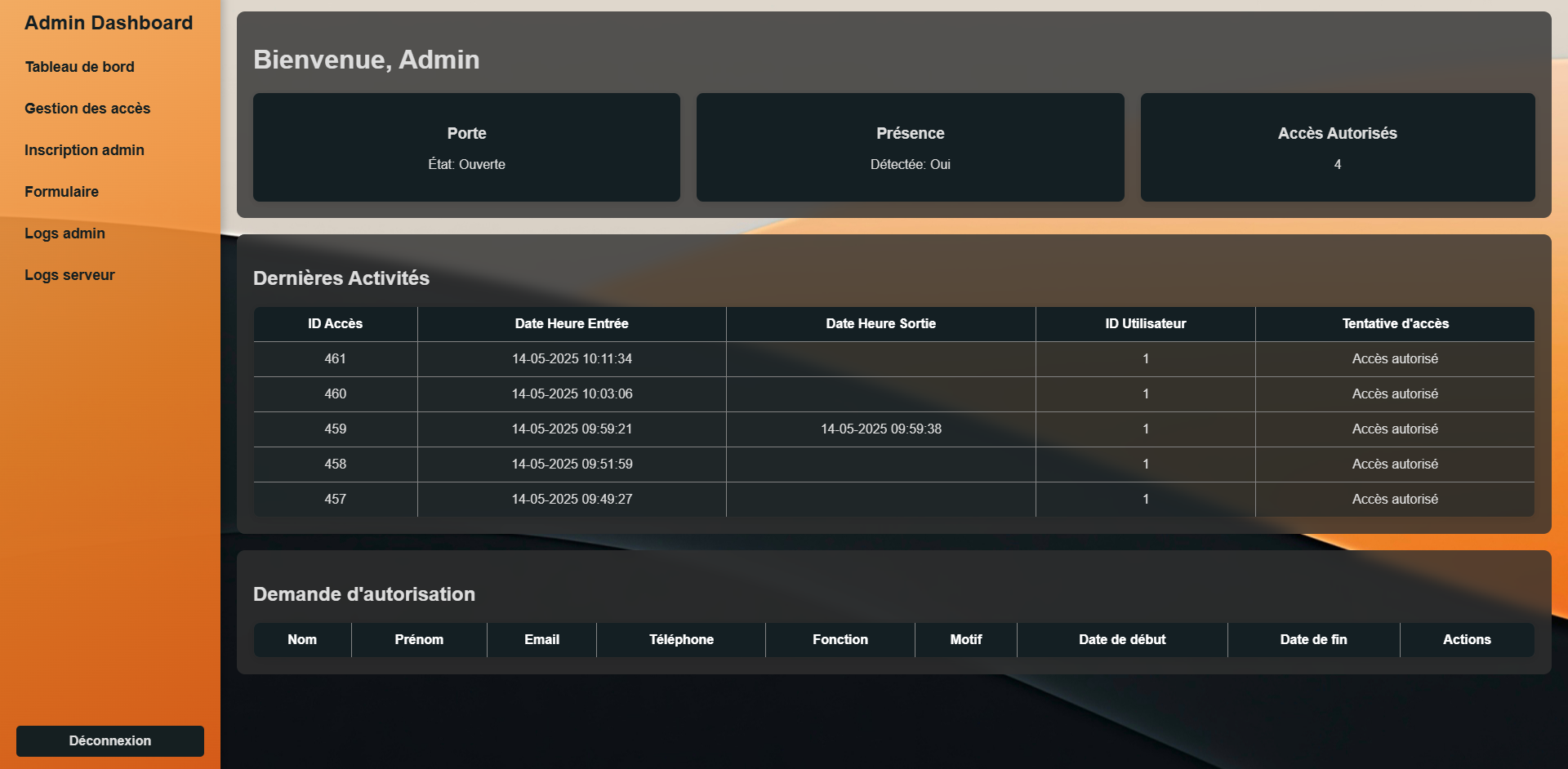
Voici la page affichée lorsque l’on clique sur « Demander l’accès » :

### 2.5.3 Dashboard

En cliquant sur l’autre bouton, « Connexion Admin », une page d’authentification s’affiche. L’administrateur doit y saisir ses identifiants. En cas d’erreur, l’accès au dashboard est refusé.



Voici à quoi ressemble le dashboard une fois connecté :



On y retrouve plusieurs informations essentielles :

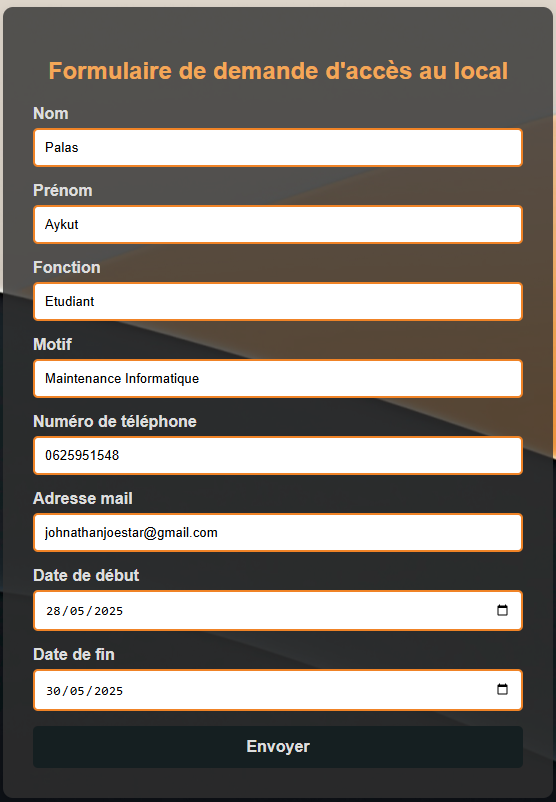
* Les dernières activités du local ;
* Le nombre de personnes actuellement autorisées à accéder ;
* La présence (ou non) d’un utilisateur dans le local ;
* L’état de la porte (ouverte ou fermée).

Sur la partie gauche du dashboard, plusieurs boutons sont disponibles. Les deux qui nous intéressent ici sont :

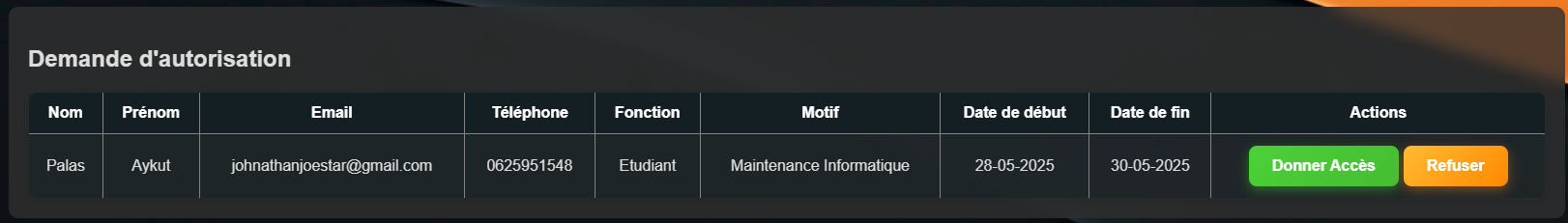
* Formulaire : permet de revenir à la page du formulaire d’accès ;
* Inscription admin : une section détaillée dans une autre partie du rapport.

Tout en bas du dashboard se trouve une section intitulée Demande d’autorisation. C’est ici que s’affichent les nouvelles demandes d’accès. Ces demandes peuvent également être consultées via le bouton Gestion des accès.

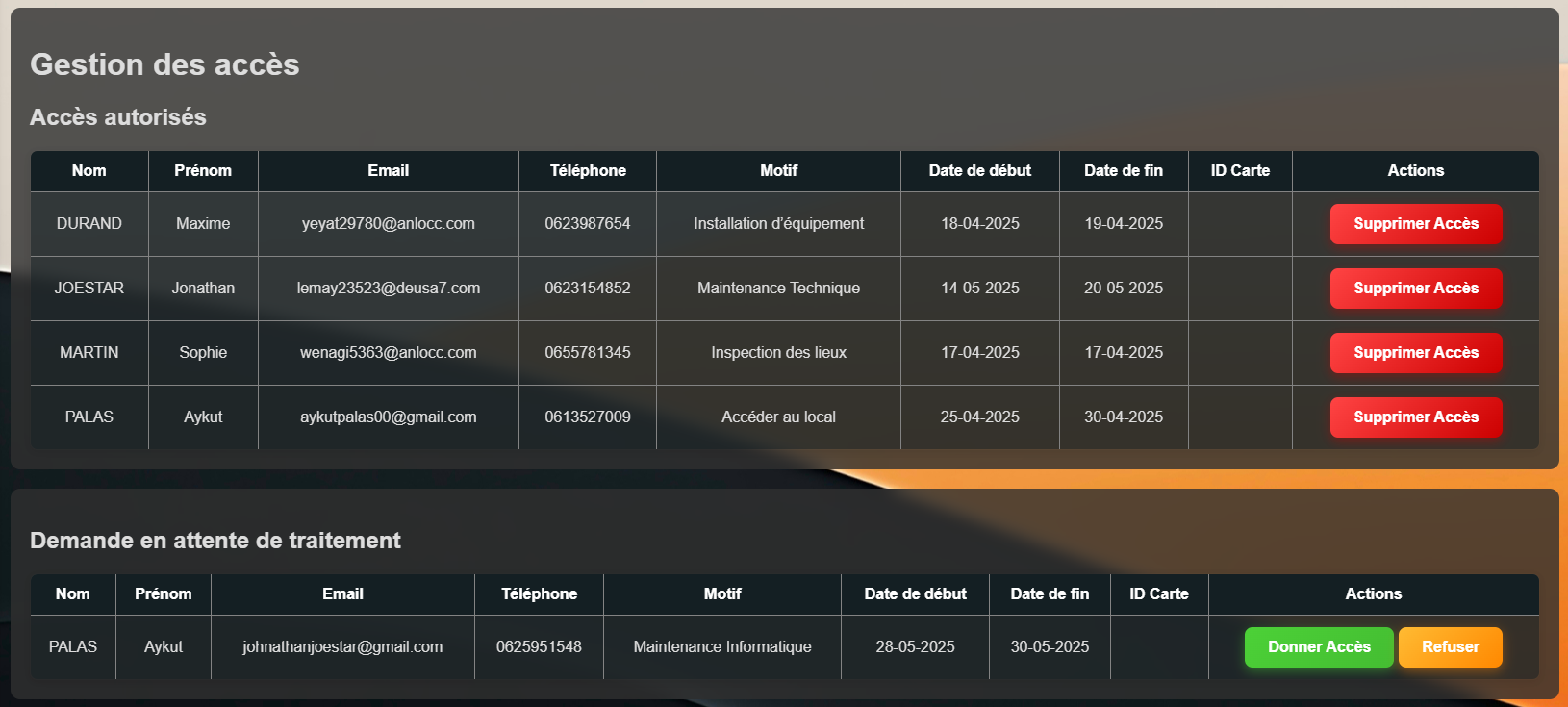
Voici un exemple de demande que j’ai créée pour illustration :



Côté site web, voici l’affichage correspondant sur le dashboard :



Et voici l’affichage de cette même demande sur la page Gestion des accès :



Cette section nous a permis de faire une présentation globale de ma partie.   
Nous allons maintenant entrer plus en détail sur la manière dont j’ai procédé pour développer et faire fonctionner l’ensemble du système. Nous verrons également la configuration de l’envoi des mails et leur affichage.

# **3.Programmation Langage**

### 3.1. HTML

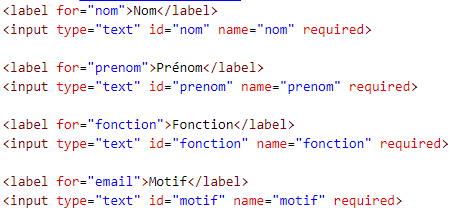
Le langage que j’ai utilisé pour réaliser la page de formulaire est le HTML. Il s’agit d’un langage fondamental pour créer des zones de texte, des tableaux, des boutons, et structurer le contenu d’une page web. Le HTML permet aussi d’interagir avec d’autres langages comme PHP, indispensable pour exécuter des scripts, gérer les bases de données, ou envoyer des e-mails.

Voici un aperçu des principales balises HTML que j’ai utilisées :

3.1.1 Zone de texte

Les zones de texte sont créées à l’aide de la balise <input type="text">. Elles permettent à l’utilisateur de saisir des informations simples comme le nom, le prénom ou la fonction.

Exemples utilisés :



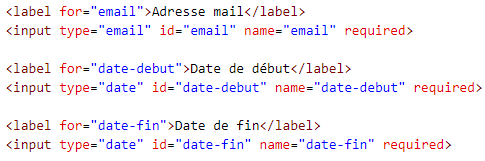
Chaque champ est associé à une étiquette <label> qui en précise la signification. L’attribut required rend le champ obligatoire. Si ce champ est laissé vide, un message d’erreur s’affiche, obligeant l’utilisateur à le remplir.

3.1.2 Les champs spécialisés

Le formulaire comporte également des champs adaptés à des types d’informations spécifiques :

* <input type="email"> : pour saisir une adresse mail (avec vérification automatique du format).
* <input type="tel"> : pour le numéro de téléphone.
* <input type="date"> : pour choisir une date via un calendrier interactif.

Exemples utilisés :



Ces champs sont également accompagnés d’un <label> pour une meilleure lisibilité.

3.1.3 Le bouton de soumission

Le formulaire se termine par un bouton de soumission, défini avec <button type="submit">.

Exemple :

<button type="submit" class="submit-btn">Envoyer</button>

Ce bouton permet d’envoyer les données vers le script PHP spécifié dans l’attribut action de la balise <form>.

3.1.4 La balise <form>

La balise <form> encadre tous les champs du formulaire et précise les paramètres d’envoi.

Exemple :

<form action="../php/formulaire.php" method="post">

* action indique le chemin du script PHP qui recevra les données.
* method="post" permet un envoi sécurisé des données, sans affichage dans l’URL.

3.1.5 Les balises de structure et de style

* <div class="form-container"> : contient tout le formulaire et permet l’application d’un style via CSS.
* <h2> : titre du formulaire.
* <label> : nomme les champs, améliore la lisibilité et l’accessibilité.

3.1.6 Conclusion

Grâce à l’utilisation combinée des balises HTML telles que <form>, <input>, <label>, <button> et <div>, le formulaire est clair, fonctionnel, et prêt à interagir avec un script PHP. Ce sont les bases essentielles d’un formulaire web efficace.

### 3.2 CSS

Je ne me suis pas personnellement chargé de la partie CSS, mais je vais tout de même en expliquer brièvement l’utilité.

Le CSS (*Cascading Style Sheets* ou *feuilles de style en cascade*) est un langage qui permet de mettre en forme une page HTML. Il ne modifie pas le contenu, mais agit uniquement sur l’apparence visuelle des éléments affichés à l’écran.

Alors que le HTML sert à structurer l’information (titres, paragraphes, formulaires, etc.), le CSS est utilisé pour rendre l’affichage plus clair, agréable et professionnel.

#### Principales fonctions du CSS :

* Définir les couleurs, polices et tailles de texte.
* Ajouter des marges, bordures, fonds, ou effets visuels.
* Organiser les éléments sur la page (alignement, colonnes, positionnement…).
* Adapter le contenu à différents types d’écrans (ordinateur, smartphone…).
* Ajouter des animations ou effets au survol.

Dans mon formulaire, le CSS est relié à la page HTML par cette ligne placée dans la balise <head> :

<link rel="stylesheet" href="../css/formulaire.css">

Cela permet de séparer la structure (HTML) de la mise en forme (CSS), ce qui rend le code plus lisible, facile à modifier et à maintenir.

En résumé, le CSS est indispensable pour donner une apparence professionnelle à un site web et améliorer l’expérience utilisateur.

### 3.3 PHP

3.3.1 Explication du PHP

Nous arrivons donc à la partie la plus importante, avec le HTML et le langage PHP. Expliquons ce qu’est le langage PHP et à quoi il sert :

Le PHP (Hypertext Preprocessor) est un langage de programmation web côté serveur. Contrairement au HTML et au CSS, qui sont exécutés dans le navigateur de l’utilisateur, le PHP fonctionne sur le serveur. Cela signifie que le code PHP est exécuté avant que la page ne soit envoyée à l’utilisateur, ce qui permet de rendre un site dynamique et interactif.

### À quoi sert le PHP dans un site web ?

Le PHP est particulièrement utile pour :

* Traiter les données d’un formulaire : lorsque l’utilisateur remplit un formulaire, les informations peuvent être récupérées et traitées par un script PHP.
* Enregistrer ou lire des données dans une base de données : par exemple, pour stocker des inscriptions, gérer des comptes utilisateurs ou afficher des informations dynamiques.
* Créer des pages personnalisées : selon l’utilisateur ou les données reçues, le PHP peut générer des pages différentes.
* Envoyer des e-mails : après l’envoi d’un formulaire, PHP peut envoyer automatiquement un e-mail de confirmation.
* Sécuriser un site : PHP permet de gérer les connexions, les mots de passe et les autorisations d’accès.

### Exemple d’utilisation dans un formulaire

Dans mon code HTML, j’ai utilisé l’attribut action dans la balise <form> pour indiquer que les données seront envoyées vers un fichier PHP :

<form action="../php/formulaire.php" method="post">

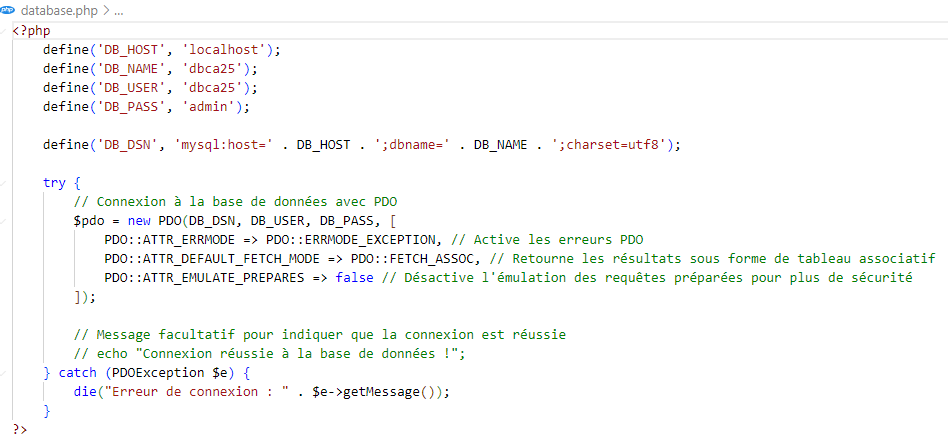
Cela signifie que toutes les données remplies par l’utilisateur dans le formulaire seront envoyées au fichier formulaire.php, qui contiendra du code PHP pour les traiter, les enregistrer ou envoyer des notifications par email.

En résumé, le PHP est un langage essentiel pour tout site web nécessitant le traitement de données, la gestion d’utilisateurs ou l’interaction avec une base de données. Il complète le HTML et le CSS en apportant la logique et les fonctionnalités nécessaires au bon fonctionnement d’un site dynamique.

3.3.2 Utilisation du PHP

Tout d’abord, le PHP sert à se connecter à la base de données, ce qui est primordial pour assurer le bon fonctionnement du système. Voici un exemple de code :

database.php :



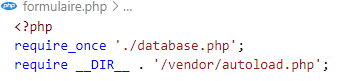
Nous pouvons voir, après les définitions « define », les différents paramètres qui servent à configurer la connexion à la base de données : le nom de la base, l’adresse IP sur laquelle elle est hébergée (ici, il s’agit de localhost), ainsi que l’identifiant et le mot de passe de connexion.

La variable « $pdo » reçoit la valeur retournée par la classe PDO. Cette méthode permet de sécuriser les connexions à la base de données, notamment contre les injections SQL.

Le code de connexion à la base de données est contenu dans le fichier database.php. Ainsi, pour les autres fichiers PHP, il suffit d’inclure « database.php » comme bibliothèque afin d’établir une connexion à la base.

Voici un extrait du fichier « formulaire.php » où l’on inclut « database.php » en tant que bibliothèque :

formulaire.php :



Il y a également une inclusion de « vendor/autoload.php », cette bibliothèque est utile pour l’envoi des mails mais nous en parlerons plus tard.

Le PHP est également utilisé pour récupérer les différents champs du formulaire, tels que « Nom », « Prénom », « Mail », via la méthode POST, et pour attribuer ces valeurs à des variables définies.

formulaire.php :

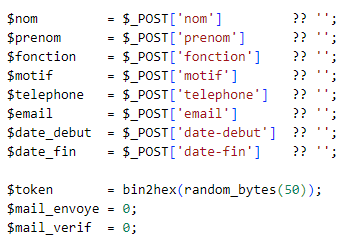


Après la balise <form action>, on trouve l’attribut method="post" ainsi que le chemin vers /php/formulaire.php. Cela indique que tous les champs du formulaire seront envoyés à formulaire.php lorsque l’utilisateur cliquera sur le bouton « submit ».

L’attribut « name » permet d’identifier les différents champs, comme prenom, fonction ou encore motif.

Voici un extrait du fichier formulaire.php où l’on récupère les différents champs envoyés via la méthode POST et où l’on attribue ces valeurs à des variables.

formulaire.php :



Nous pouvons observer les différentes variables, reconnaissables grâce au symbole $ précédant le nom des champs. Ces variables prennent la valeur de :   
$\_POST[‘nom du champ’] ?? ‘’ ;

Le tableau superglobal $\_POST indique que la valeur est obtenue via la méthode POST. Entre crochets [ ] se trouve le nom identifiant du champ concerné, par exemple « telephone ».   
Enfin, les doubles points d’interrogation ?? suivis de ' ' signifient que si le champ concerné n’a aucune valeur, la variable prendra par défaut une chaîne vide (' ').

Ensuite, les variables « $mail\_envoye » et « $mail\_verif » concernent l’envoi des mails.   
C’est un autre exemple d’ajout de variable et d’affectation de valeur.

La variable « $token » est très importante et sera utilisée dans la partie consacrée à l’envoi des mails. Nous verrons plus tard à quoi sert précisément ce token. Pour l’instant, on constate que le PHP permet sa création.

### **In**teraction du code PHP avec la base de données

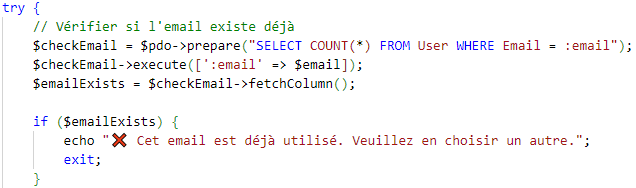
Le langage PHP permet une communication directe avec une base de données, ce qui est essentiel pour gérer les informations d’un site web. Dans mon projet, le PHP est utilisé pour enregistrer, lire et mettre à jour des données dans une base de données MySQL. Voici comment cela fonctionne et à quoi cela sert, en s’appuyant sur mon code.

Après la connexion à la base de données, déjà expliquée précédemment, nous pouvons interagir avec elle comme nous le souhaitons. Par exemple, dans le cadre d’un formulaire, nous allons prendre comme cas l’insertion d’un utilisateur dans la table User de la base de données.

Une fois connecté, le PHP récupère les données du formulaire via la méthode $\_POST, puis prépare ces données pour les insérer dans la base. Avant l’enregistrement, le code vérifie si l’adresse email existe déjà afin d’éviter les doublons. Cette vérification est effectuée à l’aide d’une requête SQL que le PHP exécute directement.

Voici un extrait de ce code :

formulaire.php :



Si l’email n’est pas encore utilisé, PHP insère les données dans la base grâce à une requête INSERT. Cela permet de conserver toutes les demandes d’accès dans la table User.   
À ce stade, chaque formulaire correspond à une ligne dans la base de données, pouvant ensuite être consultée et traitée. Dans le cas contraire, un message s’affiche : « Cet email est déjà utilisé. Veuillez en choisir un autre ».

Voici l’extrait concerné :

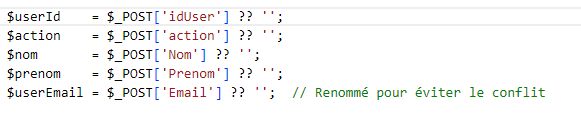
formulaire.php



Mais le rôle du PHP ne s’arrête pas là. Une fois la demande enregistrée, un administrateur peut la valider ou la refuser. Lorsqu’une décision est prise, un autre script PHP (dans mon cas, bouton.php) se charge de mettre à jour les informations de l’utilisateur. Par exemple, si l’accès est accordé, le champ Verifier passe à 1, ce qui signifie que la demande a été approuvée.   
Un identifiant unique (uid) est également ajouté pour assurer le suivi de cette autorisation.

Voici l’extrait du code bouton.php :

bouton.php



Nous pouvons tout d’abord voir que toutes les valeurs sont récupérées via la méthode POST avant d’être attribuées aux variables concernées. L’envoi se fait via un script présent dans le fichier GestionAcces, qui réagit en fonction de la décision prise par l’administrateur.

Voici un extrait du code de GestionAcces :

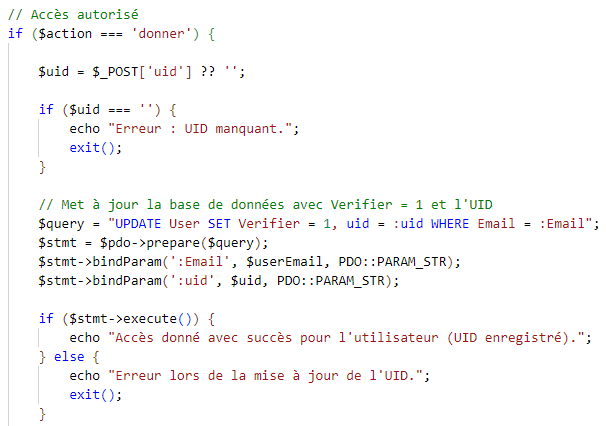
GestionAcces.php



Depuis l’extrait du code pour le bouton « Accepter », nous pouvons voir que l’attribut method="post" est présent. On remarque également une interaction avec le HTML.   
Le HTML est utilisé pour créer le bouton, attribuer les valeurs, et envoyer les données au script GestionAcces, comme expliqué précédemment.

Revenons au code « bouton.php » et regardons l’extrait du code Valider :

bouton.php :



Si le bouton Donner Accès est cliqué, le code récupère l’adresse mail de l’utilisateur, puis met à jour la colonne concernée en modifiant la valeur du champ Verifier de NULL à 1.

Le champ Verifier passe à 1 car c’est cette valeur qui détermine si l’utilisateur est autorisé à accéder au local. Le fichier accesAutorise.php affiche les personnes autorisées en fonction de cette valeur : si le champ Verifier vaut 1, alors l’utilisateur est affiché.

La page accesAutorise.php liste donc les utilisateurs autorisés à l’accès, mais aucune action ne peut y être effectuée. Pour modifier les droits, il faut passer par la page Gestion des Accès.

Voici un extrait du code présent dans accesAutorise.php :

accesAutorise.php :



Avec la clause WHERE en SQL, le code affiche les utilisateurs dont la valeur du champ Verifier est égale à 1, c’est-à-dire ceux qui ont accès.

Cette page sert également à afficher les administrateurs actuels du système. Voici un extrait du code qui sélectionne et affiche uniquement les utilisateurs ayant la fonction « Admin » :

accesAutorise.php :



Pour terminer, le PHP sert aussi à sécuriser le système. La sécurisation faisait partie de mes tâches, et pour cela, j’ai implémenté un code en PHP qui impose une authentification pour accéder au dashboard du site.

Par exemple, une personne mal intentionnée pourrait tenter d’accéder directement au dashboard ou à n’importe quelle page protégée en entrant l’URL   
« ca25.charles-poncet.net:8083/dashboard.php ». Sans protection, le hacker pourrait ainsi contourner le système de connexion.

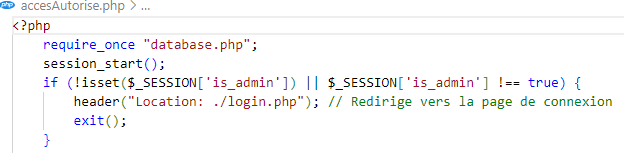
Le code PHP permet donc de vérifier l’authenticité de l’utilisateur en rendant la connexion obligatoire, grâce à la gestion des sessions.

Une session en PHP est un mécanisme qui conserve des informations sur un utilisateur pendant sa navigation entre différentes pages. Contrairement aux variables classiques, les données stockées en session ne disparaissent pas lors du changement de page. Par exemple, lorsqu’un utilisateur se connecte, son identifiant ou son nom peuvent être enregistrés dans la session.

Pour utiliser une session, on commence par appeler la fonction session\_start(), puis on peut stocker des données dans la superglobale $\_SESSION. Cela permet de créer une navigation fluide et personnalisée pour chaque utilisateur.

Voici un extrait de code contenant session\_start() au début :

accesAutorise.php :

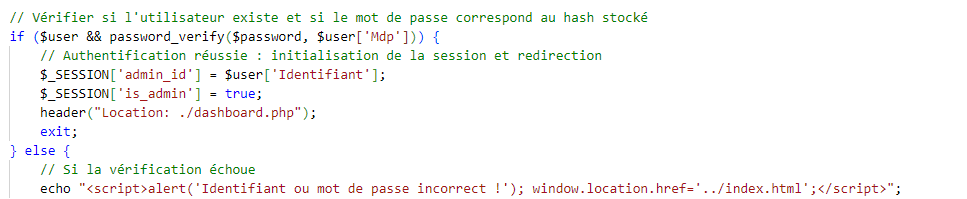


Nous observons que le fichier accesAutorise.php contient la fonction session\_start(). Une condition suit cette fonction : la variable $\_SESSION, qui fonctionne avec session\_start(), doit contenir la valeur « is\_admin ». Sinon, l’utilisateur est redirigé vers la page de connexion.   
C’est cette variable qui assure la sécurisation du site.

La valeur « is\_admin » est attribuée à cette variable dans la page de connexion login.php. Si la connexion réussit, alors $\_SESSION['is\_admin'] reçoit cette valeur, ce qui autorise l’accès à l’administrateur.

Voici un extrait du code présent dans login.php, où la valeur « is\_admin » est attribuée en cas de connexion réussie, suivi de la redirection vers la page du dashboard :

login.php :



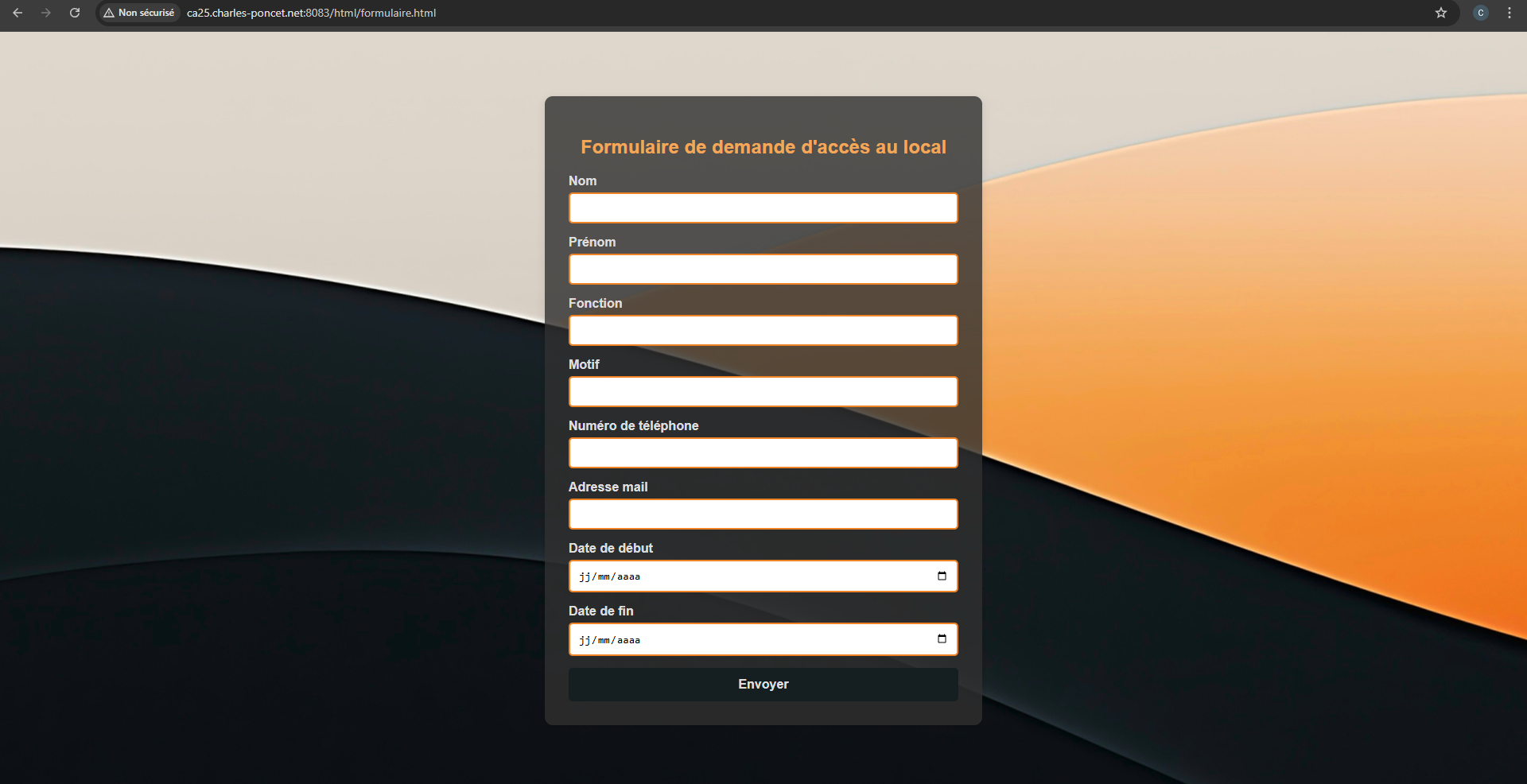
En cas de connexion échouée, l’utilisateur reçoit un message de refus et est redirigé à nouveau vers la page de connexion.

Le code PHP joue également un rôle essentiel dans l’envoi des mails, mais nous aborderons ce sujet dans une autre partie.

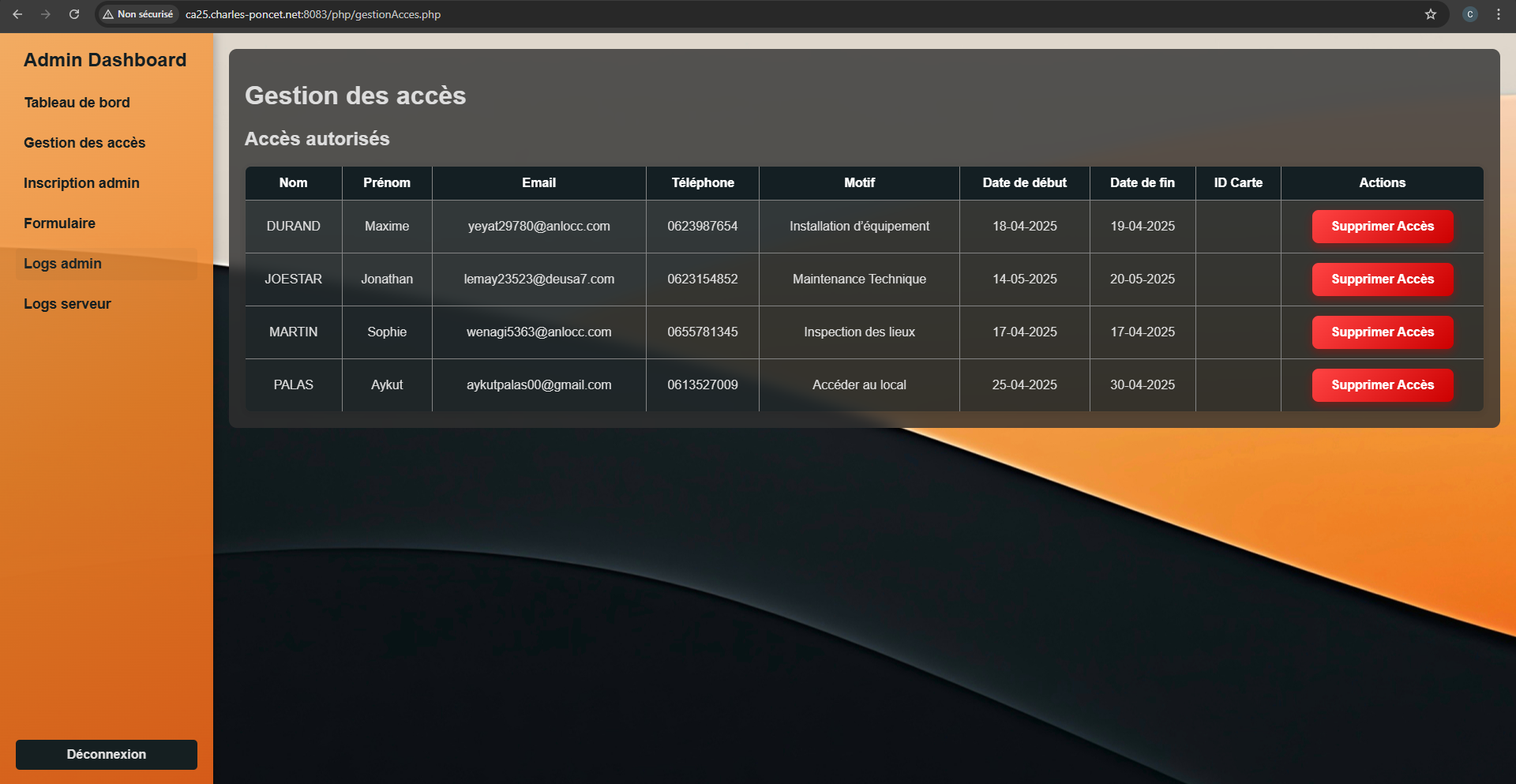
3.3.3 Pages PHP

Avec les différents extraits présentés ci-dessus, ainsi que les fonctions des différents codes écrits en PHP, voici une illustration de chaque page correspondante associée aux codes mentionnés précédemment :

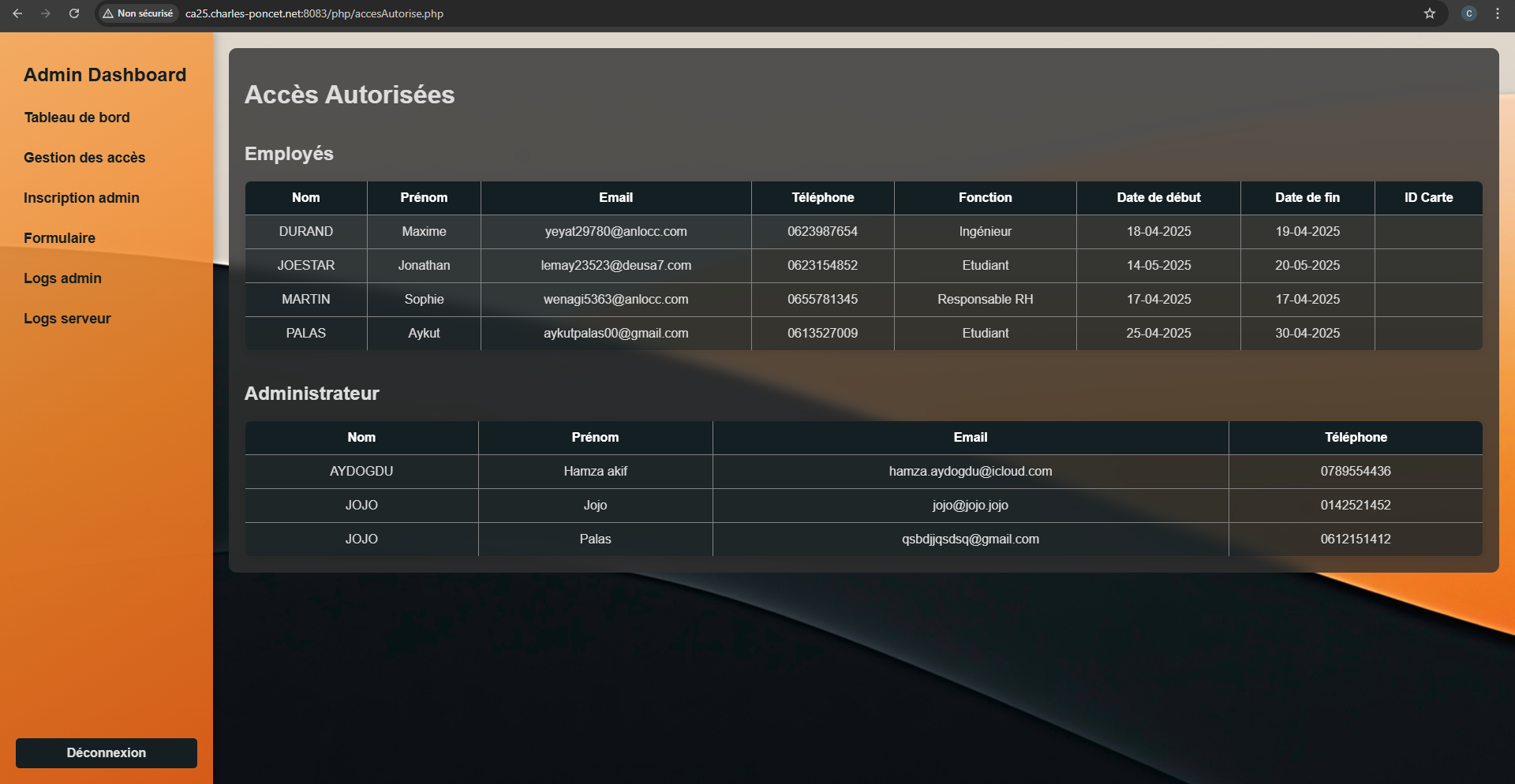
*formulaire.php :*



*gestionAcces.php :*



*accesAutorise.php :*



# **4. Envoi des mails**

### 4.1. Composer

La première étape pour l’envoi des mails consiste à installer le logiciel Composer.  
Composer est un outil indispensable en PHP pour gérer automatiquement les dépendances (bibliothèques) d’un projet. Voici comment l’installer proprement sur un système Linux.

#### Installation de Composer

L’installation se fait directement sur le serveur via le terminal, à l’aide de la commande suivante : php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

Une fois cette commande exécutée, il faut lancer la suivante pour installer Composer dans le bon répertoire :  
sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer  
  
Si jamais nous souhaitons supprimer Composer, la commande suivante permet de supprimer le fichier d’installation : php -r "unlink('composer-setup.php');"

Une fois Composer installé sur le serveur, la moitié du travail est faite.  
 Il suffit maintenant de télécharger la bibliothèque PHP Mailer, qui est essentielle pour envoyer des mails depuis un script PHP.

### 4.2 PHP Mailer

PHPMailer est une bibliothèque très utilisée en PHP pour envoyer des mails de manière plus sécurisée et plus complète que la fonction native mail().  
 Elle permet notamment :

* l’envoi via SMTP (plus fiable que mail()),
* l’ajout de pièces jointes,
* le formatage des messages en HTML..

4.2.1 Installation

Pour installer PHPMailer, de la même façon que Composer, il suffit de taper la ligne suivante dans le terminal :  
composer require phpmailer/phpmailer

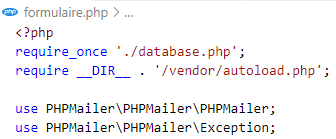
Cette commande télécharge et installe automatiquement PHPMailer dans le dossier vendor/.  
Dans mon cas, j’ai déplacé le dossier vendor/ dans mon répertoire .php (là où se trouvent tous mes fichiers PHP), afin de faciliter son utilisation et ses inclusions dans mes scripts.

4.2.2 Utilisation

Maintenant que Composer et PHP Mailer installé, nous pouvons désormais envoyer des mails. Regardons comment fonctionne PHP Mailer et comment l’utiliser.

Il faut tout d’abord inclure la bibliothèque de PHP Mailer, dans le code source de chaque fichier PHP, où il y a un envoi de mail. Nous prendrons des extraits du code pour illustrer.

Formulaire.php :



Nous incluons avec « require \_\_DIR\_\_ » vendor/autoload.php, et en-dessous, nous utilisons PHPMailer/PHPMailer et PHPMailer/Exception pour le bon fonctionnement du programme d’envoi des mails.

La seconde étape est la configuration du serveur SMTP, voici un extrait de mon fichier PHP pour voir sa configuation.

Formulaire.php :



Nous incluons une variable « $mail » qui prend la valeur de l’objet PHPMailer.   
Ainsi, nous configurons la variable « $mail ». Nous voyons que le « Host » prend la valeur de ‘smtp.gmail.com’, ce qui indique le serveur SMTP utilisé.   
La partie qui nous intéresse particulièrement est « Username » et « Password ».  
Le « Username » correspond à l’adresse du compte mail qui va se charger d’envoyer les différents mails.   
Dans notre cas, nous avons décidé de créer une adresse mail spécialement pour cette fonctionnalité. Cette adresse est : « carteacces99@gmail.com », que nous avons donc renseignée dans le champ « Username ».  
 Le « Password » ne correspond pas au mot de passe classique du compte Gmail, mais à un mot de passe d’application. Pour en configurer un, il faut se rendre dans les paramètres de sécurité du compte Gmail, activer la double authentification, puis générer un mot de passe d’application dans l’onglet Sécurité. C’est ce mot de passe d’application que l’on renseigne dans le champ « Password ».

Une fois la configuration du serveur SMTP terminée, nous pouvons passer à la rédaction du contenu du mail ainsi qu’à la définition du destinataire.  
Voici un extrait du code :

formulaire.php :



Premièrement, avec « setFrom », nous indiquons le compte mail depuis lequel nous voulons envoyer le mail, suivi du nom que l’on veut utiliser.

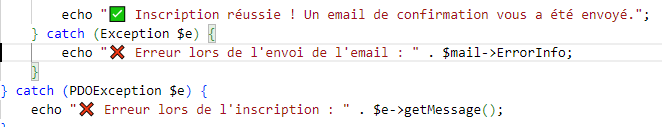
Ensuite, avec la méthode « addAddress », nous renseignons l’adresse mail de l’utilisateur, en utilisant les variables « $email », « $nom » et « $prenom » récupérées plus tôt grâce à la méthode POST.  
Le champ « Subject » correspond à l’objet du mail, tandis que « Body » définit le contenu du message.   
Nous y ajoutons donc ce que nous voulons transmettre à l’utilisateur : dans notre cas, un message indiquant que le système a bien reçu la demande via le formulaire, suivi d’un lien de confirmation.  
Le champ « AltBody » sert à fournir une version en texte brut du message, utile dans le cas où le client de messagerie du destinataire ne supporte pas le format HTML.

Une fois tout cela configuré, nous appelons la méthode « send() » à l’aide de l’objet « $mail » pour envoyer le message.

En cas d’erreur lors de l’envoi du mail, une erreur s’affichera à l’écran grâce à la structure « try / catch », qui fonctionne ainsi :

* try : Je teste un morceau de code qui pourrait provoquer une erreur.
* catch : Si une erreur se produit, je la capture pour afficher un message ou exécuter une autre action, sans que le site ne plante.

formulaire.php :

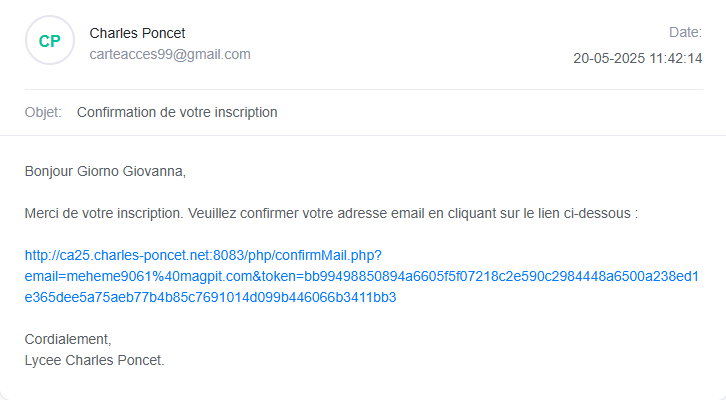
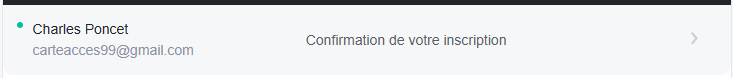


Le mail est finalement bien envoyé. Nous allons illustrer un exemple avec un formulaire, accompagné de captures d’écran des deux mails reçus, afin de montrer que l’envoi s’est bien effectué.

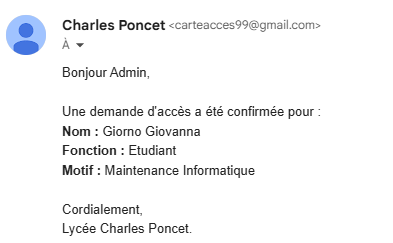
Page de formulaire :



Page de mail de l’utilisateur :



Page de mail de l’admin :



Nous pouvons donc conclure la partie sur l’envoi des mails.

# **5. Bonus**

### 5.1 Inscription Admin

Pour se connecter à la page du dashboard, il faut renseigner un nom d’utilisateur ainsi qu’un mot de passe. Afin de sécuriser la connexion administrateur, nous avons décidé de créer une page « Inscription Admin », qui se présente sous la forme d’un formulaire. L’administrateur peut ainsi remplir tous les champs nécessaires. Voici à quoi ressemble la page :



L’utilité de cette page est d’ajouter des administrateurs « SuperUser » dans la base de données. La particularité de cette page est de chiffrer le mot de passe de l’administrateur grâce à un système de « hash ». En effet, la connexion au dashboard ne serait pas sécurisée si les champs utilisateur et mot de passe étaient simplement comparés à la base de données, rendant le système facilement piratable. C’est pourquoi nous avons décidé de créer cette page « Inscription Admin ».  
 Examinons le code et observons ce qu’il fait.

inscriptionAdmin.php :

// Hashement du mdp en bcrypt

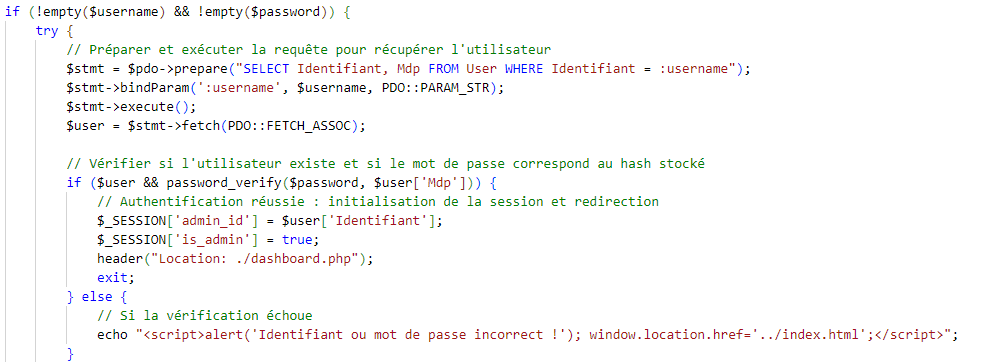
        $hashedMdp = password\_hash($mdp, PASSWORD\_DEFAULT);

Nous pouvons voir qu’une variable « $hashedMdp » prend pour valeur le résultat de la méthode « password\_hash » appliquée à « $mdp », qui est la variable contenant le mot de passe saisi par l’administrateur et transmis via la méthode POST.  
 Imaginons que le mot de passe choisi soit « admin », voyons à quoi ressemble ce mot de passe une fois enregistré dans la base de données :



Nous constatons que le mot de passe « admin » apparaît sous la forme d’une suite de caractères illisibles. Le mot de passe est donc chiffré, ce qui rend son piratage beaucoup plus difficile.  
Voyons maintenant comment cela se passe dans le fichier « login.php », qui est la page de connexion admin.

login.php :



Une fois que l’admin aura rempli les champs « Utilisateur » et « Mot de passe » puis cliqué sur « Connexion », une comparaison sera effectuée avec les données stockées dans la base pour vérifier l’authenticité des informations fournies. Cette comparaison utilise la méthode « password\_verify », spécialement conçue pour vérifier des mots de passe chiffrés, ce qui renforce la sécurité et rend le piratage beaucoup plus difficile.

### 5.2 Déconnexion

Afin de renforcer la sécurité, nous avons décidé d’ajouter un bouton de déconnexion sur le site web, permettant de fermer la session en cours. De plus, nous avons ajouté dans chaque page une condition vérifiant la variable « is\_admin » pour contrôler l’accès. L’accès à ces pages est donc réservé uniquement aux administrateurs ayant réussi la connexion, ce qui définit la variable « is\_admin » à « true ». Lorsque l’administrateur souhaite se déconnecter, il lui suffit de cliquer sur ce bouton. Personnellement, je n’ai pas travaillé directement sur ce bouton, mais j’ai étudié son fonctionnement et son code. Hamza Aydogdu en expliquera les détails plus précisément dans son rapport.

# **6. CONCLUSION**

Pour conclure, ce projet m’a beaucoup intéressé et m’a énormément servi. Je me suis beaucoup investi dans sa réalisation. J’ai pu mettre en œuvre les connaissances acquises durant mes deux années de BTS, et j’ai également appris davantage sur le langage PHP et son utilisation avec SQL. J’ai approfondi mes compétences dans le domaine des bases de données.

Ce projet m’a aussi ouvert les yeux sur la gestion d’un projet, les problèmes rencontrés ainsi que les différentes contraintes. Le travail d’équipe est un aspect essentiel, et la communication en est la clé. J’ai, par la même occasion, compris comment mener un projet du début à la fin, des fiches recettes à la finalisation.

J’ai également travaillé, en dehors de ma partie, sur d’autres aspects du projet, ce qui m’a permis d’élargir mes connaissances, notamment sur l’utilisation du Raspberry Pi, le câblage, ainsi que le fonctionnement des différents matériels.

Cependant, plusieurs améliorations auraient pu être apportées si j’avais disposé de plus de temps. L’amélioration de la sécurité est primordiale dans ma partie, comme par exemple la page d’inscription admin qui oblige la création d’un identifiant admin par défaut en cas d’absence d’administrateur. Cette situation rend la connexion vulnérable, car une personne malveillante qui réussit à se connecter au dashboard peut alors inscrire autant d’admins qu’elle le souhaite.

Pour remédier à cela, il serait nécessaire d’ajouter une page de connexion supplémentaire avec un système de validation d’authenticité par mail, afin de renforcer la sécurité d’accès à cette page en cas de tentative de piratage. Pour sécuriser davantage l’accès au site web, il faudrait aussi intégrer une protection anti-force brute sur la page de connexion, ainsi qu’un système de blacklist permettant de bannir une adresse MAC et une adresse IP.