



13/03/2025

Rapport de projet

COSME VINOUElias

Projet ACCÈS LYCÉE

Badge RFID et Interface de gestion





Table des matières

Introduction Générale	3
Présentation	3
Matériels Utilisés :	4
Logiciels Utilisés :	4
But du projet	5
Diagramme de Gantt	6
Répartition des tâches	8
Analyse UML	9
Langage UML	9
DIA	9
Diagrammes	10
Diagramme de déploiement	10
Diagramme de cas d'utilisation	11
Diagramme d'activité	14
Diagramme de séquences	16
Base De Données	19
Conceptualisation et modélisation des données	19
Présentation de JMerise	19
Utilisation du diagramme Entité-Association	19
Génération du code SQL via JMerise	20
Analyse des besoins et identification des entités et associations	21
Détail des entités	21
Détail des associations	23
Tables	24
Contrainte : ON DELETE CASCADE	26
Création du compte utilisateur	27
Gestion des horaires d'accès selon les rôles des utilisateurs	28
PhpMyAdmin	29
Utilisation de MariaDB et du moteur InnoDB	30
Le moteur de stockage InnoDB	31
Principe du développement en local avec XAMPP	32
Composants principaux de XAMPP :	32
Fonctionnement en local :	32
Technologie RFID	35
Script de connexion à la base de données	36
Language de Programmation	36



Détails du script Php	36
Script De Gestion Des Accès Selon Les Horaires.....	39
Fonctionnement du code	39
Détails du script Php	40
Affichage sur le lecteur	43
Bilan de ma contribution au projet.....	44
Annexes	45



Introduction Générale

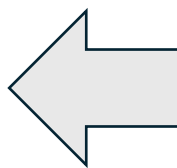
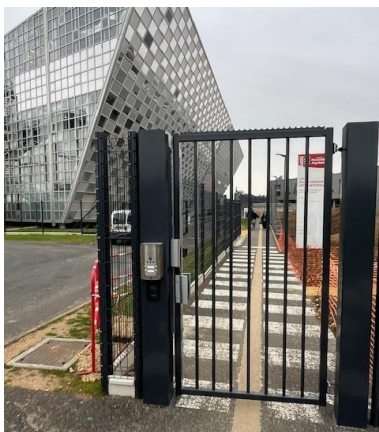


Le BTS CIEL est une formation de deux ans qui te prépare à travailler dans le secteur de l'informatique. Un des aspects cruciaux de cette formation est le Projet E6. C'est un travail pratique où tu mets en œuvre les compétences acquises durant notre formation. C'est également l'occasion de faire valoir notre esprit d'analyse et notre sens de l'organisation.

Présentation

Dans un monde où la sécurité et la gestion des accès sont devenues des enjeux majeurs, il est essentiel de mettre en place des systèmes performants et fiables.

Ce projet vise à reproduire et développer un système de gestion des accès basé sur le lycée du Pilote Innovant International.










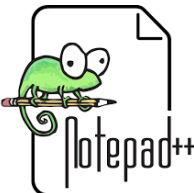





Matériels Utilisés :

Désignation	Caractéristiques
<ul style="list-style-type: none"> • 2 lecteurs de badges et des badges RFID • Volet • Une caméra IP • Un capteur de présence photoélectrique 	<ul style="list-style-type: none"> • De la marque Invéo et Minova • Matériel de récupération • De la marque D-Link DCS-5000L • De la marque OSIRIS

Logiciels Utilisés :

Désignation	Caractéristiques
<p>Base de données :</p>  <p>Pilotage du Barionet :</p>    <p>Création des interfaces :</p>   	<p>Moteur :</p>  <p>Application :</p>  <p>Application :</p>  <p>Application :</p> 



But du projet

L'objectif principal est d'automatiser l'identification des utilisateurs grâce à des badges RFID, tout en permettant une gestion centralisée des accès via un serveur. Ce système devra être capable de :

- Lire et vérifier les informations contenues dans un badge RFID
- Accorder ou refuser l'accès en fonction des droits attribués à l'utilisateur
- Enregistrer les événements d'accès dans une base de données pour assurer un suivi
- Intégrer une interface de gestion pour les administrateurs et techniciens.

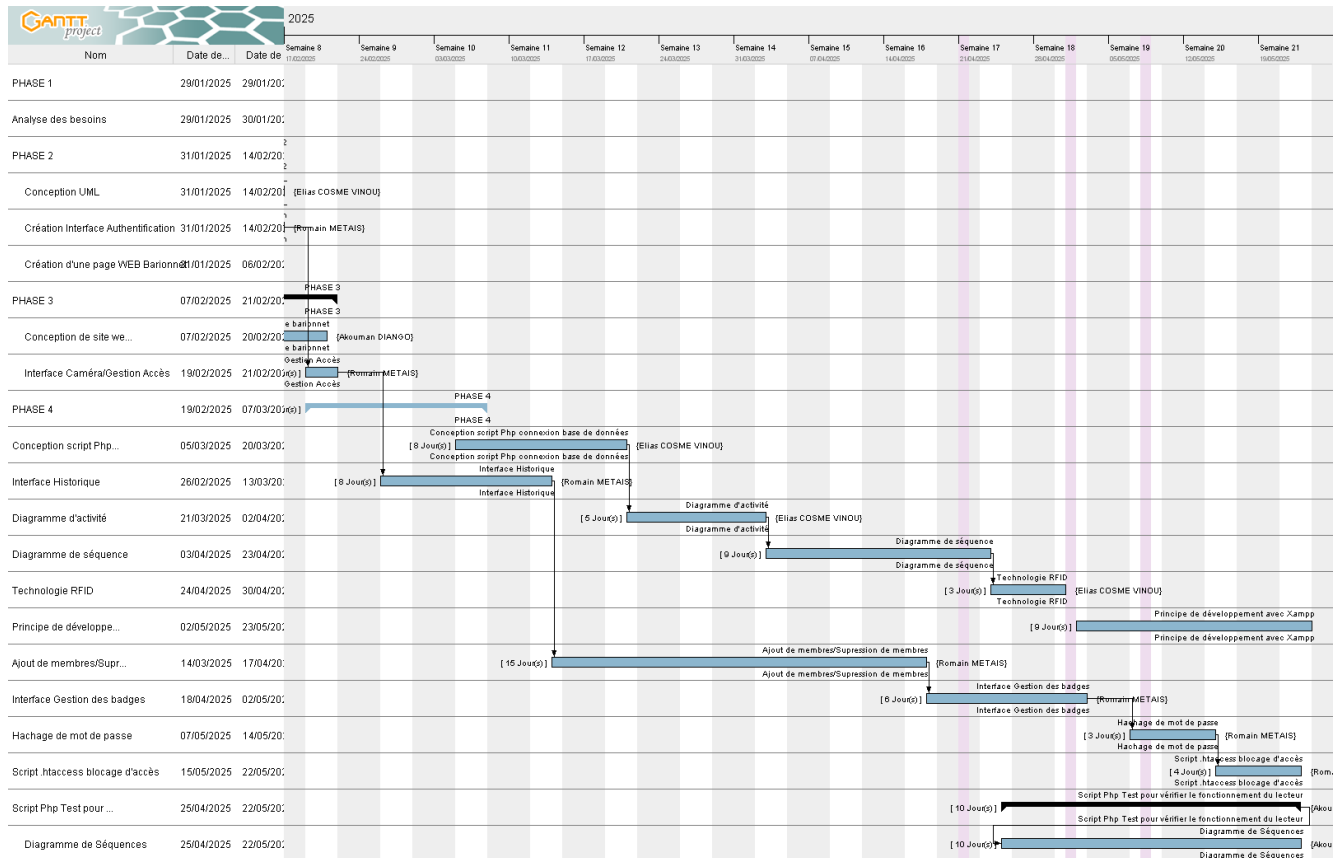
Pour mener à bien ce projet, nous avons adopté une approche structurée en plusieurs étapes :

1. Analyse des besoins et conception du système, en nous basant sur les diagrammes UML pour modéliser l'architecture et les interactions.
2. Mise en place de l'infrastructure matérielle, incluant des lecteurs RFID, un serveur, une base de données et un module Barionnet100 permettant de piloter les actionneurs.
3. Développement du système, avec l'implémentation du logiciel en utilisant des technologies comme PHP et MySQL.
4. Tests et validation, afin d'assurer le bon fonctionnement du dispositif et sa robustesse.

Ce document détaille l'ensemble des aspects du projet, depuis la conception jusqu'à l'intégration finale du système, en passant par les choix techniques et les contraintes rencontrées.



Diagramme de Gantt



Le diagramme de Gantt est un outil qui permet de représenter visuellement l'état l'avancement des différentes tâches qui constituent le projet.

Celui ci-dessus est séparé en 4 phases et relate l'avancement de chaque tâche qu'on a ou est en train d'effectuer sur le projet :

- Analyse des besoins (Phase 1)
- Conception UML (Phase 2)
- Conception Interface Authentification (Phase 2)
- Création de Page Web Sorties Barionet (Phase2)
- Conception Site Web Technicien/Agent Barionet (Phase3)
- Création Interface Gestion d'Accès (Phase3)
- Conception Base de Données (Phase4)
- Ouverture des Portes via les badges (Phase4)
- Conception Script PHP(Phase4)
- Interface Historique (Phase4)
- Diagramme d'activité (Phase4)



- Diagramme de séquences (Phase4)
- Technologie RFID (Phase4)
- Principe de développement (Phase4)
- Ajout de membres/Suppression (Phase4)
- Interface de Gestion des badges (Phase4)
- Hachage de mot de passe (Phase4)
- Script htaccesss blocage d'accès (Phase4)
- Script Php test pour vérifier le fonctionnement du lecteur Minova (Phase4)
- Diagramme de séquence (Phase4)



Répartition des tâches

Etudiant	Tâches
Akouman DIANGO	<ul style="list-style-type: none">• Pilotage du Barionet• Mise en oeuvre du lecteur RFID Minova
Elias COSME VINOUE	<ul style="list-style-type: none">• Conception UML• Mise en place et structuration de la base de données• Création du Script Php de connexion à la base de données• Script Php horaires des rôles
Romain METAIS	<ul style="list-style-type: none">• Développement de l'interface de l'agent d'accueil• Développement de l'interface du technicien• Communication avec le système RFID



Analyse UML

Langage UML

Le langage de modélisation unifié (UML) est le langage standard que de nombreux ingénieurs logiciels et de nombreuses entreprises utilisent pour avoir une vue d'ensemble de systèmes complexes.



Les différents diagrammes que nous allons vous proposer sont des diagrammes crée avec ce langage.

DIA

Dia est un logiciel libre de création de diagramme.



Dia est conçu de manière modulaire avec plusieurs paquetages de formes pour des besoins différents : diagramme de flux, diagramme de circuit électrique, diagramme UML, etc.

A partir de ce logiciel nous avons conçu les différents diagrammes un à un. Les diagrammes UML les plus adaptés à notre projet sont :

- Diagramme de déploiement
- Diagramme de cas d'utilisation
- Diagramme d'activité
- Diagramme de séquences

Un algorithme a été conçu afin d'avoir une représentation détaillée de notre projet avec le tout cumulé.



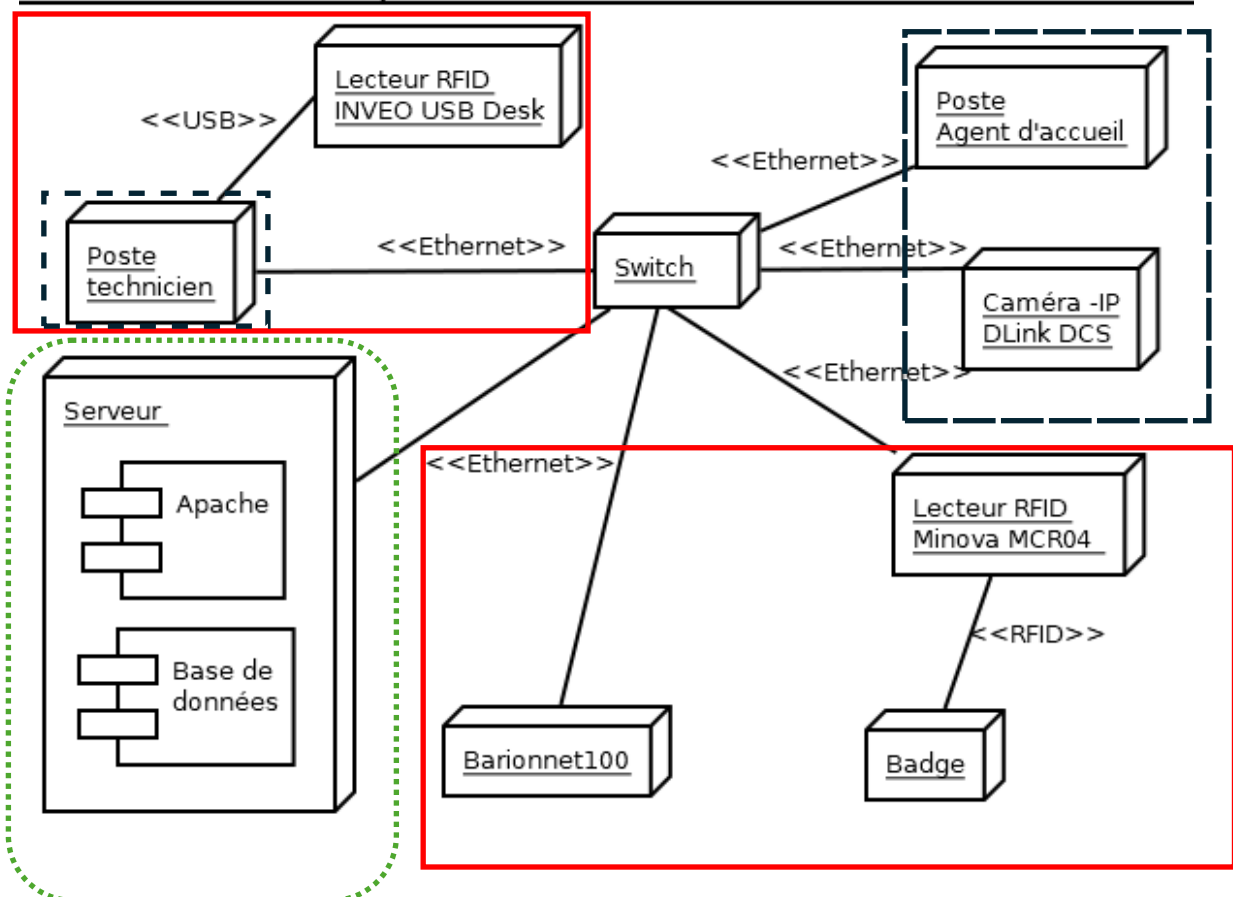
Diagrammes

Diagramme de déploiement

En UML, les diagrammes de déploiement modélisent l'architecture physique d'un système. Ces derniers affichent les relations entre les composants logiciels et matériels du système, d'une part, et la distribution physique du traitement, d'autre part.

Ils présentent la disposition physique des noeuds dans un système réparti, les artefacts qui sont stockés sur chaque noeud et les composants et autres éléments que les artefacts implémentent. Les noeuds représentent des périphériques matériels tels que des ordinateurs, des détecteurs et des imprimantes, ainsi que d'autres périphériques qui prennent en charge l'environnement d'exécution d'un système. Les chemins de communication et les relations de déploiement modélisent les connexions dans le système.

Diagramme de déploiement



— Partie de Akouman DIANGO

..... Partie de Elias COSME VINOU

- - - Partie de Romain METAIS



Ce diagramme de déploiement représente l'architecture d'un système utilisant des lecteurs RFID, un serveur, des postes de travail et des périphériques connectés via un réseau Ethernet. Voici une description détaillée de chaque composant :

Serveur (traits unis, partie de Elias COSME VINOUE)

- Héberge Apache (pour gérer l'application web).
- Contient une Base de données (stockage des informations RFID et autres données du système).

Poste technicien (traits en pointillé, partie de Romain METAIS)

- Connecté via Ethernet au réseau.
- Relié à un Lecteur RFID INVEO USB Desk via USB, permettant la lecture des badges RFID.

Poste agent d'accueil (traits en pointillé, partie de Romain METAIS)

- Connecté au réseau via Ethernet.
- Associé à une Caméra IP D-Link DCS pour la surveillance.

Switch (élément central du réseau)

- Connecte les différents équipements via Ethernet.

Barionnet100 (, partie de Akouman DIANGO)

- Connecté au réseau Ethernet.
- Probablement un contrôleur ou un module de communication.

Lecteur RFID Minova MCR04 (en rouge, partie de Akouman DIANGO)

- Connecté au réseau via Ethernet.
- Utilisé pour lire les Badges RFID.

Diagramme de cas d'utilisation

En langage de modélisation unifié (UML), un diagramme de cas d'utilisation peut servir à résumer les informations des utilisateurs de votre système (également appelés acteurs) et leurs interactions avec ce dernier. La création de ce type de diagramme UML requiert un ensemble de symboles et de connecteurs spécifiques. Lorsqu'ils sont bien conçus, les diagrammes de cas d'utilisation peuvent aider notre équipe à collaborer et représenter :

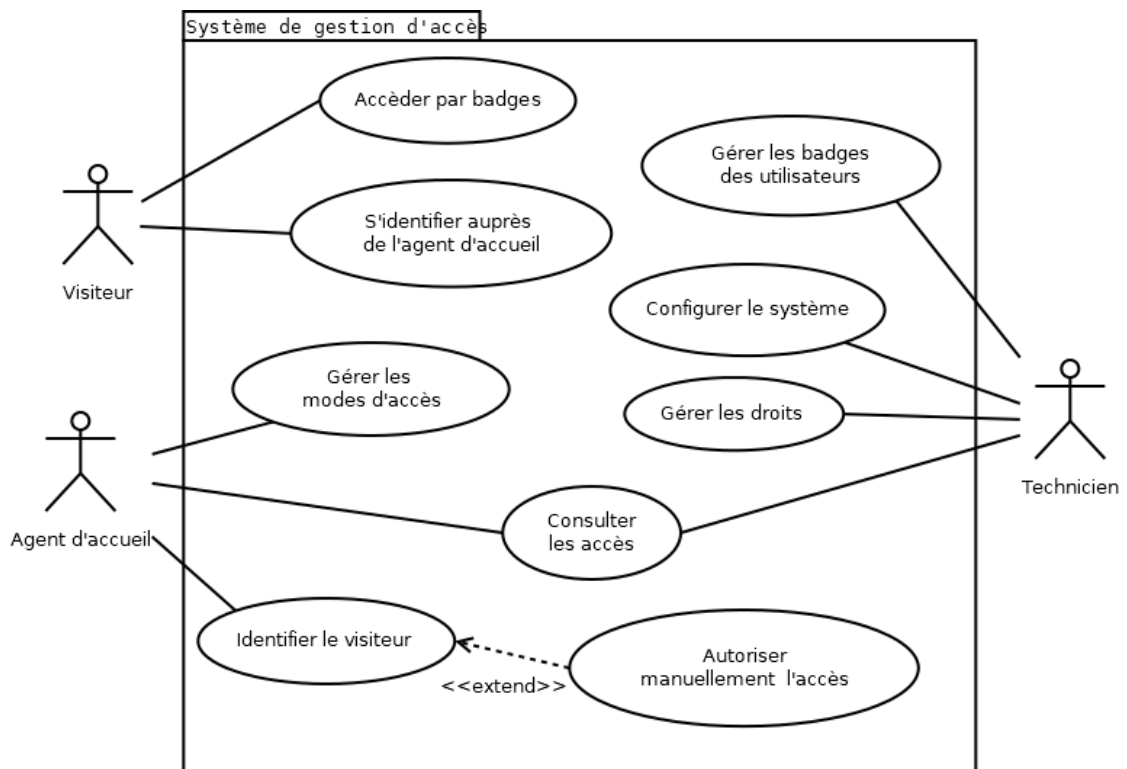
- Les scénarios dans lesquels notre système ou application interagit avec des personnes, des organisations ou des systèmes externes ;
- Les objectifs que notre système ou application permet aux entités (appelées acteurs) d'atteindre ;



- La portée de notre système.

Les diagrammes sont généralement composés :

- D'acteurs : utilisateurs qui interagissent avec un système. Un acteur peut être une personne, une organisation ou un système externe qui interagit avec votre application ou votre système. Il s'agit nécessairement d'objets externes qui produisent ou consomment des données.
- Du système : séquence spécifique d'actions et d'interactions entre les acteurs et le système. Un système peut également être appelé scénario.
- Des objectifs : résultat final de la plupart des cas d'utilisation. Un diagramme réussi doit décrire les activités et les variantes utilisées pour atteindre l'objectif.



Dans notre diagramme de cas d'utilisation les acteurs sont :

- Le visiteur
- L'agent d'accueil
- Le technicien

La balise <<extend>> présente dans le diagramme indique une relation optionnelle entre deux cas d'utilisation. Elle signifie qu'un cas d'utilisation (appelé extension) peut être inséré, sous certaines conditions, dans l'exécution d'un autre cas d'utilisation (appelé de base).



Dans notre diagramme, le cas d'utilisation « Identifier le visiteur » possède une extension vers le cas d'utilisation « Autoriser manuellement l'accès ». Cette relation signifie que, lors de l'identification d'un visiteur, il est parfois nécessaire pour l'agent d'accueil d'autoriser manuellement l'accès. Mais cette action n'est pas systématique : elle ne se produit que si une condition particulière est remplie (par exemple, un visiteur non reconnu par le système ou nécessitant une autorisation exceptionnelle).

On peut retenir que :

- Le cas d'utilisation « Identifier le visiteur » se déroule normalement.
- Mais, dans certains scénarios (non automatiques), une action supplémentaire (« Autoriser manuellement l'accès ») peut être exécutée, déclenchée par le cas principal.
- La relation <<extend>> permet donc de modéliser des options supplémentaires ou des “ajouts” qui ne sont pas toujours réalisés.

Ainsi, l'utilisation de <<extend>> rend le diagramme plus flexible et plus précis en illustrant les exceptions ou déroulés alternatifs possibles.



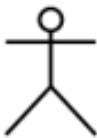
Visiteur

Peut accéder par badges et aussi s'identifier auprès de l'agent d'accueil.



Agent d'accueil

Peut gérer les modes d'accès, consulter les accès, identifier le visiteur et il peut optionnellement autoriser manuellement l'accès.



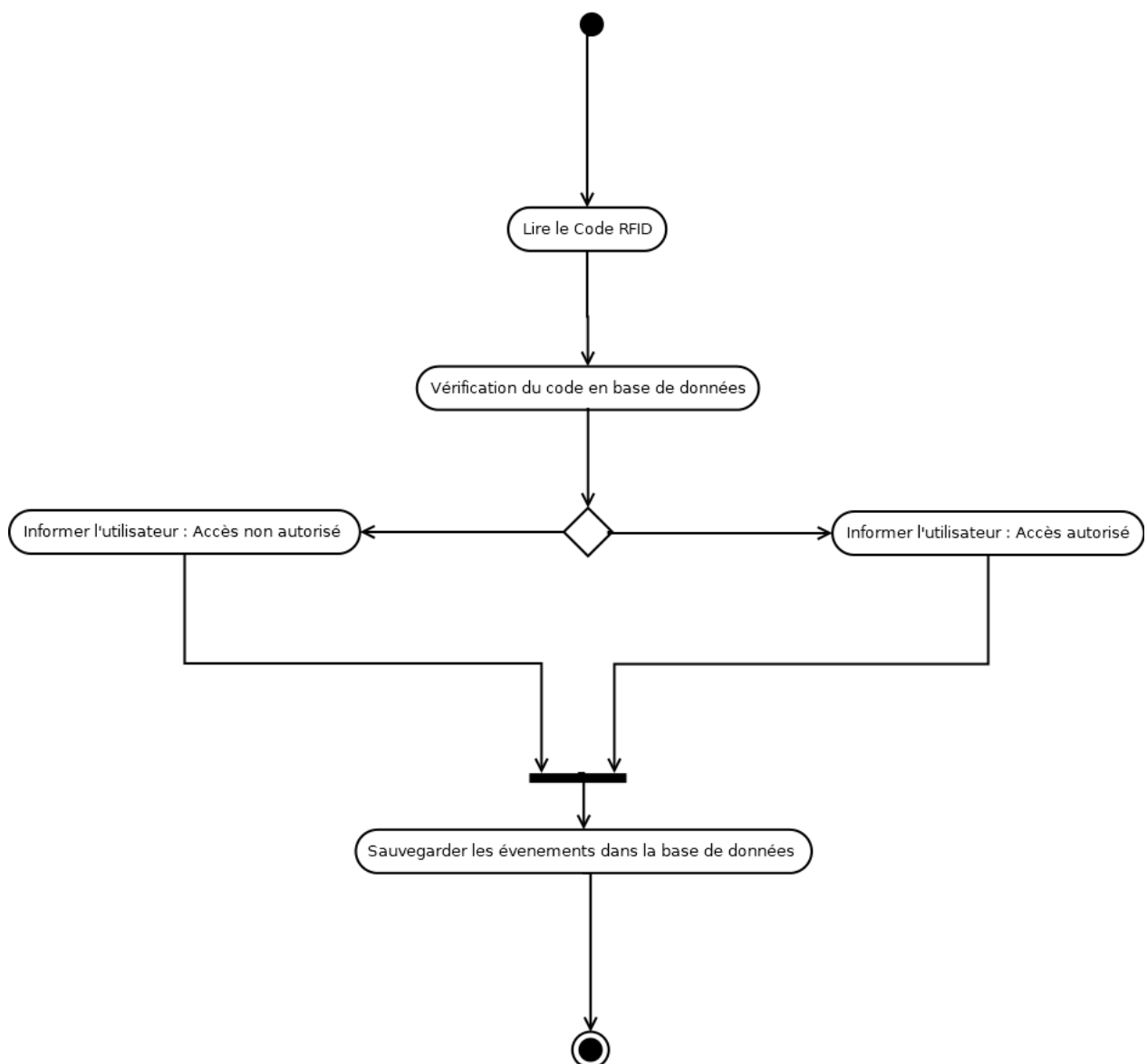
Technicien

Peut gérer les badges utilisateurs, configurer le système, gérer les droits et il peut aussi consulter les accès.



Diagramme d'activité

Le diagramme d'activité ci-dessous décrit le fonctionnement d'un système de contrôle d'accès basé sur la technologie RFID (Radio Frequency Identification). Ce système peut, par exemple, être utilisé pour contrôler l'ouverture d'un volet automatisé. J'ai donc conçu ce système en essayant qu'il réponde au maximum aux besoins utilisateurs tout en suivant les procédures strictes de structuration de diagramme d'activité.





Le diagramme d'activité ci-dessus décrit le fonctionnement d'un système de contrôle d'accès basé sur la technologie RFID (Radio Frequency Identification).

Lecture du badge RFID

- Le système commence à fonctionner lorsqu'un utilisateur présente son badge RFID devant le lecteur. Le code unique du badge est alors automatiquement détecté et transmis au système.

Vérification du code en base de données

- Le système interroge ensuite la base de données pour vérifier si le code lu correspond à un badge ayant l'autorisation d'accéder à la zone protégée.

Prise de décision (accès autorisé ou refusé)

- À ce stade, le système procède à la prise de décision automatique :
 - Si le badge n'est pas reconnu ou non autorisé, l'action d'accès est refusée. Un message d'information "Accès non autorisé" est affiché à l'utilisateur. Aucun mécanisme d'ouverture ne s'active et l'accès reste verrouillé.
 - Si le badge est autorisé, le système valide l'accès. Il actionne alors le portail (ou volet) pour l'ouvrir et informe simultanément l'utilisateur que l'accès est autorisé.

Enregistrement des événements

- Quelle que soit la décision (accès accepté ou refusé), l'événement est enregistré dans la base de données. Le code du badge, la date et l'heure de la tentative d'accès, ainsi que le résultat (accordé ou refusé) sont tous sauvegardés.
- Cette étape de traçabilité permet un suivi précis de toutes les tentatives d'accès, qu'elles soient réussies ou non.

Fin du processus

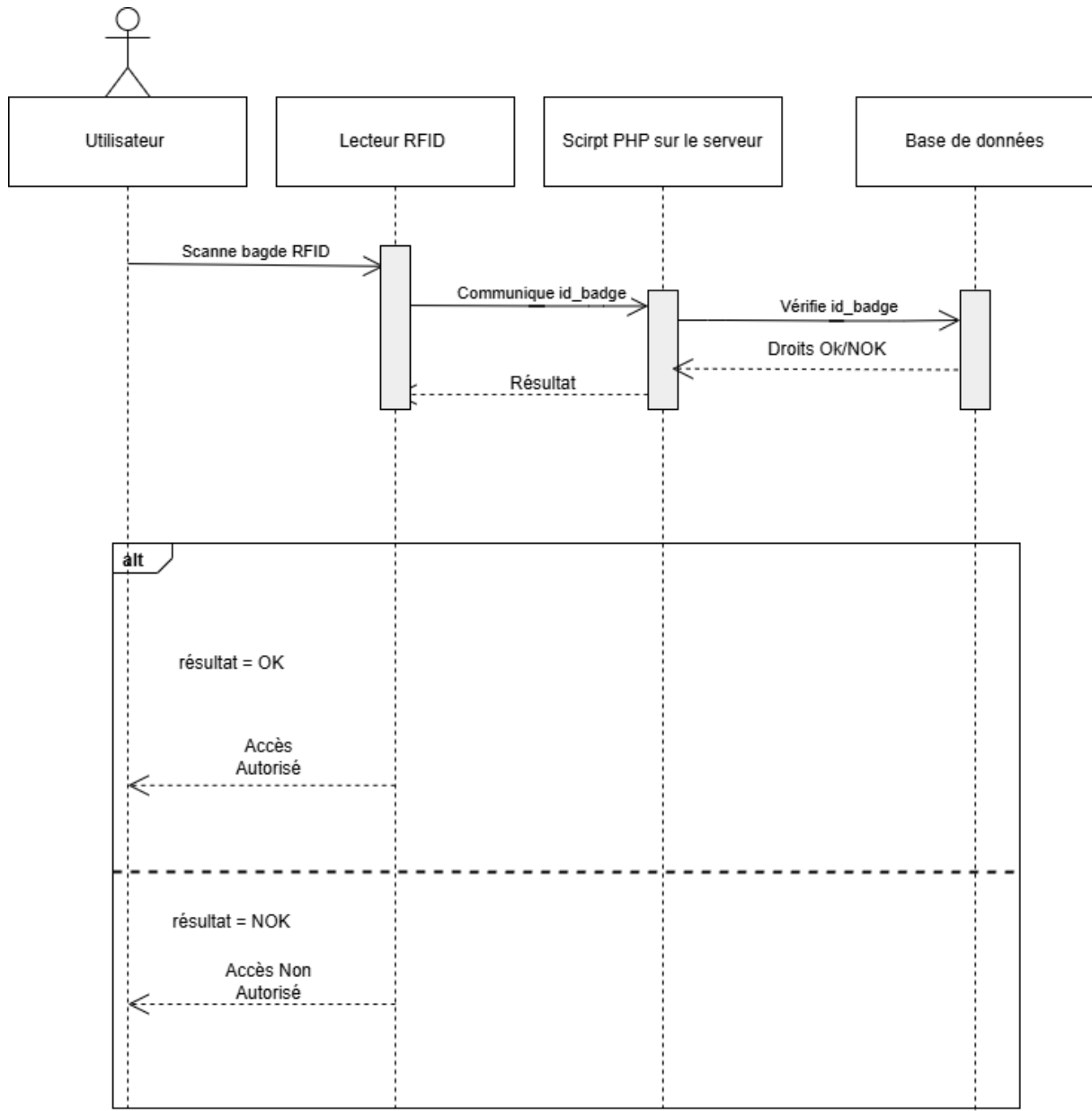
- Lorsque l'événement a bien été sauvegardé, le processus se termine, rendant le système prêt à traiter une nouvelle demande.

L'automatisation illustrée dans ce diagramme garantit que chaque tentative d'accès par badge RFID est traitée de manière fiable et sécurisée. Ce fonctionnement contribue à renforcer la sécurité du site tout en simplifiant son exploitation grâce à la gestion centralisée des droits d'accès et à la conservation de tous les historiques dans la base de données.



Diagramme de séquences

Un diagramme de séquences permet de visualiser, étape par étape, le déroulé logique des échanges entre différents acteurs et systèmes lors d'une action précise. Ici, il schématise le contrôle d'accès par badge dans un établissement (école, collège, lycée).



Dans le cadre d'un système de gestion des accès par badges, ce diagramme ci-dessus illustre le scénario typique pour contrôler l'accès à une porte via un badge RFID :



Ce que montre ce diagramme dans le contexte scolaire

Imaginons un utilisateur souhaitant entrer dans l'établissement :

Scannage du badge

L'utilisateur approche son badge RFID du lecteur situé à l'entrée.

Transmission de l'identifiant

Le lecteur RFID lit l'identifiant du badge (id_badge) et le transmet à un script PHP sur le serveur.

Vérification des droits d'accès

Le script PHP interroge la base de données contenant la liste des élèves autorisés à entrer :

- Il vérifie si cet élève est autorisé à accéder à l'établissement (présence dans la base, statut "autorisé", etc.).

Bloc alt

Le bloc alt est une notation spécifique à l'UML utilisée dans les diagrammes de séquences.

Il signifie "alternative".

Le bloc alt permet de modéliser une condition ou un choix dans le scénario décrit.

Il indique qu'il existe plusieurs chemins possibles à partir d'un point donné du processus, en fonction d'un résultat ou d'une condition.

Le bloc alt sépare visuellement ces deux alternatives dans le diagramme, montrant que le système agit différemment selon le résultat de la vérification.

Retour du résultat

La base de données envoie la réponse (OK : accès autorisé, NOK : accès refusé) au script PHP, qui la transmet au lecteur RFID.

Décision d'accès

Le bloc alt est utilisé pour spécifier les deux scénarios possibles après la vérification du badge :

- Si le résultat est OK : L'élève peut entrer, la porte s'ouvre ou un signal sonore/lumineux s'active pour valider.
- Si le résultat est NOK : L'accès est refusé, la porte reste verrouillée et l'élève est informé.

Boucle d'attente

Le système attend le prochain passage d'un badge.



Ce diagramme permet d'apporter une vision claire des interactions entre l'utilisateur, le matériel et le serveur. Et il apporte une identification des étapes clés pour le contrôle d'accès.

L'élève scanne son badge → Le lecteur envoie l'identifiant au serveur → Vérification dans la base élève → Accès autorisé ou refusé selon les droits enregistrés.

Ce diagramme est essentiel pour garantir la gestion sécurisée et fluide des accès des élèves à l'établissement.



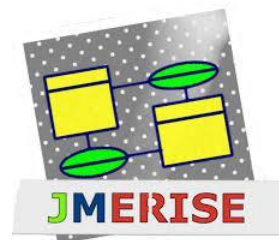
Base De Données

Conceptualisation et modélisation des données

Dans le cadre de ce projet, j'ai eu la charge de la création de la base de données. Cette étape a débuté par une phase de conceptualisation, où j'ai dû analyser les besoins pour identifier les différentes entités et les relations qui allaient composer la future base de données. J'ai structuré mes idées sous la forme d'un schéma Entité-Association (E/A), un outil graphique permettant de visualiser et organiser l'information.

Présentation de JMerise

Pour réaliser ce schéma, j'ai utilisé JMerise, un logiciel libre spécialisé dans la modélisation de bases de données. JMerise facilite la création de diagrammes Entité-Association (aussi appelés MCD pour Modèle Conceptuel de Données), outil incontournable lors de la conception d'une base de données relationnelle. Il propose une interface intuitive permettant de placer et relier graphiquement les entités, les associations, et leurs attributs.



Les avantages de JMerise :

- Création rapide de MCD et transformation automatique en Modèle Logique et Physique.
- Génération automatique du code SQL à partir du schéma conçu.
- Vérification de la cohérence du modèle.

Utilisation du diagramme Entité-Association

Le diagramme Entité-Association est une représentation graphique qui permet :

- D'identifier clairement les entités principales du système (ex : Utilisateur, Produit),
- De définir les relations ou associations entre ces entités (ex : Un utilisateur passe une commande),
- D'attribuer les attributs à chaque entité (ex : nom, adresse, date).

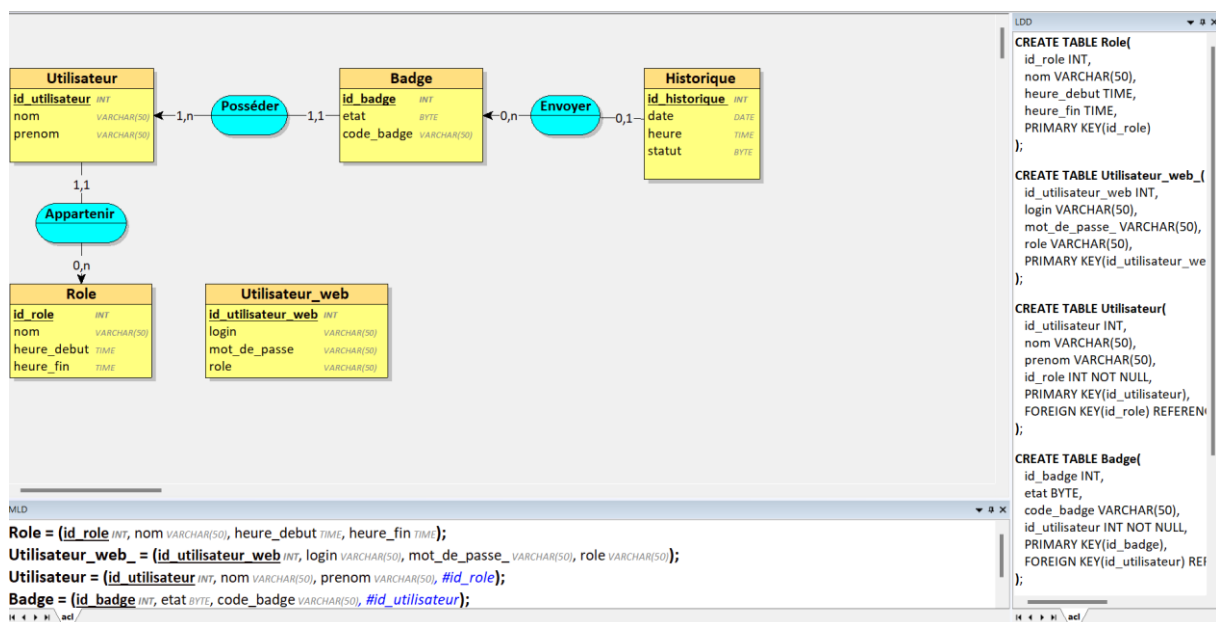
Grâce à ce diagramme, j'ai pu organiser de façon logique les différentes tables qui composeront la base de données, tout en définissant les clés primaires et étrangères nécessaires à l'intégrité des données.

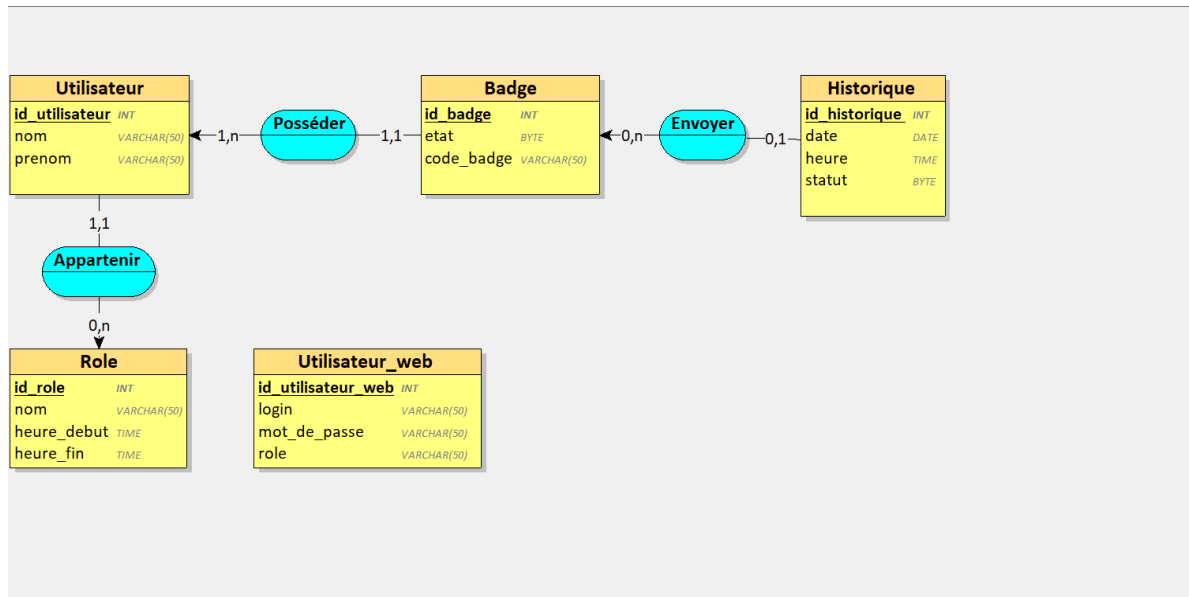


Génération du code SQL via JMerise

Une fois le schéma validé, JMerise propose une fonctionnalité précieuse : il permet de générer automatiquement le script SQL de création des tables correspondantes au modèle. Cette génération automatise l'écriture du code SQL de création de la base de données, en respectant la structure définie dans le diagramme (tables, clés primaires, clés étrangères, types de données, etc.).

Cela m'a permis de passer directement de la conception théorique (le schéma E/A) à la création concrète de la base de données dans le SGBD choisi, en évitant toute erreur de transcription et en gagnant en efficacité.





Analyse des besoins et identification des entités et associations

Afin de concevoir une base de données adaptée au projet, j'ai commencé par une analyse approfondie des besoins des utilisateurs. Cette phase a consisté à recueillir des informations sur les processus métiers, à comprendre les objectifs du système et à identifier les données à stocker ainsi que leurs interactions. Cette étude préalable m'a permis de formaliser les exigences sous forme de cas d'utilisation et de déterminer les principaux éléments à modéliser.

Détail des entités

Pour donner suite à cette analyse, j'ai identifié les entités suivantes :

1. Utilisateur

- But : Représenter chaque personne habilitée à utiliser un badge.
- Attributs :
 - id_utilisateur (clé primaire, INT)
 - nom (VARCHAR)
 - prenom (VARCHAR)

2. Badge

- But : Représenter chaque badge attribué aux utilisateurs.
- Attributs :
 - id_badge (clé primaire, INT)
 - etat (état d'activation, BYTE)



- code_badge (VARCHAR)

3. Historique

- But : Conserver les traces des événements associés à chaque badge.
- Attributs :
 - id_historique (clé primaire, INT)
 - date (DATE)
 - heure (TIME)
 - statut (BYTE)

4. Role

- But : Définir les différents rôles des utilisateurs dans le système, potentiellement associés à des plages horaires.
- Attributs :
 - id_role (clé primaire, INT)
 - nom (VARCHAR)
 - heure_debut (TIME)
 - heure_fin (TIME)

5. Utilisateur_web

- But : Gérer les accès des utilisateurs à l'interface web du système.
- **Attributs** :
 - id_utilisateur_web (clé primaire, INT)
 - login (VARCHAR)
 - mot_de_passe (VARCHAR)
 - role (VARCHAR)



Détail des associations

L'analyse métier m'a permis de formaliser les relations entre les différentes entités, en tenant compte des interactions attendues par les utilisateurs et les besoins de traçabilité.

1. Posséder

- Entre : Utilisateur et Badge
- Description : indique qu'un utilisateur peut **posséder** un ou plusieurs badges, et chaque badge est nécessairement attribué à un utilisateur.
- Cardinalité :
 - Un utilisateur possède 1 à n badges.
 - Un badge appartient obligatoirement à 1 utilisateur.

2. Envoyer

- Entre : Badge et Historique
- Description : trace les événements générés par l'utilisation des badges (ex : pointage, accès...).
- Cardinalité :
 - Un badge peut générer 0 à n événements dans l'historique.
 - Un événement historique ne concerne qu'un seul badge.

3. Appartenir

- Entre : Utilisateur et Role
- Description : chaque utilisateur remplit un ou plusieurs rôles, selon des plages horaires éventuellement définies. Cette association précise l'affectation des rôles aux utilisateurs.
- Cardinalité :
 - Un utilisateur appartient à au moins un rôle.
 - Un rôle peut être attribué à zéro, un ou plusieurs utilisateurs.

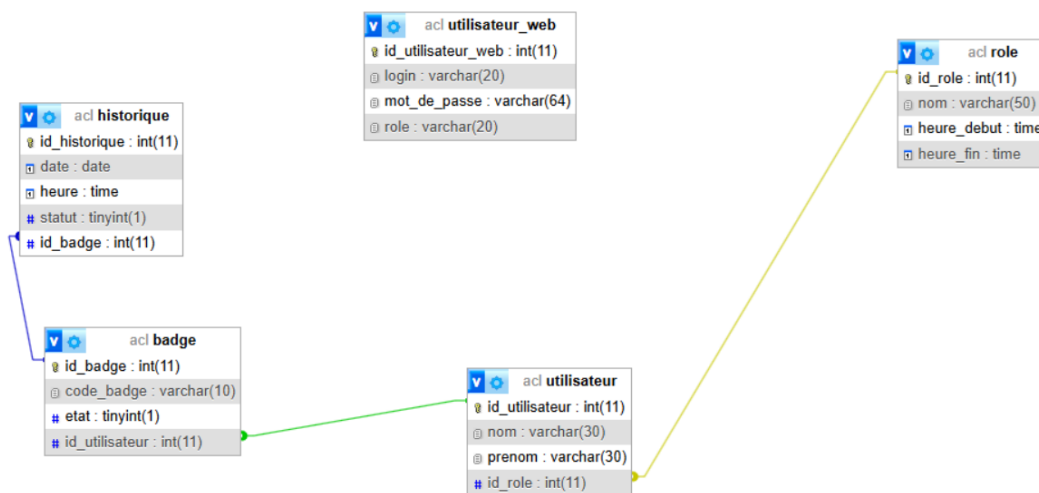
En résumé, la modélisation du schéma entité-association a été guidée par :

- La formalisation des besoins utilisateurs (qui fait quoi, qui a accès à quoi, historique des actions...),
- Le recensement des objets manipulés dans le système (utilisateurs, badges, rôles, accès web, historique),



- L'identification précise des liens entre ces objets et la gestion de leurs attributs-clés.

Cette approche m'a permis d'aboutir à un modèle conceptuel de données cohérent et fidèle aux attentes des utilisateurs, base solide pour la génération automatique du schéma physique (code SQL) via JMerise.



Tables

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> badge	Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb4_general_ci	32,0 kio	-
<input type="checkbox"/> historique	Parcourir Structure Rechercher Insérer Vider Supprimer	10	InnoDB	utf8mb4_general_ci	32,0 kio	-
<input type="checkbox"/> role	Parcourir Structure Rechercher Insérer Vider Supprimer	6	InnoDB	utf8mb4_general_ci	16,0 kio	-
<input type="checkbox"/> utilisateur	Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb4_general_ci	32,0 kio	-
<input type="checkbox"/> utilisateur_web	Parcourir Structure Rechercher Insérer Vider Supprimer	2	InnoDB	utf8mb4_general_ci	16,0 kio	-
5 tables	Somme	28	InnoDB	utf8mb4_general_ci	128,0 kio	0 0

L'architecture de la base de données, présentée ci-dessus, a été pensée pour répondre précisément aux exigences du projet, telles qu'exprimées lors de l'analyse des besoins (diagrammes de cas d'utilisation et d'activités).

Chaque table joue un rôle central dans la couverture des différents scénarios d'usage.



Traçabilité et sécurité : Table Historique

- Besoins utilisateurs couverts :
 - *Consultation des accès* (cas d'utilisation : "Consulter l'historique des accès")
 - *Audit et suivi des passages au sein de l'établissement*
- Fonction :
 - Chaque événement (entrée ou tentative d'accès) est consigné avec la date, l'heure, le statut (accepté ou refusé) et le badge concerné.
 - Cela permet aussi bien aux administrateurs qu'aux responsables de la sécurité d'avoir une **vision complète de l'activité**, d'identifier les comportements anormaux, ou de retrouver l'historique en cas d'incident.

Authentification : Table Badge

- Besoins utilisateurs couverts :
 - *Identification automatique lors de la présentation du badge* (cas d'utilisation : "Présenter son badge à la borne")
 - *Gestion des badges actifs/inactifs* (cas d'utilisation : "Désactiver un badge perdu")
- Fonction :
 - Attribue à chaque utilisateur un identifiant unique.
 - Permet la suppression temporaire ou définitive d'un droit d'accès en cas de perte ou de vol.

Gestion des personnes : Table Utilisateur

- Besoins utilisateurs couverts :
 - *Création et gestion des comptes utilisateurs* (cas d'utilisation : "Enregistrer un nouvel utilisateur")
 - *Affectation des rôles* (gestion des droits)
- Fonction :
 - Centralise toutes les informations sur les personnes autorisées.
 - Rend possible la gestion individualisée (modification d'un utilisateur particulier, changement de rôle, etc.).



Gestion des droits et automatisation des horaires : Table Rôle

- Besoins utilisateurs couverts :
 - *Contrôle d'accès différencié selon le profil* (cas d'utilisation : "Accéder à l'établissement selon son rôle")
 - *Définition des plages horaires pour chaque catégorie* (cas d'activités : "Contrôler l'accès en fonction de l'horaire")
- Fonction :
 - Permet de définir finement les autorisations d'accès pour chaque profil : lycéen, personnel, agent, etc.
 - Automatise la validation ou le refus d'accès selon la période, sans intervention humaine, rendant le système souple et évolutif.

Contrainte : ON DELETE CASCADE

Dans notre projet, chaque membre (utilisateur) peut posséder un ou plusieurs badges électroniques (badge) dans la base de données. Lorsque l'on supprime un utilisateur, il est également nécessaire de supprimer tous les badges associés afin de garantir la cohérence et la propreté des données, et ainsi éviter de conserver des éléments "orphelins".

Utilisation de ON DELETE CASCADE

Pour automatiser ce comportement, j'ai défini une contrainte d'intégrité référentielle avec l'option ON DELETE CASCADE sur la clé étrangère reliant la table badge à la table utilisateur.

Cela signifie que lorsqu'un utilisateur est supprimé, tous les badges qui lui appartiennent sont automatiquement supprimés par le système de gestion de base de données (SGBD), sans avoir besoin d'effectuer une suppression manuelle au niveau applicatif.

Actions	Propriétés de la contrainte	Colonne	Contrainte de clé étrangère (INNODB)		
			Base de données	Table	Colonne
<input type="button" value="Supprimer"/>	Badge_Utilisateur_FK ON DELETE: CASCADE ON UPDATE: RESTRICT	id_utilisateur <input type="button" value="+ Ajouter une colonne"/>	acl	utilisateur	id_utilisateur
	Nom de la contrainte ON DELETE: RESTRICT ON UPDATE: RESTRICT	<input type="button" value="+ Ajouter une colonne"/>	acl	<input type="button" value="+ Ajouter une colonne"/>	<input type="button" value="+ Ajouter une colonne"/>



La table historique contient des enregistrements relatifs à l'utilisation de chaque badge (par exemple, les dates d'accès ou d'autres actions). Elle possède une colonne `id_badge` qui référence la table `badge`.

Il a été aussi nécessaire d'appliquer la contrainte dans la table historique de la même façon.

Grâce à ce choix technique, la suppression d'un membre dans l'interface développeur est simple et sûre : il suffit de supprimer l'utilisateur, la base de données s'occupe du reste.

Le code PHP reste plus lisible, court, et il n'y a plus de risque de laisser des badges orphelins dans la table `badge`

Création du compte utilisateur

Par défaut, MySQL propose un utilisateur `root` doté de tous les privilèges, souvent sans mot de passe lors des installations locales. Cependant, utiliser ce compte dans un projet présente de gros risques de sécurité :

- Accès total à toutes les bases de données,
- Risque d'effacement ou de modification accidentelle de données critiques,
- Vulnérabilité accrue en cas de fuite de l'identifiant de connexion.

Pour renforcer la sécurité et la gestion des accès, j'ai créé un utilisateur dédié nommé `acl` pour le projet, avec un mot de passe fort, comme on peut le voir dans la configuration suivante :

- Nom d'utilisateur : `acl`
- Hôte : `%acl` ou `localhost` selon le besoin (ici, `%acl` permet l'accès depuis l'hôte local)
- Mot de passe : défini et stocké de façon sécurisée (exemple : `ac1lp2i3!`)
- Privilèges attribués : uniquement ceux nécessaires pour l'application :
 - `SELECT, INSERT, UPDATE, DELETE, FILE`
 - Aucun accès global non-autorisé

- Intégration dans le code PHP du projet :

Le compte `acl` est utilisé dans le fichier de configuration pour toutes les interactions entre l'application et la base de données, à la place de `root`.

```
10 // Définition des identifiants de connexion
11 $login = 'acl';
12 $mdp = 'ac1lp2i3!';
```



Sécurité accrue : Si la connexion de l'application est compromise, seule la base concernée par le projet est accessible, et seulement avec des droits limités.

Meilleure gestion du code : L'utilisation d'un utilisateur spécifique facilite la maintenance (identification des connexions, logs, etc.).

Respect des bonnes pratiques : En production comme en développement, il est conseillé de ne jamais utiliser root dans une application.

Adaptabilité : Si les droits de l'utilisateur doivent évoluer (par exemple suppression ou ajout de permissions), il suffit de modifier les privilèges du compte acl sans impacter les autres applications ou données.

<input type="checkbox"/> acl	%acl	Oui	SELECT, INSERT, UPDATE, DELETE, FILE	Non	Éditer les privilèges	Exporter	Verrouiller
------------------------------	------	-----	---	-----	-----------------------	----------	-------------

La création d'un compte utilisateur dédié, associé à un mot de passe fort et à des droits restreints, est une étape essentielle dans tout projet utilisant une base de données, pour garantir la sécurité, la traçabilité et la robustesse de l'application.

Gestion des horaires d'accès selon les rôles des utilisateurs

L'accès à un établissement doit être régulé en fonction du profil de chaque utilisateur, pour des raisons de sécurité, d'organisation interne et de gestion des flux. Chaque catégorie d'utilisateur (lycéen, administratif, agent, etc.) doit pouvoir entrer et sortir selon des plages horaires qui lui sont spécifiquement autorisées.

Mise en œuvre dans le projet

Pour répondre à cette exigence, j'ai créé une table role dans la base de données qui associe à chaque rôle :

- Un nom (exemple : Lycéen, Administration, Personnel, etc.)
- Une heure de début d'accès (heure_debut)
- Une heure de fin d'accès (heure_fin)

Exemple de structuration de la table :

id_role	nom	heure_debut	heure_fin
1	Lycéen	08 :00 :00	18 :00 :00
2	Administration	00 :00 :00	23 :59 :59
3	Personnel	06 :00 :00	19 :00 :00
4	Étudiant	08 :00 :00	18 :00 :00
5	Agent	07 :00 :00	18 :30 :00
6	Technicien	06 :00 :00	22 :00 :00

Lorsque l'utilisateur tente d'entrer dans l'établissement via son badge, le système :

1. Identifie son rôle via la base de données.



2. Récupère la plage horaire autorisée pour son rôle (par exemple, un lycéen de 8h à 18h).
3. Vérifie que la tentative d'entrée se situe bien dans cet intervalle.
 - Si oui, l'accès est accordé.
 - Sinon, l'accès est refusé et peut générer un rapport ou une alerte.

Avantages de cette gestion par rôle

- **Sécurité accrue** : seuls les profils autorisés peuvent accéder à certaines heures.
- **Souplesse de gestion** : il suffit de modifier une valeur dans la table role pour changer les horaires d'un type d'utilisateur pour tout le système.
- **Adaptabilité** : chaque nouveau rôle peut aisément se voir attribuer des horaires particuliers, sans modifier la logique du code.
- **Automatisation de la surveillance** : possibilité de générer des statistiques ou alertes en cas de tentatives d'accès en dehors des horaires.

L'attribution d'horaires d'accès par rôle permet d'offrir une gestion fine, sécurisée et centralisée des entrées et sorties dans l'établissement, tout en rendant le système flexible et facile à faire évoluer.

PhpMyAdmin

La base de données nous permet de stocker toutes les informations nécessaires à la réussite de ce projet comme la liste des utilisateurs. Notre SGB (système de gestion de base de données) est PhpMyAdmin.

PhpMyAdmin est un outil libre écrit en PHP qui permet de gérer facilement les bases de données MySQL ou MariaDB via une interface web conviviale. Il est largement utilisé par les développeurs et administrateurs de bases de données pour effectuer des opérations courantes sur les bases de données, sans avoir à utiliser la ligne de commande.



Fonctionnalités principales

- **Gestion des bases de données**
Créer, modifier, supprimer des bases de données, tables et utilisateurs.
- **Manipulation des données** Insérer, éditer, supprimer et rechercher des enregistrements dans les tables.



- **Exécution de requêtes SQL** Permet d'exécuter des requêtes SQL (création de tables, insertion de données, requêtes de sélection...) et d'en visualiser instantanément les résultats.
- **Sauvegarde et restauration** Exporter les bases de données (au format SQL, CSV, etc.) pour les sauvegarder, ou importer des fichiers pour restaurer des données.
- **Gestion des clés et des relations** Modifier la structure des tables, gérer les index, clés primaires et étrangères, pour garantir l'intégrité des données.
- **Interface graphique intuitive** Les opérations se font grâce à une interface web interactive, ce qui rend la gestion plus accessible, même pour les débutants.

Utilisation dans le projet

Dans le cadre de ce projet, une fois que le code SQL de création généré par JMerise a été finalisé, je l'ai importé dans phpMyAdmin pour créer la base de données. Grâce à phpMyAdmin, j'ai pu :

- Visualiser la structure des tables générées,
- Ajouter ou modifier des enregistrements pour tester la base,
- Tester les requêtes SQL,
- Exporter les données pour des besoins de sauvegarde ou de migration,
- Gérer facilement les droits utilisateurs.

Avantages de phpMyAdmin

- Accessibilité : Disponible depuis un simple navigateur web,
- Gratuit et open source,
- Compatible avec la majorité des hébergeurs web et serveurs locaux (XAMPP, WAMP, MAMP...),
- Facile à prendre en main.

PhpMyAdmin s'est révélé être un outil essentiel pour la gestion et la supervision de la base de données lors du projet, en facilitant aussi bien la phase de développement que les tests et la maintenance.

Utilisation de MariaDB et du moteur InnoDB

Dans le cadre de ce projet, la base de données a été créée sous MariaDB, un système de gestion de bases de données relationnelles libre et open source, largement utilisé comme alternative à MySQL.

MariaDB est compatible avec MySQL, tout en offrant de meilleures performances, une sécurité accrue et une communauté active.





Le moteur de stockage InnoDB

Pour ce projet, le moteur de stockage utilisé est InnoDB, qui est le moteur par défaut pour MariaDB et MySQL.

InnoDB est un moteur de stockage transactionnel, reconnu pour ses fonctionnalités avancées, notamment :

- **Support des transactions ACID**
InnoDB garantit l'intégrité des données grâce au respect des propriétés Atomique, Cohérence, Isolation et Durabilité (ACID).
- **Gestion des clés étrangères (foreign keys)**
InnoDB permet de maintenir l'intégrité référentielle entre les tables grâce à la gestion des clés étrangères.
- **Verrouillage au niveau des lignes (row-level locking)**
Cela améliore les performances dans les environnements multi-utilisateurs et réduit les conflits lors de l'accès simultané aux données.
- **Rétablissement automatique**
En cas de panne ou de coupure, InnoDB est capable de restaurer un état cohérent de la base de données grâce à ses mécanismes de journalisation (logs).



Principe du développement en local avec XAMPP



Pour faciliter le développement et les tests de mon application, j'ai utilisé XAMPP en environnement local.

XAMPP est un package tout-en-un qui permet de simuler un serveur web sur un ordinateur personnel, sans avoir besoin d'infrastructure distante.

Composants principaux de XAMPP :

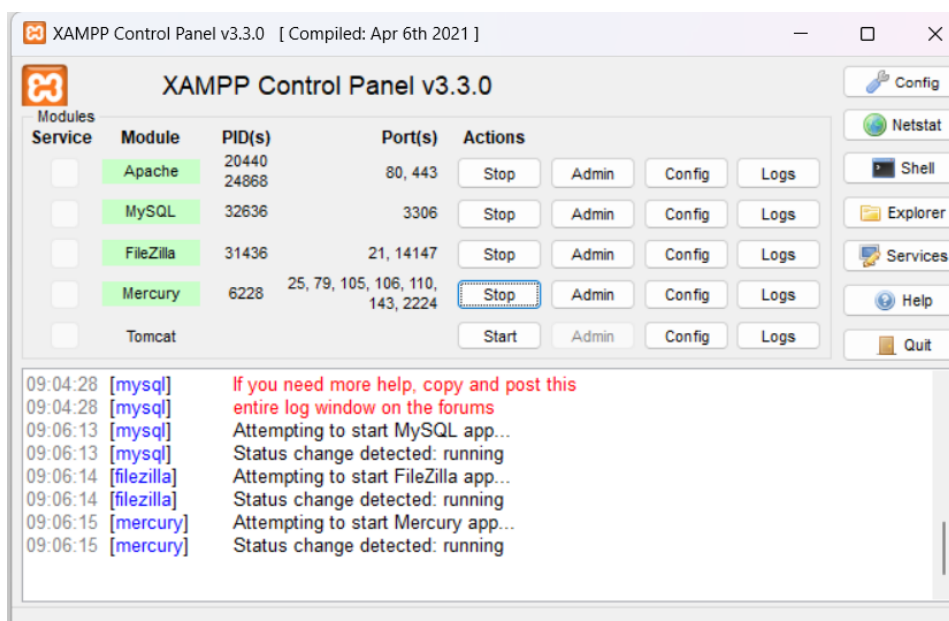
- **Apache** : serveur web qui permet d'héberger des pages web dynamiques.
- **MySQL/MariaDB** : système de gestion de bases de données relationnelles.
- **PhpMyAdmin** : interface web pour gérer la base de données facilement.
- (Autres composants : FileZilla, Mercury, Tomcat... utiles selon les projets.)

Fonctionnement en local :

1. Installation de XAMPP

J'ai installé XAMPP sur mon poste de travail, ce qui m'a permis d'avoir tous les outils nécessaires pour travailler comme sur un vrai serveur.

2. Lancement des services Apache et MySQL/MariaDB

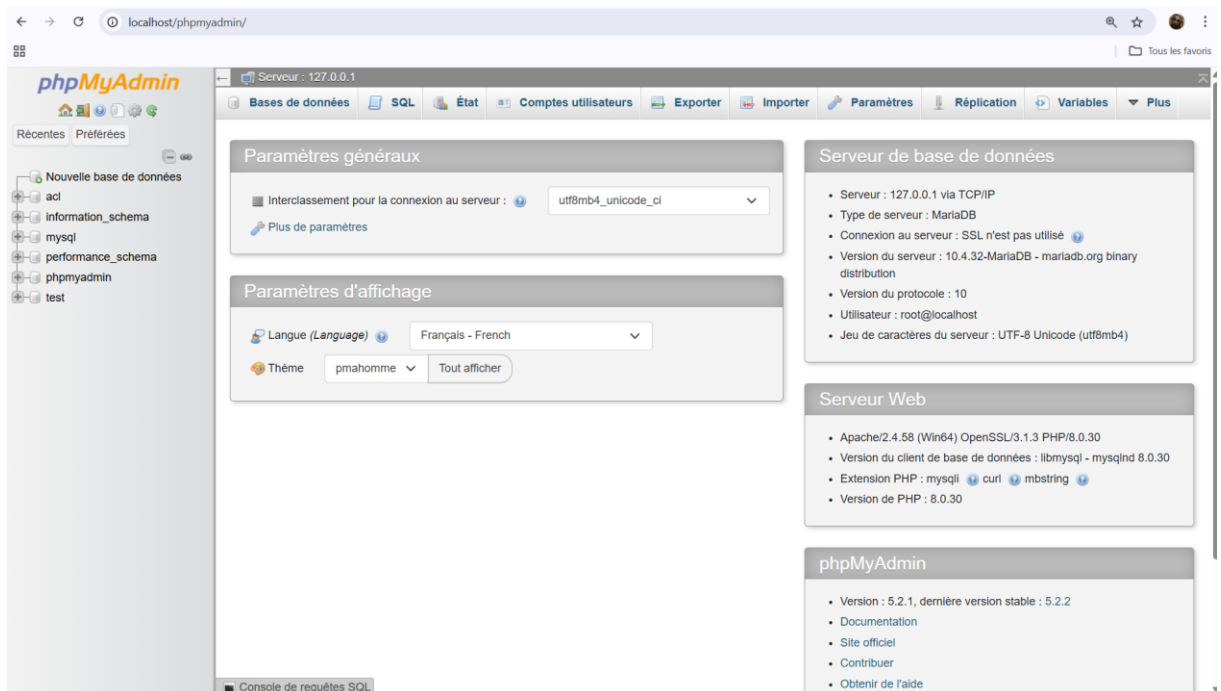




Grâce au XAMPP Control Panel (voir capture ci-dessus), j'ai démarré les services Apache et MySQL.

- Apache gère l'hébergement en local de mes fichiers PHP.
- MySQL/MariaDB gère la base de données de l'application (dans mon cas il s'agit de MariaDB).

3. Accès à PhpMyAdmin



En ouvrant un navigateur web et en allant à l'adresse <http://localhost/phpmyadmin/>, j'accède à l'interface de gestion de la base de données.

Cela me permet de :

- Créer et tester la structure de la base de données générée par JMerise.
- Manipuler les données pour les tests.
- Gérer les utilisateurs et les droits de la base de données.

4. Développement et tests

En local, je peux développer, tester et déboguer le fonctionnement de mon application (connexion à la base de données, requêtes SQL, gestion des utilisateurs, etc.) en toute sécurité avant de déployer sur un serveur réel.

Avantages du local avec XAMPP :

- Tests sans risque sur un environnement isolé.
- Facilité d'installation et de configuration.



- Tous les outils nécessaires regroupés dans une seule interface.
- Rapidité pour lancer, arrêter ou redémarrer les services.

XAMPP m'a permis de disposer rapidement d'un environnement complet pour développer et manipuler la base de données MariaDB, outil essentiel pour la réussite et la fiabilité de mon projet.

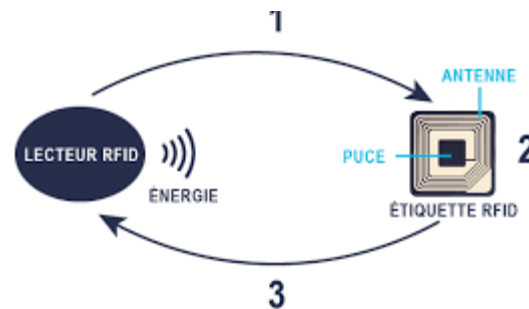


Technologie RFID

La technologie RFID est une méthode pour mémoriser et récupérer des données à distance en utilisant des marqueurs appelés « radio-étiquettes ». Ces radio-étiquettes sont des petits objets qui peuvent être incorporés ou collés, ici on va s'en servir sur un badge (ou carte) RFID. Quand le badge va passer devant le lecteur, celui-ci va envoyer un signal à l'étiquette. L'étiquette retourne alors son identification numérique (Code RFID).



Pour que l'utilisateur puisse accéder à l'établissement, il va passer sa carte devant le lecteur, le calculateur embarqué (Barionnet) va récupérer le code RFID et va envoyer ce code au serveur. Le serveur va ensuite interroger la base de données (Base de données (MariaDB)). Ainsi on va pouvoir vérifier que l'utilisateur correspondant à ce code RFID existe réellement, et que l'utilisateur essaie d'accéder à la salle dans la bonne plage horaire par rapport à son statut (Etudiant, Personnel).



Le technicien et l'agent d'accueil ont la possibilité d'effectuer plusieurs actions depuis le client. Ils sont les seuls à pouvoir s'identifier à l'application grâce à des identifiants stockés en base de données. Il peut effectuer des opérations CRUD (Create, Read, Update, Delete) sur les utilisateurs (étudiants et personnels du lycée) ainsi que sur les plages horaires en fonction du rôle de la personne. Pour finir, ils peuvent consulter l'historique des entrées. L'historique est constitué de toutes les tentatives d'accès aux salles (échouées ou réussies). Un étudiant doit passer sa carte RFID devant le lecteur, si le créneau horaire le permet, il peut accéder à la salle. Le personnel de l'établissement aura des restrictions différentes, lorsqu'ils passeront leur carte RFID devant le lecteur, la porte s'ouvrira automatiquement à des horaires différentes.



Script de connexion à la base de données

Ce script a pour objectif principal de centraliser et de sécuriser les paramètres de connexion à la base de données pour l'ensemble de l'application. Il prépare l'environnement nécessaire pour que d'autres scripts PHP puissent accéder à la base de données MySQL de manière optimale et sécurisée.

Language de Progammation

PHP (Hypertext Preprocessor) est un langage de programmation libre, principalement utilisé côté serveur pour le développement web. Il permet de créer des pages et des applications dynamiques, d'interagir avec des bases de données, et de gérer la logique métier de nombreux sites et systèmes d'information. Son code s'exécute sur le serveur web et peut générer dynamiquement du HTML, communiquer avec des API, envoyer des emails, gérer des fichiers, etc.



Grâce à PHP, toutes les étapes de contrôle d'accès (vérification, enregistrement, réponse à la borne, gestion des erreurs) sont automatisées de manière fiable, rapide et sécurisée. PHP est donc un élément clé pour que le système soit dynamique et réactif en fonction des données de la base et de l'activité réelle des utilisateurs.

Détails du script Php

1. Définition des paramètres de connexion à la base de données

```
$host = 'localhost';  
$DB = 'acl';  
$port = '3306';
```

\$host='localhost' est l'adresse du serveur de base de données (ici la machine locale)

\$DB='acl' est le nom de la base de données à consulter

\$port='3306' est le numéro du port du serveur

2. Construction de la chaîne DSN (Data Source Name)

```
// Définition du Data Source Name (DSN)  
$dsn = 'mysql:host=' . $host . ';dbname=' . $DB . ';charset=utf8;port=' . $port;
```

La DSN est une chaîne d'information qui indique au PDO comment se connecter à la base de données. Elle précise le type de base (ici, mysql), l'hôte, le nom de la base, l'encodage (utf8) et le port.



3. Définition des identifiants d'accès à la base de données

```
// Définition des identifiants de connexion
$login = 'root';
$mdp = '';
```

\$login='root' est le nom d'utilisateur pour la base données

\$mdp=' ' est le mot de passe (vide ici pour l'instant)

Ces deux variables stockent les informations nécessaires pour s'authentifier auprès du serveur.

4. Configuration des options de connexion PDO

```
// Définition des options de PHP Data Objects (PDO)
$pdo_options = [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
];
```

PDO (pour *PHP Data Objects*) est une extension native de PHP qui permet de se connecter et d'interagir de manière sécurisée avec plusieurs types de bases de données (MySQL, PostgreSQL, SQLite, etc.).

C'est une méthode moderne et recommandée, car :

- Elle centralise la gestion des connexions et requêtes.
- Elle permet des connexions sécurisées (préparation de requêtes, gestion des erreurs).
- Elle facilite le changement de type de base de données sans modifier tout le code de l'application.

`PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION` permet de gérer les erreurs sous forme d'exceptions

`PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC` permet de récupérer les résultats sous forme de tableau d'associatifs.



5. Déclaration des constantes de connexion pour l'application

```
// Définir les constantes pour la connexion à la base de données
define('DB_DSN', $dsn);
define('DB_USER', $login);
define('DB_PASS', $mdp);
```

- **define()** permet de créer des constantes PHP.
- On utilise ces constantes plus tard dans le code pour établir la connexion sans réécrire les valeurs. Mon camarade projet Romain METAIS s'en occupe.

Ce code prépare donc tous les paramètres nécessaires à une connexion sécurisée à une base de données MySQL :

- Il déclare le serveur, la base, le port, l'utilisateur et le mot de passe.
- Il construit une chaîne de connexion (DSN).
- Il configure les options de gestion des erreurs et du format des résultats.
- Enfin, il définit des constantes à réutiliser lors de l'instanciation de l'objet PDO, pour établir la connexion proprement dite ailleurs dans l'application.



Script De Gestion Des Accès Selon Les Horaires

L'objectif de ce script est de vérifier, lors de chaque tentative d'entrée dans l'établissement, si l'utilisateur a le droit d'accéder aux locaux en fonction de son badge et de la plage horaire associée à son rôle. L'ensemble du processus est automatisé depuis la lecture du badge jusqu'à l'ouverture de la porte (via Barionet), et l'enregistrement dans l'historique.

Fonctionnement du code

1. Récupération des informations du badge

Lorsque le badge est présenté au lecteur, le script reçoit deux paramètres (UID et devID).

→ Il récupère les informations de l'utilisateur associé au badge (si le badge est valide et actif).

2. Récupération des horaires d'accès

À partir de l'ID du rôle, le script interroge la table role pour obtenir la plage horaire autorisée à ce profil.

3. Contrôle du créneau horaire

- Il compare l'heure actuelle à l'intervalle autorisé pour le rôle.
- Si l'horaire est respecté, l'accès est accordé :
 - Insertion d'un enregistrement positif dans la table historique
 - Déclenchement d'un relais pour l'ouverture de porte via Barionet
 - Affichage d'un message de validation sur l'écran connecté
- Sinon, l'accès est refusé :
 - Insertion d'un refus dans l'historique
 - Affichage d'un message d'accès refusé (hors horaires)

4. Badges invalides ou désactivés

- Si le badge n'est pas reconnu ou est désactivé, l'accès est refusé et l'incident est enregistré.

5. Gestion d'incidents et journalisation

- Une erreur de connexion à la base entraîne l'affichage d'un message d'erreur spécifique et une trace dans les logs serveur.



Détails du script Php

1. Récupération des paramètres de requête

```
1 <?php
2 $code_badge = htmlspecialchars($_GET["UID"]);
3 $dev_id = htmlspecialchars($_GET["devID"]);
4 require '../config.php';
```

Ces deux variables récupèrent, depuis les paramètres GET de l'URL, le code du badge (UID) et l'identifiant du dispositif (devID).

On utilise htmlspecialchars() pour éviter les risques d'injection.

2. Connexion à la base de données

```
4 require '../config.php';
5
6 try {
7     $conn = new PDO(DB_DSN, DB_USER, DB_PASS, $pdo_options);
```

Le fichier de configuration (config.php) définit tous les paramètres nécessaires pour une connexion sécurisée à la base de données avec PDO.

Cela centralise la configuration et protège l'accès.

3. Récupération du badge et de l'utilisateur associé

```
// 1. Vérification du badge + récupération utilisateur + rôle (changer b.id_utilisateur si besoin)
$sql = "SELECT b.id_badge, u.id_utilisateur, u.nom, u.prenom, u.id_role
        FROM badge b
        JOIN utilisateur u ON b.id_utilisateur = u.id_utilisateur
        WHERE b.code_badge = :code_badge AND b.etat = 1";
$reponse = $conn->prepare($sql);
$reponse->bindValue(':code_badge', $code_badge, PDO::PARAM_STR);
$reponse->execute();
```

Cette requête permet :

- De vérifier si le badge présenté est bien actif (b.etat = 1)
- D'identifier l'utilisateur lié au badge, ainsi que son rôle (id_role), ce qui sera utile pour contrôler les horaires d'accès.

4. Vérification de la validité du badge

```
if ($reponse->rowCount() > 0) {
    $badge_info = $reponse->fetch(PDO::FETCH_ASSOC);
    $id_badge = $badge_info['id_badge'];
    $id_role = $badge_info['id_role'];
```

Si le badge est valide et associé à un utilisateur, on récupère son identifiant et celui du rôle de l'utilisateur.



5. Récupération des horaires autorisés selon le rôle

```
// 2. Récupérer les horaires via id_role (clé étrangère)
$sqlHoraire = "SELECT heure_debut, heure_fin FROM role WHERE id_role = :id_role";
$getHoraire = $conn->prepare($sqlHoraire);
$getHoraire->bindValue(':id_role', $id_role, PDO::PARAM_INT);
$getHoraire->execute();
$horaire = $getHoraire->fetch(PDO::FETCH_ASSOC);
```

Cette section interroge la table role pour obtenir la plage horaire d'accès autorisée pour l'utilisateur concerné.

6. Vérification de l'heure actuel avec les horaires autorisés

```
if ($horaire && $heureActuelle >= $horaire['heure_debut'] && $heureActuelle <= $horaire['heure_fin']) {
```

On prend l'heure du serveur et on la compare à la plage définie pour ce rôle :

- Si l'utilisateur est dans le bon créneau horaire, accès possible ;
- Sinon, accès refusé.

6. Enregistrement dans l'historique et gestion de l'accès

Cas accès autorisé :

```
$insert = $conn->prepare("INSERT INTO historique (id_badge, date, heure, statut) VALUES (:id_badge, CURDATE(), CURTIME(), 1)");
$insert->bindValue(':id_badge', $id_badge, PDO::PARAM_INT);
$insert->execute();

// Activation du Barionet
$barionet_ip = "172.22.21.164";
$urlOn = "http://$barionet_ip/rc.cgi?o=1,10";
file_get_contents($urlOn);

echo $dev_id.',LCDCLR,LCDSET;0;13;5;Badge valide,LCDSET;8;35;3;Acces autorise';
```

Enregistrement : On ajoute une entrée dans la table historique pour garder la trace d'un accès réussi (statut = 1).

Ouverture de la porte : On envoie une requête HTTP à un contrôleur Barionet pour activer le relais (ouvrir la porte).

Retour pour affichage : Un message de validation à destination de l'interface utilisateur (ex. écran LCD).

Cas accès refusé :

```
$insert = $conn->prepare("INSERT INTO historique (id_badge, date, heure, statut) VALUES (:id_badge, CURDATE(), CURTIME(), 0)");
$insert->bindValue(':id_badge', $id_badge, PDO::PARAM_INT);
$insert->execute();

echo $dev_id.',LCDCLR,LCDSET;0;13;4;Hors horaires,LCDSET;8;35;3;Acces refuse';
```

Enregistrement du refus : On garde la trace du refus dans l'historique.

Affichage utilisateur : Message d'accès refusé (hors horaires).



7.Cas du badge inconnu ou désactivé

```
$getId = $conn->prepare("SELECT id_badge FROM badge WHERE code_badge = :code_badge");
$getId->bindValue(':code_badge', $code_badge, PDO::PARAM_STR);
$getId->execute();
$id_badge = $getId->fetchColumn();
if ($id_badge) {
    $insert = $conn->prepare("INSERT INTO historique (id_badge, date, heure, statut) VALUES (:id_badge, CURDATE(), CURTIME(), 0)");
    $insert->bindValue(':id_badge', $id_badge, PDO::PARAM_INT);
    $insert->execute();
}
echo $dev_id.',LCDCLR,LCDSET;0;13;4;Badge invalide,LCDSET;8;35;3;Acces refuse';
```

Si le badge n'est pas valide ou désactivé :

- Si possible, on log l'événement dans l'historique (refus pour cause de badge invalide).
- Affichage d'un message « badge invalide » à l'utilisateur.

8.Gestion des erreurs de connexion à la base

```
66 } catch (PDOException $e) {
67     echo $dev_id.',LCDCLR,LCDSET;0;10;0;ERREUR BD';
68     error_log("Erreur de base de données : " . $e->getMessage());
69 }
70 ?>
```

Toute erreur lors de l'accès à la base déclenche :

- Un message d'alerte pour l'interface,
- Un enregistrement de l'erreur dans les logs du serveur (pour analyse et correction).

Ce script gère intégralement l'accès physique à un établissement en :

- Vérifiant le badge présenté,
- Identifiant l'utilisateur et son rôle,
- Contrôlant les horaires autorisés,
- Enregistrant toutes les tentatives dans un historique,
- Pilotant l'appareil d'ouverture de porte,
- Et en affichant le message adapté sur le terminal d'accès.

Son architecture modulaire (requêtes préparées, gestion centralisée des horaires, journalisation systématique) garantit robustesse, sécurité et évolutivité.



Affichage sur le lecteur





Bilan de ma contribution au projet

Participer à ce projet de gestion des accès par badge au sein du lycée a été une expérience extrêmement enrichissante, autant sur le plan technique que personnel.

Dès le départ, il a fallu comprendre le fonctionnement réel de l'établissement : comment les élèves, enseignants et agents circulent, quels sont les enjeux de sécurité ou de contrôle, et comment assurer une traçabilité efficace des accès. Cette phase d'échange avec les utilisateurs et d'analyse des besoins m'a permis de développer mon écoute, ma capacité à poser les bonnes questions et à formaliser les attentes réelles, et pas seulement les demandes exprimées.

La conception de la base de données a été une grande étape. J'ai dû mettre en pratique une vision globale : penser aux liens entre les différentes informations, prévoir les évolutions possibles, tout en restant simple et efficace. Ce travail m'a poussé à prendre du recul et à me projeter dans un usage quotidien du système, à la place des utilisateurs.

Sur le plan technique, développer le script de connexion en PHP m'a permis de plonger dans la programmation côté serveur, de comprendre l'importance de la sécurisation des données et de la centralisation des paramètres dans une vraie application.

J'ai appris que chaque détail compte, aussi bien pour la robustesse technique que pour la maintenabilité dans le temps.

Ce projet m'a également sensibilisé à l'importance du travail en équipe, chacun ayant son domaine d'intervention et devant collaborer pour que l'ensemble soit cohérent, fiable, et utile.

Ce que j'en retire aujourd'hui, c'est bien plus que des compétences techniques.

C'est une vraie méthodologie d'analyse, une logique de pensée "d'ingénieur des données" c'est à dire toujours relier la technique aux besoins humains et organisationnels, anticiper les usages, sécuriser, documenter.

C'est dans cet état d'esprit que je souhaite m'orienter vers le métier de Data Ingénieur. Pour être le lien entre la technique des données et la réalité des besoins, et contribuer à des solutions à la fois performantes, évolutives et utiles à tous.

En somme, cette expérience m'a appris à allier rigueur technique et écoute des besoins, à être force de proposition, et à poser les bases solides qui serviront à tout projet de gestion de données dans le futur.



Annexes

Script PHP de connexion à la base de données

```

1  <?php
2  // Paramètres de la base de données
3  $host = 'localhost';
4  $DB = 'acl';
5  $port = '3306';
6
7  // Définition du Data Source Name (DSN)
8  $dsn = 'mysql:host=' . $host . ';dbname=' . $DB . ';charset=utf8;port=' . $port;
9
10 // Définition des identifiants de connexion
11 $login = 'acl';
12 $mdp = 'ac1lp2i3!';
13
14 // Définition des options de PHP Data Objects (PDO)
15 $pdo_options = [
16     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
17     PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
18 ];
19
20 // Définir les constantes pour la connexion à la base de données
21 define('DB_DSN', $dsn);
22 define('DB_USER', $login);
23 define('DB_PASS', $mdp);
24 ?>

```

Script PHP de gestion des accès par badges selon les horaires

```

<?php
$code_badge = htmlspecialchars($_GET["UID"]);
$dev_id = htmlspecialchars($_GET["devID"]);
require '../config.php';

try {
    $conn = new PDO(DB_DSN, DB_USER, DB_PASS, $pdo_options);

    // 1. Vérification du badge + récupération utilisateur + rôle (changer
    b.id_utilisateur si besoin)
    $sql = "SELECT b.id_badge, u.id_utilisateur, u.nom, u.prenom, u.id_role
            FROM badge b
            JOIN utilisateur u ON b.id_utilisateur = u.id_utilisateur
            WHERE b.code_badge = :code_badge AND b.etat = 1";
    $reponse = $conn->prepare($sql);
    $reponse->bindValue(':code_badge', $code_badge, PDO::PARAM_STR);
    $reponse->execute();

    if ($reponse->rowCount() > 0) {
        $badge_info = $reponse->fetch(PDO::FETCH_ASSOC);
        $id_badge = $badge_info['id_badge'];
        $id_role = $badge_info['id_role'];
    }
}

```



```

// 2. Récupérer les horaires via id_role (clé étrangère)
$sqlHoraire = "SELECT heure_debut, heure_fin FROM role WHERE id_role =
:id_role";
$getHoraire = $conn->prepare($sqlHoraire);
$getHoraire->bindValue(':id_role', $id_role, PDO::PARAM_INT);
$getHoraire->execute();
$horaire = $getHoraire->fetch(PDO::FETCH_ASSOC);

$heureActuelle = date("H:i:s");

// 3. Vérifier l'accès horaire
if ($horaire && $heureActuelle >= $horaire['heure_debut'] && $heureAc-
tuelle <= $horaire['heure_fin']) {
    // Insertion historique (statut accepté)
    $insert = $conn->prepare("INSERT INTO historique (id badge, date,
heure, statut) VALUES (:id badge, CURDATE(), CURTIME(), 1)");
    $insert->bindValue(':id badge', $id badge, PDO::PARAM_INT);
    $insert->execute();

    // Activation du Barionet
    $barionet_ip = "172.22.21.164";
    $urlOn = "http://$barionet_ip/rc.cgi?o=1,10";
    file_get_contents($urlOn);

    echo $dev_id.',LCDCLR,LCDSET;0;13;5;Badge valide,LCDSET;8;35;3;Ac-
ces autorise';
} else {
    // Insertion historique (statut refusé - hors horaires)
    $insert = $conn->prepare("INSERT INTO historique (id badge, date,
heure, statut) VALUES (:id badge, CURDATE(), CURTIME(), 0)");
    $insert->bindValue(':id badge', $id badge, PDO::PARAM_INT);
    $insert->execute();

    echo $dev_id.',LCDCLR,LCDSET;0;13;4;Hors ho-
raires,LCDSET;8;35;3;Acces refuse';
}
} else {
    // Badge inconnu ou désactivé (code comme avant)
    $getId = $conn->prepare("SELECT id badge FROM badge WHERE code badge =
:code badge");
    $getId->bindValue(':code badge', $code badge, PDO::PARAM_STR);
    $getId->execute();
    $id badge = $getId->fetchColumn();
    if ($id badge) {
        $insert = $conn->prepare("INSERT INTO historique (id badge, date,
heure, statut) VALUES (:id badge, CURDATE(), CURTIME(), 0)");
        $insert->bindValue(':id badge', $id badge, PDO::PARAM_INT);

```




```
$insert->execute();  
}  
echo $dev id.',LCDCLR,LCDSET;0;13;4;Badge invalide,LCDSET;8;35;3;Acces  
refuse';  
}  
} catch (PDOException $e) {  
    echo $dev id.',LCDCLR,LCDSET;0;10;0;ERREUR BD';  
    error log("Erreur de base de données : " . $e->getMessage());  
}  
?>
```

Matériel à disposition

