25/10/2025

**Rapport Projet AP4A**

**Automne 2025**

**Simulation d’un système de badges et lecteurs de badges**

Journet Célia

Table of Contents

[**INTRODUCTION** 1](#_Toc212454736)

[**Besoins du Projet** 2](#_Toc212454737)

[**Description Fonctionnelle** 2](#_Toc212454738)

[Liste des fonctionnalités exigées 2](#_Toc212454739)

[Spécialisation des classes, d’après le sujet : 3](#_Toc212454740)

[**Règles de Gestion** 6](#_Toc212454741)

[**Conception du Projet** 8](#_Toc212454742)

# **INTRODUCTION**

La sécurisation des accès dans un campus universitaire, où un grand nombre de personnes aux accès variés se mélangent, est une nécessité. L'objectif était de concevoir et développer en C++ un simulateur de contrôle d'accès modélisant un système de badges et de lecteurs de badges.

Ce projet m’a permis de mettre en application les principes de la programmation orientée objet (POO) pour créer une architecture modulaire, simulant les interactions entre un serveur central, des lecteurs de badges, des badges et des utilisateurs, le tout centralisé dans un ordonnanceur. Les défis principaux ont résidé dans la modélisation fine des droits d'accès et dans l'ordonnancement des événements sur une journée type. Les résultats des transactions sont tracés via une console et enregistrés dans des fichiers de logs.

Ce rapport détaillera dans un premier temps l'analyse et la conception du système. Puis, nous présenterons les choix d'implémentation des classes principales et les algorithmes de gestion des accès. Enfin, nous validerons le fonctionnement du simulateur à travers ses résultats et proposerons une analyse critique ainsi que des pistes d'amélioration.

# **Besoins du Projet**

## **Description Fonctionnelle**

### Liste des fonctionnalités exigées

**Fonctionnalités principales :**

* Gestion des**personnels** (étudiants, enseignants, sécurité, etc.) avec leurs**badges nominatifs**
* Gestion des**lecteurs de badges**associés à des portes (bâtiments, salles)
* **Contrôle d'accès**avec droits variables selon le type de personnel
* **Simulation temporelle**d'une journée type via un ordonnanceur

**Traitement des accès :**

* **Workflow :** Badge présenté → Lecteur → Serveur → Vérification droits → Autorisation/Refus
* **Service central :**demandeAcces(Badge, LecteurBadge) avec validation des droits
* **Configuration**des droits via fichier chargé au démarrage

**Contraintes techniques :**

* **Langage :** C++ orienté objet
* **Communication**par appels de méthodes
* **Compilation multi-plateforme**(CMakeLists.txt ou .pro)
* **Mécanismes C++ avancés :** héritage, polymorphisme, STL, etc.

**Sorties et visualisation :**

* **Journalisation :** Fichier de logs avec les demandes et résultat
* **Visualisation console**
* **Informations tracées :** heure, nom, prénom, statut, type et lieu du lecteur, résultat

### Spécialisation des classes, d’après le sujet :

N’ayant pas de diagramme UML de conception, la structure détaillée des classes et leurs responsabilités sont décrites ci-dessous.

**Classe Badge :**

* Chaque badge doit être **nominatif** et associé de manière unique à une personne
* Le badge doit contenir les **informations d'identification** de la personne (nom, statut, identifiant unique)
* Le badge doit permettre de **récupérer les droits d'accès** de la personne via des méthodes appropriées
* Les badges peuvent être organisés selon une **hiérarchie d'héritage** selon le type d'accès autorisé
* Le badge doit pouvoir **initier une demande d'accès** lorsqu'il est présenté sur un lecteur

**Classe Lecteur de Badge :**

* Les lecteurs doivent être **associés à des portes spécifiques** (bâtiments ou salles de travail)
* La classe LecteurBadge doit servir de **classe mère** pour différents types de lecteurs
* Les lecteurs doivent pouvoir **communiquer avec le serveur** pour transmettre les demandes d'accès
* Chaque lecteur doit avoir une **localisation spécifique** (bâtiment, salle, zone)
* Les lecteurs doivent pouvoir **détecter la présentation d'un badge** et initier la procédure de vérification

**Classe Serveur :**

**Rôle principal :**

* **Cœur du système** de contrôle d'accès
* **Centralise** la gestion des droits et autorisations
* **Coordonne** la communication entre badges et lecteurs

**Fonctionnalités clés :**

* **Service central** : demandeAcces(Badge, LecteurBadge) pour traiter toutes les requêtes
* **Vérification des droits** selon le type de personnel et la localisation
* **Gestion de configuration** via fichier chargé au démarrage
* **Décision d'accès** : autorisation ou refus avec justification

**Traitement des demandes :**

* **Réception** des demandes depuis les lecteurs
* **Analyse** des informations badge + lecteur
* **Validation** des droits d'accès
* **Retour** de la décision au lecteur
* **Journalisation** systématique du résultat

**Caractéristiques techniques :**

* **Mode passif** : répond aux requêtes externes
* **Utilise le polymorphisme** pour gérer les différents types
* **Connaissance complète** des objets du système

**Classe Scheduler - Description Fonctionnelle**

**Rôle principal :**

* **Moteur de la simulation** temporelle
* **Ordonnance** le déroulement des événements
* **Anime** l'ensemble du système sur une base de temps

**Fonctionnalités clés :**

* **Gestion du temps** simulation d'une "journée type"
* **Séquencement** des demandes d'accès
* **Activation** des processus aux moments appropriés
* **Coordination** globale de la simulation

**Processus de simulation :**

* **Initialisation** de la séquence temporelle
* **Déclenchement** des présentations de badges
* **Organisation** du flux d'événements
* **Gestion** de la fréquence et répartition des accès

**Caractéristiques techniques :**

* **Boucle d'ordonnancement** principale
* **Base de temps** prédéfinie et configurable
* **Séquence d'appels** coordonnée entre objets
* **Flexibilité** dans l'organisation des appels

La différence de développement entre Badge/Lecteur de Badge et Serveur/Scheduler s’explique par le fait que le Serveur et le Scheduler sont des éléments centraux et nécessitent une logique métier élaborée tandis que les Badges et Lecteurs ont des rôles plus spécialisés et limités

## **Règles de Gestion**

Voici toutes les règles émises par le sujet. Ici nous ne détaillerons pas ce qui a été fait ou non, ceci sera fait plus tard.

**Gestion des droits d'accès :**

* Les droits d'accès et priorités diffèrent selon le type de personnel
* Un même personnel peut disposer de droits d'accès multiples
* La configuration des droits d'accès est centralisée au niveau du serveur
* Les droits doivent pouvoir être chargés via un fichier de description au démarrage

**Hiérarchie des personnels :**

* **Étudiants** : droits d'accès basiques selon leur filière
* **Enseignants** : droits étendus, pouvant inclure des accès de type sécurité
* **Personnel de sécurité** : droits étendus avec priorités élevées
* **Personnel administratif** : droits limités aux zones administratives
* **Enseignants-chercheurs** : droits combinés (enseignement + accès spécifique au laboratoire)

**Gestion des priorités :**

* Les personnels de sécurité disposent de la priorité la plus élevée
* Les enseignants et enseignants-chercheurs ont une priorité standard
* Les étudiants et personnels administratifs ont une priorité basique

**Processus d'accès :**

* Une demande d'accès est initiée par le scan d'un badge sur un lecteur
* Le serveur analyse chaque demande et vérifie les droits correspondants
* Les autorisations d'accès sont accordées ou refusées en fonction des droits
* Les portes d'entrée s'ouvrent uniquement après autorisation du serveur

**Journalisation :**

* Toutes les tentatives d'accès doivent être journalisées
* Les accès autorisés et les tentatives infructueuses sont enregistrés dans des fichiers séparés
* Les logs doivent contenir : heure d'accès, nom de la personne, statut, résultat

# **Conception du Projet**

### Diagramme de classes UML :

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 1 : Diagramme de classes UML

Le diagramme pourra être retrouvé dans le dossier envoyé sous le nom : « UML\_AP4A\_TP1B\_Celia\_Journet.png ».

# **Annexe**