Système de gestion de présence MIT/MISA : Description Détaillée

L3-MIT 2025

Herilala RAKOTO

Iantso Christian RAZAFINDRAZAKA

Diary Santatrasoa HERIMAMPIONONA

Brice Privat NARIVELO

1. Introduction générale

1.a Présentation du système ou de l'application :

Le système modélisé correspond à une application de gestion de présence, spécifiquement conçue pour répondre aux besoins d'un environnement éducatif ou institutionnel. Cette application vise à automatiser et à optimiser le suivi des présences des étudiants, ainsi que d'autres utilisateurs tels que les enseignants ou les délégués, dans divers contextes comme les cours, les examens ou les activités extrascolaires. Elle repose sur une combinaison de technologies modernes, notamment la reconnaissance faciale pour une identification rapide, le scan de QR codes pour une prise de présence simplifiée, une validation manuelle pour les cas spécifiques, ainsi qu'une fonctionnalité de consultation d'historique et de génération de rapports détaillés. L'application est pensée pour être intuitive, sécurisée et adaptable aux exigences des utilisateurs, tout en offrant une interface conviviale et des outils permettant une gestion efficace des données de présence.

1.b Objectif de la modélisation UML :

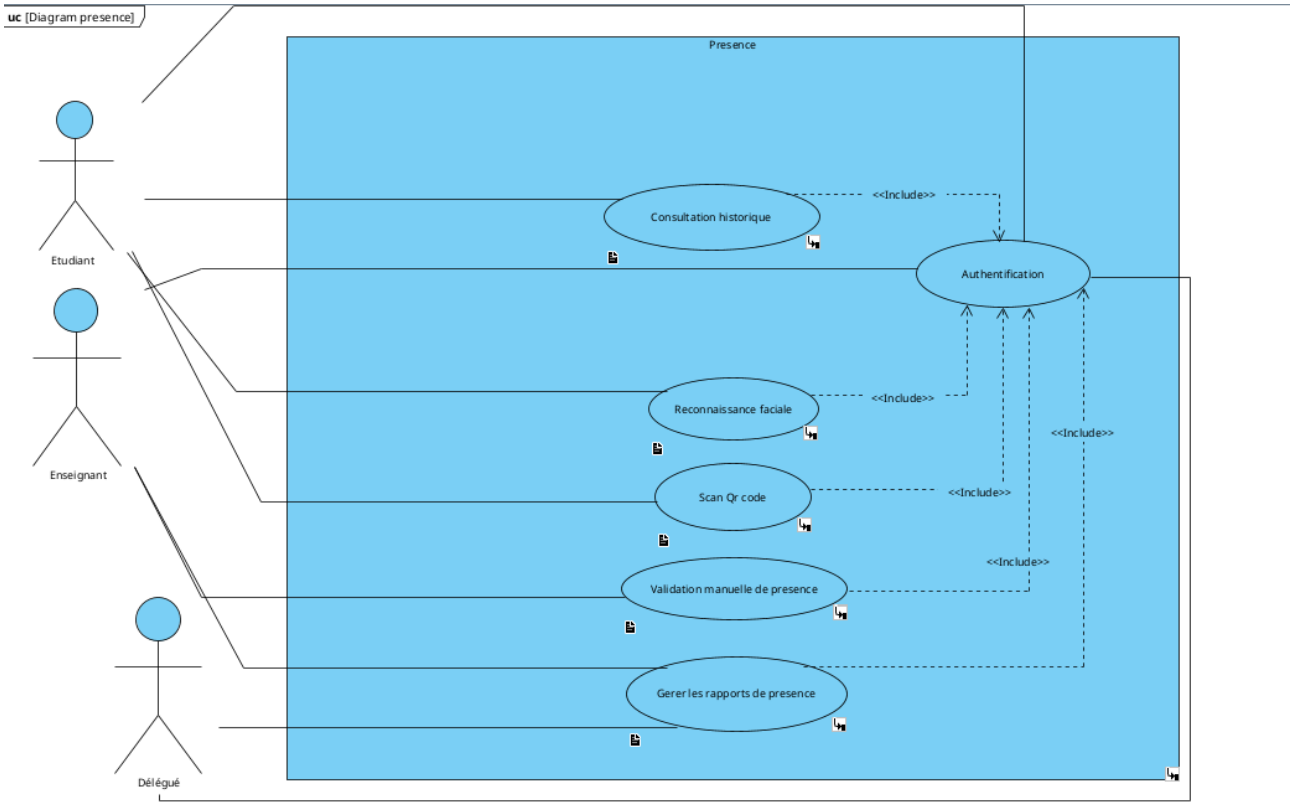
La modélisation UML (Unified Modeling Language) a pour principal objectif de fournir une représentation visuelle et structurée du système, permettant de clarifier ses fonctionnalités et ses composants. Elle sert de base à la conception en mettant en évidence les interactions entre les différents acteurs et le système à travers le diagramme de cas d'utilisation, tout en détaillant la structure interne avec le diagramme de classes, qui inclut les attributs, méthodes et relations entre les entités. Cette approche facilite la communication entre les développeurs, les analystes et les parties prenantes, réduit les ambiguïtés dans les spécifications, et constitue un guide solide pour le développement, la maintenance et l'évolution future de l'application. Elle permet également d'identifier les besoins fonctionnels et de s'assurer que toutes les exigences sont bien prises en compte avant la phase de codage.

1.c Portée des diagrammes présentés :

Les diagrammes présentés dans cette analyse couvrent l'ensemble des interactions principales entre les acteurs identifiés (Étudiant, Enseignant, Délégué) et les fonctionnalités offertes par le système, comme illustré dans le diagramme de cas d'utilisation. Ce dernier met en lumière les cas d'utilisation spécifiques tels que la consultation de l'historique, la reconnaissance faciale, le scan de QR code, la validation manuelle et la gestion des rapports de présence, avec leurs relations d'inclusion. Par ailleurs, le diagramme de classes offre une vue détaillée de la structure du système, en montrant les classes principales (Utilisateur, Historique, Présence, Contenu, etc.), leurs attributs (comme l'identifiant, le rôle ou la date), leurs méthodes (comme consulter ou générer un rapport), et les associations entre elles (comme la relation un-à-plusieurs entre Utilisateur et Historique). Cette

portée englobe ainsi les aspects fonctionnels et structurels essentiels pour une compréhension complète du système.

2. Diagramme de cas d’utilisation



2.a Présentation globale des cas d’utilisation (liste et vue d'ensemble) :

Le diagramme de cas d’utilisation offre une vue d’ensemble des interactions entre les acteurs et les fonctionnalités principales du système de gestion de présence. Les acteurs identifiés incluent **l’Étudiant, l’Enseignant et le Délégué,** chacun jouant un rôle spécifique dans l’utilisation de l’application.

Les cas d’utilisation principaux sont les suivants :

Consultation historique, permettant aux utilisateurs de visualiser l’historique des présences ;**Reconnaissance faciale**, utilisée pour une identification automatique des individus ; **Scan QR code**, offrant une méthode rapide pour enregistrer une présence ; **Validation manuelle de présence**, permettant une vérification manuelle dans des cas particuliers ; et **Gérer les rapports de présence**, qui permet de générer et de gérer des rapports détaillés.

Ces cas sont organisés autour du système "Présence", avec des relations d’inclusion (« include ») reliant chaque cas d’utilisation à **l’authentification**, soulignant que cette dernière est une

étape préalable essentielle pour accéder aux fonctionnalités. Cette vue d'ensemble illustre comment les acteurs interagissent avec le système de manière cohérente et structurée pour répondre aux besoins de suivi et de gestion des présences.

2.b Description détaillé

2.b.1 Consultation historique :

◦ **Nom du cas d'utilisation :** Consultation historique

◦ **Acteurs impliqués :** Étudiant

◦ **But / Description succincte :** Permettre à l'étudiant d'accéder et de visualiser l'historique des présences enregistrées pour son propre compte, afin de suivre ses données de présence passées.

◦ **Préconditions / Postconditions :**

- **Préconditions :** L'étudiant doit être authentifié avec succès dans le système, et les données d'historique correspondant à son profil doivent être disponibles dans la base de données.
- **Postconditions :** L'historique des présences de l'étudiant est affiché, et aucune modification des données n'est effectuée à moins d'une action supplémentaire autorisée.

◦ **Déclencheur (Trigger) :** La demande explicite de l'étudiant de consulter son historique via l'interface de l'application (par exemple, en sélectionnant une option dédiée dans le menu).

◦ **Résumé du scénario principal :** L'étudiant s'authentifie dans le système, accède à l'interface de consultation, sélectionne l'option "Historique", et le système récupère et affiche les données d'historique (dates, statuts, etc.) spécifiques à son profil en temps réel.

◦ **Scénarios alternatifs ou exceptions :**

- Si les données d'historique de l'étudiant sont indisponibles ou incomplètes, un message d'erreur est affiché, et l'étudiant est invité à contacter un administrateur.
- En cas d'absence d'authentification ou si l'utilisateur n'est pas un étudiant, l'accès est refusé, et l'utilisateur est redirigé vers la page de connexion ou une page d'erreur.
- Si une erreur survient lors de la récupération des données (par exemple, problème avec le système), le processus est interrompu, et un message d'alerte est présenté.

2.b.2 Gérer les rapports de présence :

◦ **Nom du cas d'utilisation :** Gérer les rapports de présence

◦ **Acteurs impliqués : Enseignants** et le **Délégués**

◦ **But / Description succincte :** Permettre à l'enseignant ou au délégué de générer, consulter et exporter des rapports de présence pour une période ou un cours spécifique, afin de suivre et d'analyser les données de présence des étudiants.

◦ **Préconditions / Postconditions :**

- **Préconditions :** L’utilisateur (enseignant ou délégué) doit être authentifié avec succès dans le système, et les données de présence doivent être disponibles dans la base de données.
- **Postconditions :** Un rapport de présence est généré et peut être affiché ou exporté avec succès, ou un message d’erreur est affiché en cas d’échec.

◦ **Déclencheur (Trigger) :** La demande explicite de l’utilisateur de gérer un rapport via l’interface de l’application qui affiche le tableau de données filtré suivant deux périodes.

◦ **Résumé du scénario principal :** L’utilisateur s’authentifie, accède à l’interface, sélectionne l’option "Historique", choisit une période ou un cours, le système récupère les données de présence (demande de génération, récupération des données, calcul des statistiques, consolidation), génère le rapport, et propose son téléchargement ou affichage.

◦ **(Optionnel) Scénarios alternatifs ou exceptions :**

- Si l’authentification échoue, le système affiche un message d’erreur et bloque l’accès.
- Si aucune donnée n’est disponible pour la période ou le cours sélectionné, le système affiche un message indiquant l’absence de données.
- En cas de conflit dans les données (par exemple, incohérences), le système peut générer un rapport partiel ou signaler le problème à l’utilisateur.
- Si une erreur survient pendant la génération, le processus est interrompu, et un message d’échec est affiché.

2.b.3 Reconnaissance faciale :

◦ **Nom du cas d’utilisation :** Reconnaissance faciale

◦ **Acteurs impliqués :** Étudiant

◦ **But / Description succincte :** Permettre à l’utilisateur de s’authentifier ou de vérifier son identité via une analyse biométrique faciale, en capturant une image et en la comparant aux données biométriques enregistrées.

◦ **Préconditions / Postconditions :**

- **Préconditions :** L’utilisateur doit être dans un environnement permettant la capture d’image (caméra disponible), et ses données biométriques doivent être préalablement enregistrées dans le système.
- **Postconditions :** L’identité de l’utilisateur est confirmée avec succès (succès) ou un échec est signalé (échec/retry/failed), selon le résultat de la comparaison.

◦ **Déclencheur (Trigger) :** La demande explicite de l’utilisateur d’initier une reconnaissance faciale via l’interface de l’application (par exemple, en sélectionnant une option d’authentification biométrique).

◦ **Résumé du scénario principal :** L’utilisateur demande une reconnaissance faciale, l’interface capture une image (jusqu’à 5 tentatives maximum), le système compare les données d’image avec les données biométriques enregistrées, et renvoie un résultat : succès si le score de comparaison est ≥85%, ou un échec nécessitant un repositionnement si le score est <85%, avec une limite de tentatives avant basculement vers une méthode alternative.

◦ **(Optionnel) Scénarios alternatifs ou exceptions :**

- Si le score de comparaison est <85% après la première tentative, le système demande un repositionnement et retente la capture.
- Si aucune correspondance n’est trouvée après plusieurs tentatives, le système renvoie un résultat "no match" et bloque l’accès.
- Après 5 tentatives infructueuses, le système propose une méthode alternative d’authentification.
- En cas d’échec technique (caméra défectueuse), un message d’erreur est affiché.

2.b.4 Scan QR code :

◦ **Nom du cas d’utilisation :** Scan QR code

◦ **Acteurs impliqués : Étudiant**

◦ **But / Description succincte :** Permettre à l’étudiant de scanner un QR code via l’application pour valider et enregistrer sa présence de manière rapide et sécurisée.

◦ **Préconditions / Postconditions :**

- **Préconditions :** L’étudiant doit avoir ouvert l’application et disposé d’un QR code valide généré par le système.
- **Postconditions :** La présence de l’étudiant est enregistrée avec succès si le QR code est valide, ou un message d’erreur est affiché si le QR code est invalide.

◦ **Déclencheur (Trigger) :** L’action de l’étudiant d’ouvrir l’application et de scanner un QR code via l’interface.

◦ **Résumé du scénario principal :** L’étudiant ouvre l’application, scanne un QR code, le système vérifie sa validité (y compris la validation de la date d’expiration), et enregistre la présence avec une notification de succès si le QR code est valide, ou affiche un message d’erreur si le QR code est invalide.

◦ **(Optionnel) Scénarios alternatifs ou exceptions :**

- Si le QR code est invalide ou expiré, le système renvoie un message d’erreur et ne procède pas à l’enregistrement.
- En cas de problème technique lors du scan (par exemple, caméra défectueuse), un message d’erreur est affiché, et l’étudiant est invité à réessayer.
- Si plusieurs tentatives échouent, le système peut bloquer l’accès temporairement et suggérer une assistance.

2.b.5 Validation manuelle de présence :

◦ **Nom du cas d’utilisation :** Validation manuelle de présence

◦ **Acteurs impliqués : Enseignant**

◦ **But / Description succincte :** Permettre à l’enseignant de valider manuellement la présence ou l’absence d’un étudiant pour un cours spécifique, en s’assurant que l’identité est confirmée et les données enregistrées correctement.

◦ **Préconditions / Postconditions :**

- **Préconditions :** L’enseignant doit être authentifié avec succès dans le système, et la liste des étudiants pour le cours sélectionné doit être disponible.
- **Postconditions :** La présence ou l’absence de l’étudiant est enregistrée avec succès, ou un message d’erreur est affiché si l’étudiant est inconnu ou si une erreur survient.

◦ **Déclencheur (Trigger) :** La demande explicite de l’enseignant de valider une présence via l’interface de l’application (par exemple, en sélectionnant un cours et un étudiant).

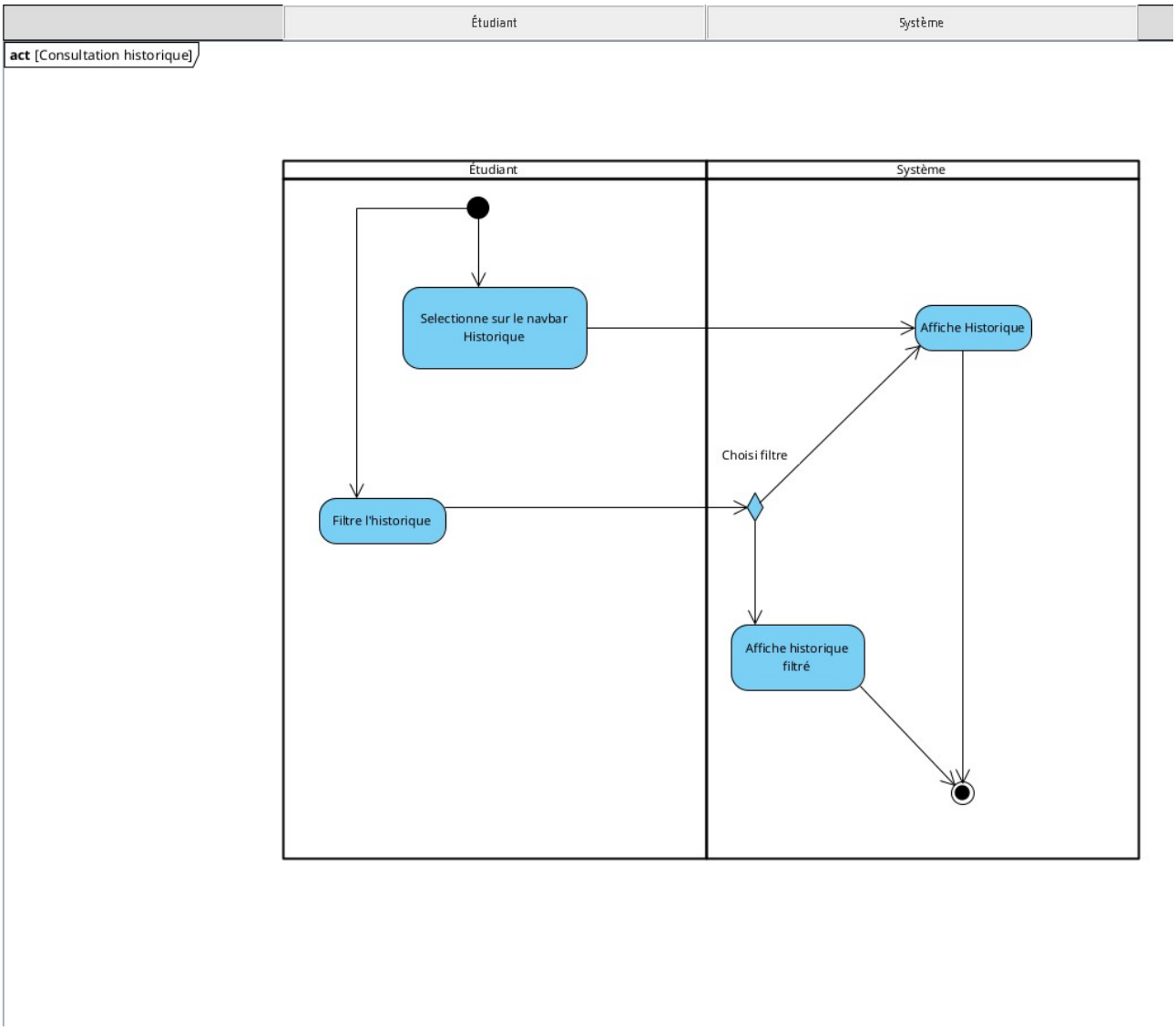
◦ **Résumé du scénario principal :** L’enseignant s’authentifie, sélectionne un cours, le système affiche la liste des étudiants, l’enseignant choisit un étudiant et marque sa présence ou absence, le système vérifie la validité des données et enregistre la présence, ou affiche un message d’erreur si l’étudiant est inconnu.

◦ **(Optionnel) Scénarios alternatifs ou exceptions :**

- Si l’authentification échoue, le système affiche un message d’erreur et bloque l’accès.
- Si l’étudiant sélectionné est inconnu, le système affiche un message d’erreur et ne procède pas à l’enregistrement.
- En cas de conflit (par exemple, présence déjà validée), le système peut demander une confirmation ou ignorer la nouvelle entrée.
- Si une erreur technique survient lors de l’enregistrement, un message d’échec est affiché, et l’opération peut être répétée.

3. Diagrammes d’activités :

3.a Consultation historique



3.a.a Objectif du diagramme :

Décrire le déroulement du cas d'utilisation permettant à un étudiant de consulter son historique de présence, en mettant en évidence les étapes de sélection, de filtrage et d’affichage des données**.**

3.a.b Description de chaque étape importante :

- **Sélectionne sur la barre Historique :** L’étudiant initie le processus en sélectionnant l’option "Historique" dans l’interface, marquant le point de départ de la consultation.
- **Filtre Historique :** L’étudiant peut choisir d’appliquer un filtre pour affiner les données (par exemple, par date ou période), offrant une personnalisation de l’affichage.
- **Choisir Filtre :** Une décision est prise pour déterminer si un filtre est appliqué (oui/non), influençant la suite du processus.
- **Affiche Historique Filtré :** Si un filtre est choisi, le système affiche les données historiques filtrées, sinon il affiche l’historique complet.

- **Affiche Historique :** Le système présente l’historique des présences de l’étudiant, constituant l’objectif final de cette interaction.

3.d.c Rôle des décisions, boucles, synchronisations, etc. :

- **Décision (Choisir Filtre) :** Cette étape agit comme un point de bifurcation, permettant à l’étudiant de décider d’utiliser un filtre ou non, ce qui détermine si l’affichage est filtré ou non.
- **Synchronisation :** Le diagramme implique une synchronisation implicite entre l’action de l’étudiant (sélection/filtrage) et la réponse du système (affichage), assurant que chaque étape suit l’autre de manière séquentielle.
- **Absence de boucles :** Aucun mécanisme de boucle n’est explicitement représenté, indiquant que le processus est linéaire avec une seule itération par consultation.

3.b Reconnaissance faciale

3.b.a Objectif du diagramme

Ce diagramme d'activités illustre le déroulement dynamique du cas d'utilisation « Enregistrement de la présence par reconnaissance faciale ».
Il met en évidence les interactions entre l'étudiant et le système de contrôle de présence, via un processus de reconnaissance biométrique (faciale), et détaille le comportement du système face à la validation ou à l'échec de cette reconnaissance.

3.b.b Description de chaque étape importante

1. L'étudiant initialise le processus

- L'étudiant active son appareil (mobile, borne ou ordinateur) pour lancer la procédure de reconnaissance.
- Il entre dans l'interface utilisateur dédiée à la reconnaissance faciale.

2. Transmission de la demande au système

- L'interface envoie une demande de scan biométrique au système central.

3. Le système capture l'image

- Le système utilise la caméra pour prendre une photo du visage de l'étudiant.
- Cette étape est critique car elle déclenche la suite de la vérification.

4. Vérification des données biométriques

- Le système analyse l'image et compare les données biométriques extraites avec celles déjà enregistrées pour cet utilisateur.
- Une décision conditionnelle est prise à ce niveau.

5a. Si les données sont valides

- Le système confirme l'identité de l'étudiant.
- Il enregistre l'étudiant comme présent dans le registre.
- Le processus prend fin.

5b. Si les données sont invalides

- Le système signale l'échec de la reconnaissance.
- Il demande une nouvelle tentative (retour à l'étudiant pour un nouveau scan).
- Le processus boucle jusqu'à ce que les données soient valides ou qu'un seuil d'échec soit éventuellement atteint (non représenté ici).

3.b.c Rôle des structures de contrôle

Décisions (nœud de branchement)

- Le losange représente une condition : « Données biométriques valides ? »
- Deux branches sont définies :
  - Oui → enregistrement de la présence.
  - Non → nouvelle tentative de scan.

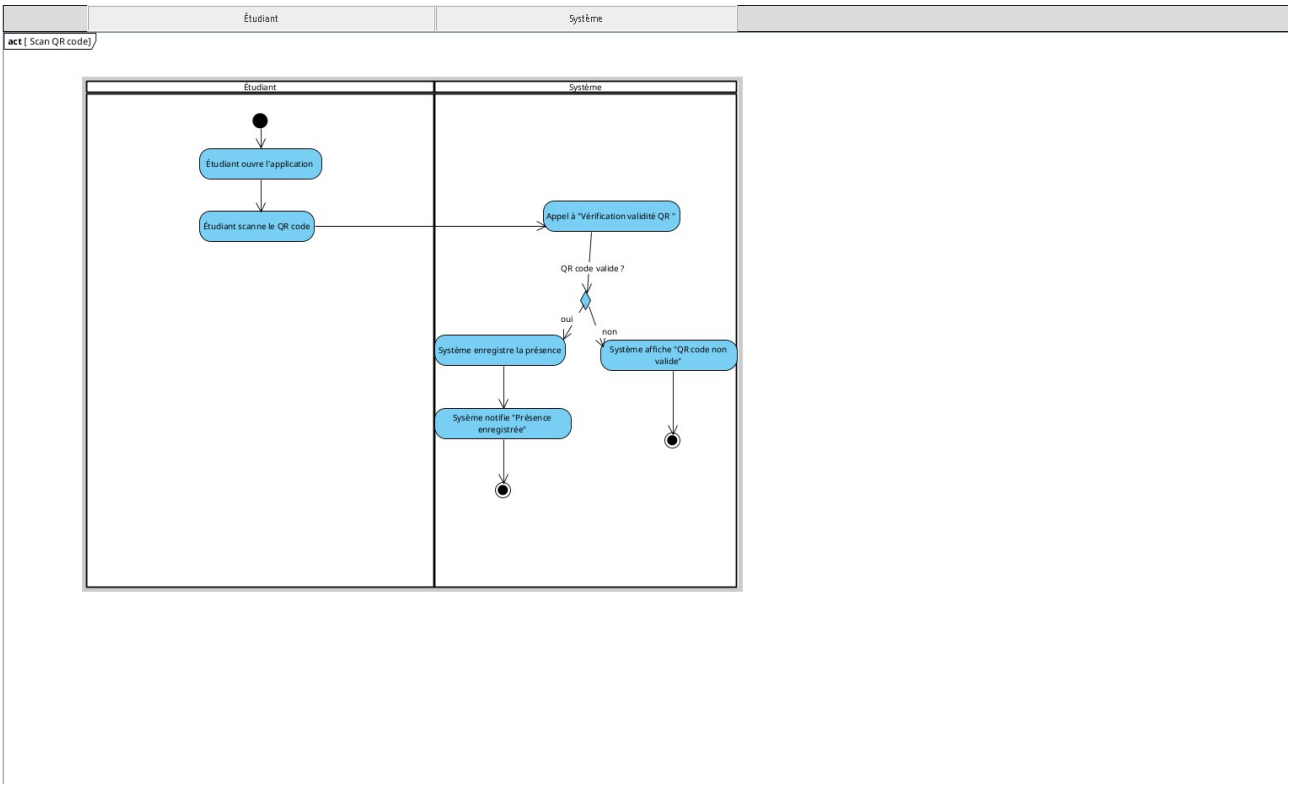
Boucle

- Implémentée implicitement par le retour du système vers l'étudiant si la reconnaissance échoue.
- Cela permet à l'utilisateur de refaire un scan autant de fois que nécessaire, jusqu'à validation.

Synchronisation

- Ce diagramme ne comporte pas de synchronisation parallèle (fork ou join) car le processus est séquentiel : une seule branche active à la fois (de l'étudiant vers le système, puis retour éventuel).

3.c Scan QR code



3.c.a Objectif du diagramme

Ce diagramme d'activités décrit le déroulement du cas d'utilisation « Enregistrement de la présence par scan de QR code ».
Il met en évidence les étapes de l'interaction entre l'étudiant et le système, depuis l'ouverture de l'application jusqu'à la validation (ou non) du QR code, et l'enregistrement de la présence.

3.c.b Description de chaque étape importante

1. L'étudiant lance l'application

- L'étudiant accède à l'application de présence via son appareil personnel.

2. Scan du QR code

- L'étudiant scanne le QR code fourni (par exemple affiché sur un support physique ou écran de l'établissement).

3. Envoi au système pour vérification

- Le système reçoit le QR code scanné.
- Il procède à une vérification de sa validité (vérification d'authenticité, de correspondance avec la séance, de la date/heure, etc.).

4. Décision sur la validité du QR code

- Si le QR code est valide :
    - Le système enregistre la présence de l'étudiant.

    - Il notifie que la présence a été enregistrée.
- Si le QR code n'est pas valide :
    - Le système affiche un message d'erreur indiquant que le QR code est invalide.

3.c.c Rôle des structures de contrôle

Décisions

- Le diagramme contient une condition sous forme de losange : « QR code valide ? »
- Deux chemins sont définis :
    - Oui → enregistrement et notification de la présence.
    - Non → affichage d'un message d'erreur.

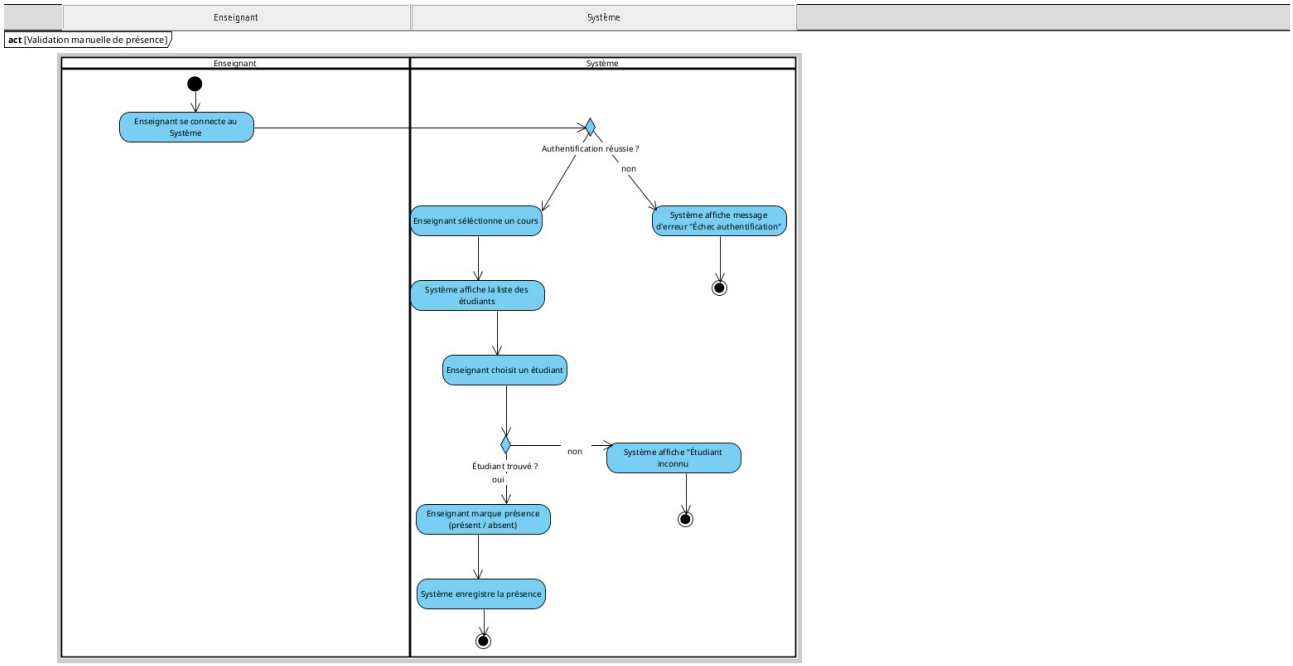
Boucles

- Aucune boucle explicite n'est prévue dans ce diagramme. Le processus est linéaire, sans répétition.

Synchronisation

- Il n'y a pas de parallélisme dans ce processus. Toutes les activités sont séquentielles.

3 . d Validation manuelle de presence



3.d.a Objectif du diagramme

Ce diagramme d'activités décrit le déroulement du cas d'utilisation « Validation manuelle de la présence ».
Il illustre les interactions entre l'enseignant et le système pour marquer manuellement la présence ou l'absence d'un étudiant.

3.d.b Description de chaque étape importante

1. Connexion de l'enseignant

- L'enseignant accède au système et procède à l'authentification.

2. Vérification d'identité

- Le système vérifie si l'authentification est réussie.
    - Si oui, l'enseignant accède à ses cours.
    - Si non, un message d'erreur s'affiche et le processus s'arrête.

3. Sélection du cours

- L'enseignant choisit un cours dans la liste disponible.
- Le système affiche alors la liste des étudiants inscrits.

4. Choix de l'étudiant

- L'enseignant sélectionne un étudiant spécifique.
- Le système vérifie si l'étudiant existe dans la liste :
    - Si oui, il peut continuer.
    - Si non, un message "Étudiant inconnu" s'affiche et l'activité s'arrête.

5. Marquage de la présence

- L'enseignant indique si l'étudiant est présent ou absent.
- Le système enregistre cette information dans la base de données.

3.d.c Rôle des structures de contrôle

Décisions

- Deux points de décision sont présents :
    - Authentification réussie ?
    - Étudiant trouvé ?
- Ces décisions permettent de gérer les erreurs (connexion ou identité inconnue).

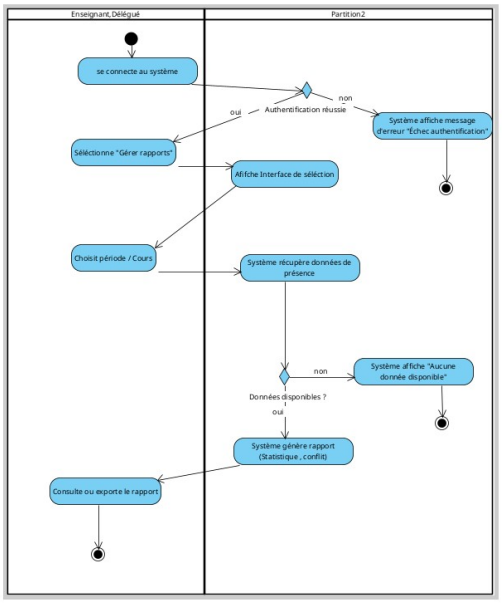
Boucles

- Le diagramme ne montre pas explicitement de boucle, mais on peut imaginer qu'un enseignant puisse valider la présence de plusieurs étudiants à la suite (boucle implicite possible en dehors de ce diagramme).

Synchronisation

- Les activités sont séquentielles. Aucun traitement parallèle n'est prévu.

3. e Gerer les rapports de presence :



3.e.a Objectif du diagramme :

Décrire le déroulement du cas d'utilisation permettant à un enseignant ou un délégué de gérer la génération, la consultation et l’exportation de rapports de présence, en tenant compte de l’authentification et des options de traitement des données.

3.e.b Description de chaque étape importante :

- **Se connecte au système :** L’enseignant ou le délégué initie le processus en se connectant au système, marquant le point de départ.
- **Sélectionne "Gérer rapports" :** L’utilisateur choisit l’option de gestion des rapports dans l’interface.
- **Authentification réussie :** Une décision vérifie si l’authentification est réussie ; si non, le système affiche un message d’erreur.
- **Affiche Interface de sélection :** Si l’authentification réussit, le système affiche une interface permettant de sélectionner les critères du rapport.
- **Choisit période / Cours :** L’utilisateur sélectionne une période ou un cours spécifique pour générer le rapport.
- **Système récupère données de présence :** Le système collecte les données de présence correspondantes.

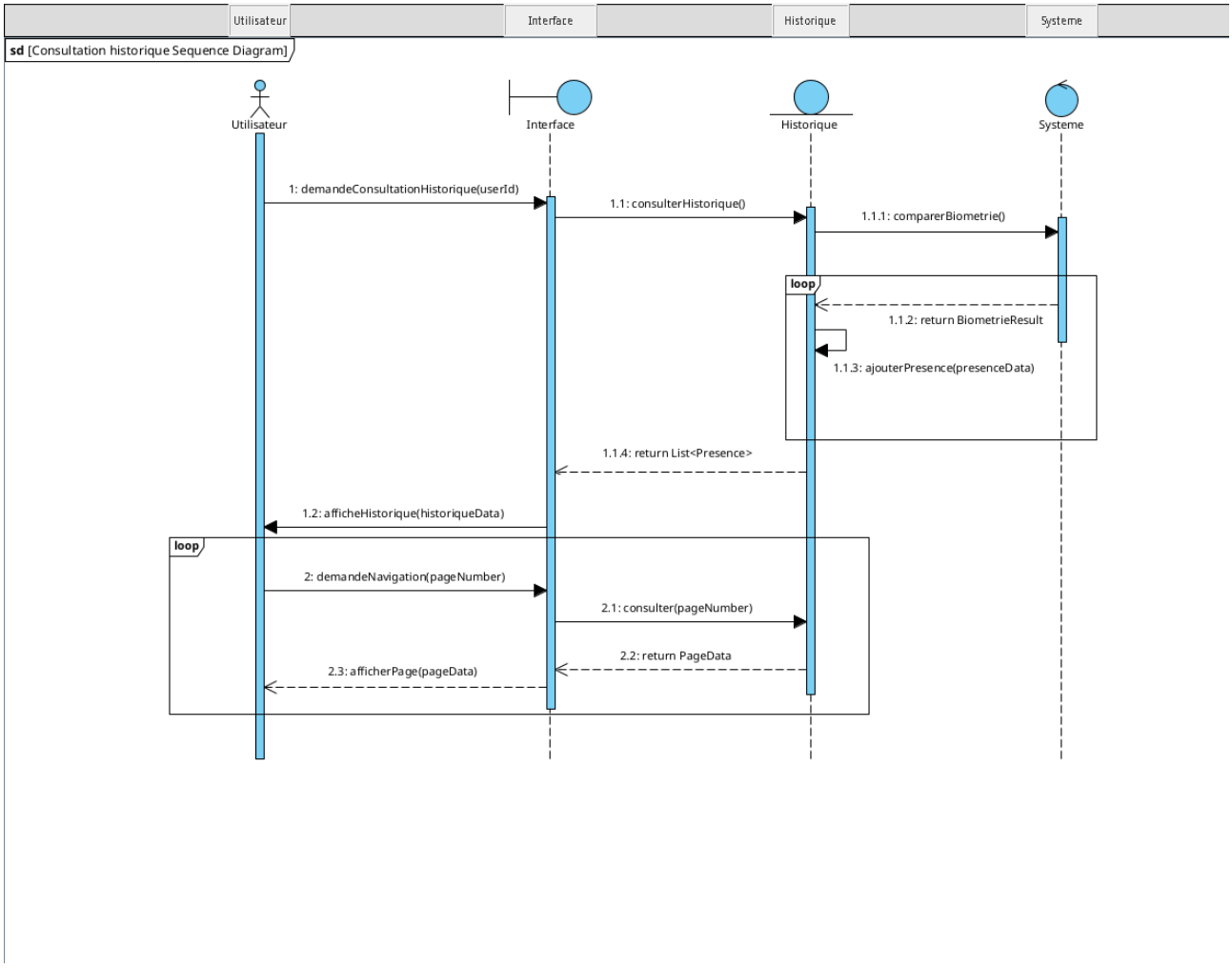
- **Données disponibles ? :** Une décision détermine si des données sont disponibles ; si non, le système affiche un message indiquant l’absence de données.
- **Système génère rapport (Statistique ou conflit) :** Si des données sont disponibles, le système génère un rapport, soit sous forme statistique, soit en signalant d’éventuels conflits.
- **Consulte ou exporte le rapport :** L’utilisateur peut choisir de consulter le rapport à l’écran ou de l’exporter, constituant la fin du processus.

3.e.c Rôle des décisions, boucles, synchronisations, etc. :

- **Décisions (Authentification réussie ? et Données disponibles ?) :** Ces points de décision permettent de bifurquer le flux en fonction de la réussite de l’authentification et de la disponibilité des données, avec des messages d’erreur en cas d’échec.
- **Synchronisations :** Le diagramme implique une synchronisation entre les actions de l’utilisateur (connexion, sélection, consultation/exportation) et les réponses du système (authentification, récupération de données, génération), assurant une progression séquentielle.
- **Absence de boucles :** Aucun mécanisme de boucle n’est explicitement représenté, indiquant que le processus est linéaire avec une seule itération par génération de rapport, bien que des tentatives répétées puissent être implicites en cas d’échec d’authentification (non détaillé ici).
- **Branches alternatives :** Les chemins pour gérer les conflits dans les données ou l’absence de données représentent des scénarios alternatifs, assurant une gestion robuste des cas d’erreur.

4. Diagrammes de séquence

4.a Consultation historique



4.a.a Objectif :

Illustrer l'interaction temporelle entre objets/acteurs pour permettre à un utilisateur de consulter son historique de présence, en détaillant les échanges de messages et les processus itératifs impliqués.

4.a.b Liste des participants (acteurs, classes, composants) :

- **Utilisateur :** L'acteur principal initiant la consultation de l'historique.
- **Interface :** Composant intermédiaire gérant les interactions avec l'utilisateur.
- **Historique :** Composant ou classe contenant les données historiques des présences.
- **Système :** Composant principal traitant les requêtes et effectuant les vérifications biométriques.

4.a.c Description du scénario en mettant l'accent sur :

- **L'ordre des messages :**
    1. L'Utilisateur envoie demandeConsultationHistorique(userId) à l'Interface.
    2. L'Interface transmet consulterHistorique() à l'Historique.
    3. L'Historique envoie comparerBiometrie() au Système pour valider l'identité.
    4. Dans une boucle, l'Historique :

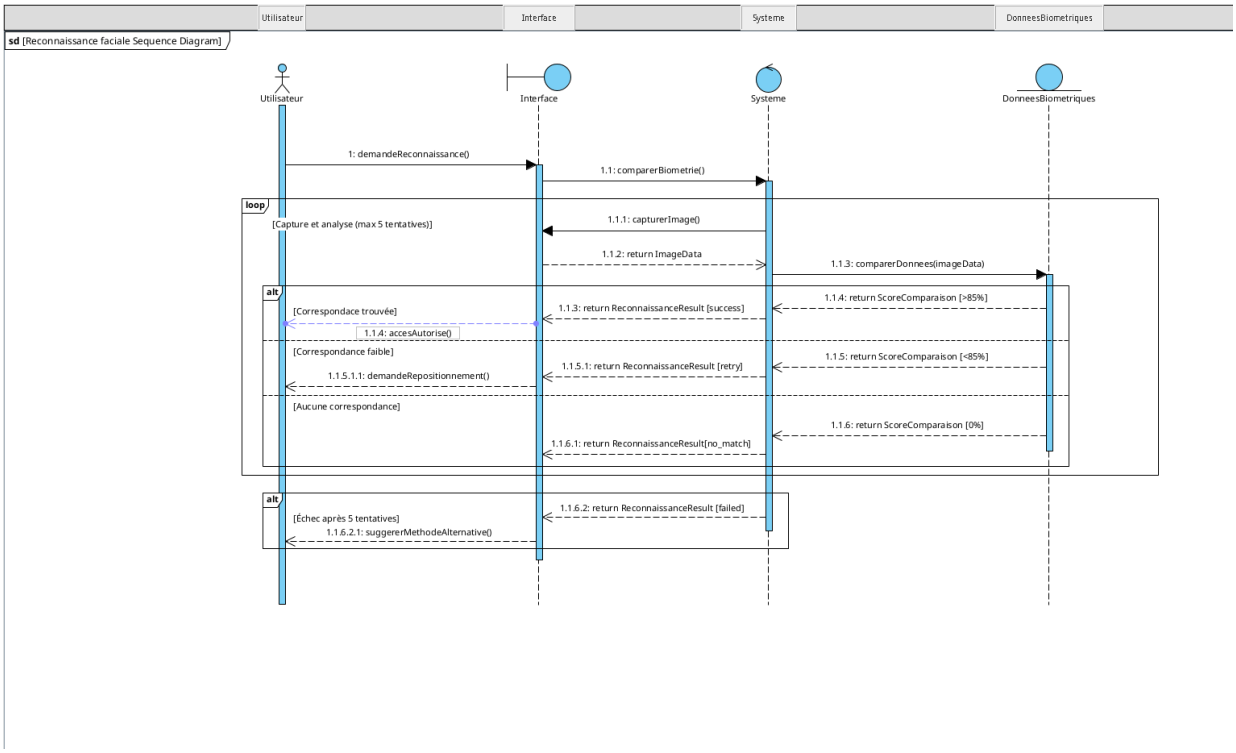
    - Retourne BiometrieResult au Système.
    - Envoie ajouterPresence(preseenceId) pour mettre à jour les données.
5. L’Historique retourne return List<Presence> à l’Interface.
6. L’Interface affiche les données via afficheHistorique(historiqueData) à l’Utilisateur.
7. Dans une boucle, l’Utilisateur demande demandeNavigation(pageNumber) à l’Interface.
8. L’Interface envoie consulter(pageNumber) à l’Historique.
9. L’Historique retourne return PageData à l’Interface.
10. L’Interface affiche les données paginées via affichePage(pageData) à l’Utilisateur.

- **Les messages importants et leur signification :**
    1. demandeConsultationHistorique(userId) : Initiates the process by specifying the user whose history is to be consulted.
    2. comparerBiometrie() : Critical for security, verifies the user’s identity via biometric data.
    3. return List<Presence> : Delivers the core data set of presence records to be displayed.
    4. demandeNavigation(pageNumber) : Allows pagination, enabling the user to navigate through large datasets.
    5. return PageData : Provides the paginated subset of data for efficient display.

4.a.d Les conditions ou boucles éventuelles :

- **Boucle (loop) pour la biométrie et les présences :** Une boucle dans l’interaction entre Historique et Système gère la validation biométrique et l’ajout des présences, permettant un traitement itératif pour chaque enregistrement pertinent.
- **Boucle (loop) pour la navigation :** Une seconde boucle permet à l’Utilisateur de demander et afficher des pages successives de l’historique, facilitant la navigation dans un ensemble de données potentiellement volumineux.
- **Absence de conditions explicites :** Aucun point de décision ou condition alternative (comme un "if") n’est représenté, indiquant que le processus suit un flux linéaire tant que les données sont disponibles et la biométrie validée.

4.b Reconnaissance faciale



4.b.a Objectif :

Illustrer l’interaction temporelle entre objets/acteurs pour permettre à un utilisateur de s’authentifier ou de vérifier son identité via une reconnaissance faciale, en détaillant les étapes de capture, comparaison et gestion des résultats.

4.b.b Liste des participants (acteurs, classes, composants) :

- **Utilisateur :** L’acteur principal initiant la demande de reconnaissance faciale.
- **Interface :** Composant intermédiaire gérant les interactions avec l’utilisateur et transmettant les demandes au système.
- **Système :** Composant principal responsable de la capture d’image, de la comparaison biométrique et de la gestion des résultats.
- **DonneesBiometriques :** Composant ou classe contenant les données biométriques enregistrées pour la comparaison.

4.b.c Description du scénario en mettant l’accent sur :

- **L’ordre des messages :**
    1. L’Utilisateur envoie demandeReconnaissance() à l’Interface.
    2. L’Interface transmet comparerBiometrie() au Système.
    3. Dans une boucle (capture et analyse, max 5 tentatives) :
        - Le Système envoie capturerImage() à lui-même.
        - Le Système retourne return ImageData à l’Interface.
        - Le Système envoie comparerDonnees(imageData) aux DonneesBiometriques.
        - Les DonneesBiometriques retournent return ScoreComparaison au Système.

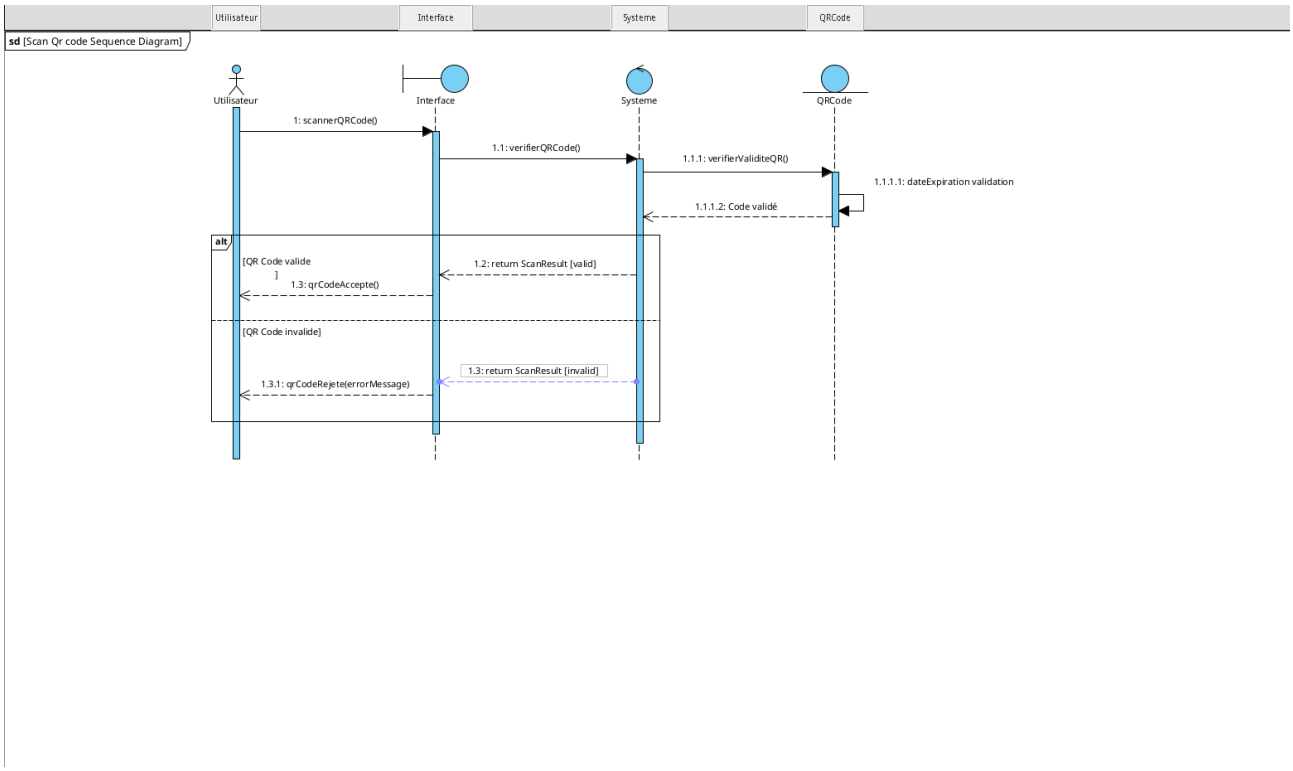
4. Selon les résultats (branche alternative) :
    - Si ScoreComparaison >= 85% : Le Système retourne return ReconnaissanceResult(success) à l'Interface, qui transmet accesAutorise() à l'Utilisateur.
    - Si ScoreComparaison < 85% : Le Système retourne return ReconnaissanceResult(retry) à l'Interface, qui transmet demandeRepositionnement() à l'Utilisateur.
    - Si ScoreComparaison = 0% : Le Système retourne return ReconnaissanceResult(no_match) à l'Interface.
    - Après 5 tentatives échouées : Le Système retourne return ReconnaissanceResult(failed) à l'Interface, qui transmet suggererMethodeAlternative() à l'Utilisateur.

- **Les messages importants et leur signification :**
    1. demandeReconnaissance() : Initiates the facial recognition process by the user.
    2. capturerImage() : Triggers the capture of the user's facial image, a critical step in the process.
    3. comparerDonnees(imageData) : Performs the biometric comparison, central to validating identity.
    4. return ScoreComparaison : Provides the comparison score, determining the success or failure of recognition.
    5. return ReconnaissanceResult(success/retry/no_match/failed) : Communicates the outcome, enabling appropriate user feedback or action (access, repositioning, or alternative method).

4.b.d Les conditions ou boucles éventuelles :

- **Boucle (loop - Capture et analyse, max 5 tentatives) :** Une boucle gère les tentatives de capture et d'analyse de l'image, limitée à 5 essais, permettant des ajustements si la reconnaissance échoue initialement.
- **Branche alternative (alt) :** Plusieurs branches alternatives gèrent les différents résultats de la comparaison :
    - Correspondance trouvée (score ≥ 85%) mène à un accès autorisé.
    - Correspondance faible (score < 85%) déclenche une demande de repositionnement.
    - Aucune correspondance (score = 0%) indique un échec d'identité.
    - Échec après 5 tentatives suggère une méthode alternative, assurant une gestion robuste des cas d'échec.

4.c Scan QR code



4.c.a Objectif :

Illustrer l'interaction temporelle entre objets/acteurs pour permettre à un utilisateur de scanner un QR code, en détaillant les étapes de vérification et les résultats possibles.

4.c.b Liste des participants (acteurs, classes, composants) :

- **Utilisateur :** L'acteur principal initiant le scan du QR code.
- **Interface :** Composant intermédiaire gérant les interactions avec l'utilisateur et transmettant les demandes au système.
- **Système :** Composant principal responsable de la vérification du QR code.
- **QRCode :** Composant ou classe contenant les données du QR code et effectuant la validation.

4.c.c Description du scénario en mettant l'accent sur :

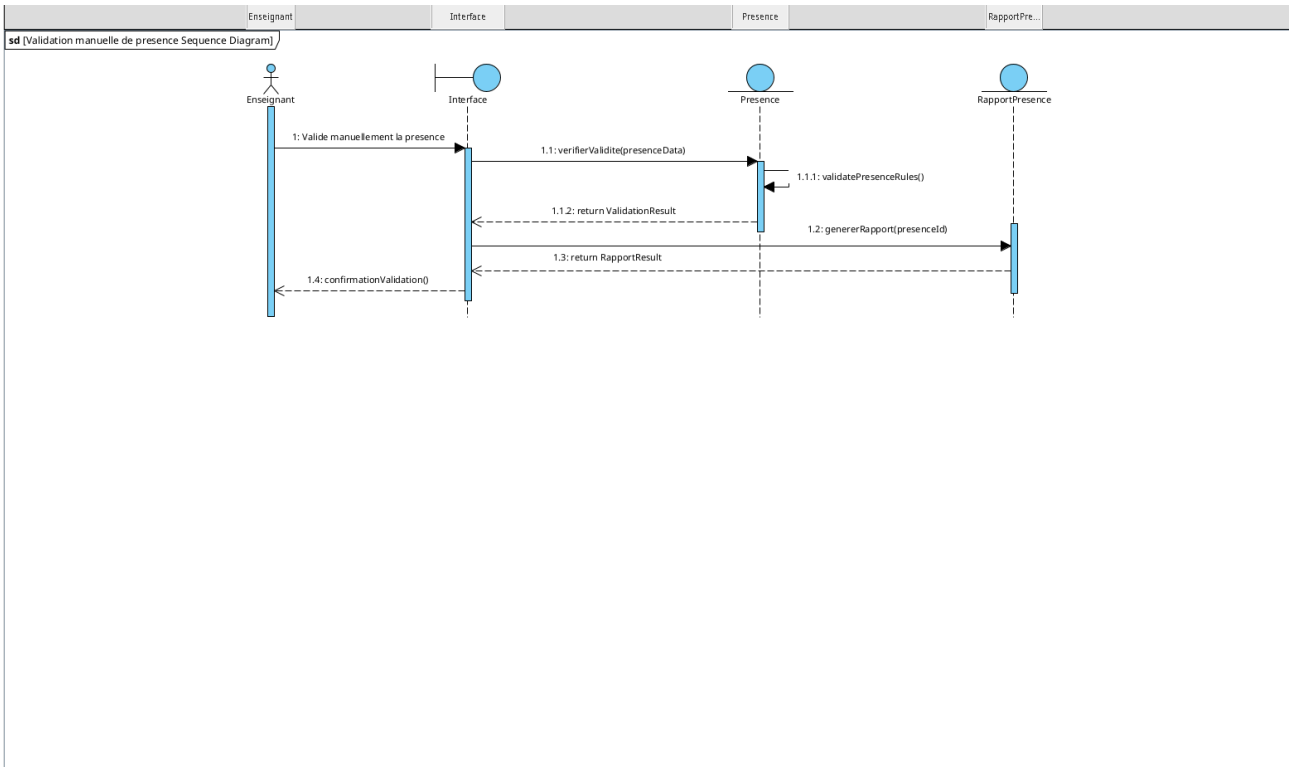
- **L'ordre des messages :**
    1. L'Utilisateur envoie scannerQRCode() à l'Interface.
    2. L'Interface transmet verifierQRCode() au Système.
    3. Le Système envoie verifierValideQR() au QRCode.
    4. Le QRCode effectue dateExpirationValidation() pour vérifier la validité temporelle.
    5. Le QRCode retourne Code valide au Système si la validation réussit.
    6. Selon les résultats (branche alternative) :
        - Si valide : Le Système retourne return ScanResult(valid) à l'Interface, qui transmet qrCodeAccepte() à l'Utilisateur.

    - Si invalide : Le Système retourne return ScanResult(invalid) à l’Interface, qui transmet qrCodeRejete(errorMessage) à l’Utilisateur.
- **Les messages importants et leur signification :**
    1. scannerQRCode() : Initiates the QR code scanning process by the user.
    2. verifierQRCode() : Triggers the verification process, a key step in validating the QR code.
    3. verifierValideQR() : Performs the core validation of the QR code’s integrity and expiration.
    4. dateExpirationValidation() : Ensures the QR code is still valid based on its expiration date.
    5. return ScanResult(valid/invalid) : Communicates the outcome, enabling appropriate user feedback (acceptance or rejection).

4.c.d Les conditions ou boucles éventuelles :

- **Branche alternative (alt) :** Une branche alternative gère les deux cas possibles :
    - QR Code valide mène à une acceptation et un retour positif.
    - QR Code invalide mène à un rejet avec un message d’erreur.
- **Absence de boucles :** Aucun mécanisme de boucle n’est représenté, indiquant que le processus est linéaire avec une seule tentative de scan, sans itération explicite (des tentatives répétées pourraient être implicites en cas d’échec, mais non détaillées ici).
- **Absence de conditions explicites :** Aucune condition supplémentaire (comme un "if" ou une boucle) n’est illustrée en dehors de la branche alternative, suggérant un flux direct basé sur la validité du QR code.

4.d Validation manuelle de presence



4.d.a Objectif :

Illustrer l’interaction temporelle entre objets/acteurs pour permettre à un enseignant de valider manuellement la présence d’un étudiant, en détaillant les étapes de vérification des données et la génération d’un rapport.

4.d.b Liste des participants (acteurs, classes, composants) :

- **Enseignant :** L’acteur principal initiant la validation manuelle de la présence.
- **Interface :** Composant intermédiaire gérant les interactions avec l’enseignant et transmettant les demandes.
- **Presence :** Composant ou classe contenant les données de présence à valider.
- **RapportPresence :** Composant ou classe responsable de générer le rapport de validation.

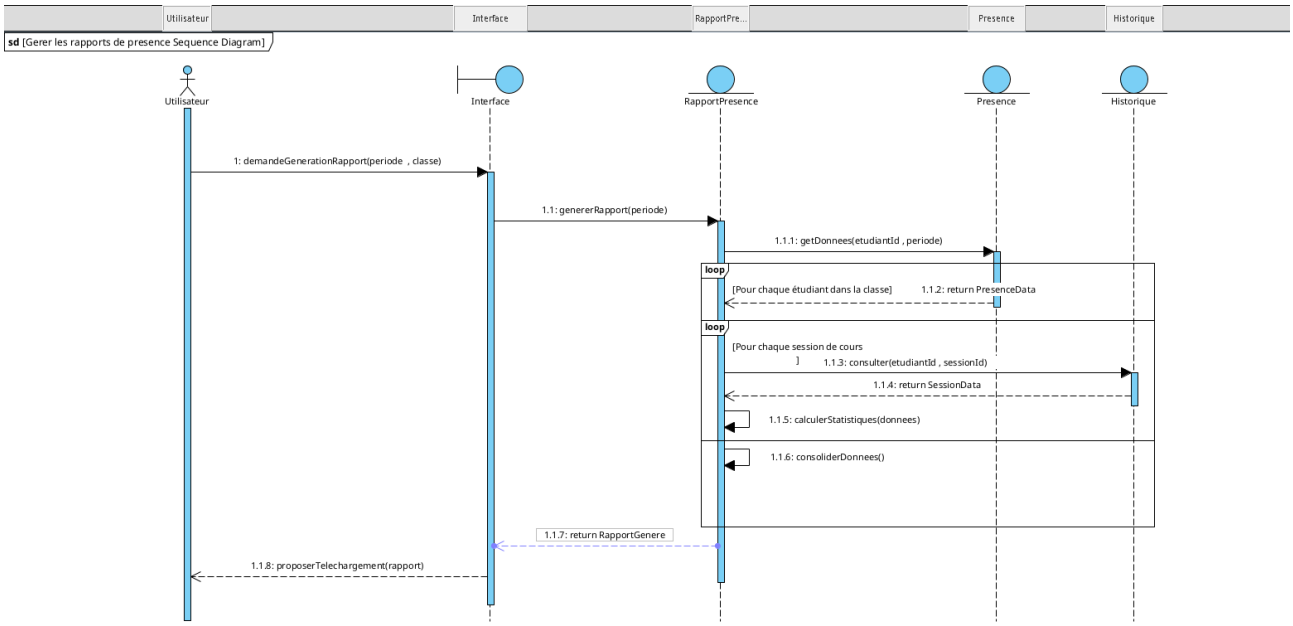
4.d.c Description du scénario en mettant l’accent sur :

- **L’ordre des messages :**
    1. L’Enseignant envoie valideManuellementLaPresence à l’Interface.
    2. L’Interface transmet verifierValidePresenceData() à la Presence.
    3. La Presence envoie validatePresenceRules() pour vérifier les règles de validation.
    4. La Presence retourne return ValidationResult à l’Interface.
    5. L’Interface retourne return RapportResult à l’Enseignant.
    6. La Presence envoie genererRapport(presenceId) au RapportPresence.
    7. Le RapportPresence retourne les données générées à la Presence.
    8. L’Enseignant envoie confirmationValidation() à l’Interface pour confirmer le processus.
- **Les messages importants et leur signification :**
    1. valideManuellementLaPresence : Initiates the manual validation process by the teacher.
    2. verifierValidePresenceData() : Triggers the verification of presence data, a key step in ensuring accuracy.
    3. validatePresenceRules() : Checks compliance with predefined presence rules, critical for validation integrity.
    4. return ValidationResult : Provides the result of the validation check, guiding the next steps.
    5. genererRapport(presenceId) : Generates a report based on the validated presence data.
    6. return RapportResult : Delivers the final report to the teacher for confirmation or use.
    7. confirmationValidation() : Confirms the validation, finalizing the process.

4.d.d Les conditions ou boucles éventuelles :

- **Absence de boucles :** Aucun mécanisme de boucle n’est représenté, indiquant que le processus est linéaire avec une seule itération par validation.
- **Absence de conditions explicites :** Aucune condition ou branche alternative (comme un "if" ou "alt") n’est illustrée, suggérant un flux direct basé sur la réussite implicite de la validation, sans gestion explicite d’échecs ou d’erreurs dans ce diagramme.

4.e Gerer les rapports de presence



4.e.a Objectif :

Illustrer l'interaction temporelle entre objets/acteurs pour permettre à un utilisateur de générer, traiter et télécharger un rapport de présence basé sur une période et une classe spécifique.

4.e.b Liste des participants (acteurs, classes, composants) :

- **Utilisateur :** L'acteur principal initiant la demande de génération de rapport.
- **Interface :** Composant intermédiaire gérant les interactions avec l'utilisateur et transmettant les demandes.
- **RapportPresence :** Composant ou classe responsable de générer et consolider les données du rapport.
- **Presence :** Composant ou classe fournissant les données de présence des étudiants.
- **Historique :** Composant ou classe fournissant les données historiques pour compléter le rapport.

4.e.c Description du scénario en mettant l'accent sur :

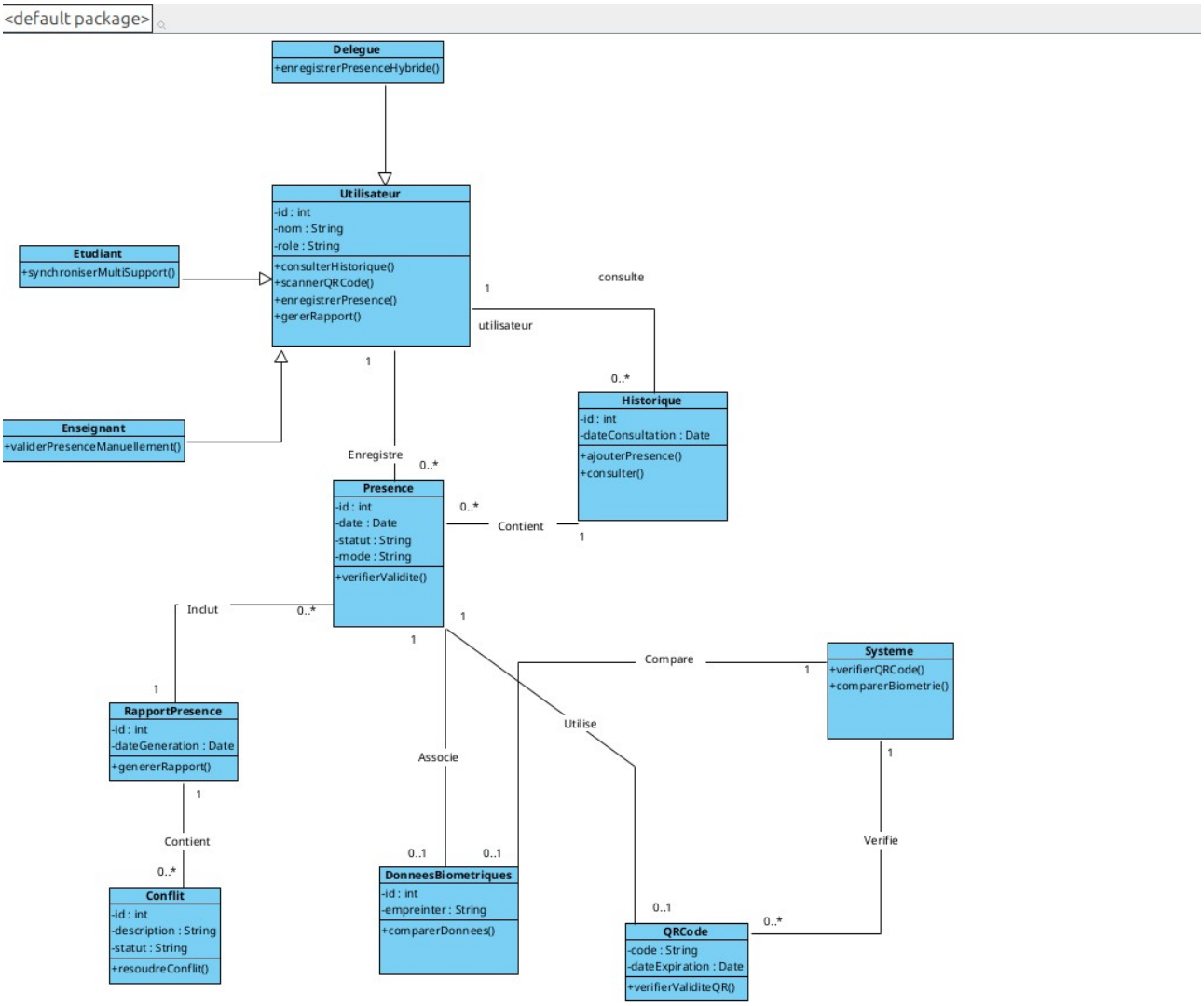
- **L'ordre des messages :**
    1. L'Utilisateur envoie demandeGenerationRapport(periode, classe) à l'Interface.
    2. L'Interface transmet genererRapport(periode) à RapportPresence.
    3. Dans une boucle (pour chaque étudiant dans la classe) :
        - RapportPresence envoie getDonneesEtudiant(periode) à Presence.
        - Presence retourne return PresenceData à RapportPresence.
    4. Dans une boucle (pour chaque session de cours) :
        - RapportPresence envoie consulterEtudiant(sessionId) à Historique.

        - Historique retourne return SessionData à RapportPresence.
    5. RapportPresence envoie calculerStatistiques(donnees) pour analyser les données.
    6. RapportPresence envoie consoliderDonnees() pour finaliser le rapport.
    7. RapportPresence retourne return RapportGenere à l’Interface.
    8. L’Interface propose proposerTelechargement(rapport) à l’Utilisateur.
- **Les messages importants et leur signification :**
    1. demandeGenerationRapport(periode, classe) : Initiates the report generation process with specific parameters.
    2. genererRapport(periode) : Triggers the generation of the report based on the given period.
    3. getDonneesEtudiant(periode) : Requests student presence data, a key step in data collection.
    4. return PresenceData : Provides the raw presence data for each student.
    5. consulterEtudiant(sessionId) : Retrieves session-specific data from the historical records.
    6. return SessionData : Supplies session data to enrich the report.
    7. calculerStatistiques(donnees) : Computes statistical insights from the collected data.
    8. return RapportGenere : Delivers the finalized report to the user.
    9. proposerTelechargement(rapport) : Offers the user the option to download the generated report.

4.e.d Les conditions ou boucles éventuelles :

- **Boucle (pour chaque étudiant dans la classe) :** Une boucle itère sur tous les étudiants de la classe pour collecter leurs données de présence, assurant une couverture complète.
- **Boucle (pour chaque session de cours) :** Une seconde boucle itère sur chaque session de cours pour intégrer les données historiques, permettant une analyse détaillée.
- **Absence de conditions explicites :** Aucune condition ou branche alternative (comme un "if" ou "alt") n’est illustrée, suggérant un flux direct basé sur la disponibilité implicite des données, sans gestion explicite d’erreurs dans ce diagramme.

5. Diagramme de classes



5.a Vue d'ensemble de l'architecture logicielle

Le diagramme de classes représente une architecture logicielle conçue pour gérer les présences et les rapports dans un système éducatif. Il met en évidence les interactions entre les utilisateurs (Étudiant, Enseignant/Délégué), les entités de données (Presence, Historique, RapportPresence, QRCode, DonneesBiometriques, Conflit, Contient) et les composants fonctionnels (Utilisateur, Systeme, Verifie, Compare). Cette structure permet l'enregistrement hybride des présences, la consultation d'historiques, la validation manuelle et la génération de rapports, avec une intégration de la biométrie et des QR codes pour la sécurité.

5.b Description des principales classes

- **Utilisateur**
    - **Attributs importants :** id: int, nom: String, role: String

    - **Méthodes principales :** consulterHistorique(), scannerQRCode(), enregistrerPresence(), genererRapport()
    - **Relations :** Utilisé par 0..* Enseignant/Délégué et 0..* Étudiant (généralisation), associé à 1 Historique (consultation).
- **Enseignant**
    - **Attributs importants :** (hérités de Utilisateur)
    - **Méthodes principales :** validerPresenceManuellement()
    - **Relations :** Hérite de Utilisateur, associé à 1 Systeme pour la validation.
- **Délégué**
    - **Attributs importants :** (hérités de Utilisateur)
    - **Méthodes principales :** enregistrerPresenceHybride()
    - **Relations :** Hérite de Utilisateur, associé à 1 Systeme pour l'enregistrement hybride.
- **Étudiant**
    - **Attributs importants :** (hérités de Utilisateur)
    - **Méthodes principales :** synchroniserMultiSupport()
    - **Relations :** Hérite de Utilisateur, associé à 1 Presence (enregistrement).
- **Historique**
    - **Attributs importants :** +ajouterPresence(): Date, +consulter()
    - **Méthodes principales :** ajouterPresence(), consulter()
    - **Relations :** Associé à 1 Utilisateur (consultation), composé de 0..* Presence (agrégation).
- **Presence**
    - **Attributs importants :** id: int, date: Date, statut: String, mode: String
    - **Méthodes principales :** verifierValide()
    - **Relations :** Composé de 0..* Historique (agrégation), associé à 1 Étudiant (enregistrement), associé à 1 RapportPresence (génération).
- **RapportPresence**
    - **Attributs importants :** id: int, dateGeneration: Date
    - **Méthodes principales :** genererRapport()
    - **Relations :** Associé à 1 Presence (génération), associé à 1 Contient (contenu), associé à 1 Utilisateur (génération).
- **QRCode**
    - **Attributs importants :** code: String, dateExpiration: Date
    - **Méthodes principales :** verifierValideQR()
    - **Relations :** Associé à 0..1 Systeme (vérification), associé à 0..* DonneesBiometriques (validation).
- **DonneesBiometriques**
    - **Attributs importants :** id: int, empreinte: String
    - **Méthodes principales :** comparerDonnees()
    - **Relations :** Associé à 0..1 QRCode (validation), associé à 1 Systeme (comparaison).
- **Conflit**
    - **Attributs importants :** id: int, description: String, statut: String
    - **Méthodes principales :** resoudreConflit()
    - **Relations :** Associé à 0..* Presence (conflit potentiel).
- **Contient**
    - **Attributs importants :** (aucun attribut spécifique)

    - **Méthodes principales :** (aucune méthode spécifique)
    - **Relations :** Associé à 1 RapportPresence (contenu), composé de 0..* Presence (agrégation).
- **Systeme**
    - **Attributs importants :** (aucun attribut spécifique)
    - **Méthodes principales :** verifierQRCode(), comparerBiometrie()
    - **Relations :** Associé à 1 Enseignant/Délégué (validation/enregistrement), associé à 1 QRCode (vérification), associé à 1 DonneesBiometriques (comparaison).
- **Verifie**
    - **Attributs importants :** (aucun attribut spécifique)
    - **Méthodes principales :** verifier()
    - **Relations :** Associé à 1 Systeme (vérification), associé à 0..* QRCode (validation).
- **Compare**
    - **Attributs importants :** (aucun attribut spécifique)
    - **Méthodes principales :** comparer()
    - **Relations :** Associé à 1 Systeme (comparaison), associé à 1 DonneesBiometriques (comparaison).

5.c Explication des relations

- **Cardinalités :**
    - Utilisateur -> Historique : 1 (un utilisateur consulte un historique), 0..* (un historique peut être consulté par plusieurs utilisateurs).
    - Utilisateur -> Enseignant/Délégué/Étudiant : 0..* (plusieurs utilisateurs peuvent être de ces types), 1 (chaque enseignant/délégué/étudiant est un utilisateur).
    - Historique -> Presence : 0..* (un historique contient plusieurs présences), 1 (chaque présence appartient à un historique).
    - Presence -> Étudiant : 1 (une présence est liée à un étudiant), 0..* (un étudiant peut avoir plusieurs présences).
    - RapportPresence -> Presence : 1 (un rapport est généré à partir d'une présence), 0..* (une présence peut être incluse dans plusieurs rapports via Contient).
    - RapportPresence -> Contient : 1 (un rapport contient des données), 0..* (un rapport peut contenir plusieurs présences).
    - QRCode -> Systeme : 0..1 (un QR code peut être vérifié par un système), 1 (le système vérifie un QR code).
    - DonneesBiometriques -> QRCode : 0..* (plusieurs données biométriques peuvent valider un QR code), 0..1 (un QR code peut être lié à des données biométriques).
    - Systeme -> Enseignant/Délégué : 1 (le système est utilisé par un enseignant/délégué), 1 (un enseignant/délégué utilise un système).
    - Conflit -> Presence : 0..* (un conflit peut concerner plusieurs présences), 0..* (une présence peut être liée à plusieurs conflits).
- **Navigabilité :**
    - La navigabilité est unidirectionnelle dans la plupart des cas (par exemple, Utilisateur -> Historique, Enseignant -> Systeme), indiquant que l'initiation vient d'une classe vers une autre (par exemple, un utilisateur consulte un historique, un enseignant valide via le système).

    - Certaines relations (comme Presence -> RapportPresence via Contient) suggèrent une navigabilité bidirectionnelle implicite pour la gestion des données.
- **Multidirectionnalité si applicable :**
    - La relation entre RapportPresence, Contient et Presence montre une multidirectionnalité implicite, où le rapport contient des présences via Contient, et les présences peuvent être liées à plusieurs rapports, permettant une gestion complexe des données rapportées.