MIAGE_ 2021 - 2022

UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Master 2 MIAGE parcours ITN

Dossier de conception
Odoru

Quentin DOURIS
Christian MICHIELAN
Trung LE DUC

Enseignants : Cédric TEYSSIE
Patrice TORGUET

1. Introduction

Ce projet a pour objectif de mettre en pratique la création et la manipulation d'Applications en Mode Service (AMS). Ainsi, nous allons mettre en place une architecture composée de différents micro-services.

Pour réaliser ce projet, nous allons travailler dans un environnement JAVA avec le framework open source Spring au sein de l'environnement de développement intégré : IntelliJ IDEA. De plus, nous allons utiliser l'outil Git et l'interface GitHub pour la gestion des différentes versions de notre projet. Nous allons également utiliser pour la gestion du projet le tableau Kanban proposé par GitHub. Enfin, pour réaliser les différents diagrammes de modélisation UML, nous avons utilisé l'outil Visual Paradigm Community.

La réalisation de ce projet a pour objectif de concevoir la plateforme pour le club de danse Odoru pour gérer ses membres, de planifier des cours, des intervenants et de gérer l'assiduité des membres du club aux différents cours.

L'ensemble des règles de gestion nous ont été transmises dans le sujet de ce projet. Pour certaines règles, nous nous sommes permis de formuler des hypothèses pour traiter le sujet de ce projet. L'ensemble de ces hypothèses vous sont présentées dans ce dossier de conception lorsque nous avons eu à le faire.

Ainsi, nous allons concevoir, développer et mettre en place l'application du centre de danse Odoru.

Ce présent document correspond au dossier de conception. Ainsi, seront présentés en grande partie les différents diagrammes de conception réalisés avec la réflexion que nous avons adoptée pour aboutir à une solution fonctionnelle.

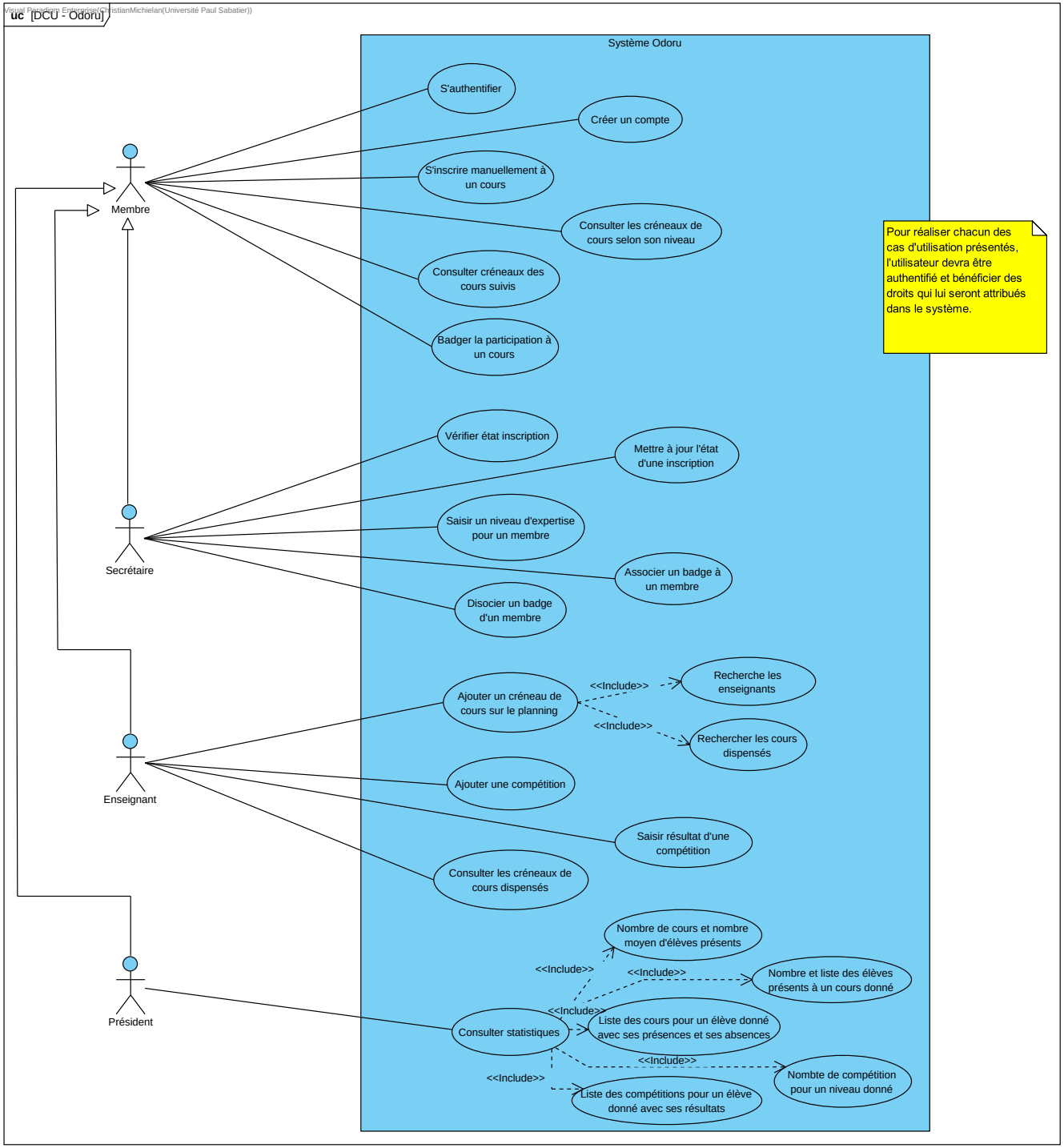
Une présentation vidéo du projet est disponible à l'url suivante : https://youtu.be/yLuFsjBxark

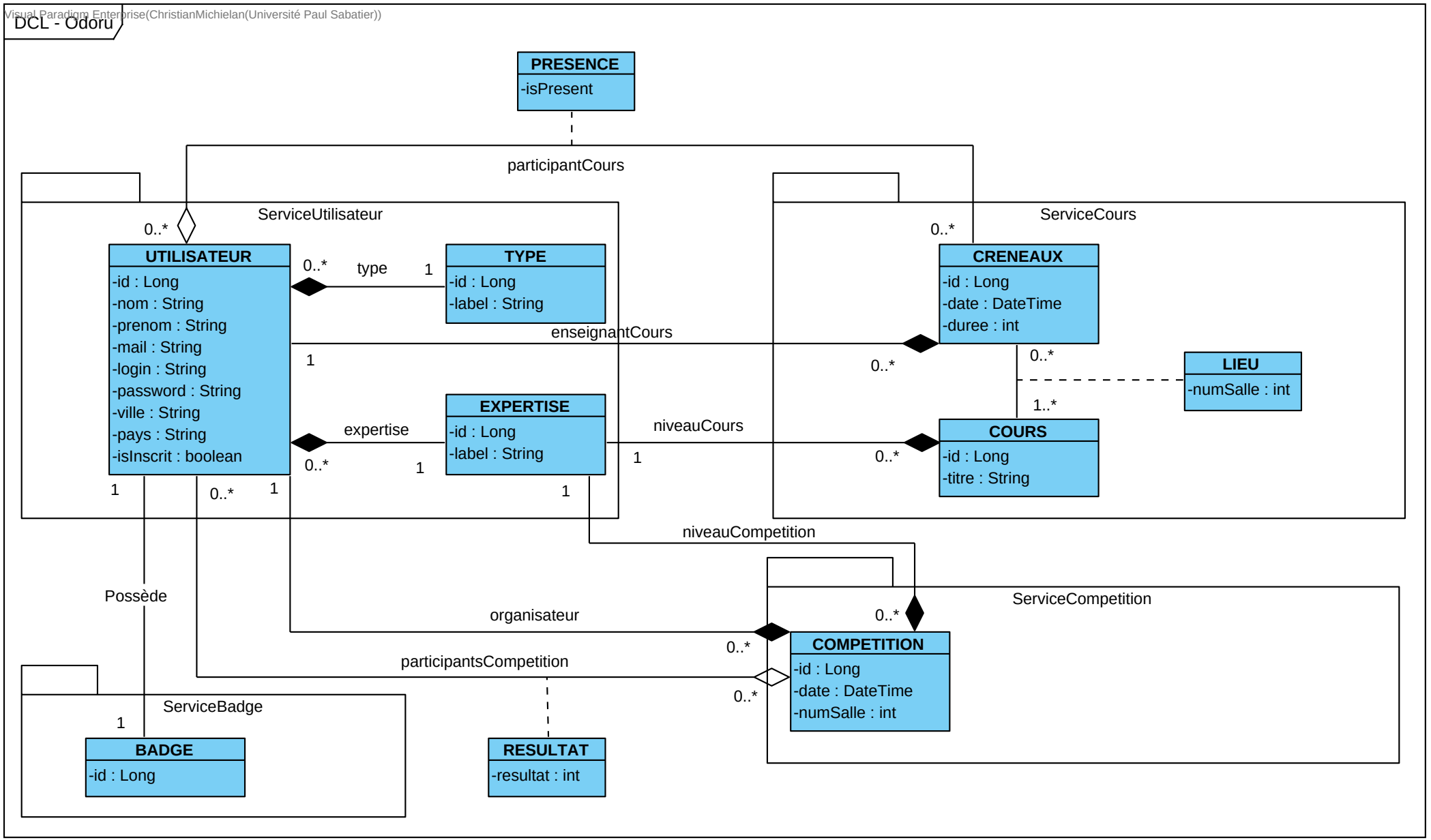
2. Diagrammes UML

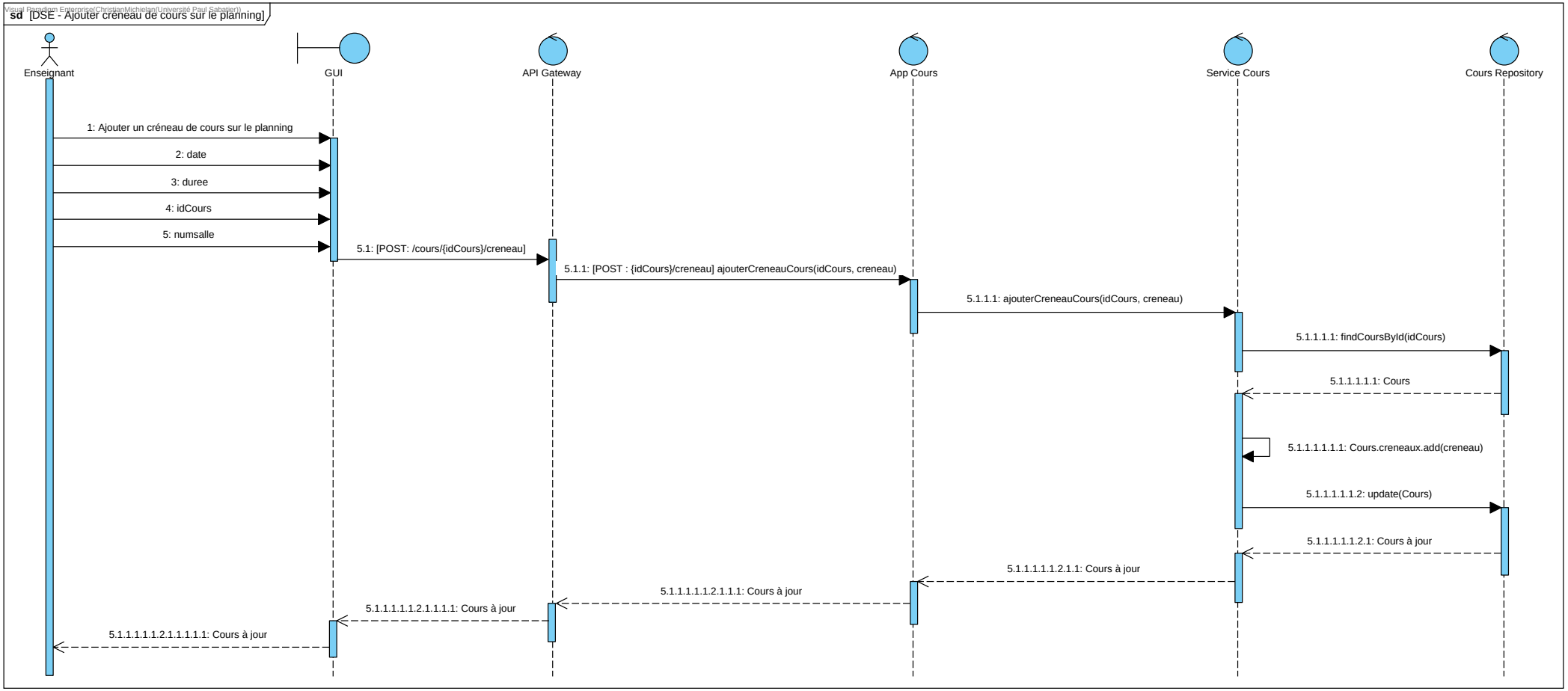
Ci-dessous sont présentés les différents diagrammes UML correspondants à notre analyse du projet Odoru.

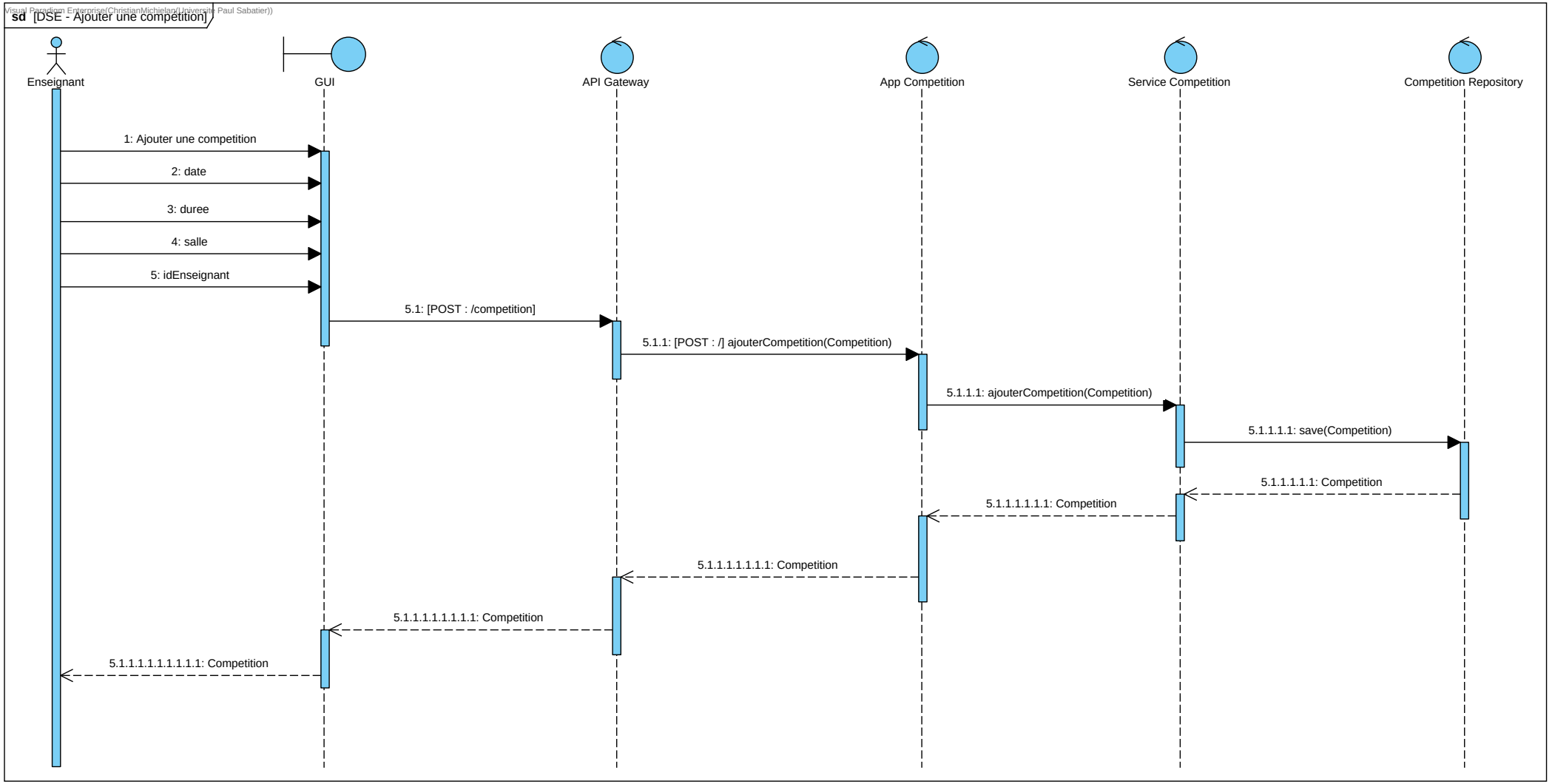
NB : Les “APP” proposés sur les diagrammes de séquences correspondent à des contrôleurs REST.

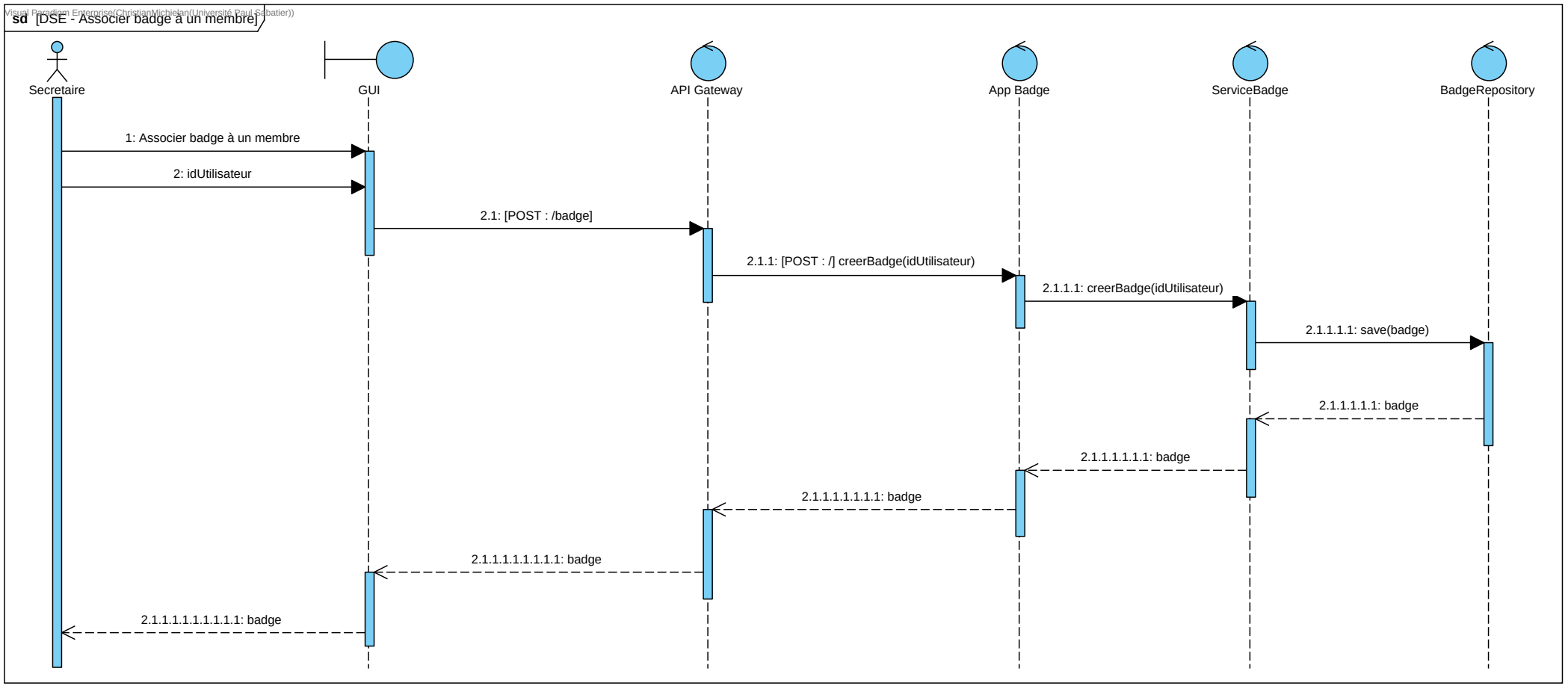
uc [DCU - Odoru]

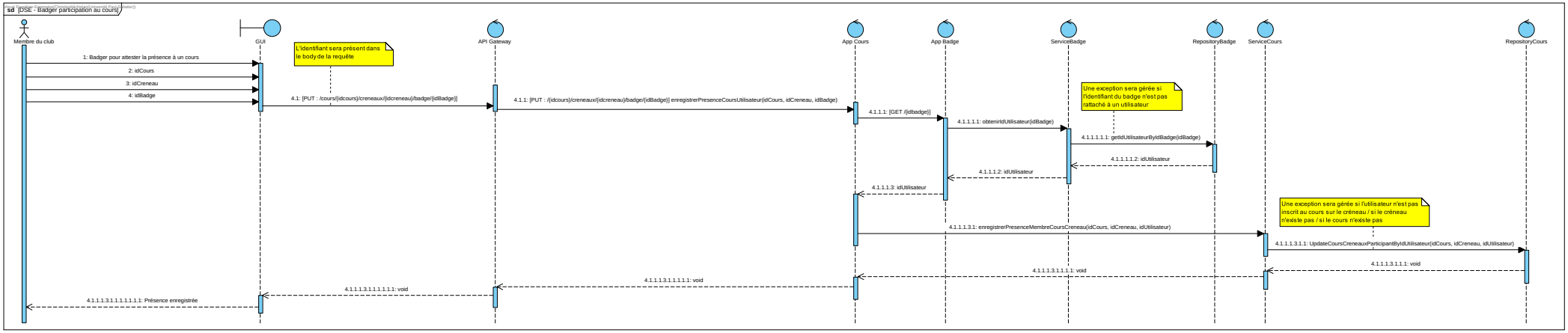


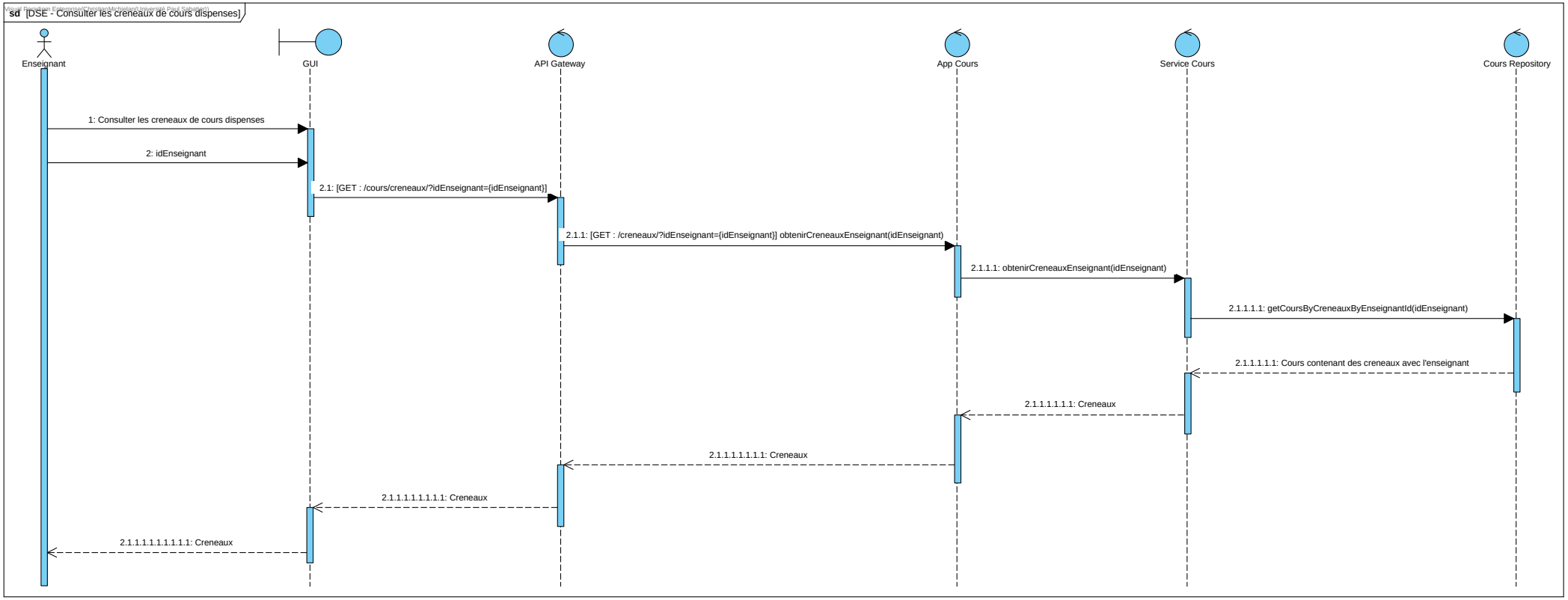


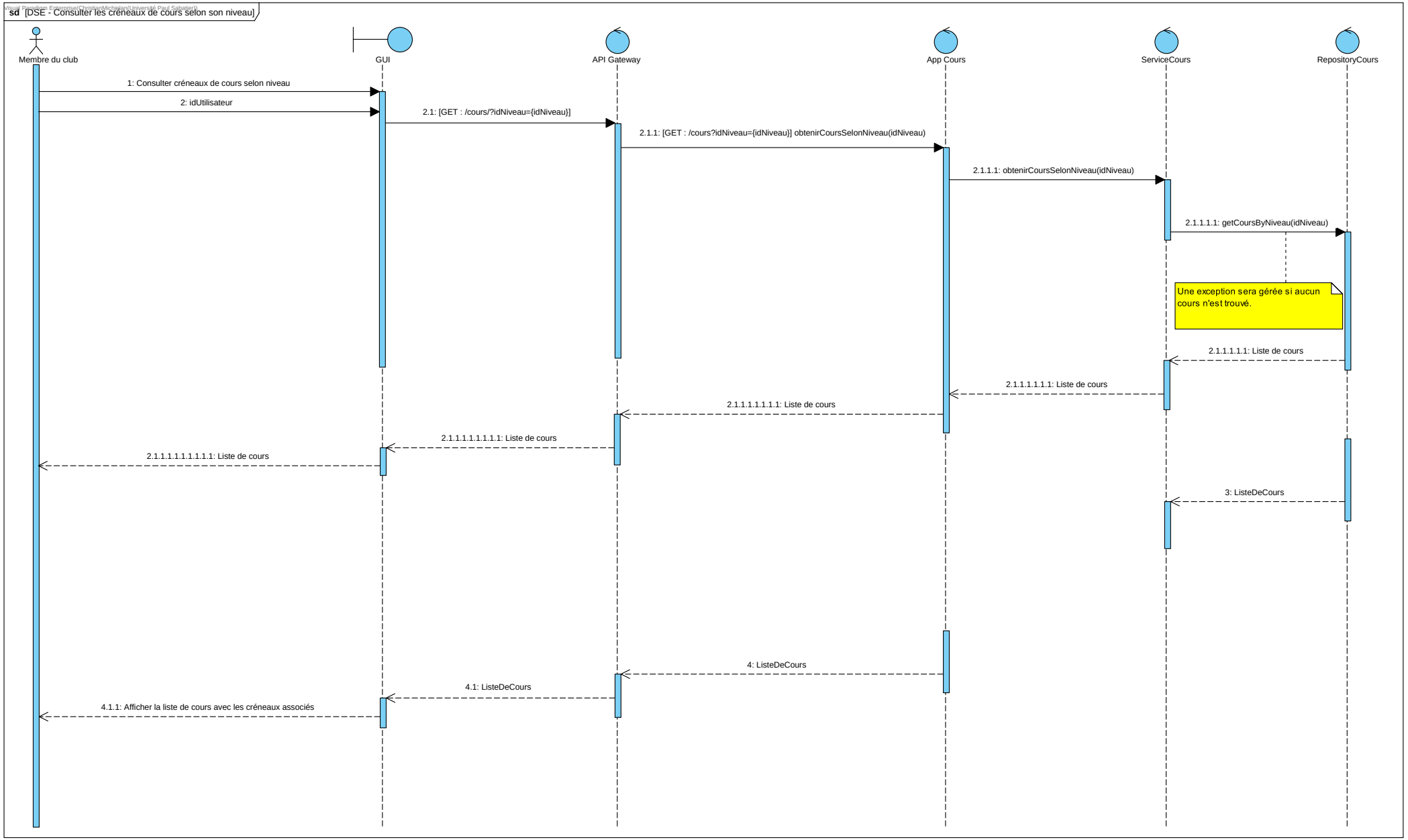

Visual Paradigm Enterprise(ChristianMichelan(Université Paul Sabatier))
sd [DSE - Ajouter créneau de cours sur le planning]
Enseignant
GUI
API Gateway
App Cours
Service Cours
Cours Repository
1: Ajouter un créneau de cours sur le planning
2: date
3: duree
4: idCours
5: numsalle
5.1: [POST: /cours/{idCours}/creneau]
5.1.1: [POST : {idCours}/creneau] ajouterCreneauCours(idCours, creneau)
5.1.1.1: ajouterCreneauCours(idCours, creneau)
5.1.1.1.1: findCoursById(idCours)
5.1.1.1.1.1: Cours
5.1.1.1.1.1.1: Cours.creneaux.add(creneau)
5.1.1.1.1.2: update(Cours)
5.1.1.1.1.2.1: Cours à jour
5.1.1.1.1.2.1.1: Cours à jour
5.1.1.1.1.2.1.1.1: Cours à jour
5.1.1.1.1.2.1.1.1.1: Cours à jour
5.1.1.1.1.2.1.1.1.1.1: Cours à jour

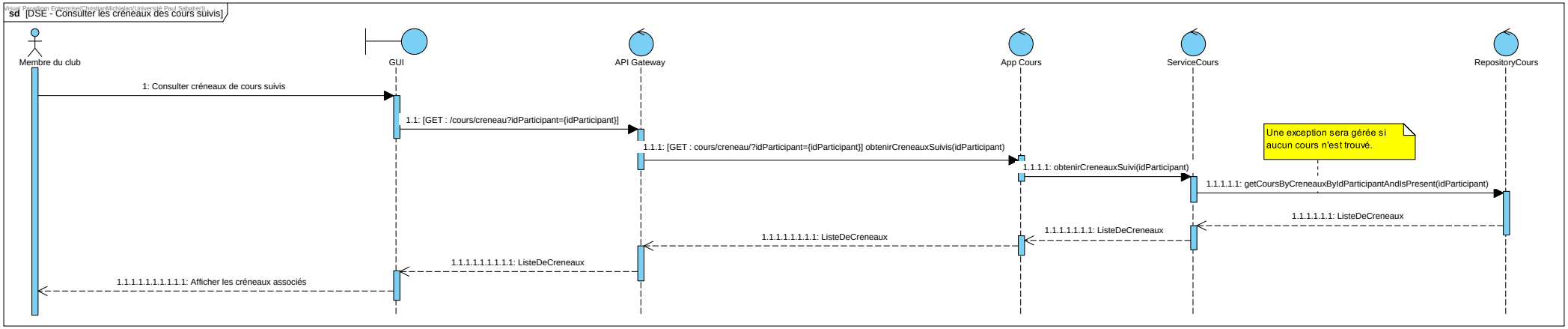


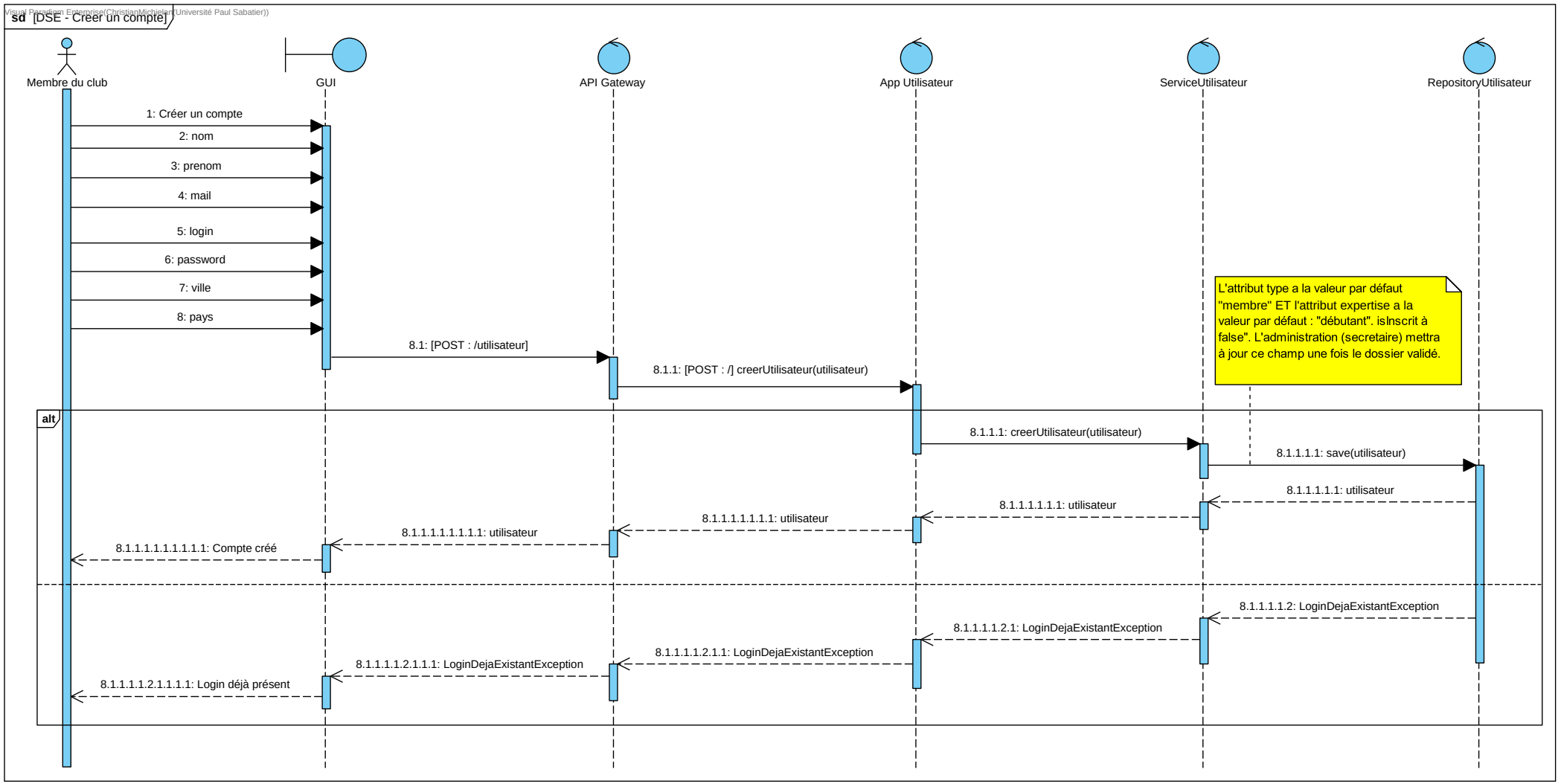


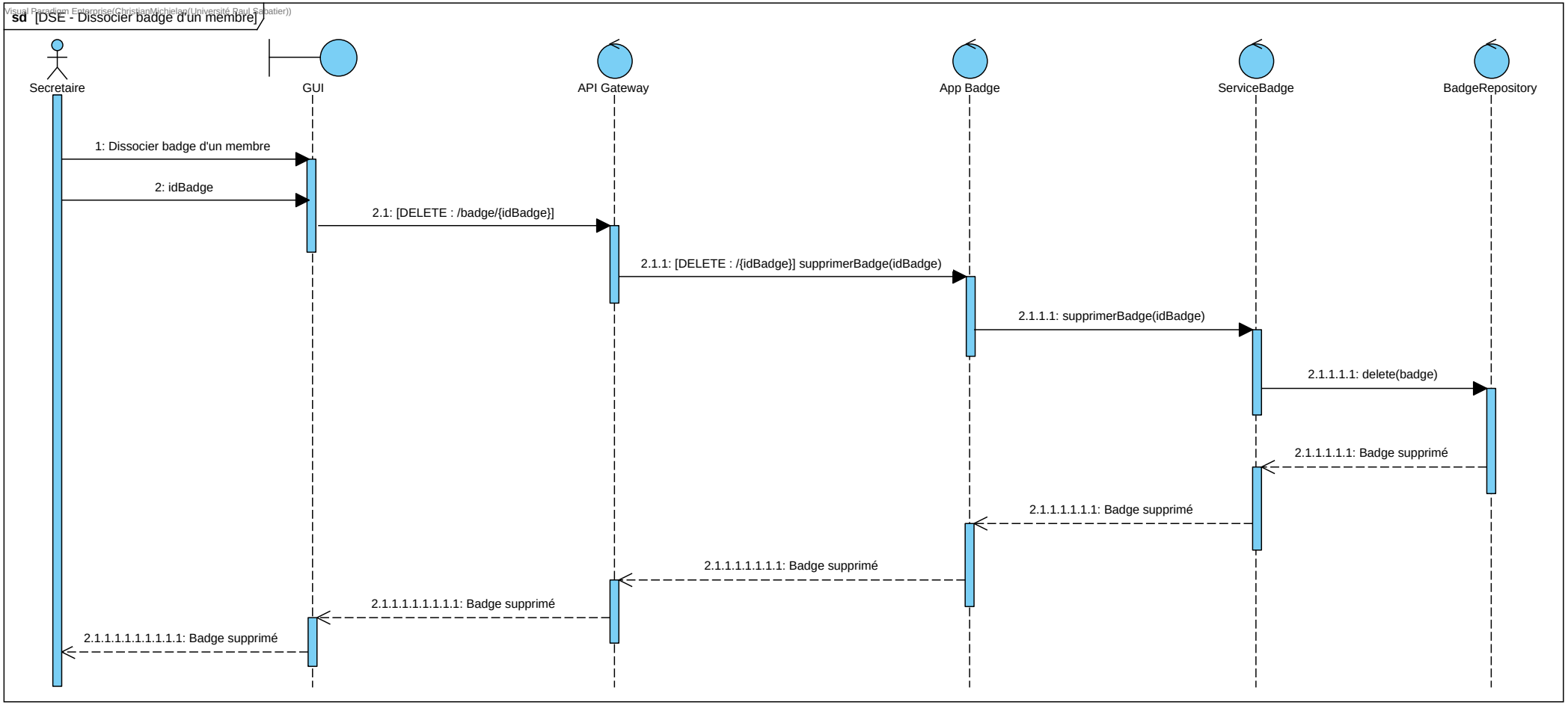

sd [DSE - Badger participation au cours]
Membre du club
GUI
API Gateway
App Cours
App Badge
ServiceBadge
RepositoryBadge
ServiceCours
RepositoryCours
1: Badger pour attester la présence à un cours
2: idCours
3: idCreneau
4: idBadge
L'identifiant sera présent dans le body de la requête
4.1: [PUT : /cours/{idcours}/creneaux/{idcreneau}/badge/{idBadge}]
4.1.1: [PUT : /{idcours}/creneaux/{idcreneau}/badge/{idBadge}] enregistrerPresenceCoursUtilisateur(idCours, idCreneau, idBadge)
4.1.1.1: [GET /{idbadge}]
4.1.1.1.1: obtenirIdUtilisateur(idBadge)
Une exception sera gérée si l'identifiant du badge n'est pas rattaché à un utilisateur
4.1.1.1.1.1: getIdUtilisateurByIdBadge(idBadge)
4.1.1.1.1.2: idUtilisateur
4.1.1.1.2: idUtilisateur
4.1.1.1.3: idUtilisateur
Une exception sera gérée si l'utilisateur n'est pas inscrit au cours sur le créneau / si le créneau n'existe pas / si le cours n'existe pas
4.1.1.1.3.1: enregistrerPresenceMembreCoursCreneau(idCours, idCreneau, idUtilisateur)
4.1.1.1.3.1.1: UpdateCoursCreneauxParticipantByIdUtilisateur(idCours, idCreneau, idUtilisateur)
4.1.1.1.3.1.1.1: void
4.1.1.1.3.1.1.1.1: void
4.1.1.1.3.1.1.1.1.1: void
4.1.1.1.3.1.1.1.1.1.1: Présence enregistrée

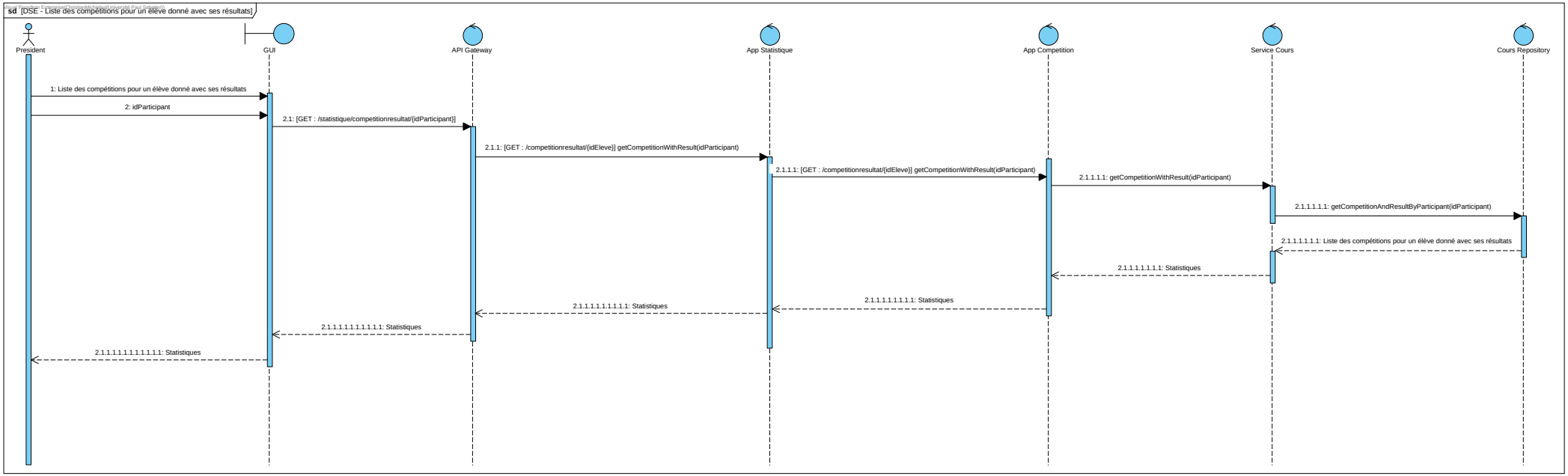

sd [DSE - Consulter les créneaux de cours dispenses]
Enseignant
GUI
API Gateway
App Cours
Service Cours
Cours Repository
1: Consulter les creneaux de cours dispenses
2: idEnseignant
2.1: [GET : /cours/creneaux/?idEnseignant={idEnseignant}]
2.1.1: [GET : /creneaux/?idEnseignant={idEnseignant}] obtenirCreneauxEnseignant(idEnseignant)
2.1.1.1: obtenirCreneauxEnseignant(idEnseignant)
2.1.1.1.1: getCoursByCreneauxByEnseignantId(idEnseignant)
2.1.1.1.1.1: Cours contenant des creneaux avec l'enseignant
2.1.1.1.1.1.1: Creneaux
2.1.1.1.1.1.1.1: Creneaux
2.1.1.1.1.1.1.1.1: Creneaux
2.1.1.1.1.1.1.1.1.1: Creneaux

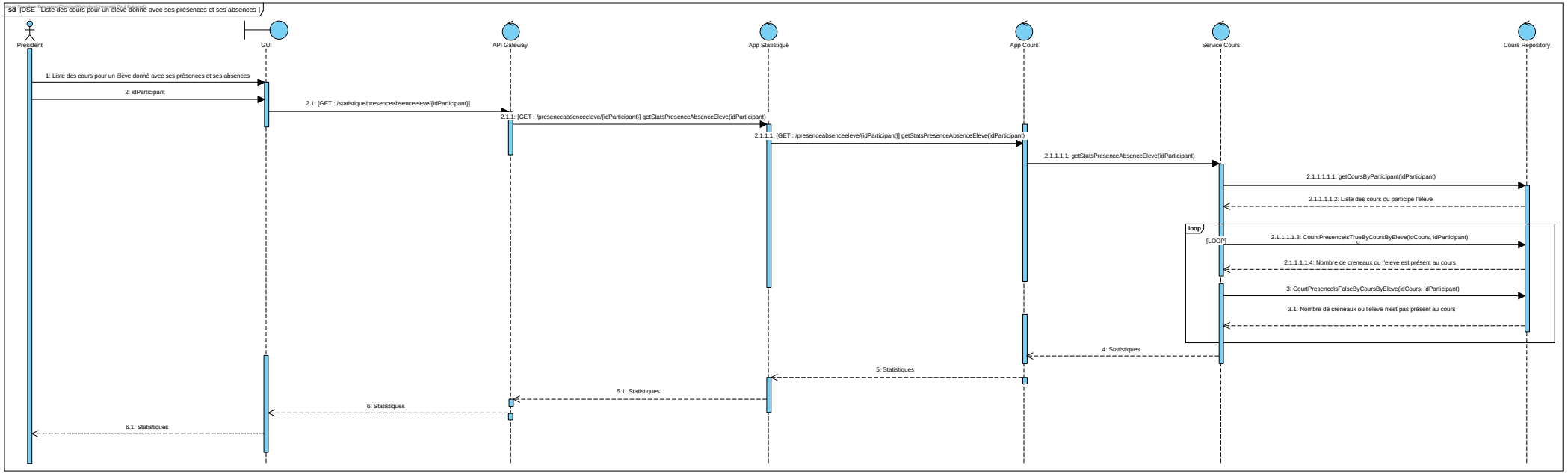


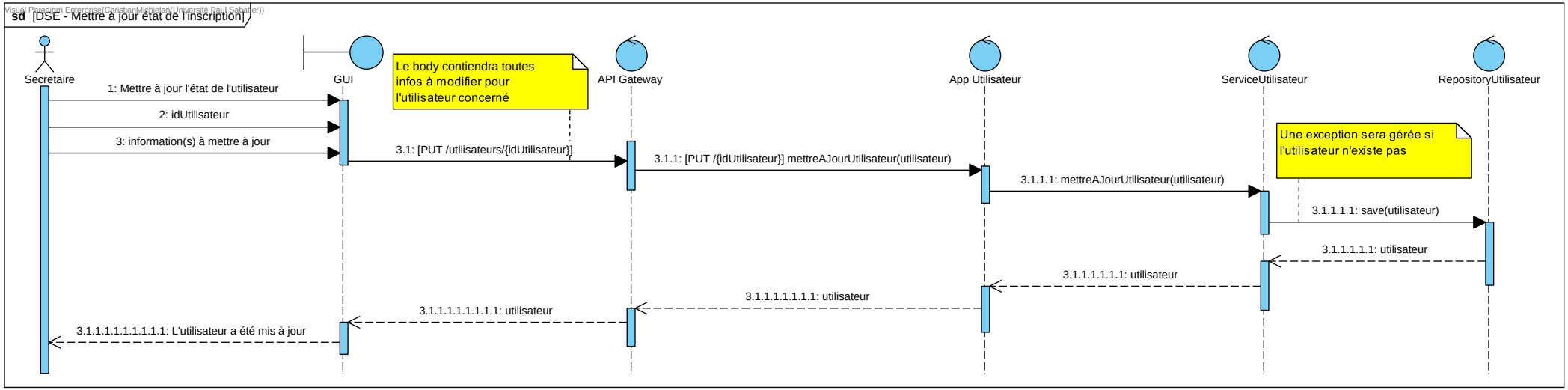

sd [DSE - Consulter les créneaux des cours suivis]
Membre du club
GUI
API Gateway
App Cours
ServiceCours
RepositoryCours
1: Consulter créneaux de cours suivis
1.1: [GET : /cours/creneau?idParticipant={idParticipant}]
1.1.1: [GET : cours/creneau/?idParticipant={idParticipant}] obtenirCreneauxSuivis(idParticipant)
1.1.1.1: obtenirCreneauxSuivi(idParticipant)
1.1.1.1.1: getCoursByCreneauxByIdParticipantAndIsPresent(idParticipant)
Une exception sera gérée si aucun cours n'est trouvé.
1.1.1.1.1.1: ListeDeCreneaux
1.1.1.1.1.1.1: ListeDeCreneaux
1.1.1.1.1.1.1.1: ListeDeCreneaux
1.1.1.1.1.1.1.1.1: ListeDeCreneaux
1.1.1.1.1.1.1.1.1.1: Afficher les créneaux associés

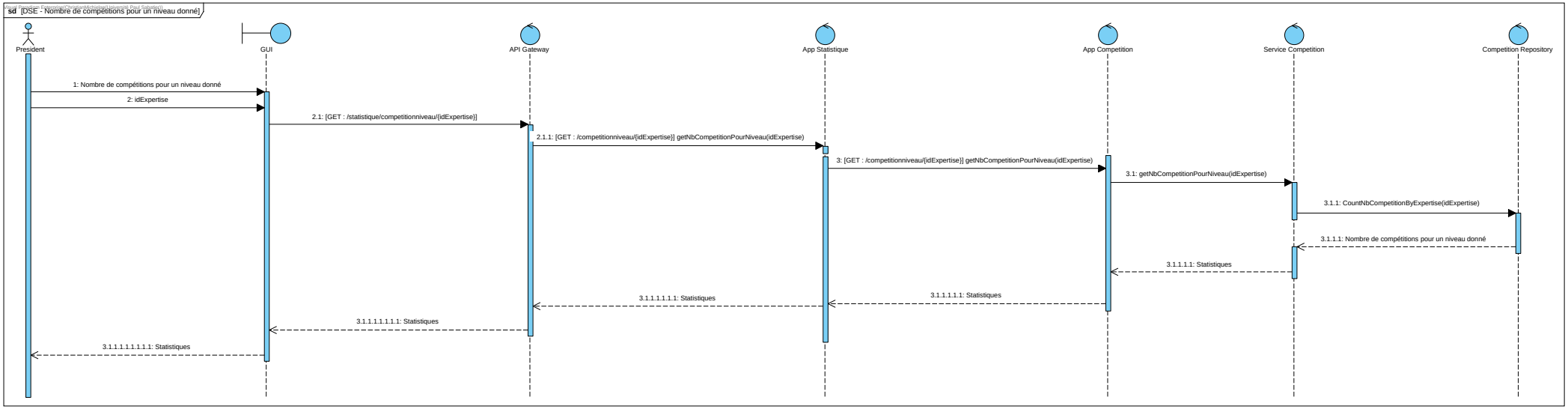


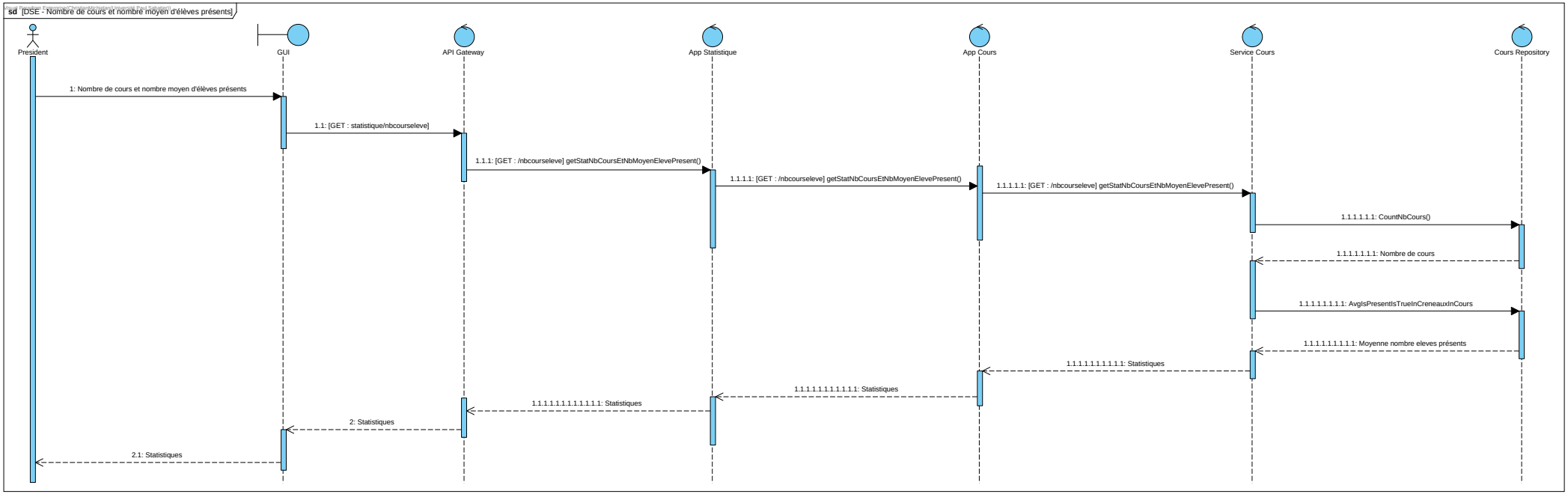


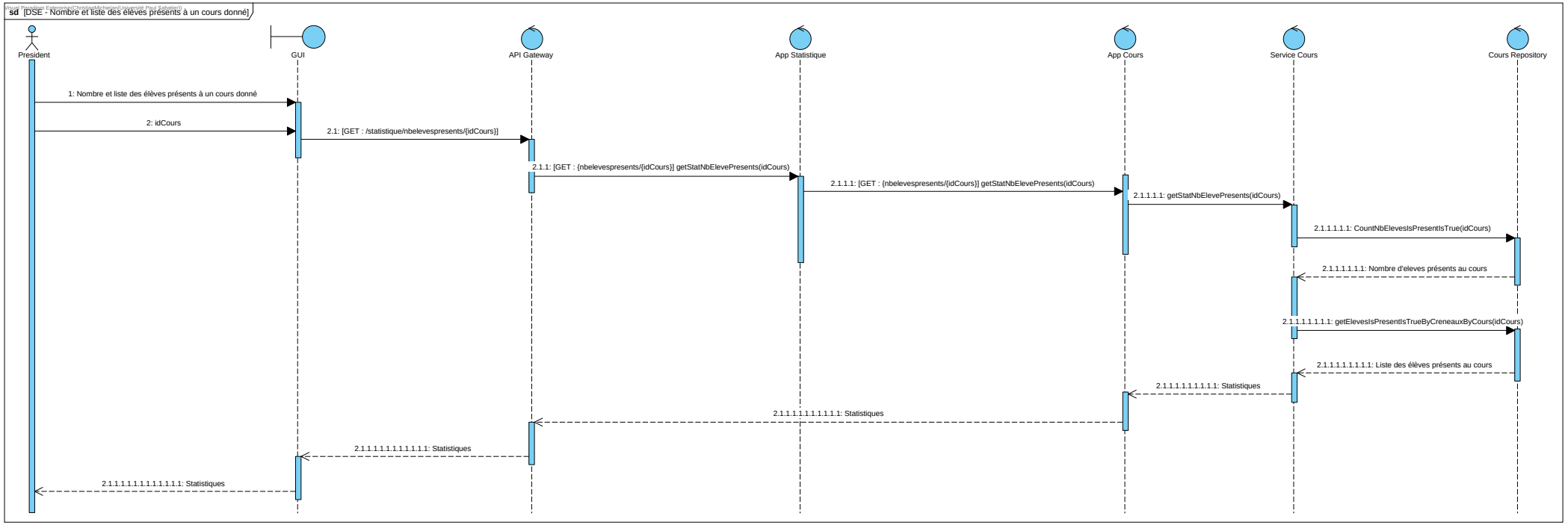

sd [DSE - Liste des compétitions pour un élève donné avec ses résultats]
President
GUI
API Gateway
App Statistique
App Competition
Service Cours
Cours Repository
1: Liste des compétitions pour un élève donné avec ses résultats
2: idParticipant
2.1: [GET : /statistique/competitionresultat/{idParticipant}]
2.1.1: [GET : /competitionresultat/{idEleve}] getCompetitionWithResult(idParticipant)
2.1.1.1: [GET : /competitionresultat/{idEleve}] getCompetitionWithResult(idParticipant)
2.1.1.1.1: getCompetitionWithResult(idParticipant)
2.1.1.1.1.1: getCompetitionAndResultByParticipant(idParticipant)
2.1.1.1.1.1.1: Liste des compétitions pour un élève donné avec ses résultats
2.1.1.1.1.1.1: Statistiques
2.1.1.1.1.1.1.1: Statistiques
2.1.1.1.1.1.1.1.1: Statistiques
2.1.1.1.1.1.1.1.1.1: Statistiques
2.1.1.1.1.1.1.1.1.1.1: Statistiques

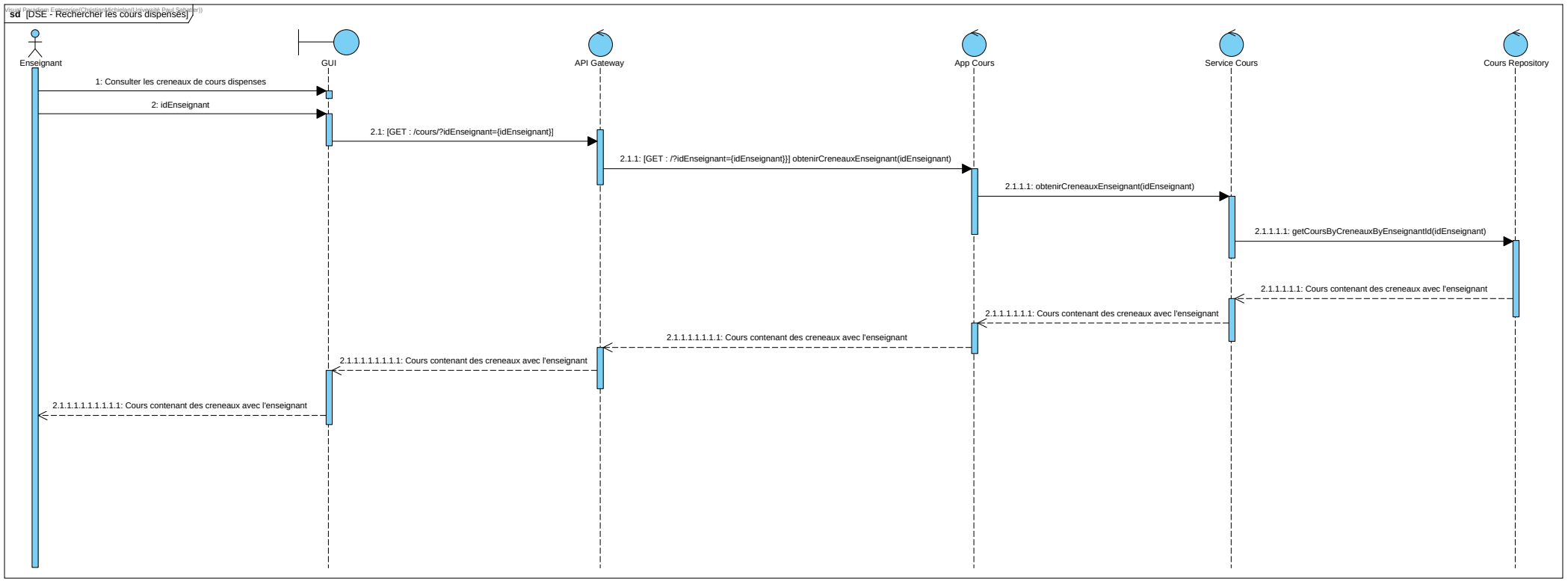

sd [DSE - Liste des cours pour un élève donné avec ses présences et ses absences]
Président
GUI
API Gateway
App Statistique
App Cours
Service Cours
Cours Repository
1: Liste des cours pour un élève donné avec ses présences et ses absences
2: idParticipant
2.1: [GET : /statistique/presenceabsenceeleve/{idParticipant}]
2.1.1: [GET : /presenceabsenceeleve/{idParticipant}] getStatsPresenceAbsenceEleve(idParticipant)
2.1.1.1: [GET : /presenceabsenceeleve/{idParticipant}] getStatsPresenceAbsenceEleve(idParticipant)
2.1.1.1.1: getStatsPresenceAbsenceEleve(idParticipant)
2.1.1.1.1.1: getCoursByParticipant(idParticipant)
2.1.1.1.1.2: Liste des cours ou participe l'élève
loop
[LOOP]
2.1.1.1.1.3: CountPresenceIsTrueByCoursByEleve(idCours, idParticipant)
2.1.1.1.1.4: Nombre de creneaux ou l'eleve est présent au cours
3: CourtPresenceIsFalseByCoursByEleve(idCours, idParticipant)
3.1: Nombre de creneaux ou l'eleve n'est pas présent au cours
4: Statistiques
5: Statistiques
5.1: Statistiques
6: Statistiques
6.1: Statistiques

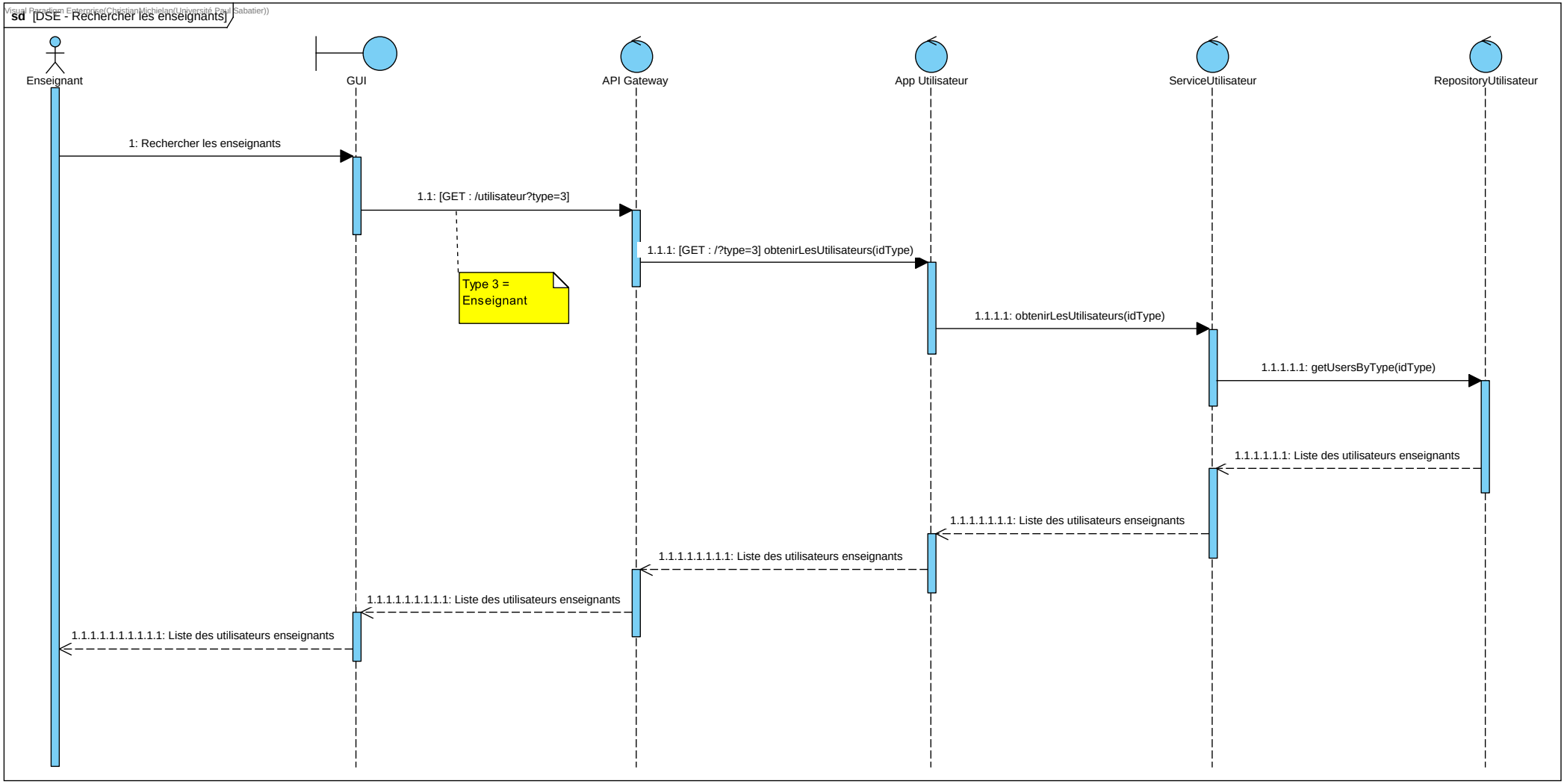


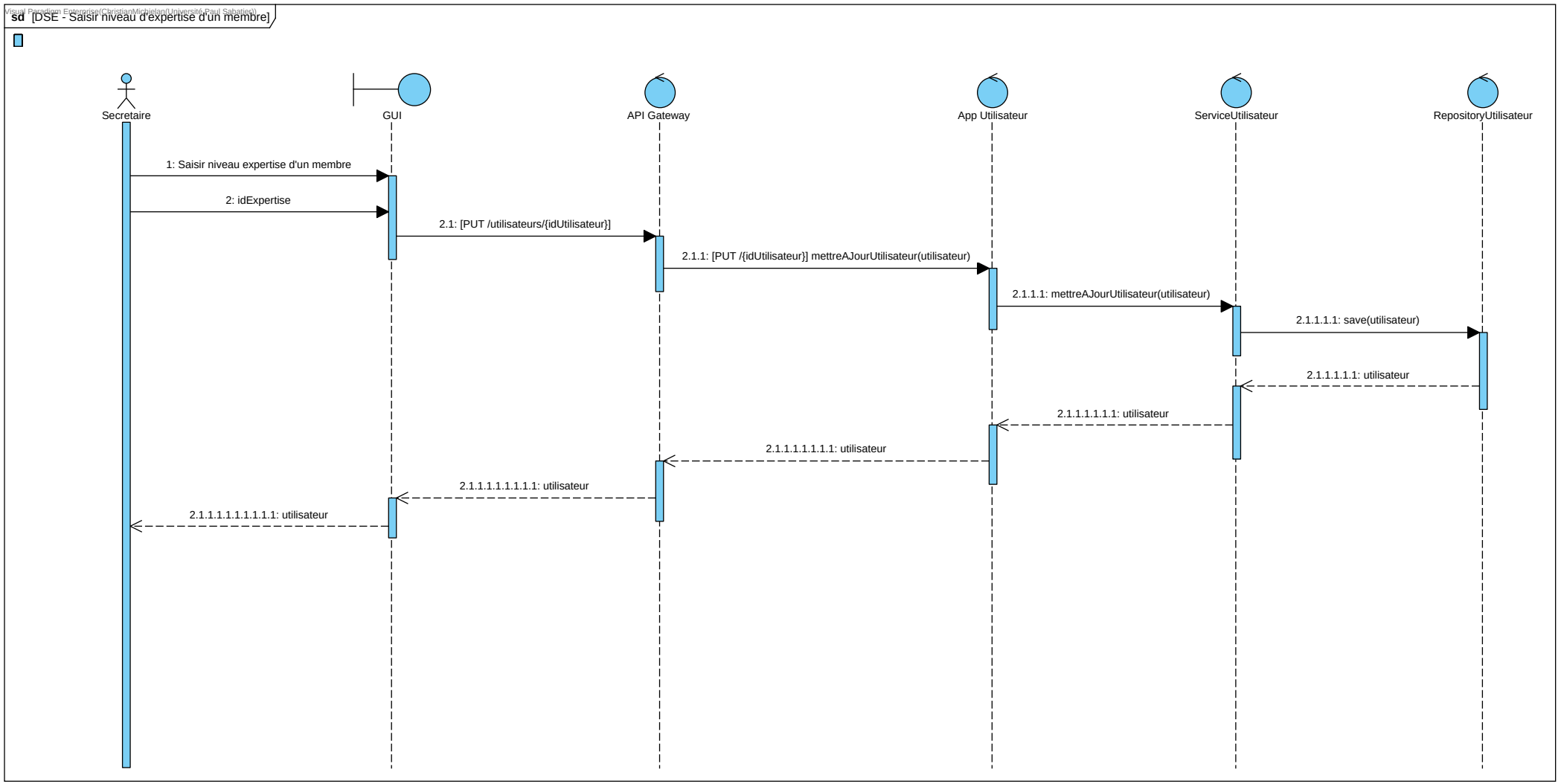

sd [DSE - Nombre de compétitions pour un niveau donné]
President
GUI
API Gateway
App Statistique
App Competition
Service Competition
Competition Repository
1: Nombre de compétitions pour un niveau donné
2: idExpertise
2.1: [GET : /statistique/competitionniveau/{idExpertise}]
2.1.1: [GET : /competitionniveau/{idExpertise}] getNbCompetitionPourNiveau(idExpertise)
3: [GET : /competitionniveau/{idExpertise}] getNbCompetitionPourNiveau(idExpertise)
3.1: getNbCompetitionPourNiveau(idExpertise)
3.1.1: CountNbCompetitionByExpertise(idExpertise)
3.1.1.1: Nombre de compétitions pour un niveau donné
3.1.1.1.1: Statistiques
3.1.1.1.1.1: Statistiques
3.1.1.1.1.1.1: Statistiques
3.1.1.1.1.1.1.1: Statistiques
3.1.1.1.1.1.1.1.1: Statistiques

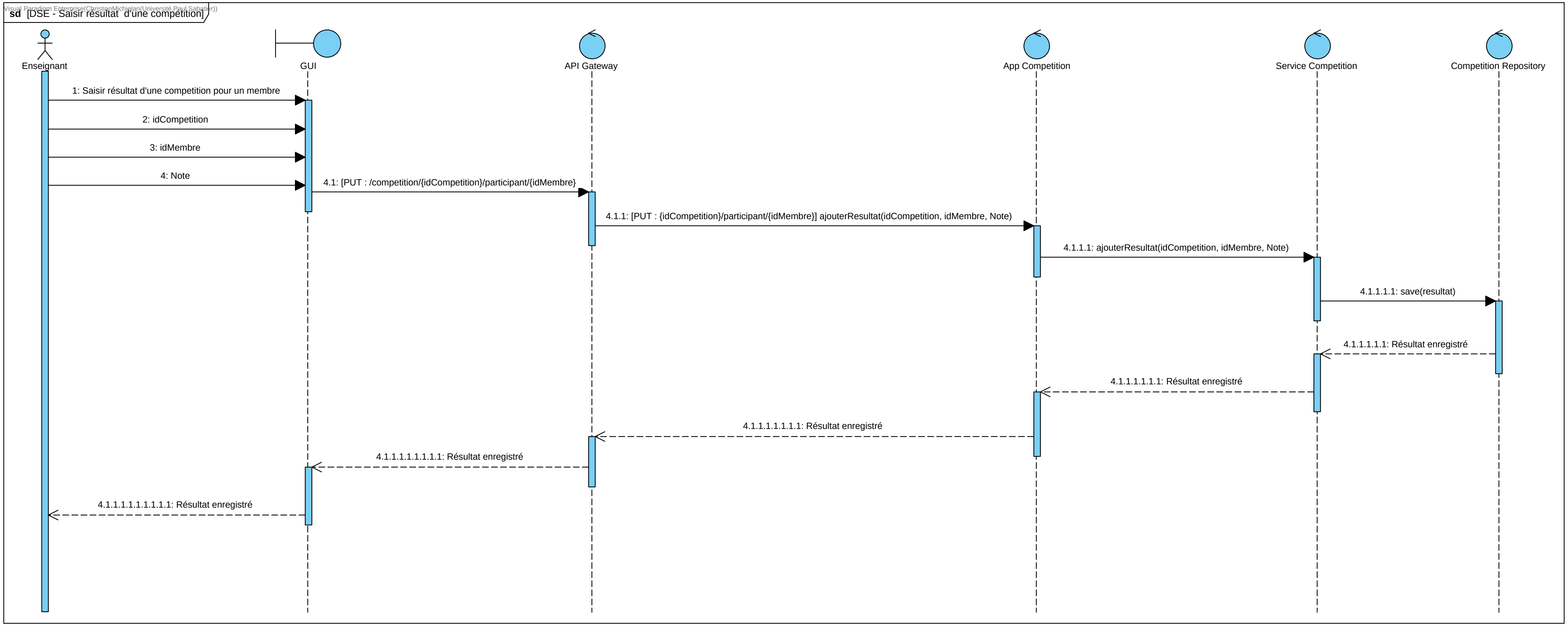

sd [DSE - Nombre de cours et nombre moyen d'élèves présents]
President
GUI
API Gateway
App Statistique
App Cours
Service Cours
Cours Repository
1: Nombre de cours et nombre moyen d'élèves présents
1.1: [GET : statistique/nbcourseleve]
1.1.1: [GET : /nbcourseleve] getStatNbCoursEtNbMoyenElevePresent()
1.1.1.1: [GET : /nbcourseleve] getStatNbCoursEtNbMoyenElevePresent()
1.1.1.1.1: [GET : /nbcourseleve] getStatNbCoursEtNbMoyenElevePresent()
1.1.1.1.1.1: CountNbCours()
1.1.1.1.1.1.1: Nombre de cours
1.1.1.1.1.1.1.1: AvgIsPresentIsTrueInCreneauxInCours
1.1.1.1.1.1.1.1.1: Moyenne nombre eleves présents
1.1.1.1.1.1.1.1.1.1: Statistiques
1.1.1.1.1.1.1.1.1.1.1: Statistiques
1.1.1.1.1.1.1.1.1.1.1.1: Statistiques
2: Statistiques
2.1: Statistiques

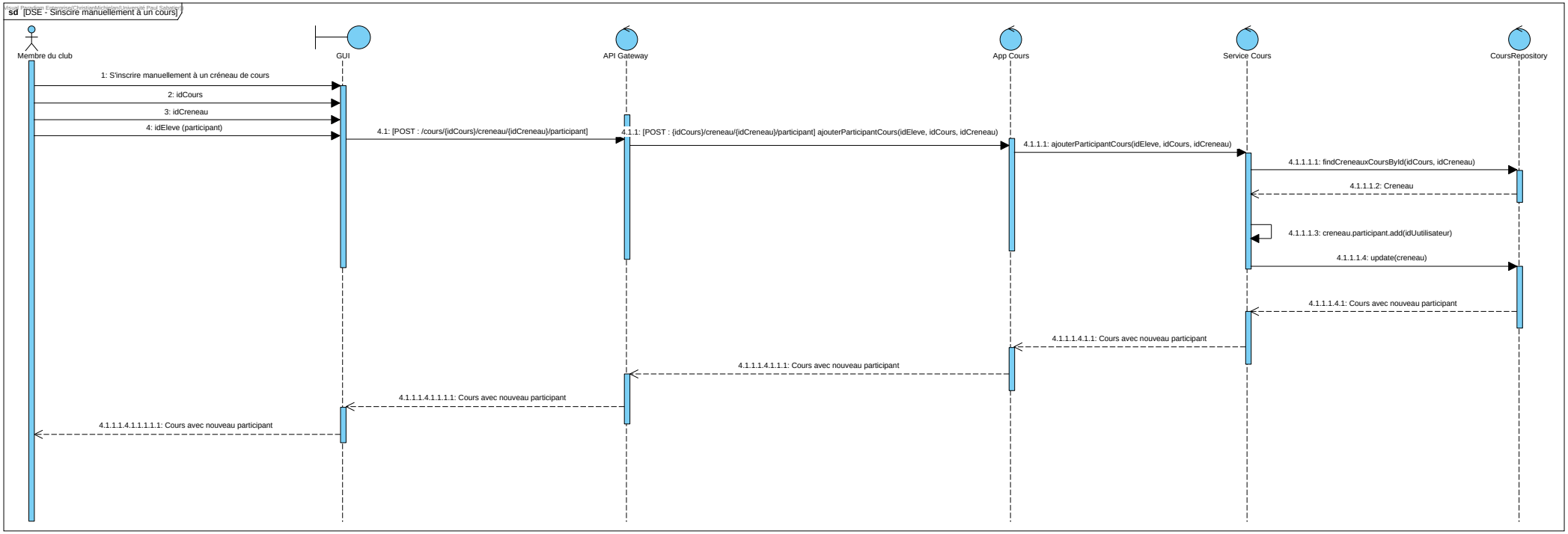

sd [DSE - Nombre et liste des élèves présents à un cours donné]
President
GUI
API Gateway
App Statistique
App Cours
Service Cours
Cours Repository
1: Nombre et liste des élèves présents à un cours donné
2: idCours
2.1: [GET : /statistique/nbelevespresents/{idCours}]
2.1.1: [GET : {nbelevespresents/{idCours}] getStatNbElevePresents(idCours)
2.1.1.1: [GET : {nbelevespresents/{idCours}] getStatNbElevePresents(idCours)
2.1.1.1.1: getStatNbElevePresents(idCours)
2.1.1.1.1.1: CountNbElevesIsPresentIsTrue(idCours)
2.1.1.1.1.1.1: Nombre d'eleves présents au cours
2.1.1.1.1.1.1: getElevesIsPresentIsTrueByCreneauxByCours(idCours)
2.1.1.1.1.1.1.1: Liste des élèves présents au cours
2.1.1.1.1.1.1.1.1: Statistiques
2.1.1.1.1.1.1.1.1.1: Statistiques
2.1.1.1.1.1.1.1.1.1.1: Statistiques
2.1.1.1.1.1.1.1.1.1.1.1: Statistiques

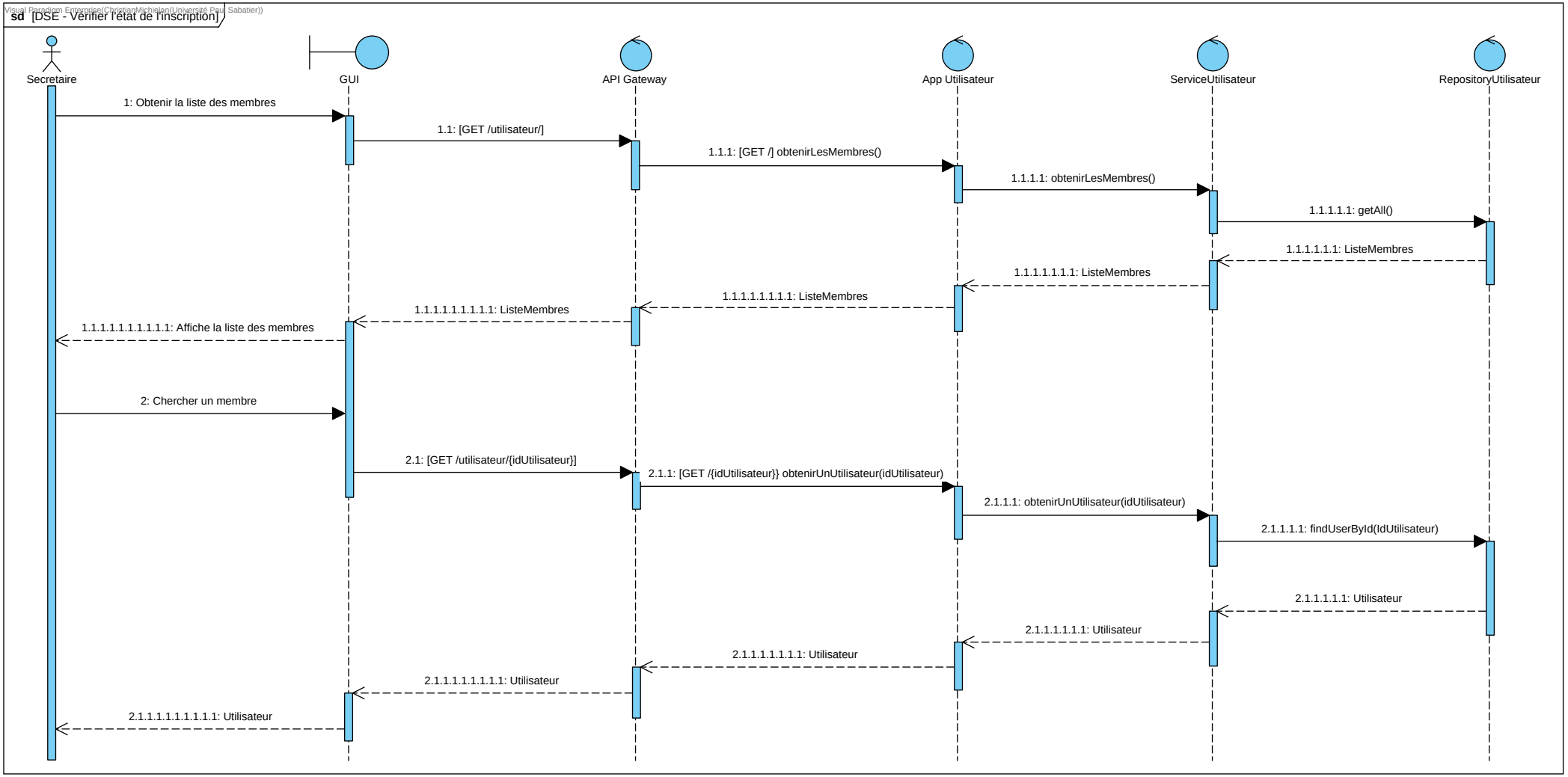

sd [DSE - Rechercher les cours dispenses]
Enseignant
GUI
API Gateway
App Cours
Service Cours
Cours Repository
1: Consulter les creneaux de cours dispenses
2: idEnseignant
2.1: [GET : /cours/?idEnseignant={idEnseignant}]
2.1.1: [GET : /?idEnseignant={idEnseignant}}] obtenirCreneauxEnseignant(idEnseignant)
2.1.1.1: obtenirCreneauxEnseignant(idEnseignant)
2.1.1.1.1: getCoursByCreneauxByEnseignantId(idEnseignant)
2.1.1.1.1.1: Cours contenant des creneaux avec l'enseignant
2.1.1.1.1.1.1: Cours contenant des creneaux avec l'enseignant
2.1.1.1.1.1.1.1: Cours contenant des creneaux avec l'enseignant
2.1.1.1.1.1.1.1.1: Cours contenant des creneaux avec l'enseignant
2.1.1.1.1.1.1.1.1.1: Cours contenant des creneaux avec l'enseignant

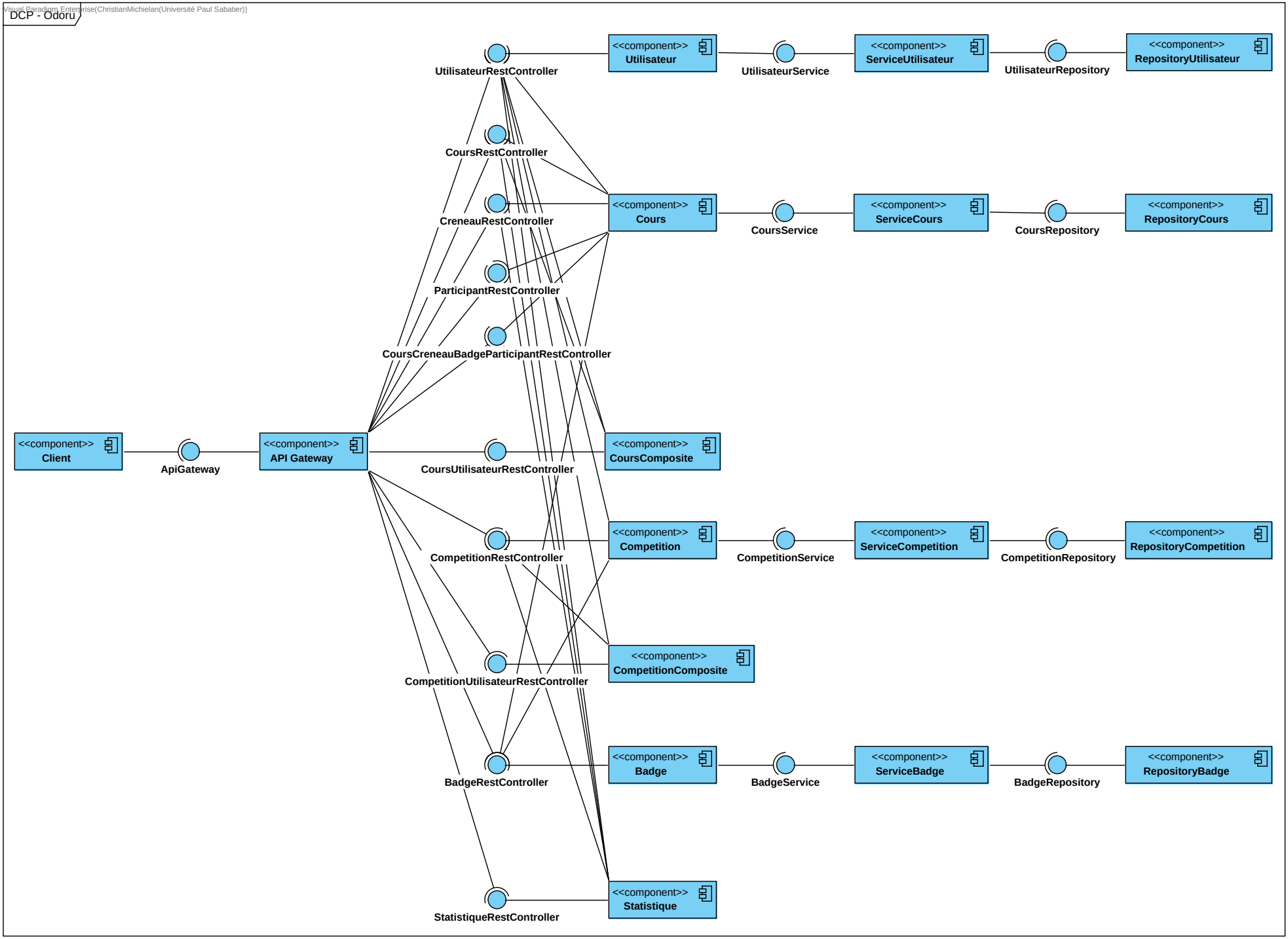



Visual Paradigm Enterprise(ChristianMichelan(Université Paul Sabatier))
sd [DSE - Saisir niveau d'expertise d'un membre]
Secretaire
GUI
API Gateway
App Utilisateur
ServiceUtilisateur
RepositoryUtilisateur
1: Saisir niveau expertise d'un membre
2: idExpertise
2.1: [PUT /utilisateurs/{idUtilisateur}]
2.1.1: [PUT /{idUtilisateur}] mettreAJourUtilisateur(utilisateur)
2.1.1.1: mettreAJourUtilisateur(utilisateur)
2.1.1.1.1: save(utilisateur)
2.1.1.1.1.1: utilisateur
2.1.1.1.1.1.1: utilisateur
2.1.1.1.1.1.1.1: utilisateur
2.1.1.1.1.1.1.1.1: utilisateur
2.1.1.1.1.1.1.1.1.1: utilisateur

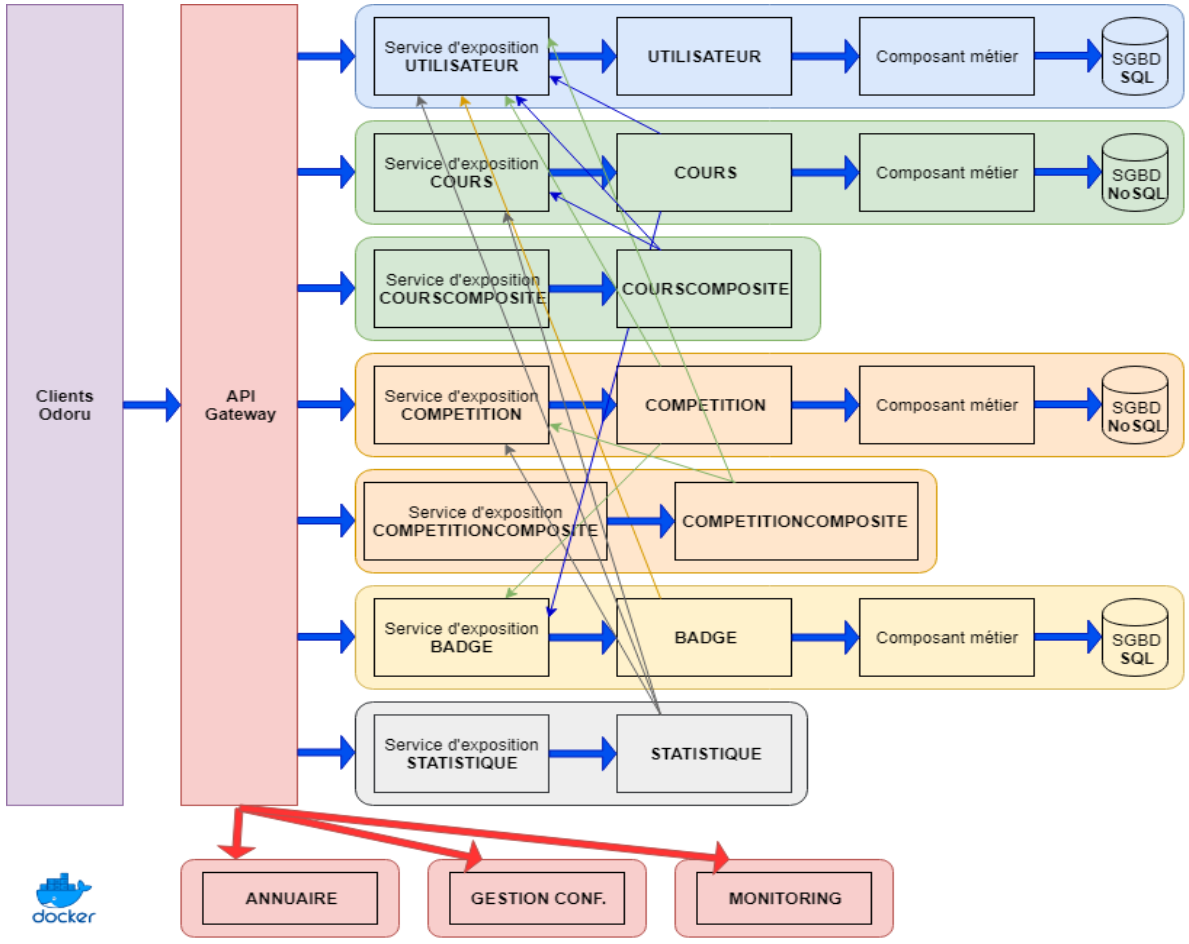

Visual Paradigm Enterprise(ChristianMichielan(Université Paul Sabatier))
sd [DSE - Saisir résultat d'une compétition]
Enseignant
GUI
API Gateway
App Competition
Service Competition
Competition Repository
1: Saisir résultat d'une competition pour un membre
2: idCompetition
3: idMembre
4: Note
4.1: [PUT : /competition/{idCompetition}/participant/{idMembre}
4.1.1: [PUT : {idCompetition}/participant/{idMembre}] ajouterResultat(idCompetition, idMembre, Note)
4.1.1.1: ajouterResultat(idCompetition, idMembre, Note)
4.1.1.1.1: save(resultat)
4.1.1.1.1.1: Résultat enregistré
4.1.1.1.1.1.1: Résultat enregistré
4.1.1.1.1.1.1.1: Résultat enregistré
4.1.1.1.1.1.1.1.1: Résultat enregistré
4.1.1.1.1.1.1.1.1.1: Résultat enregistré


sd [DSE - S'inscrire manuellement à un cours]
Membre du club
GUI
API Gateway
App Cours
Service Cours
CoursRepository
1: S'inscrire manuellement à un créneau de cours
2: idCours
3: idCreneau
4: idEleve (participant)
4.1: [POST : /cours/{idCours}/creneau/{idCreneau}/participant]
4.1.1: [POST : {idCours}/creneau/{idCreneau}/participant] ajouterParticipantCours(idEleve, idCours, idCreneau)
4.1.1.1: ajouterParticipantCours(idEleve, idCours, idCreneau)
4.1.1.1.1: findCreneauxCoursById(idCours, idCreneau)
4.1.1.1.2: Creneau
4.1.1.1.3: creneau.participant.add(idUutilisateur)
4.1.1.1.4: update(creneau)
4.1.1.1.4.1: Cours avec nouveau participant
4.1.1.1.4.1.1: Cours avec nouveau participant
4.1.1.1.4.1.1.1: Cours avec nouveau participant
4.1.1.1.4.1.1.1.1: Cours avec nouveau participant
4.1.1.1.4.1.1.1.1.1: Cours avec nouveau participant





3. Architecture micro-services



Pour réaliser ce projet avec une architecture micro-service, nous avons développé et déployé 8 projets :

- **API-Gatway :** Service qui offre une interface commune pour interragir avec les autres services.
- **Annuaire :** Service qui enregistre les différents micro-services au sein d'un annuaire Eureka Server.
- **BadgeService :** Service qui s'occupe de la gestion des badges.
- **CompetitionService :** Service qui s'occupe de la gestion des compétitions.
- **CompetitionCompositeService :** Service qui s'occupe d'agréger les données entre le service Compétition et le service Utilisateur.
- **ConfigServer :** Service qui s'occupe de la gestion des configurations.
- **CoursService :** Service qui s'occupe de la gestion des cours.
- **CoursCompositeService :** Service qui s'occupe d'agréger les données entre le service Cours et le service Utilisateur.
- **StatistiqueService :** Service qui s'occupe de la gestion des statistiques.
- **UtilisateurService :** Service qui s'occupe de la gestion des utilisateurs.

Concernant le choix des bases de données, nous avons choisi d'utiliser des bases de données NoSQL pour stocker les objets qui nécessitent une structure complexe avec imbrication. Notre choix s'est porté sur MongoDB car il s'agit de celle que nous connaissons le mieux mais également car elle permet au travers des collections de documents de persister et d'utiliser plus facilement des structures complexes d'objets. Pour les objets plus simples à manipuler car ne nécessitant pas une structure imbriquée, notre choix s'est porté vers des bases de données relationnelles avec MySQL.

7. Open API

Documentation de l’API accessible à l’url suivante :

https://app.swaggerhub.com/apis-docs/christian.michielan/Odoru/1.0.0#/

7. Correspondances entre identifiants applicatifs et labels

Pour manipuler notre application et permettre l’affichage des données, la future application qui doit consommer nos API pour son IHM devra connaître la correspondance entre nos identifiants de type d’utilisateurs, nos niveaux d'expertises et leurs labels associés.

7.1 Types d’utilisateurs Odoru

Nous distinguerons pour nos données les correspondances suivantes concernant les types d’utilisateurs :

N°	Label
1	Membre
2	Secrétaire
3	Enseignant
4	Président

7.2 Niveaux d’expertise des cours Odoru

Nous distinguerons pour nos données les correspondances suivantes concernant les niveaux d’expertise des cours et des utilisateurs :

N°	Label
1	Débutant
2	Moyen
3	Expert