**Rapport de conception 21/03/12**

[Sous-titre du document]

Anthony Morales, Chen-Yang Gao, Irvin genieys, Simon Maby

Table des matières

[I. Introduction 4](#_Toc320209345)

[II. Conception de la Base de données 4](#_Toc320209346)

[a. Schéma de la base de données 4](#_Toc320209347)

[b. Vues et types 4](#_Toc320209348)

[III. Conception logiciel 6](#_Toc320209349)

[a. Diagramme des classes 6](#_Toc320209350)

[b. Diagrammes de séquence 11](#_Toc320209351)

[c. Persistance des données 20](#_Toc320209352)

[IV. Conclusion 21](#_Toc320209353)

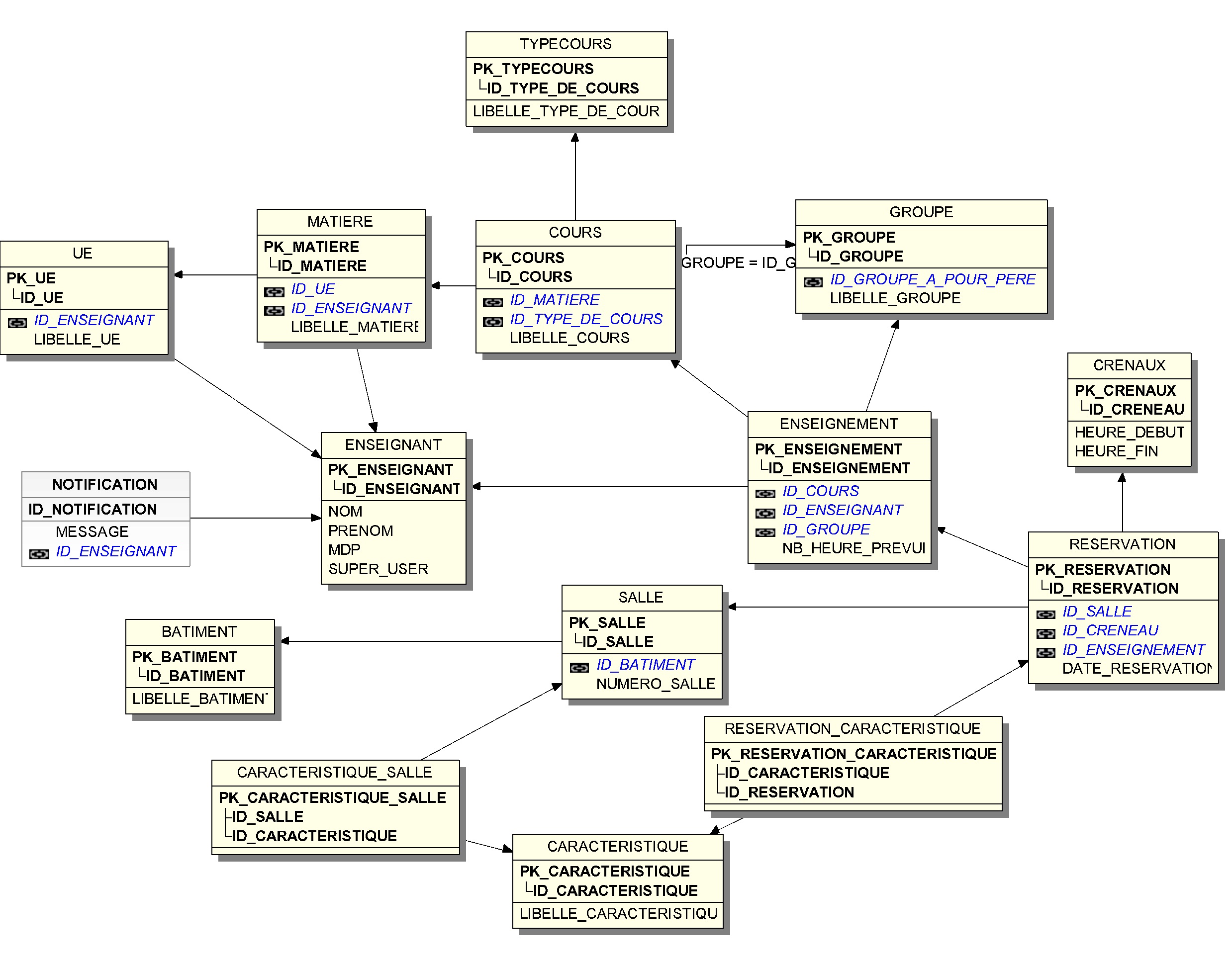
[V. Annexes 22](#_Toc320209354)

# Introduction

Nous présenterons dans ce rapport le compte rendu détaillé de la conception du logiciel Slab. Nous présenterons dans un premier temps la conception de la base de données puis nous présenterons la conception objet du logiciel.

# Conception de la Base de données

## Schéma de la base de données



Le schéma précédent présente le modèle logique de notre base de données. Nous avons décidé d’ajouter la table notification qui contient un message et référence un enseignant. Cette table sert à donc à stocker tout nouvel événement dont doit être informé un enseignant comme la réponse à l’un de ses demandes.

Nous avons dans un premier temps considéré qu’une notification devait avoir une référence vers une réservation cependant nous avons décidé de la remplacer par une référence vers un enseignant car avec notre première idée nous n’aurions pas pu représenter les notifications de suppression de réservation.

## Vues et types

De manière à utiliser notre base de données plus simplement nous avons décidé de créer des types objets représentant les tables de la base de données ainsi que des vues sur ces types.

La vue principalement utilisée dans notre projet est la vue VEnseignant qui représente la table Enseignant sous forme de vue objet. Cette vue nous permet d’avoir simplement pour un enseignant la liste de ses enseignements et de ses réservations. Celle-ci nous a grandement facilité un certain nombre de requêtes notamment pour connaitre le planning d’un enseignant. Nous avons donc ainsi évité d’effectuer des jointures compliqués.

Nous avons créé une vue VSalle ainsi qu’une vue réservation qui nous permettent d’avoir pour une salle et pour une réservation la liste de leurs caractéristiques.

De plus l’application étant développé en java et donc dans un langage objet, l’utilisation des vues était cohérente et simplifié le lien entre l’application et la base de données.

Ces types sont disponibles dans le fichier types.sql. (cf annexe).

Représentation de la vue Enseignant :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID\_enseignant | Nom | Prenom | Enseignement 1 | Réservation 1 |
| Enseignement 2 | Réservation 2 |
| Enseignement 3 | Réservation 3 |
| Enseignement 4 | Réservation 4 |

Représentation de la vue VReservation :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID\_Reservation | Salle | Créneau | Enseignement | Date | Caractéristique1 |
| Caractéristique2 |
| Caractéristique3 |

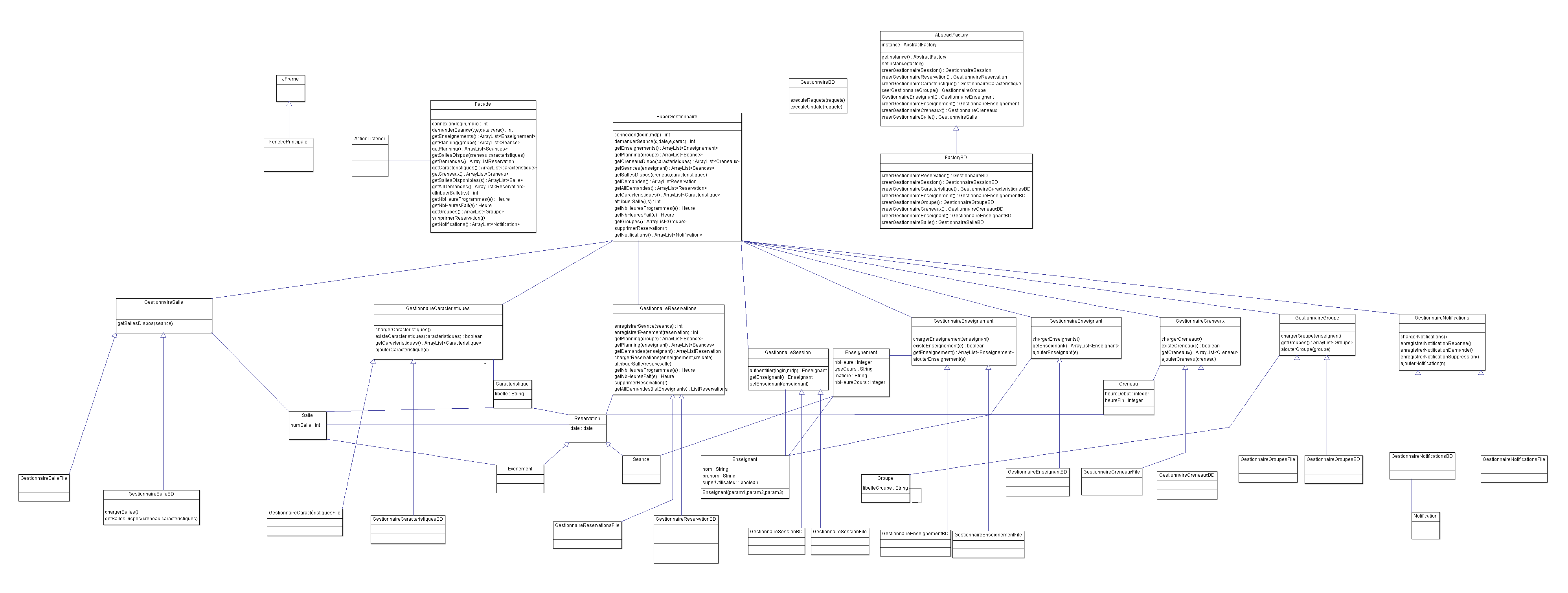
Représentation de la vue VSalle :

|  |  |  |
| --- | --- | --- |
| ID\_Salle | Numero\_Salle | Caractéristique1 |
| Caractéristique2 |
| Caractéristique3 |

# Conception logiciel

## Diagramme des classes

Nous décrirons dans cette partie les classes principales de notre modèle et leur rôle. Nous expliquerons notamment le principe des gestionnaire qui se calque sur le modèle des courtiers. Tous les gestionnaires sont abstraits et devront être implémentés pour un type de données persistantes spécifique, dans notre cas une base de données et des fichiers.



Facade :

Cette classe sert de point d’entrée dans la business logic, elle simplifie donc l’utilisation des différentes fonctionnalités de notre programme à une interface utilisateur.

Nous avons choisi de ne conserver qu’une seule façade pour notre application, en effet après de longues réflexions nous avons estimé que le but premier de cette classe est de simplifier l’utilisation des classes métiers et que le fait de la découper ne ferait que compliquer l’utilisation du programme par une interface utilisateur, en effet cette dernière aura accès à toutes les fonctionnalités via une seule classe.

De plus lors de la conception de nos maquettes nous avons remarqué que chaque fonctionnalité correspondait plus ou moins à un onglet et que toutes les fonctionnalités étaient liées et qu’il y a donc un sens à ce qu’elles soient accessibles au travers de la même interface. Cependant si nous devions développer de nouvelles fonctionnalités ne s’apparentant pas à la gestion de l’emploi du temps (ex : ajout d’une enseignant…) nous utiliserions dans ce cas une façade différente.

SuperGestionnaire :

La classe de SuperGestionnaire sert à superviser le travail de tous les autres gestionnaires, en effet le rôle de la façade étant simplement de fournir un point d’entrée pour accéder aux classes métier nous ne voulions pas que celle-ci ai du traitement à effectuer. Nous avons donc introduit une classe entre la façade et les différents gestionnaire de manière à ce qu’il existe une classe capable de contrôler le travail des gestionnaire et de faire en sorte de contrôler les données venant de l’interface utilisateur (via la façade) de façon à ce que les gestionnaires n’aient pas à se préoccuper de ce problème et de simplement faire les taches qui leurs sont assignées.

La seconde raison pour laquelle nous avons choisi d’introduire cette classe est par souci de découplage. En effet nous voulions éviter que les gestionnaires se connaissent entre eux ce qui aurait introduit un couplage fort dans notre programme, ainsi seul le super gestionnaire a connaissance des autres gestionnaires et fait en sorte de leur donner les informations dont ils ont besoin.

Par exemple pour obtenir le planning de l’enseignant connecté à l’application, le super gestionnaire va d’abord demander au gestionnaire de session l’instance de l’enseignant connecté avant de la fournir au gestionnaire de réservations pour obtenir l’ensemble de ses séances.

GestionnaireSession :

Cette classe sert à garder une instance en mémoire de l’enseignant connecté à l’application. La présence de cette classe simplifie notamment l’utilisation des différentes fonctionnalités, par exemple pour avoir le planning personnel de l’enseignant connecté l’interface utilisateur n’aura pas à fournir l’enseignant mais le super gestionnaire ira chercher cette information dans cette classe.

Sa principale fonction est donc de fournir l’enseignant connecté.

GestionnaireRéservations :

Cette classe sera utilisée pour effectuer des taches en rapport avec tous les types de réservations comme les séances ou les évènements.

Elle permettra d’obtenir le planning d’un enseignant (l’ensemble de ses séances), la liste des demandes d’un enseignant mais aussi de supprimer une réservation ou d’attribuer une salle à une réservation.

GestionnaireCaracteristiques :

La classe GestionnaireCaractéristiques fournit des méthodes permettant de gérer les caractéristiques des salles. Son principal rôle est de stocker en mémoire l’ensemble des caractéristiques présentes dans les données persistantes, cela permettra de réduire la communication entre le programme et les données brutes, d’autant plus que les caractéristiques sont des données qui ne changeront pas au cours de l’exécution de l’application ce qui explique leur durée de vie dans notre programme.

Le super gestionnaire pourra donc demander ces caractéristiques pour permettre à l’interface utilisateur de les afficher.

GestionnaireSalle :

Cette classe accomplie des taches propres aux salles, la principale fonctionnalité de cette classe est de fournir la liste des salles disponibles pour une réservation donnée.

GestionnaireEnseignements :

La classe gestionnaire enseignement est chargé de stocker en mémoire les enseignements de l’enseignant connecté. De la même manière que pour les caractéristiques les enseignements sont des données qui ne changeront pas au cours de l’exécution, nous les chargeons donc en mémoire et leur durée de vie est celle de l’application.

GestionnaireEnseignants :

Cette classe fournit la liste des enseignants stockés dans les données persistantes, elle ne sera utilisé que si l’enseignant connecté est un responsable.

GestionnaireCreneaux :

La classe GestionnaireCreneaux conserve la liste des créneaux, cette liste sera chargé lors de la connexion de l’enseignant et conservé tout au long de l’exécution du programme.

GestionnaireGroupes :

Cette classe fournit l’ensemble des groupes liés à l’enseignant connecté qui seront chargés lors de son authentification.

GestionnaireNotifications :

La classe GestionnaireNotification est en charge des taches se rapportant aux notifications, elle permet notamment d’enregistrer des notifications dans les données persistantes.

GestionnaireBD :

Cette classe fournit une connexion avec une base de donnée et fournit deux méthodes permettant d’effectuer des requètes. La premiere sert à exécuter une requete d’intérrogation et la seconde une requete de modification.

Pour maintenir une et une seule connexion au cours de l’exécution et être accéssible par tous les gestionnaires cette classe suivra le patron de conception singleton.

AbstractFactory :

La classe AbstractFactory représente une fabrique abstraite d’objets et permettra de construire les gestionnaires appropriés selon la nature des données persistantes que l’on veut utiliser.

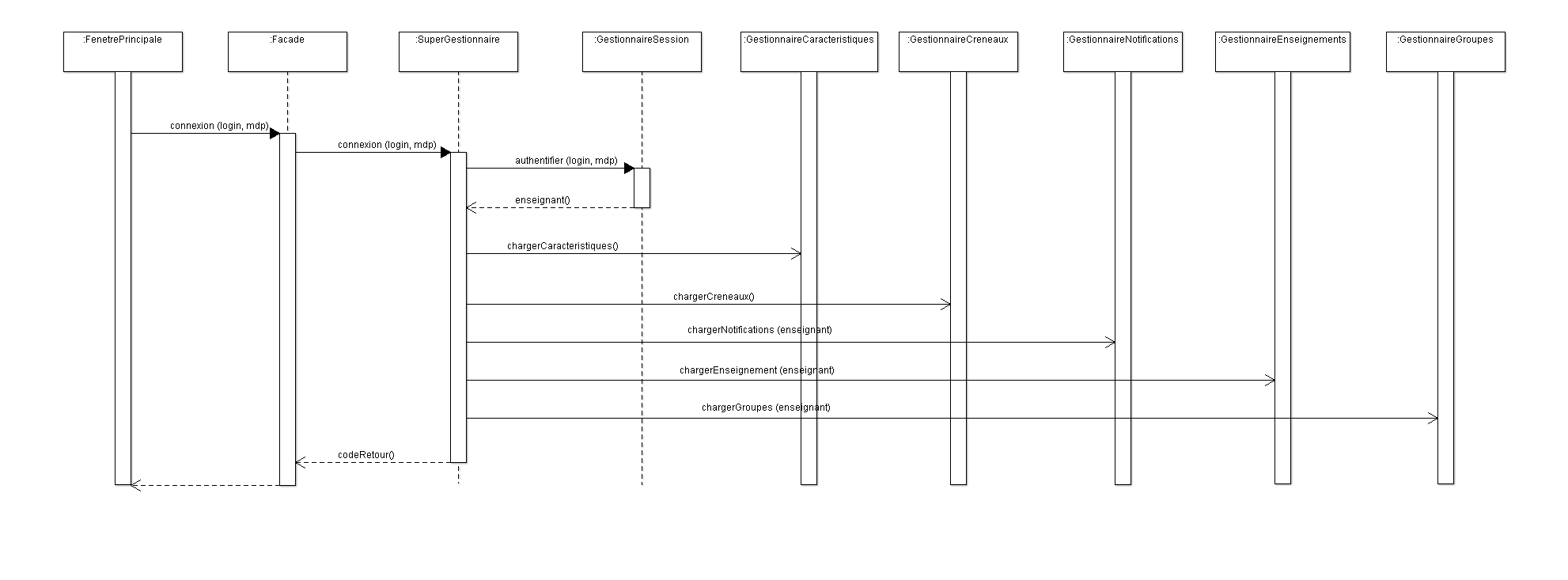
FactoryBD :

Cette classe implémente les méthodes de la classe AbstractFactory et permet de construire des gestionnaires capables d’interagir avec une base de données.

Enfin les classes Créneau, Salle, Enseignant, Enseignement, Notification, Caracteristique, Reservation, Seance, Evenement et Groupe servent à représenter en mémoire les données stockées en mémoire persistante.

## Diagrammes de séquence

Connexion :

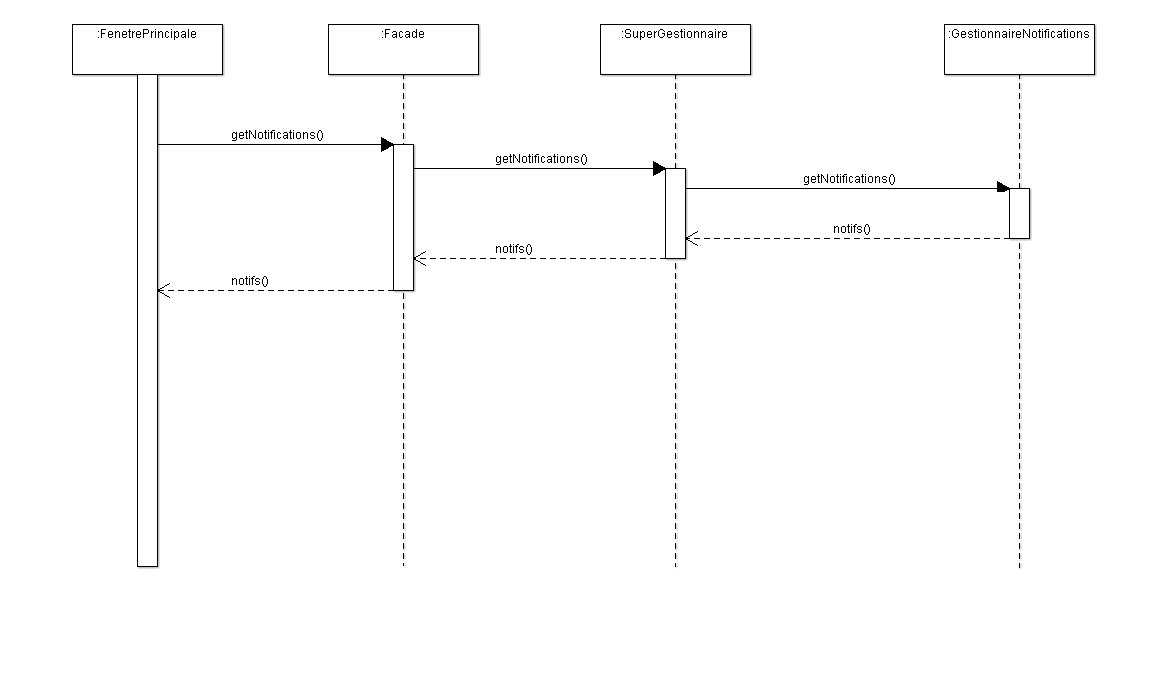


Pour se connecter un utilisateur, la classe FenetrePrincipale appelle chez la façade la méthode « connexion » dont le seul but est de demander au super gestionnaire la connexion de l’utilisateur en lui donnant son identifiant et son mot de passe. Le super gestionnaire prend ensuite le relais et demande au gestionnaire de session d’authentifier l’utilisateur en allant vérifier dans les données persistantes l’existence de celui-ci et le cas échéant construit un objet Enseignant et le donne au super gestionnaire.

Si le gestionnaire de session a accompli l’authentification avec succès nous allons charger en mémoire les données dont aura besoin l’utilisateur et qui ne changerons pas au cours de l’exécution c’est-à-dire les caractéristiques des salles, les créneaux, les enseignements, les groupes et les notifications de l’utilisateur. Ces chargements sont délégués par le super gestionnaire aux gestionnaires responsables de ces objets.

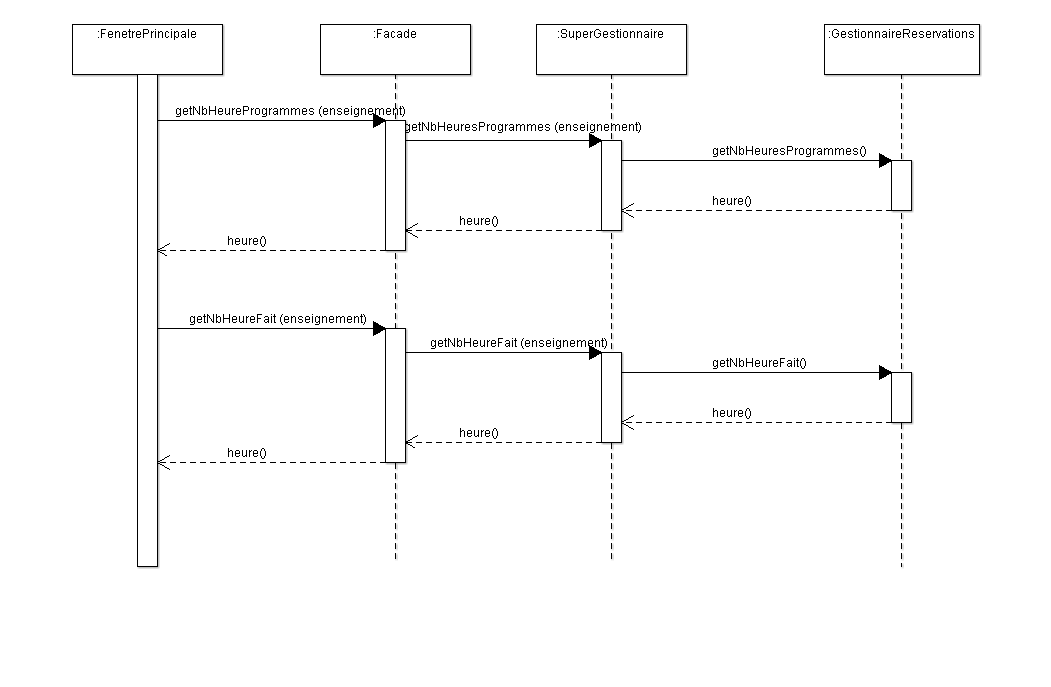
Après cela le super gestionnaire examine l’enseignant retourné par le gestionnaire de session et envoi un code de retour à la façade pour qu’elle puisse communiquer à la fenêtre principale si l’authentification a réussi et si oui, si l’enseignant est une responsable.

Visualisation des notifications :



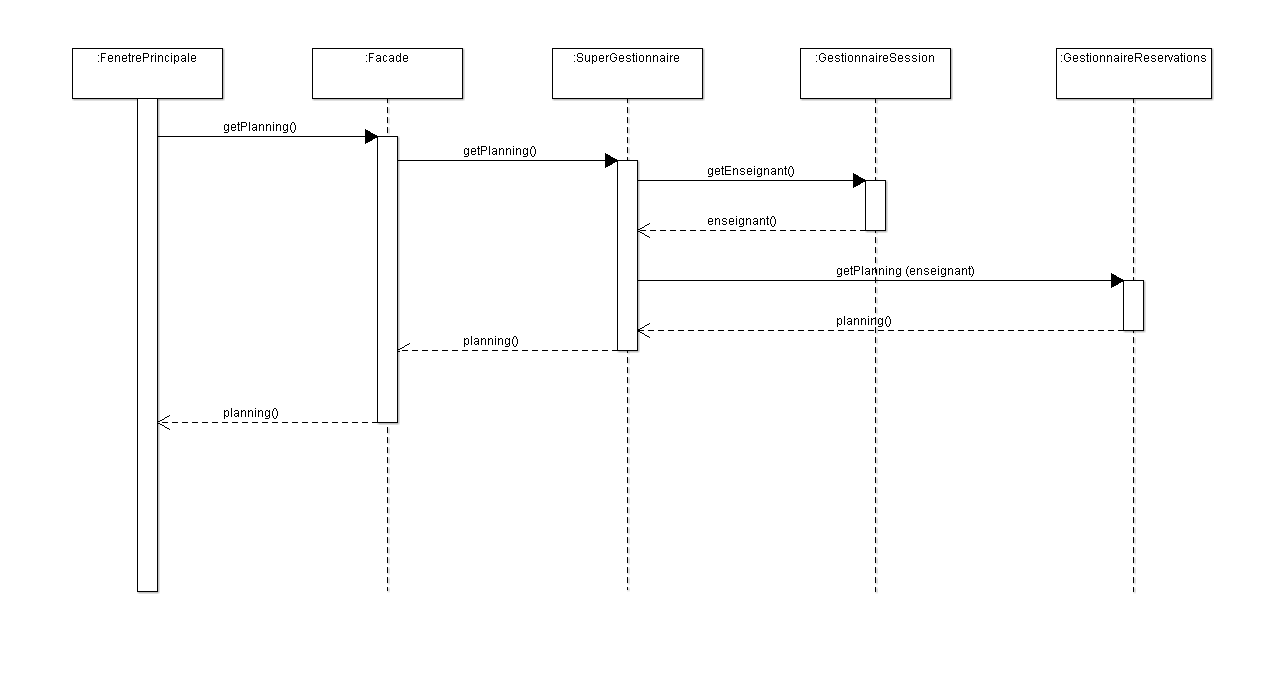
Pour visualiser les notifications, l’interface utilisateur va demander à la façade la liste des notifications de l’enseignant via la méthode getNotifications(). La façade n’étant qu’un point d’entrée pour l’utilisation des classes métiers elle demande simplement au super gestionnaire de lui donner la liste des notifications de l’enseignant connecté. Etant donné que les notifications font partie des objets que nous chargeons dès la connexion le super gestionnaire n’a qu’à les récupérer en les demandant au gestionnaire de notification qui garde ces connexions en mémoire depuis la connexion.

Visualisation de l’état du service :



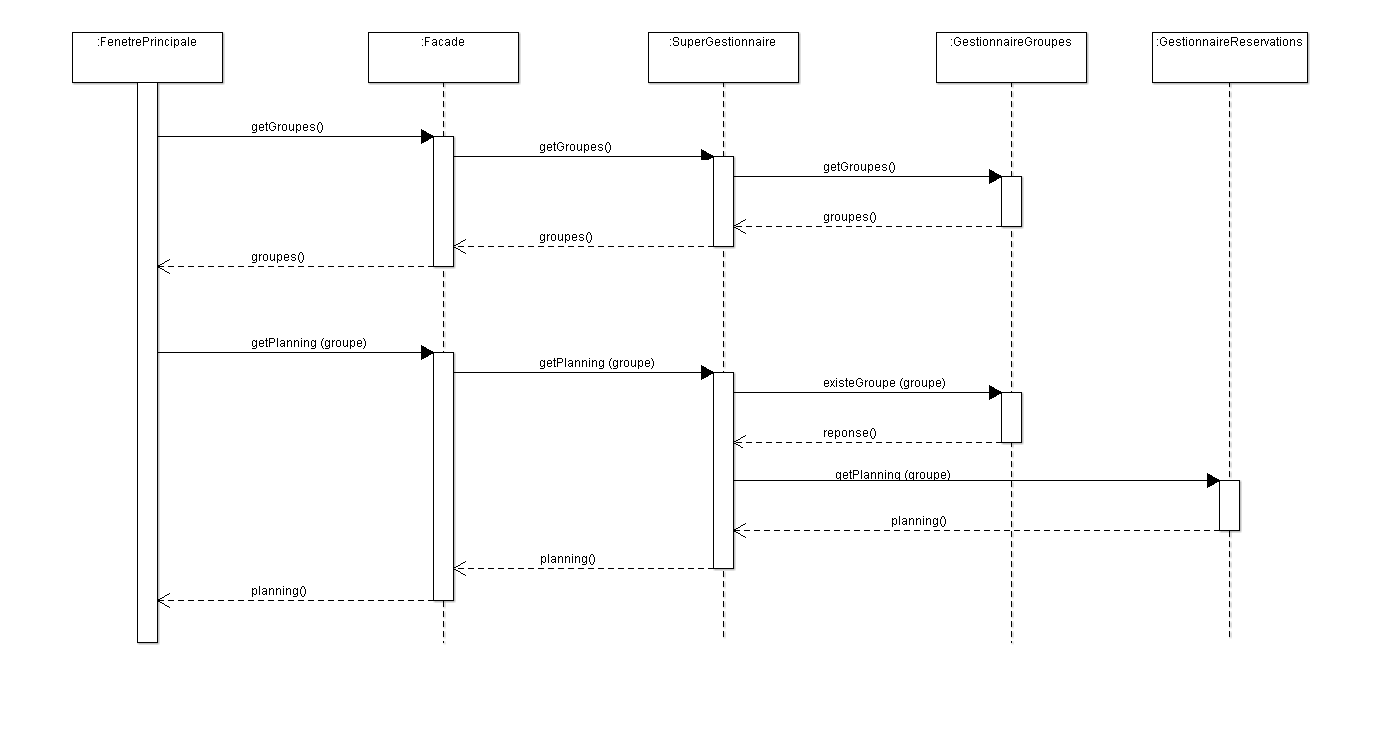
Pour visualiser l’état du service d’un enseignant, la façade fournit les méthodes getNbHeureProgramme et getNbHeureFait qui donnent respectivement pour un enseignement le nombre d’heure qui est programmé dans le planning et le nombre d’heure déjà effectué. Pour utiliser ces fonctionnalités l’interface utilisateur doit préalablement demander la liste des enseignements grâce à la méthode getEnseignements qui donnent les enseignements de l’enseignant connecté, en effet ces enseignements ont été chargés à la connexion dans le gestionnaire d’enseignements.

Visualisation du planning :



Pour visualiser le planning de l’enseignant connecté, l’interface utilisateur dispose de la methode getPlanning dans la facade qui appelle simplement la methode getPlanning chez le super gestionnaire. Le super gestionnaire demande ensuite au gestionnaire de session de lui donner l’objet représentant l’enseignant connecté et demande au gestionnaire de reservation de lui donner toutes les réservations programmées de cet enseignant. Pour cela le gestionnaire de réservations va chercher dans les données persistantes ces réservations et les donne au super gestionnaire qui va les fournir à l’interface utilisateur via la facade. L’interface utilisateur pourra ensuite afficher à l’utilisateur son planning grâce à la liste de réservations reçues.

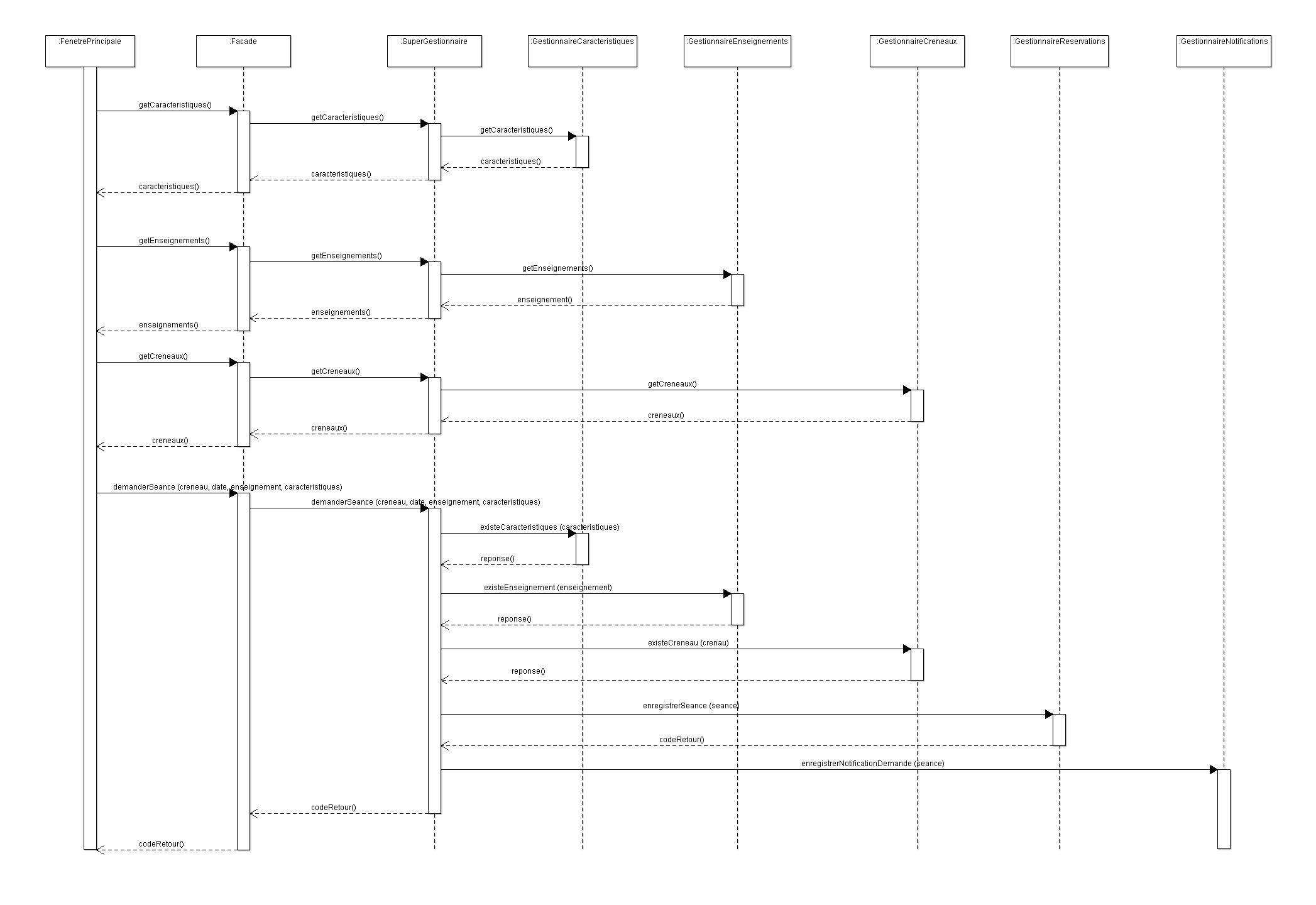
Visualisation du planning d’un groupe :



L’application doit aussi permettre d’afficher le planning d’un groupe d’étudiant. Comme nous avons pu le voir dans la description de la connexion, le super gestionnaire demande, lors de l’authentification de l’enseignant, au gestionnaire de groupes de charger en mémoire tous les groupes de l’enseignant connecté. L’interface utilisateur peut donc ensuite demander ces groupes grâce à la méthode getGroupes qui récupère ces données. La façade fournit une méthode getPlanning qui attend un groupe en paramètre et demande au super gestionnaire le planning de ce groupe. Celui va ensuite vérifier si le groupe donné existe bien parmi les groupes chargés et demandera ensuite le planning de ce groupe au gestionnaire de réservations.

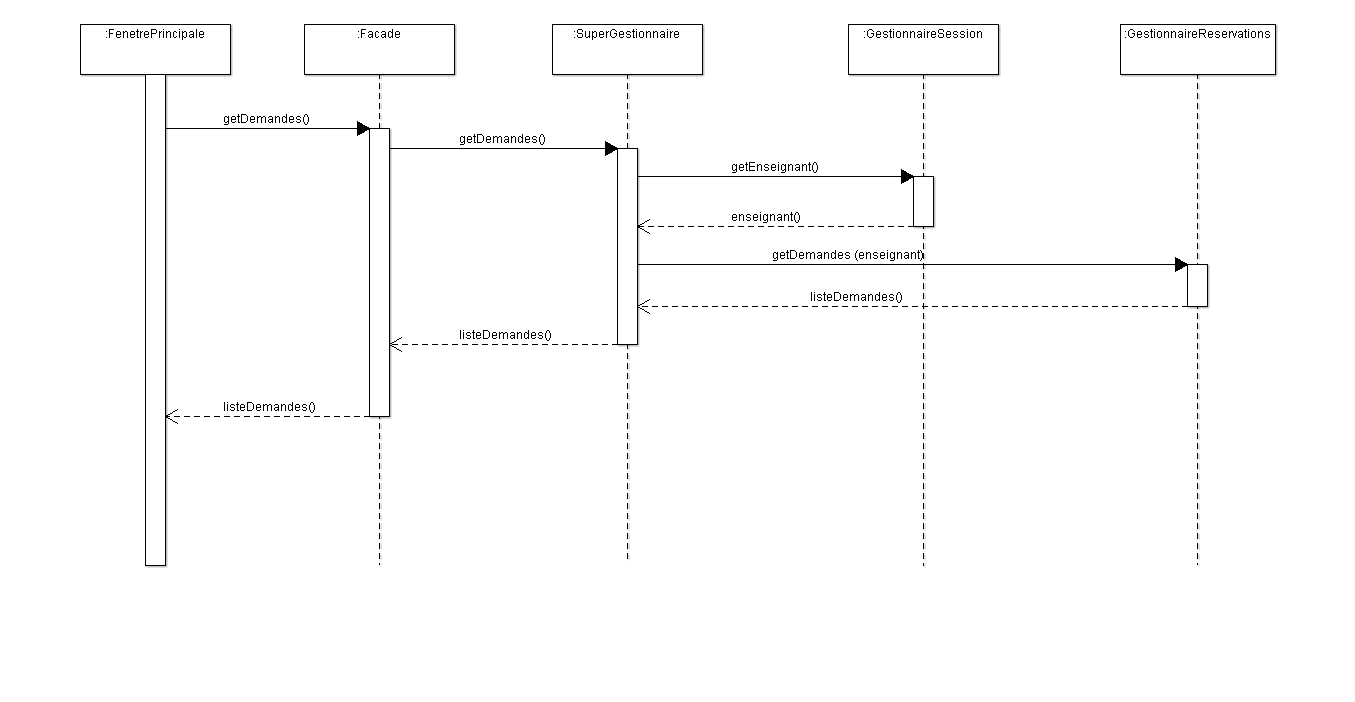
Comme pour la visualisation du planning personnel l’interface utilisateur pourra afficher le planning de ce groupe grâce à la liste des réservations correspondantes renvoyée par la méthode.

Faire une nouvelle demande :



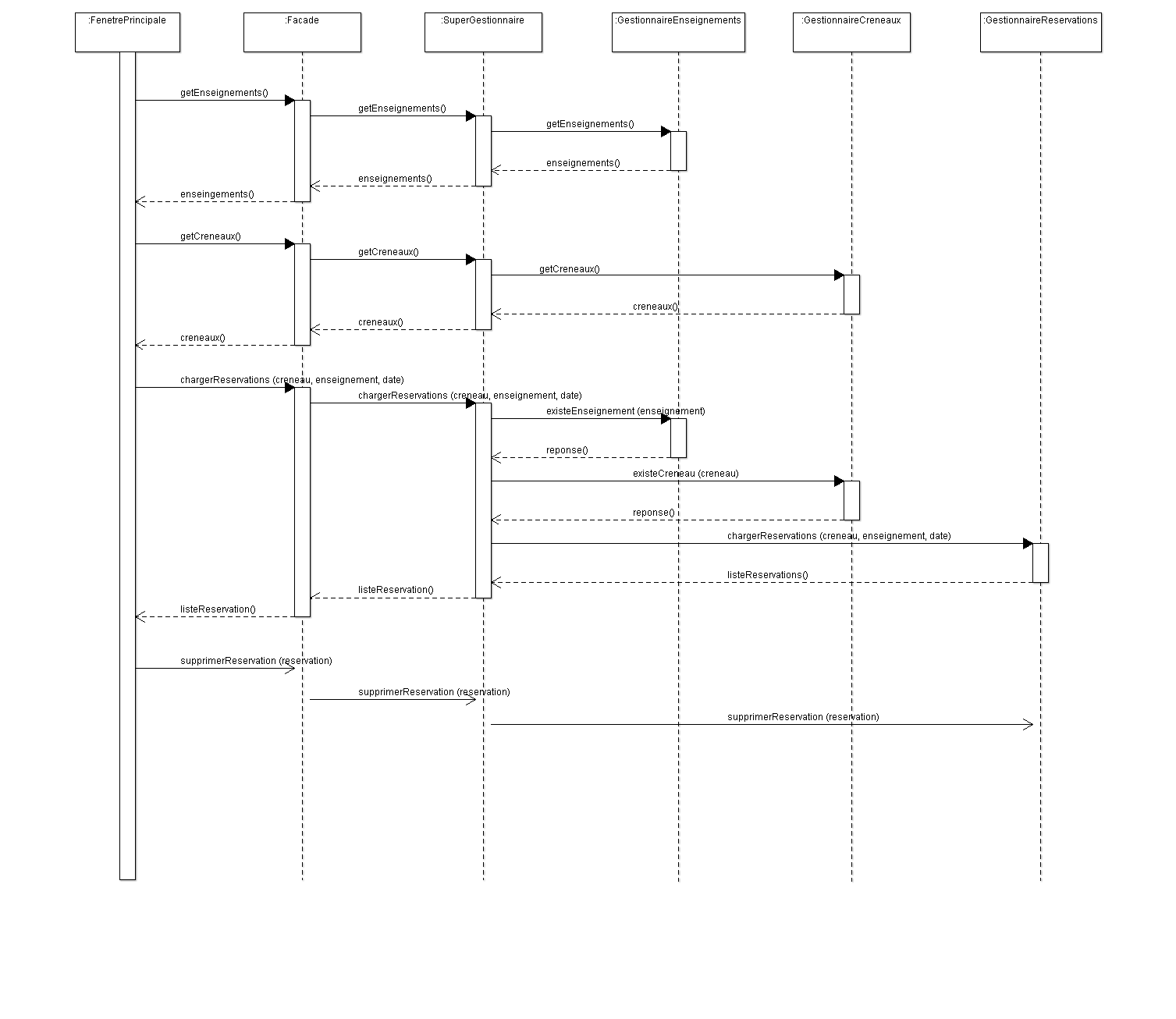
Ce diagramme représente la demande d’une séance par un enseignant. Pour effectuer une demande l’interface utilisateur doit fournir à la business logic une séance, un créneau, une date et une liste de caractéristiques. L’interface peut donc charger depuis les classes métiers les informations disponibles pour l’enseignant connecté. L’interface peut ensuite appeler la méthode demanderSeance avec l’enseignement, le créneau, la date et les caractéristiques souhaitées. La façade délègue le travail au super gestionnaire qui, par mesure de sécurité vérifie que les informations fournis existent bien dans les données persistantes via les différents gestionnaires appropriés et que la date donnée est cohérente, en effet il est incohérent de demander une réservation pour une date antérieure à la date du jour ou pour une dimanche. Si les informations sont correctes le super gestionnaire demande au gestionnaire de réservations d’enregistrer la demande dans les données persistantes. Une fois les données enregistrées le gestionnaire de réservations renvoie un code de retour au super gestionnaire indiquant si l’enregistrement a bien eu lieu et l’information remonte jusqu’à l’interface utilisateur.

Visualisation de ses demandes personnelles :



Un enseignant peut visualiser ses demandes en cours. Pour cela la façade fournit une méthode getDemandes. Cette méthode demande au super gestionnaire les demandes de l’enseignant qui demande donc au gestionnaire de session l’instance de l’enseignant connecté et demande au gestionnaire de réservations de chercher dans les données persistantes les demandes de cet enseignant. La liste des demandes remonte donc ensuite vers l’interface utilisateur qui pourra les afficher à l’utilisateur.

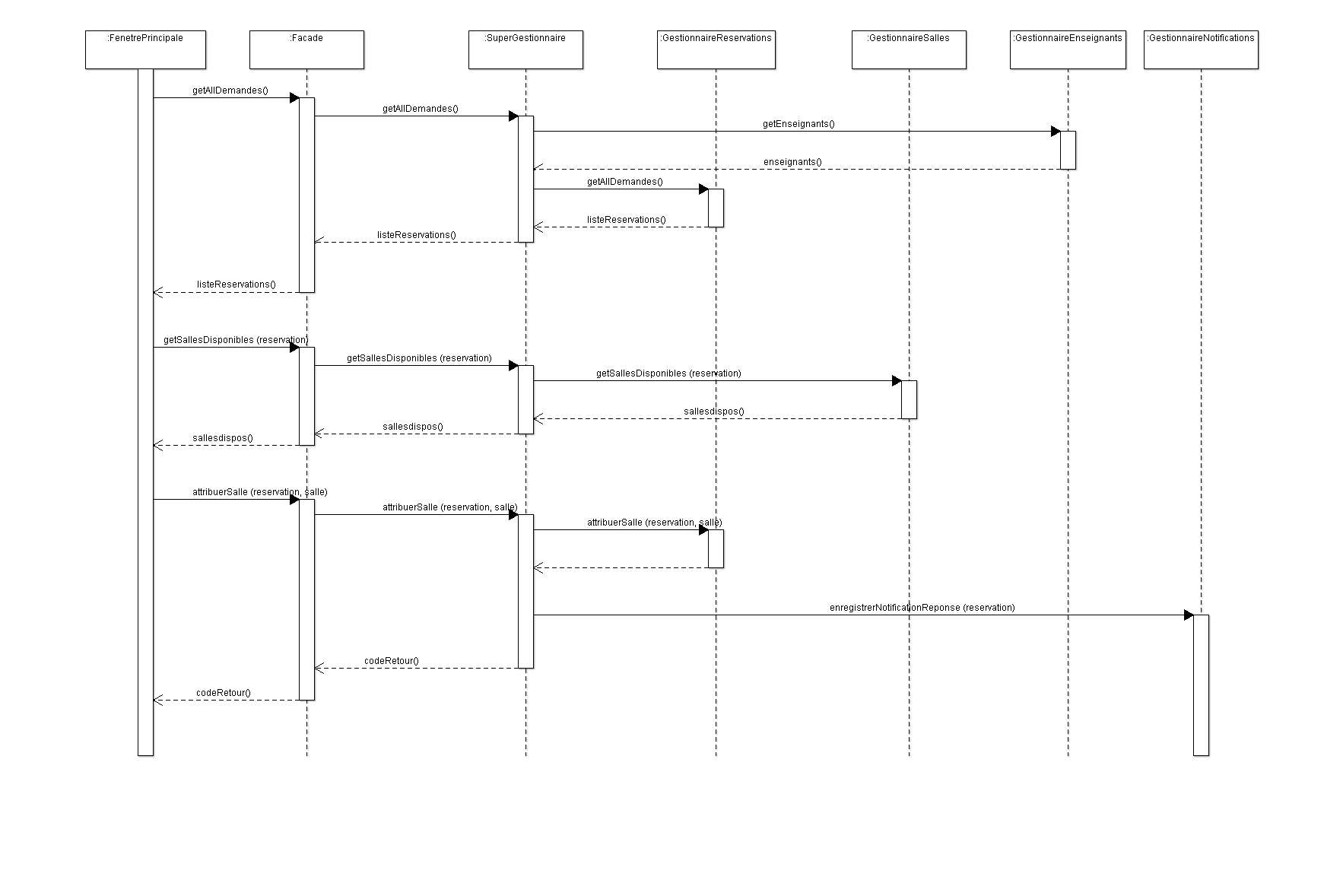
Suppression d’une séance :



Ce diagramme montre le cas d’utilisation de suppression d’une séance par un enseignant. Pour cela la façade fournit la méthode supprimerSeance qui attend une séance en paramètre. La façade fournit aussi une méthode permettant de charger les réservations pour un enseignement, un créneau et une date donnée, ceci facilitera le travail de l’utilisateur qui pourra effectuer une recherche et ne pas à avoir à parcourir la liste de ses réservations pour en supprimer une. La méthode chargerReservations dans la façade demande donc au super gestionnaire de charger les réservations correspondant à un enseignement, une date et un créneau. Le super gestionnaire vérifie que l’enseignement et le créneau existe bien dans les données persistantes et demande au gestionnaire de réservations les séances correspondantes qui remontent jusqu’à l’interface utilisateur.

L’interface peut donc ensuite demander la suppression d’une séance grâce à la méthode supprimer séance.

Attribution d’une salle :



Ce diagramme montre la principale fonctionnalité liée à un responsable à savoir répondre à une demande.

La façade fournit à un responsable la méthode getAllDemandes qui donne l’ensemble des demandes en cours. Pour cela la façade demande au super gestionnaire ces demandes et celui-ci récupère l’ensemble des enseignants préalablement chargés dans le gestionnaire d’enseignants. Le super gestionnaire demande ensuite au gestionnaire de réservations l’ensemble des demandes de cette liste d’enseignants.

A partir de ces demandes l’interface pourra faire appel à la méthode getSallesDisponibles qui donnera pour une demande donnée la liste des salles qui sont disponibles.

Enfin la façade fournit la méthode attribuerSalle qui permet, comme son nom l’indique, d’attribuer une salle à la réservation donnée. Pour cela la façade appelle la méthode attribuerSalle chez le super gestionnaire qui demande au gestionnaire de réservation d’enregistrer cette demande dans les données persistantes puis délègue au gestionnaire de notifications la tâche d’enregistrer une notification de la réponse.

## Persistance des données

Comme nous l’avons présenté précédemment nous avons tenu au cours de notre conception à ce que notre business logic soit le plus indépendante possible de l’interface utilisateur et de la représentation des données qu’il utilise.

Nous pensons que notre modèle de gestionnaires permet une indépendance totale peu importe le type de persistance des données, en effet les gestionnaires sont tous abstraits et décrivent les fonctionnalités attendus. Pour utiliser un certain type de persistance il suffit donc d’implémenter ces méthodes abstraites de manière à pouvoir utiliser ce type de persistance. De plus la classe AbstractFactory permet la création automatique du type de gestionnaire dont l’application a besoin.

Il nous a donc été très simple d’implémenter la fonctionnalité de login avec une persistance sous forme de fichiers. Nous avons simplement créé une nouvelle classe GestionnaireSessionFile qui authentifie un enseignant à partir d’un fichier xml.

# Conclusion

# Annexes

Types.sql :

drop type ty\_Enseignant;

drop type nt\_reservation;

drop type ty\_ref\_reservation;

drop type ty\_Reservation ;

drop type ty\_Salle ;

drop type ty\_creneau ;

drop type nt\_caracteristique ;

drop type ty\_ref\_caracteristique ;

drop type ty\_caracteristique ;

drop type nt\_enseignement ;

drop type ty\_ref\_ENseignement;

drop type ty\_Enseignement ;

drop type ty\_cours;

drop type ty\_type\_cours ;

drop type ty\_matiere;

drop type ty\_groupe;

drop view VEnseignant;

drop view VReservation;

drop view VCreneau;

drop view VGroupe;

drop view VCours;

drop view VType\_cours;

drop view VMatiere;

drop view VSalle;

drop view VCaracteristique;

drop view VEnseignement;

---------------------TYPES OBJETS-----------------------------

create or replace type ty\_caracteristique as object (ID\_Caracteristique number (10), libelle\_caracteristique char (255))

/

create or replace type ty\_ref\_caracteristique as object (RefCa ref ty\_caracteristique)

/

create or replace type nt\_caracteristique as table of ty\_ref\_caracteristique

/

create or replace type ty\_salle as object (ID\_SALLE NUMBER(10), numero\_salle CHAR(255), carac nt\_caracteristique)

/

create or replace type ty\_groupe as object (ID\_GROUPE number(10), libelle\_groupe char(255), ID\_GROUPE\_PERE number(10))

/

create or replace type ty\_creneau as object (ID\_CRENEAU NUMBER(10),heure\_debut char(255), heure\_fin CHAR(255))

/

create or replace type ty\_matiere as object (ID\_matiere NUMBER(10), libelle\_matiere char(255))

/

create or replace type ty\_type\_cours as object (ID\_type\_cours NUMBER(10), libelle\_type\_cours char (255))

/

create or replace type ty\_cours as object(ID\_COURS number(10), matiere ref ty\_matiere, type\_cours ref ty\_type\_cours)

/

create or replace type ty\_enseignement as object(ID\_ENSEIGNEMENT NUMBER(10), cours ref ty\_cours, groupe ref ty\_groupe, nb\_heure\_prevue NUMBER(4))

/

create or replace type ty\_ref\_enseignement as object (RefE ref ty\_enseignement)

/

create or replace type nt\_enseignement as table of ty\_ref\_enseignement

/

create or replace type ty\_Reservation as object (ID\_RESERVATION NUMBER(10), salle ref ty\_salle, creneau ref ty\_creneau,

enseignement ref ty\_enseignement, dateRes date, caracteristiques nt\_caracteristique)

/

create or replace type ty\_ref\_reservation as object (RefR ref ty\_reservation)

/

create or replace type nt\_reservation as table of ty\_ref\_reservation

/

create or replace type ty\_enseignant as object (ID\_ENSEIGNANT NUMBER(10), nom CHAR(255), prenom CHAR(255),

liste\_Enseignement nt\_enseignement, liste\_Reservations nt\_reservation)

/

---------------------------VUES----------------------------

create or replace view VCreneau of ty\_creneau

with object identifier(ID\_CRENEAU)

as select C.ID\_CRENEAU,C.HEURE\_DEBUT, C.HEURE\_FIN from crenaux C;

create or replace view VGroupe of ty\_groupe

with object identifier(ID\_GROUPE)

as select G.ID\_GROUPE, G.libelle\_groupe, G.ID\_GROUPE\_A\_POUR\_PERE from groupe G;

create or replace view VMatiere of ty\_matiere

with object identifier(ID\_MATIERE)

as select M.ID\_MATIERE, M.LIBELLE\_MATIERE from matiere M;

create or replace view VType\_cours of ty\_type\_cours

with object identifier(id\_type\_cours)

as select T.id\_type\_de\_cours, T.libelle\_type\_de\_cours from typecours T;

create or replace view VCours of ty\_cours

with object identifier(id\_cours)

as select C.id\_cours, make\_ref(VMatiere, C.id\_matiere), make\_ref(VType\_cours, C.id\_type\_de\_cours) from cours C;

create or replace view VCaracteristique of ty\_caracteristique

with object identifier(ID\_CARACTERISTIQUE)

as select C.ID\_CARACTERISTIQUE, C.LIBELLE\_CARACTERISTIQUE from caracteristique C;

create or replace view VSalle of ty\_salle

with object identifier(ID\_SALLE)

as select S.ID\_SALLE, S.NUMERO\_SALLE,

cast(

multiset(

select make\_ref(VCaracteristique, C.ID\_CARACTERISTIQUE) from caracteristique\_salle C where S.ID\_SALLE = C.ID\_SALLE) as nt\_caracteristique) from salle S;

create or replace view VEnseignement of ty\_enseignement

with object identifier(id\_enseignement)

as select E.id\_enseignement, make\_ref(VCours, E.id\_cours), make\_ref(VGroupe, E.id\_groupe), E.nb\_heure\_prevue from enseignement E;

create or replace view VReservation of ty\_reservation

with object identifier(id\_reservation)

as select R.id\_reservation, make\_ref(VSalle, R.id\_salle), make\_ref(VCreneau, R.id\_creneau), make\_ref(VEnseignement, R.id\_enseignement), R.date\_reservation,

cast (

multiset(

select make\_ref(VCaracteristique, RC.ID\_CARACTERISTIQUE) from reservation\_caracteristique RC where RC.ID\_RESERVATION = R.ID\_RESERVATION) as nt\_caracteristique)

from reservation R;

create or replace view VEnseignant of ty\_enseignant

with object identifier(ID\_ENSEIGNANT)

as select ens.ID\_ENSEIGNANT, ens.NOM, ens.PRENOM,

cast(

multiset(

select make\_ref(VEnseignement, E.id\_enseignement) from enseignement E where ens.id\_enseignant = E.id\_enseignant) as nt\_enseignement),

cast(

multiset(

select make\_ref(VReservation, R.id\_reservation) from reservation R where ens.id\_enseignant = (select Ebis.id\_enseignant from enseignement Ebis where R.id\_enseignement = Ebis.id\_enseignement)) as nt\_reservation) from enseignant ens;

------------------------REQUETES-----------------------------

-- Enseignements d'un professeur donné

select Ens.RefE.nb\_heure\_prevue, deref(deref(Ens.RefE.cours).type\_cours).libelle\_type\_cours, deref(deref(Ens.RefE.cours).matiere).libelle\_matiere, deref(Ens.RefE.groupe).libelle\_groupe from table( select E.liste\_enseignement from VEnseignant E where nom = 'Stratulat') Ens;

-- Demandes en cours d'un enseignant

select Res.RefR.id\_reservation, deref(Res.RefR.creneau).heure\_debut, deref(Res.RefR.creneau).heure\_fin,deref(Res.RefR.enseignement).nb\_heure\_prevue, deref(deref(deref(Res.RefR.enseignement).cours).matiere).libelle\_matiere,

deref(deref(deref(Res.RefR.enseignement).cours).type\_cours).libelle\_type\_cours, deref(deref(Res.RefR.enseignement).groupe).libelle\_groupe,

Res.RefR.dateRes from table(select E.liste\_Reservations from VEnseignant E where nom = 'Stratulat') Res where Res.RefR.salle is null;

select Res.RefR.id\_reservation, deref(Res.RefR.creneau).heure\_debut, deref(Res.RefR.creneau).heure\_fin,deref(Res.RefR.enseignement).nb\_heure\_prevue, deref(deref(deref(Res.RefR.enseignement).cours).matiere).libelle\_matiere,deref(deref(deref(Res.RefR.enseignement).cours).type\_cours).libelle\_type\_cours, deref(deref(Res.RefR.enseignement).groupe).libelle\_groupe,Res.RefR.dateRes from table(select E.liste\_Reservations from VEnseignant E where nom = 'Stratulat') Res where Res.RefR.salle is null;

-- savoir si un groupe est disponible pour un creneau donné

-- vérifier si groupe pere disponible

select id\_reservation from reservation r where r.id\_creneau = (select id\_creneau from crenaux where heure\_debut = '8h00' and heure\_fin = '9h45') and (select id\_groupe from enseignement where id\_enseignement = r.id\_enseignement) = (select id\_groupe from groupe where libelle\_groupe = 'IG4') and id\_salle is not null;

-- Salles qui ont toutes les caracteristiques souhaitées

select id\_salle from caracteristique\_salle cars where (select count(\*) from

(select id\_caracteristique from caracteristique c where c.libelle\_caracteristique = 'videoprojecteur' or c.libelle\_caracteristique = 'grande'

MINUS

select id\_caracteristique from caracteristique\_salle cs where cs.id\_salle = cars.id\_salle)) = 0;

--select s.id\_salle from salle s where (select count(\*) from

--(

--(select id\_caracteristique from caracteristique c where c.libelle\_caracteristique = 'videoprojecteur' or c.libelle\_caracteristique = 'grande')

-- MINUS

--(select id\_caracteristique from caracteristique\_salle cs where --cs.id\_salle = s.id\_salle)

--)) = 0;

Select s.ID\_SALLE

from salle s, reservation r

where r.ID\_RESERVATION = 5 and

s.ID\_SALLE in (select ID\_SALLE from salle sa where

(select count(ID\_CARACTERISTIQUE) from caracteristique where

ID\_CARACTERISTIQUE in ((

SELECT ID\_CARACTERISTIQUE from reservation\_caracteristique where ID\_RESERVATION = r.ID\_RESERVATION) MINUS(

SELECT ID\_CARACTERISTIQUE from CARACTERISTIQUE\_SALLE where ID\_SALLE = sa.ID\_SALLE))) = 0)

and s.ID\_SALLE not in (select sa.ID\_SALLE from salle sa, reservation re

WHERE sa.ID\_SALLE = re.ID\_SALLE and re.DATE\_RESERVATION = r.DATE\_RESERVATION

and re.ID\_CRENEAU = r.ID\_CRENEAU);

-- groupes d'un enseignant

select distinct(libelle\_groupe) from groupe g, enseignement e where e.id\_groupe = g.id\_groupe and e.id\_enseignant = 1;

-- toutes les demandes

select Res.RefR.id\_reservation, deref(Res.RefR.creneau).heure\_debut, deref(Res.RefR.creneau).heure\_fin, deref(Res.RefR.enseignement).nb\_heure\_prevue, deref(deref(deref(Res.RefR.enseignement).cours).matiere).libelle\_matiere,deref(deref(deref(Res.RefR.enseignement).cours).type\_cours).libelle\_type\_cours, deref(deref(Res.RefR.enseignement).groupe).libelle\_groupe,Res.RefR.dateRes from table(select E.liste\_Reservations from VEnseignant E ) Res where Res.RefR.salle is NULL;