



Projet SE :
***Conception de
l'architecture & conception
détaillée***



SIMPLIFY

Par Emma Mathieu, Thomas Fournier, Salim Belkhir et Ayoub Hakemi

Sommaire

Sommaire	1
<i>1 - Contexte</i>	2
<i>2 - Organisation</i>	2
3 - Problèmes rencontrés	3
<i>4 - Solutions apportées</i>	4
<i>5 - Remarques</i>	5
5.1 Remarques positives	5
5.2 Remarques négatives	5
<i>6 - Résultat</i>	6
6.1 Bases de données	6
6.2 Diagrammes de classes et diagrammes de séquence	13
6.2.1 Le panier	13
6.2.2 La livraison	15
6.2.3 Les événements	17
6.2.4 Le plat	19
6.2.6 L'opinion	23
6.2.7 Le paiement	26
6.2.8 Le placement des tables	28
6.2.9 La réservation	30
6.2.10 Le restaurant	33
6.2.11 La table	35
6.2.12 L'utilisateur	38

1 - Contexte

La troisième étape de ce projet consistait à réaliser la conception de l'architecture du projet. Celle-ci comprenait, l'architecture de la base de données, l'architecture des uses-cases avec des diagrammes de classes et une représentation des fonctionnalités des uses-cases à travers des diagrammes de séquences.

2 - Organisation

Pour réaliser cette étape, nous avons décidé d'utiliser "PlantUml", un outil open-source permettant de créer des diagrammes UML avec un langage pour pouvoir pallier les problèmes de limitation sur Lucid (logiciel utilisé dans l'étape précédente) et créer des diagrammes plus personnalisables notamment au niveau des diagrammes de séquences.

À partir de cette étape, nous avons décidé d'organiser un tableau pour séparer le restant du projet en 3 sprints avec la liste des uses-cases à implémenter pour chacun :

	Architecture (Classes et séquence)	Implémentation	Test	Numéro itération	Date architecture	Date implémentation	Date test
Gestion de compte	@Thomas Fournier	@Ayoub_HK	@Emma Mathieu	1	19/12	21/12	22/12
Restaurant	@Salim B	@Thomas Fournier	@Ayoub_HK	1			
Gestion des plats	@Emma Mathieu	@Salim B	@Thomas Fournier	1			
Table	@Ayoub_HK	@Emma Mathieu	@Salim B	1			
Païement	@Salim B	@Thomas Fournier	@Ayoub_HK	2	23/12	26/12	27/12
Placement	@Emma Mathieu	@Salim B	@Thomas Fournier	2			
Notification	@Ayoub_HK	@Emma Mathieu	@Salim B	2			
Panier	@Thomas Fournier	@Ayoub_HK	@Emma Mathieu	2			
Réservation	@Salim B	@Emma Mathieu	@Ayoub_HK	3	29/12	01/01	02/02
Èvènement	@Thomas Fournier	@Ayoub_HK	@Salim B	3			
Gestion des avis	@Ayoub_HK	@Thomas Fournier	@Emma Mathieu	3			
Gestions des livraisons	@Emma Mathieu	@Salim B	@Thomas Fournier	3			

Tableau organisation projet sur notion

Pour établir ce tableau, nous avons essayé d'équilibrer les uses-cases en termes de difficultés et en fonction des liaisons entre eux. Par exemple, nous avons déterminé que les uses-cases du premier sprint doivent être réalisés avant ceux du deuxième pour éviter les problèmes de conflits notamment.

Après avoir établi le tableau précédent, nous avons organisé une réunion afin d'établir la base de données ensemble. Celle-ci est présentée dans la dernière partie de ce rapport.

Par ailleurs, à la fin de chaque itération, avant de réaliser l'implémentation, nous avons organisé des réunions afin de valider et commenter les diagrammes UML réalisés.

3 - Problèmes rencontrés

Au cours de cette phase, nous avons rencontré plusieurs problèmes.

Pour commencer, nous avons eu du mal à concevoir la base de données notamment au niveau de la représentation d'une commande. En effet, les livraisons et les réservations ont énormément de ressemblance mais aussi quelques différences. Ce sont toutes les deux des commandes. Il a fallu visualiser leur représentation.

Ensuite, nous avons eu des problèmes pour représenter les classes symbolisant les Frame FXML dans le diagramme de classes. Cela est dû à nos différentes interprétations d'une frame, est-ce un component (composant) ou une page web ?

D'autre part, il a été très difficile de mettre en place le design des facades pour le premier sprint. En effet, nous avons tous un point de vue différent quant à son utilisation et représentation. Plusieurs réunions ont été établies.

Par ailleurs, nous avons des questions quant à la représentation des DAO utilisés dans les diagrammes de classe notamment au niveau du degré de détail que nous voulions et du fait de les mettre en Singleton.

Enfin, il y a eu quelques soucis à propos des diagrammes de séquence, encore une fois liée au niveau du détail attendu.

4 - Solutions apportées

Nous avons trouvé des solutions aux problèmes cités ci-dessus :

Premièrement, au niveau de l'architecture pour une commande, nous avons décidé de créer une classe "parent" Order qui contient toutes les informations d'une livraison ou d'une réservation (l'id, l'id du restaurant, de l'utilisateur...). Ensuite, nous avons défini une classe pour indiquer le type de commande, TypeOrder. l'id 1 correspond à une livraison et le 2 à une réservation. Pour finir, nous avons ajouté un état à chaque commande qui lui aussi vient avec une autre table, State_order. Il a fallu faire des schémas pour résoudre ce problème.

Au sujet des difficultés quand à représenter les classes de Frame FXML, c'est après avoir étudié un exemple plus complet pour comprendre la définition exacte d'une frame (page, composant) que l'on a pu se mettre d'accord quant à la manière de les représenter dans le diagramme de classe. En général, on considère les frame comme étant des pages mais aussi des composants que l'on peut inclure dans d'autres pages.

Le problème concernant les facades a été le plus important de cette phase.

Nous avons essayé de contacter le professeur responsable du projet mais sans succès. Il a donc fallu trouver une solution pour savoir comment gérer les facades. Nous avons donc organisé plusieurs réunions, proposé des diagrammes de classes différents en justifiant les choix de design de chacun mais sans succès.

Certains pensaient que la facade ne devait contenir que les méthodes qui interagissent directement avec "l'objet" de l'use-case. Par exemple, pour la facade du plat, seules les méthodes qui concernent directement un plat sont mises à l'intérieur. Les méthodes concernant les opinions des plats sont dans la facade des opinions. Le but étant de faire appel à la facade des opinions dans la facade des plats pour avoir les méthodes.

Une autre solution proposée était de réunir toutes les méthodes dans la facade du plat qui apparaissent dans la frame du plat. Dans ce cas là, si la fonctionnalité “consulter les opinions d’un plat” est sur la frame du plat, la facade du plat doit avoir la méthode pour consulter l’opinion d’un plat.

Pour résoudre ce problème, nous avons décidé que chacun allait réaliser le diagramme de classe du sprint 1 selon sa propre idée de design ainsi que son implémentation (seulement pour le sprint 1). De ce fait, nous avons fait une dernière réunion pour montrer le travail réalisé en justifiant les choix avec un exemple concret du projet. Nous avons enfin pu nous mettre d’accord pour avoir une marche à suivre pour les prochains sprints et cela nous a permis d’éviter d’être en retard par rapport aux prévisions. (Au final, nous avons mixé nos solutions) Nous avons opté pour la solution avec la facade présentant toutes les méthodes liées aux fonctionnalités possibles sur les frames de l’use-case mais avec quelques détails en moins.

Pour les DAO, nous avons réglé ce problème en même temps que celui de la facade. Nous avons décidé de mettre les Facades en Singleton et les Enfin, concernant les diagrammes de séquence, nous avons déterminé un niveau de détail commun à travers une autre réunion. Les diagrammes de séquence n’ont servi qu’à décrire globalement une fonctionnalité sans pour autant rentrer dans les détails.

5 - Remarques

5.1 Remarques positives

Le fait d’avoir utilisé PlantUml pour les diagrammes a été une bonne décision, cela a facilité la correction des diagrammes et leur génération.

Nous avons pu consolider la notion de frame pour le FXML, s’entraîner à faire une base de données correcte, réaliser des diagrammes de classes très utiles pour la partie implémentation et créer des diagrammes de séquences pour illustrer certaines fonctionnalités dans le cas ou celui en charge de l’implémentation avait du mal à savoir le principe d’une des méthodes du diagramme de classe.

5.2 Remarques négatives

Cette étape nous a montré à quel point il était difficile de se mettre d'accord sur un choix de design sachant que personne n'arrivait à valider les justifications des propositions.

Comme il s'agissait d'un problème de conception et non de programmation, tout dans les cas la solution allait marcher mais c'était difficile de savoir qu'elle était la plus optimale.

Cela nous a également montré l'importance du choix du niveau de détail dans les diagrammes qui s'avère plus compliqué à choisir en fonction des situations.

Enfin, il était parfois difficile de bien comprendre le point de vue de l'auteur des diagrammes, il a fallu éclaircir certains points avec celui-ci.

6 - Résultat

6.1 Bases de données



Au niveau de la base de données, nous avons décidé d'utiliser une base de données PostGres notamment parce qu'il existe une API facile à utiliser et compatible avec Java, l'API JDBC. Pour héberger celle-ci, nous avons utilisé "Always data", un site d'hébergeur de base de données gratuit avec notamment un SGDB pour PostGresql (phpPgAdmin).

Voici la liste des tables réalisées pour le projet, extrait de phpPgAdmin :



	Table	Owner	T
<input type="checkbox"/>	Event	simplify	
<input type="checkbox"/>	Meal	simplify	
<input type="checkbox"/>	Meal_ordered	simplify	
<input type="checkbox"/>	Notification	simplify	
<input type="checkbox"/>	Opinion	simplify	
<input type="checkbox"/>	Opinion_meal	simplify	
<input type="checkbox"/>	Opinion_restaurant	simplify	
<input type="checkbox"/>	Order	simplify	
<input type="checkbox"/>	Payment	simplify	
<input type="checkbox"/>	Reserve_Table	simplify	
<input type="checkbox"/>	Respo_restaurant	simplify	
<input type="checkbox"/>	Restaurant	simplify	
<input type="checkbox"/>	Role	simplify	
<input type="checkbox"/>	State_order	simplify	
<input type="checkbox"/>	Table	simplify	
<input type="checkbox"/>	TypeOrder	simplify	
<input type="checkbox"/>	User	simplify	

Les illustrations suivantes vont présenter les tables de manières plus précises avec les attributs de chacune. La colonne “constraints” indique les contraintes associées aux attributs. Une clé de couleur or correspond à la clé primaire et une clé argentée à une clé étrangère. La fonction “nextval” permet de calculer automatiquement un identifiant :





- **Table Event** représentant les événement associé à un restaurant :

Column	Type	Not Null	Default	Constraints
idEvent	integer	NOT NULL	nextval('sq_id_event'::regclass)	
idRestaurant	integer	NOT NULL		
date	timestamp without time zone	NOT NULL		
description	character varying			
title	character varying	NOT NULL		



- **Table Meal**, représentant les plats d'un restaurant :

Column	Type	Not Null	Default	Constraints
idMeal	integer	NOT NULL	nextval('id_meal'::regclass)	
idRestaurant	integer	NOT NULL		
description	character varying			
title	character varying	NOT NULL		
price	numeric	NOT NULL		



- **Table Meal_ordered**, représentant une table association entre un plat et une réservation :

Column	Type	Not Null	Default	Constraints
idMeal	integer	NOT NULL		 
idOrder	integer	NOT NULL		 
quantity	integer	NOT NULL		





- **Table Notification**, représentant les notifications :

Column	Type	Not Null	Default	Constraints
idNotification	integer	NOT NULL	nextval('id_notification'::regclass)	
title	character varying	NOT NULL		
description	character varying	NOT NULL		
idUser	integer	NOT NULL		





- **Table Opinion**, représentant les opinions des utilisateurs :
-

Column	Type	Not Null	Default	Constraints
idOpinion	integer	NOT NULL	nextval('id_opinion'::regclass)	
idUser	integer	NOT NULL		
comment	character varying	NOT NULL		






- **Table Opinion_meal**, représentant une table association entre un plat et une opinion :

Column	Type	Not Null	Default	Constraints
idMeal	integer	NOT NULL		 
idOpinion	integer	NOT NULL		 




- **Table Opinion_restaurant**, représentant une table association entre une opinion et un restaurant :

Column	Type	Not Null	Default	Constraints
idRestaurant	integer	NOT NULL		 
idOpinion	integer	NOT NULL		 





- **Table Order**, représentant une commande qui peut être soit une réservation soit une livraison, elle est déterminée en fonction de son type :

Column	Type	Not Null	Default	Constraints
idOrder	integer	NOT NULL	nextval('id_order'::regclass)	
idTypeOrder	integer	NOT NULL		
idRestaurant	integer	NOT NULL		
idUser	integer	NOT NULL		
idState	integer	NOT NULL		
date	date	NOT NULL	now()	



- **Table Payment**, représentant un paiement :

Column	Type	Not Null	Default	Constraints
idPayment	integer	NOT NULL	nextval('id_payment'::regclass)	
idOrder	integer			
idUser	integer			
amount	character varying(255)			
date	date			
numbercard	character varying(255)			


- **Table Reserve_Table**, représentant une table d'association entre une table et une réservation :

Column	Type	Not Null	Default	Constraints
idOrder	integer	NOT NULL		 
idTable	integer	NOT NULL		 


- **Table Respo_restaurant**, représentant une table association entre un utilisateur et un restaurant, cela indique qui est responsable de quel restaurant :

Column	Type	Not Null	Default	Constraints
idUser	integer	NOT NULL		
idRestaurant	integer	NOT NULL		



- **Table Restaurant**, représentant les restaurants :

Column	Type	Not Null	Default	Constraints
idRestaurant	integer	NOT NULL	nextval('id_resto'::regclass)	
name	character varying	NOT NULL		
address	character varying	NOT NULL		
nbOfStars	integer	NOT NULL		
width	numeric	NOT NULL		
length	numeric	NOT NULL		
phoneNumber	character varying(12)			
email	character varying(40)			

- **Table Role**, indiquant les rôles des utilisateurs :



Column	Type	Not Null	Default	Constraints
idRole	integer	NOT NULL		
label	character varying(30)	NOT NULL		

- **Table State_order**, représentant les états d'une commande (DELIVERED, CANCELLED...) :



Column	Type	Not Null	Default	Constraints
idState	integer	NOT NULL	nextval('id_state_livraison'::regclass)	
description	character varying	NOT NULL		

La contrainte "1" indique que la description doit être unique.




- **Table Table**, représentant les tables d'un restaurant :

Column	Type	Not Null	Default	Constraints
idTable	integer	NOT NULL	nextval('sq_id_table'::regclass)	
idRestaurant	integer	NOT NULL	1	
name	character varying	NOT NULL		
description	character varying			
booked	boolean		false	
x	integer	NOT NULL	'-1'::integer	
y	integer	NOT NULL	'-1'::integer	

- **Table TypeOrder**, représentant le type de commande possible :

Column	Type	Not Null	Default	Constraints
idTypeOrder	integer	NOT NULL		
label	character varying	NOT NULL		

- **Table User**, représentant les utilisateurs :

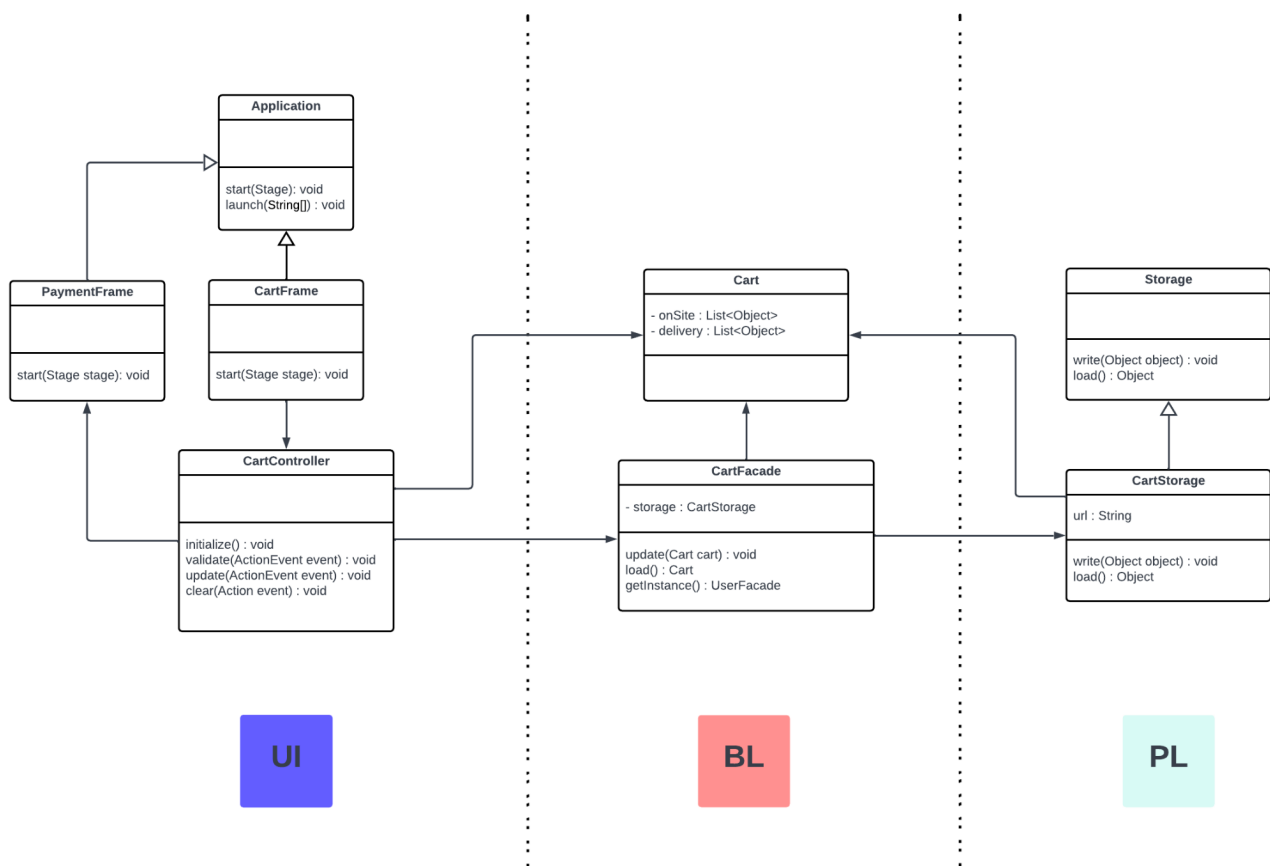
Column	Type	Not Null	Default	Constraints
idUser	integer	NOT NULL	nextval('mgr_id_user'::regclass)	
password	character varying(255)	NOT NULL		
email	character varying(255)	NOT NULL		
name	character varying(50)	NOT NULL		
firstname	character varying(50)	NOT NULL		
address	character varying(200)			
phone	character varying(10)			
idRole	integer	NOT NULL		
ban	boolean			
askDelete	boolean			

6.2 Diagrammes de classes et diagrammes de séquence

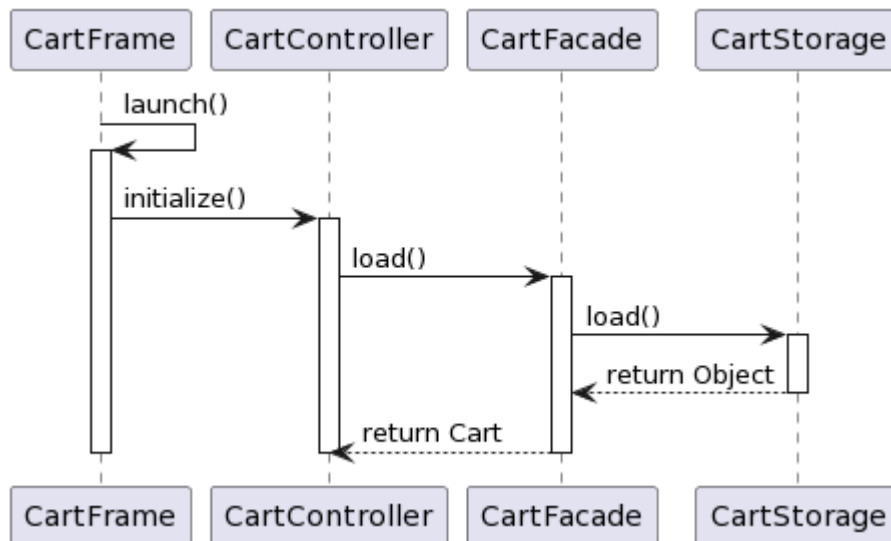
Certains diagrammes ont été réalisés avec Lucid et la majorité avec PlantUML.
Pour plus de visibilité, ils sont disponibles dans le dossier “diagrammes” du projet.
Nous avons décidé d'utiliser une facade, des singletons (les DAO ne sont pas en singletons mais la connexion à la base de données est unique).
Voici les différents diagrammes de classe réalisés au cours du projet :

6.2.1 Le panier

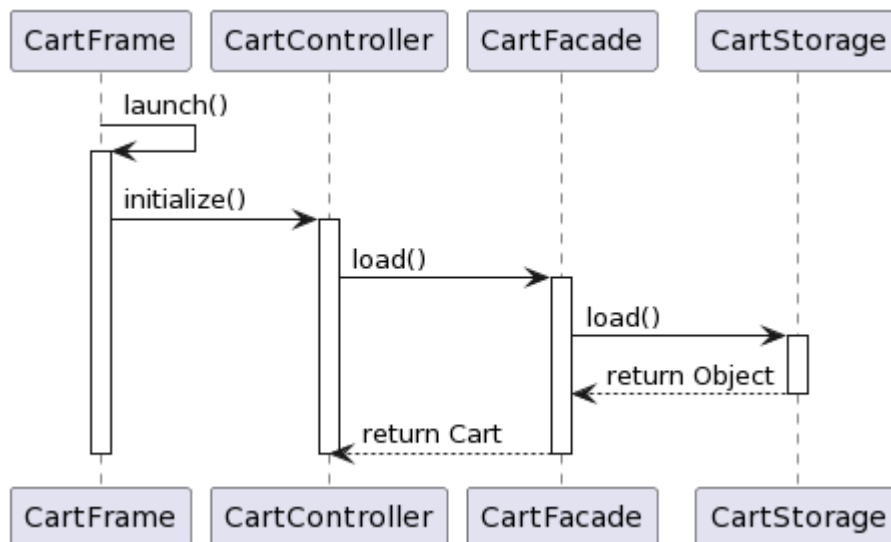
Le diagramme de classe pour le **Cart** :



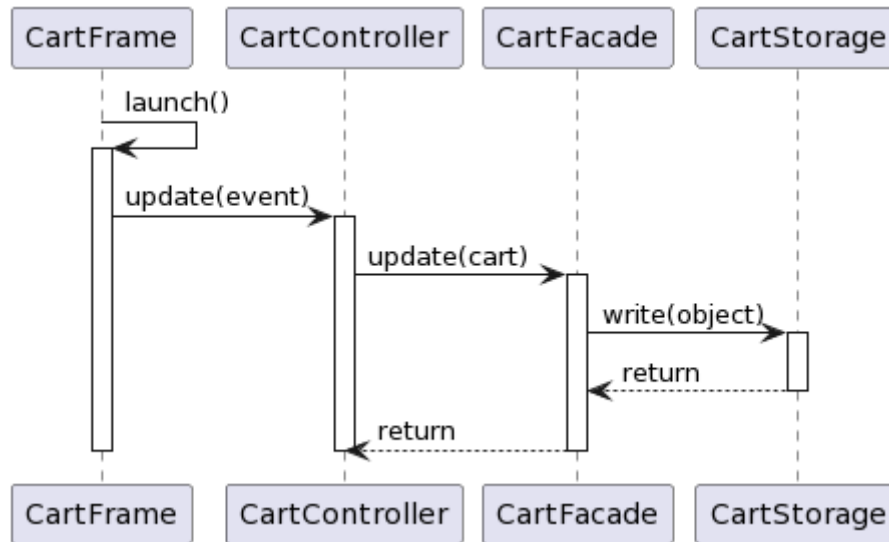
Le diagramme de séquence correspondant à la fonctionnalité “**clear**” du Cart :



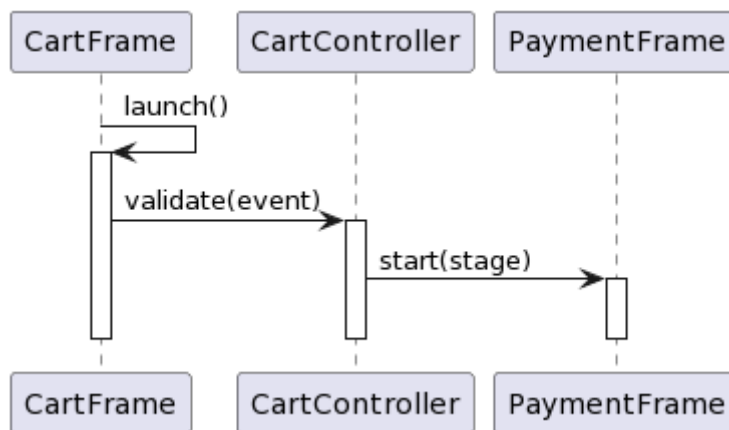
Le diagramme de séquence correspondant à la fonctionnalité “**initialize**” du Cart :



Le diagramme de séquence correspondant à la fonctionnalité “**update**” du Cart :

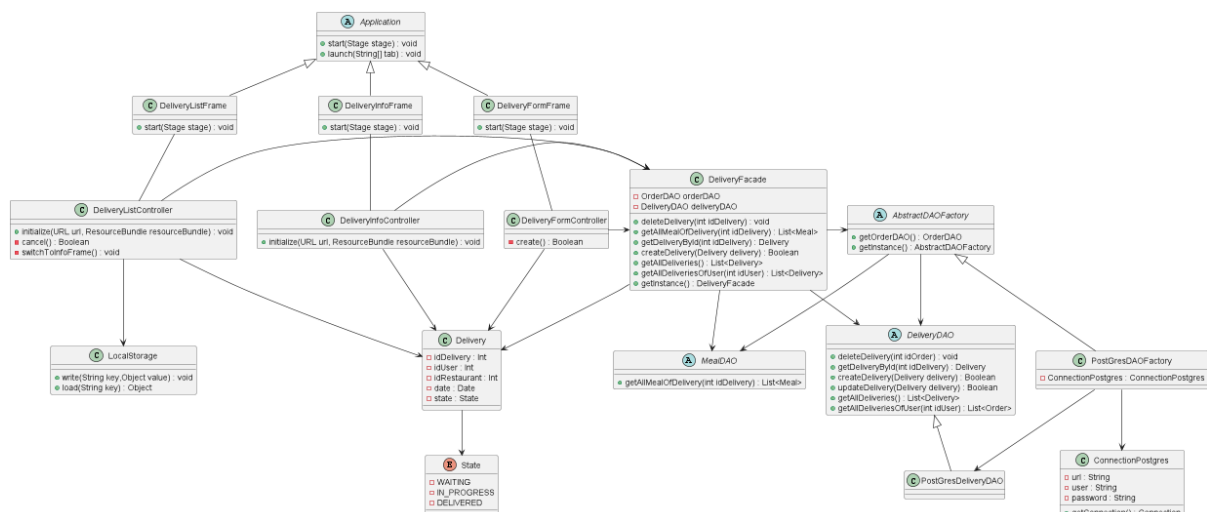


Le diagramme de séquence correspondant à la fonctionnalité “**validate**” du Cart :

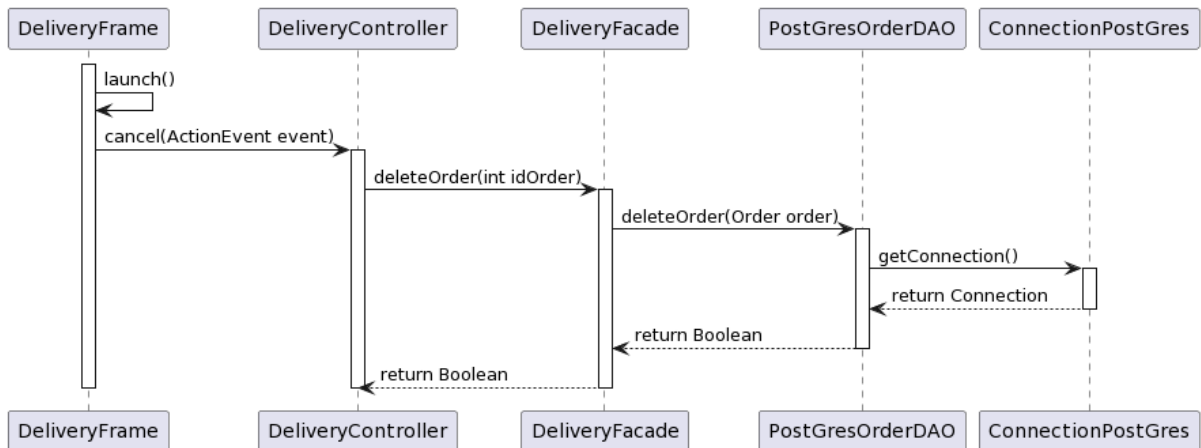


6.2.2 La livraison

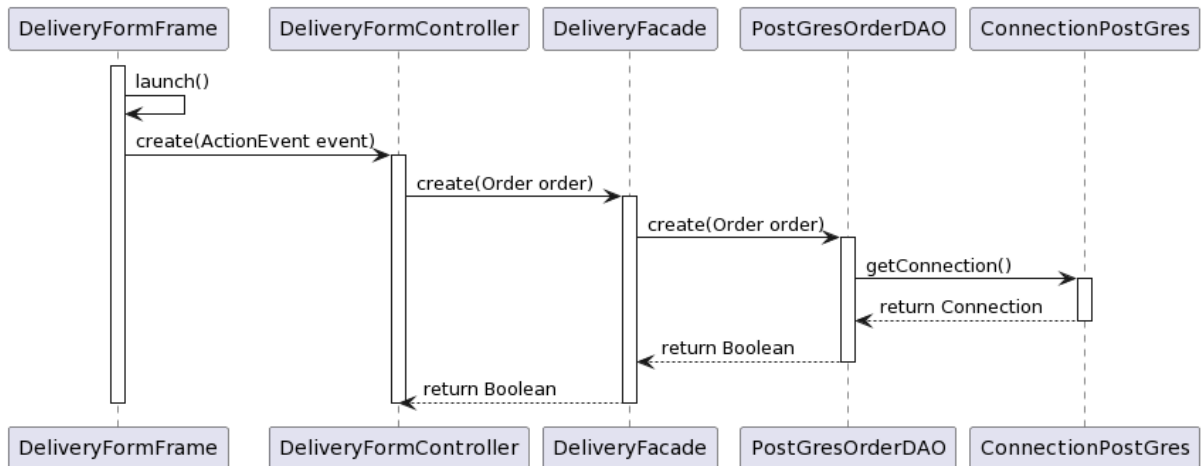
Le diagramme de classe pour le **Delivery** :



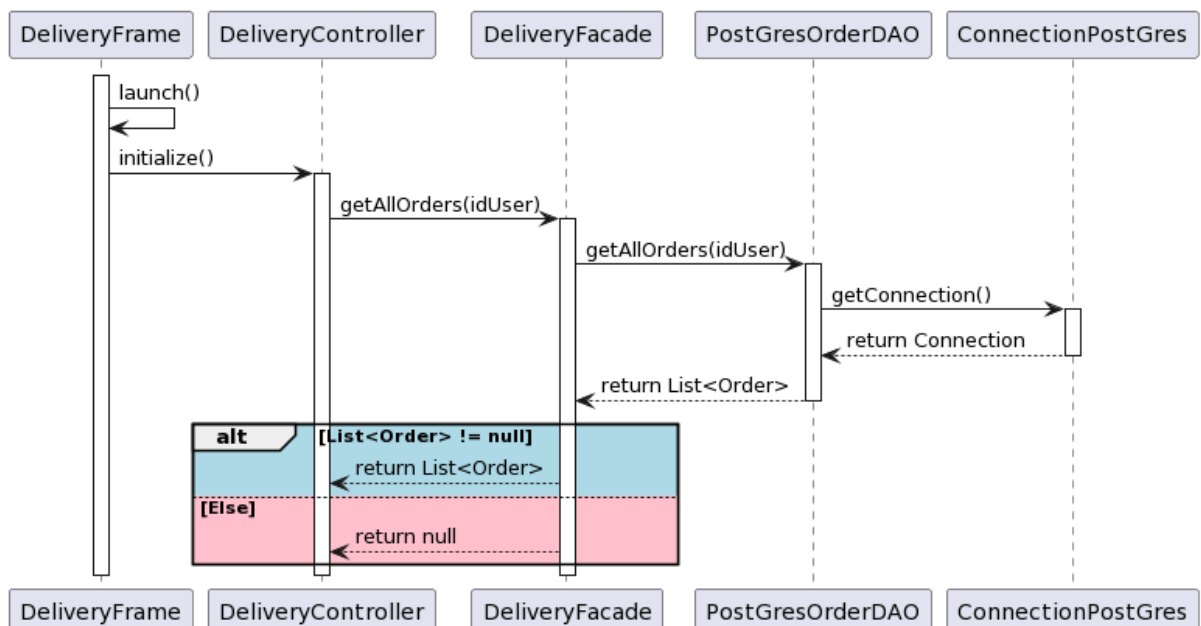
Le diagramme de séquence correspondant à la fonctionnalité “**cancel**” du Delivery :



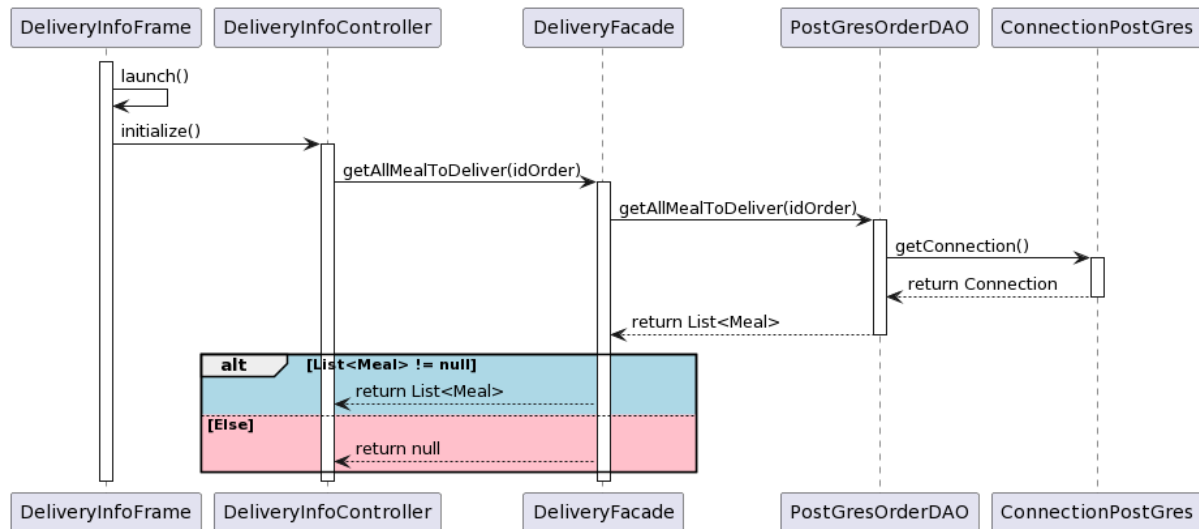
Le diagramme de séquence correspondant à la fonctionnalité “**create**” du Delivery :



Le diagramme de séquence correspondant à la fonctionnalité “**initialize**” du Delivery :

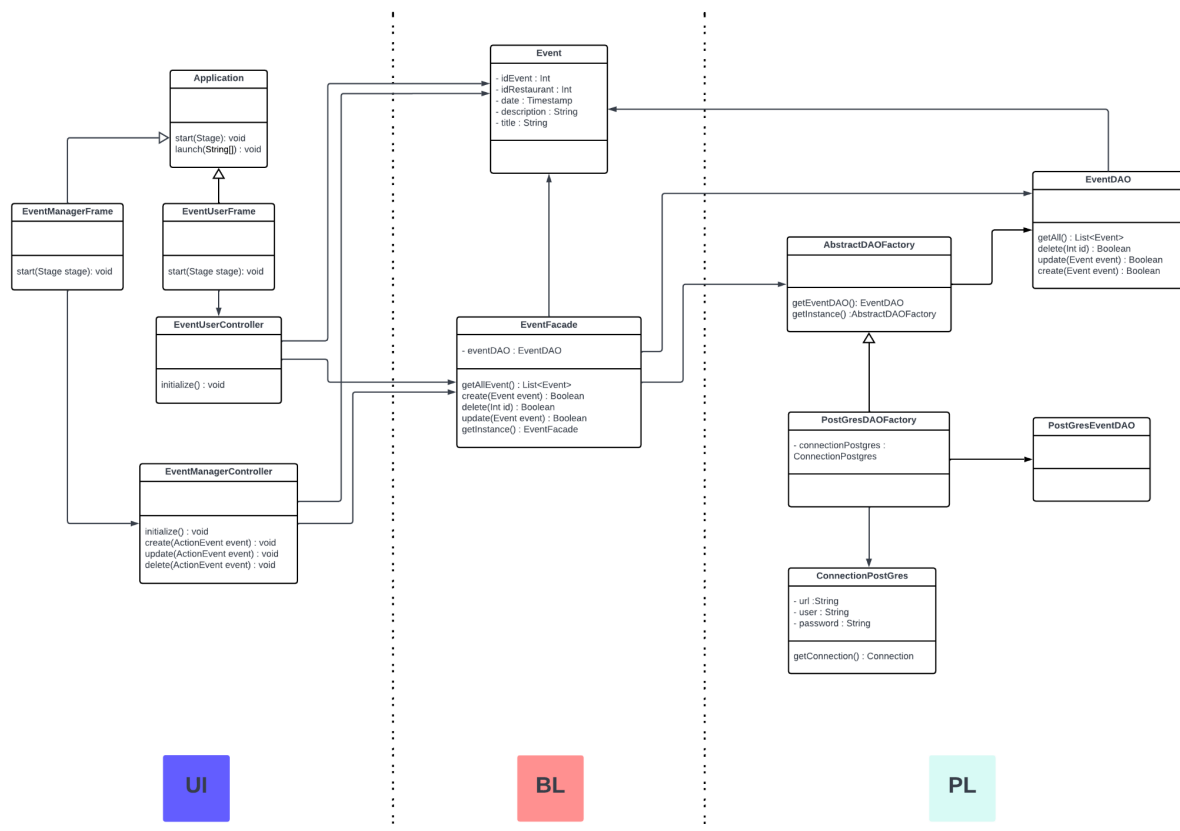


Le diagramme de séquence correspondant à la fonctionnalité “**initialize**” du Delivery pour la page avec les informations d’une livraison :

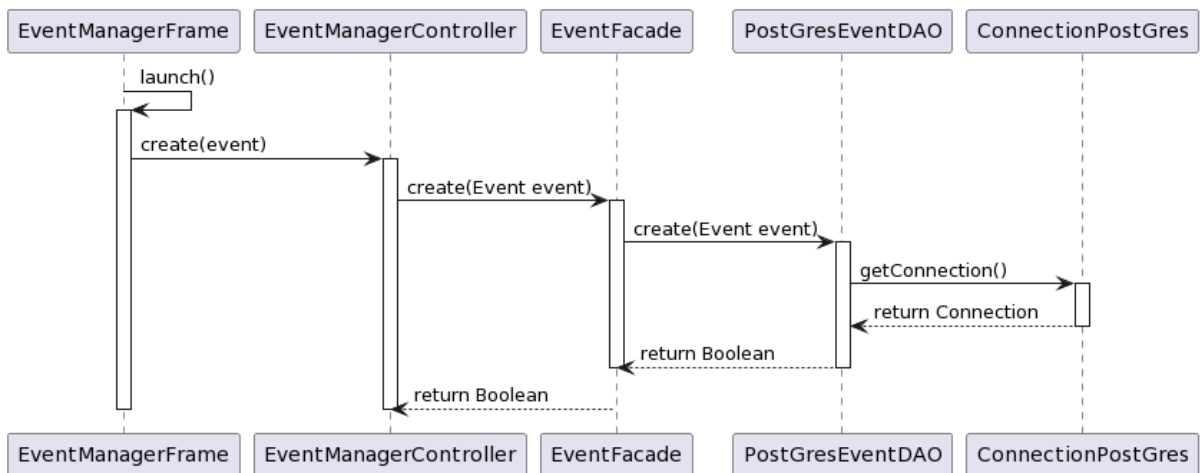


6.2.3 Les événements

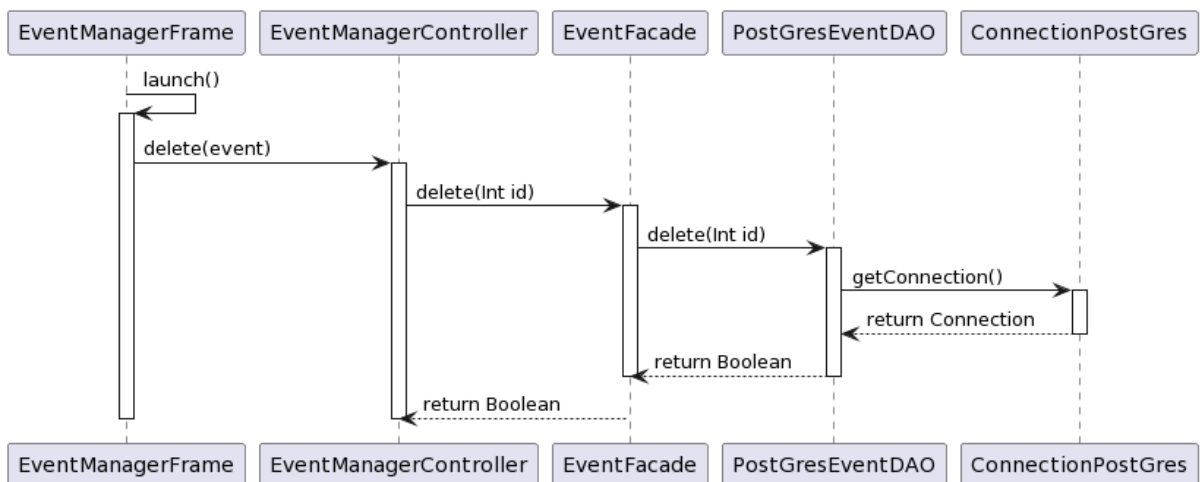
Le diagramme de classe pour le **Event** :



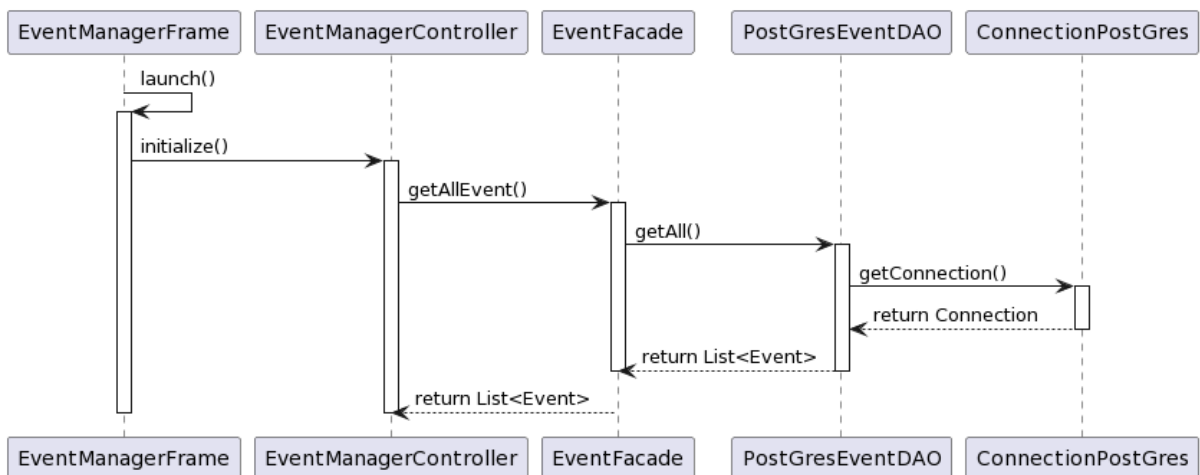
Le diagramme de séquence correspondant à la fonctionnalité “**create**” de Event :



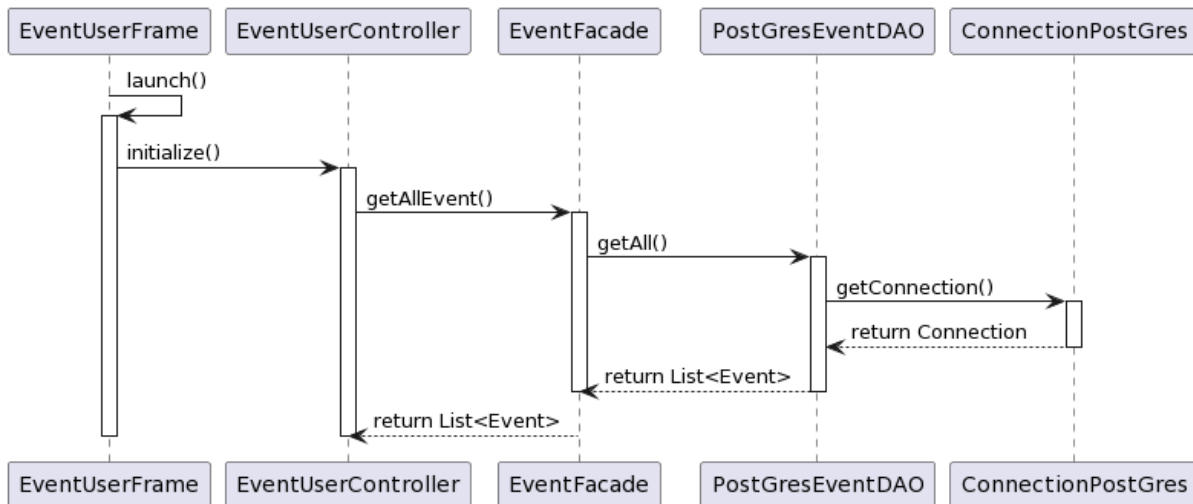
Le diagramme de séquence correspondant à la fonctionnalité “**delete**” de Event :



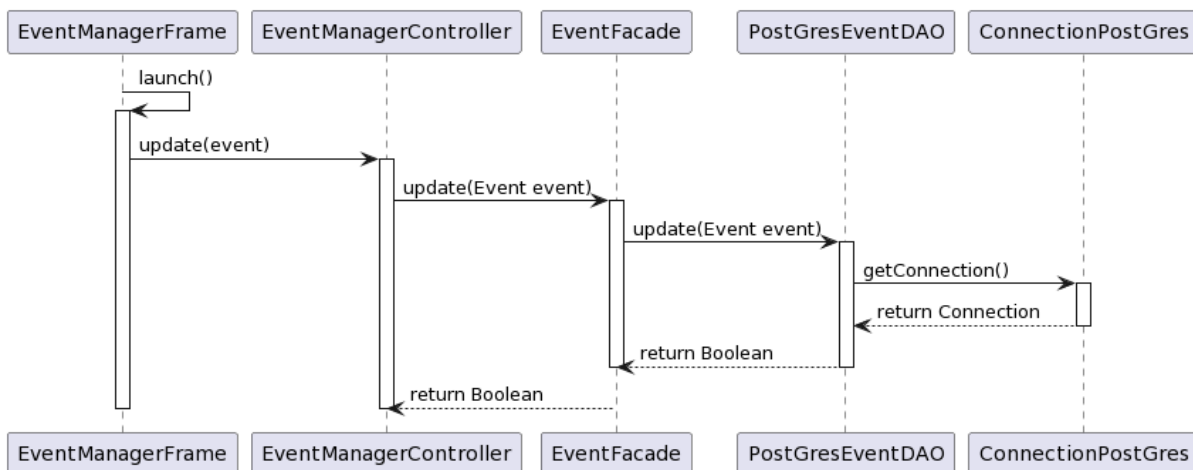
Le diagramme de séquence correspondant à la fonctionnalité “**initialize**” de Event coté manager :



Le diagramme de séquence correspondant à la fonctionnalité “**initialize**” de Event coté User :

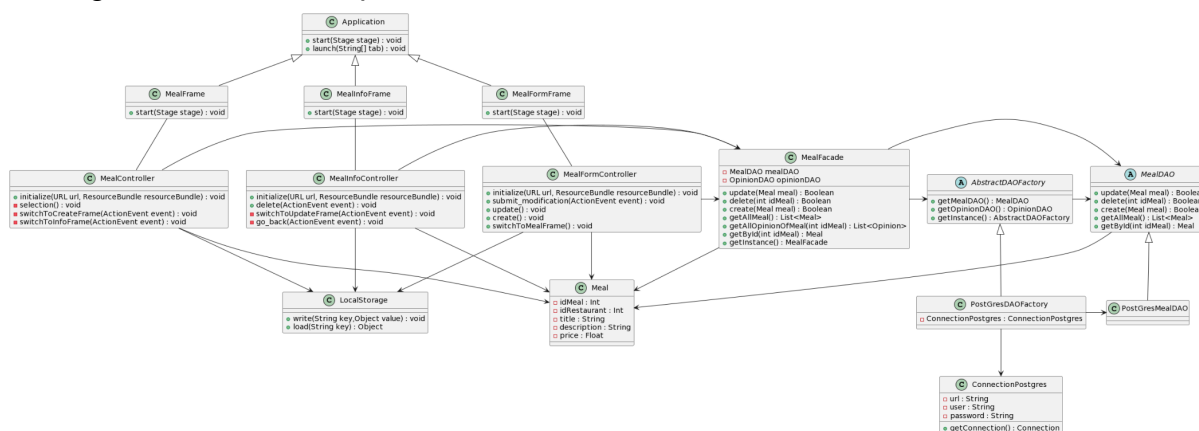


Le diagramme de séquence correspondant à la fonctionnalité “**update**” de Event :

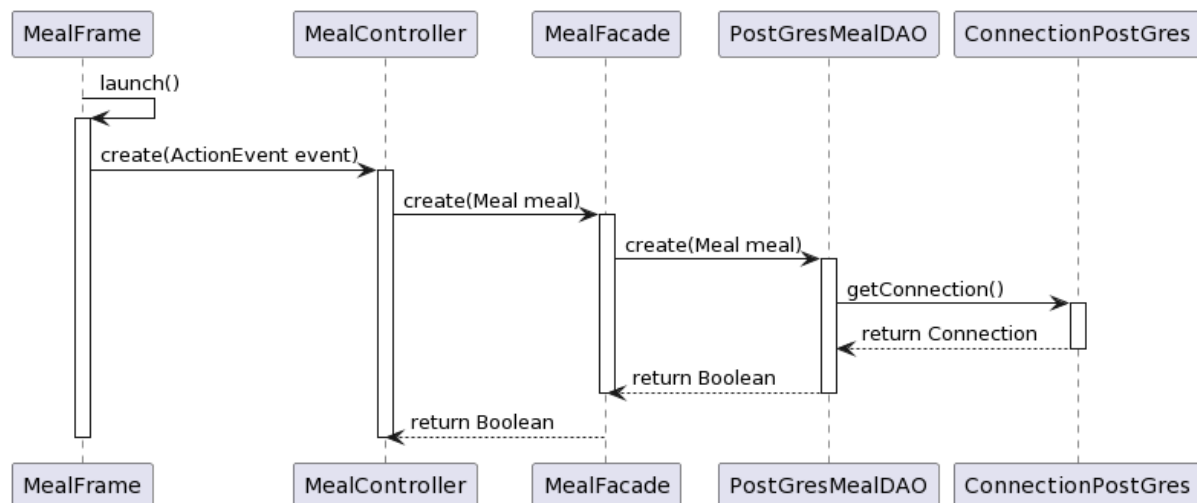


6.2.4 Le plat

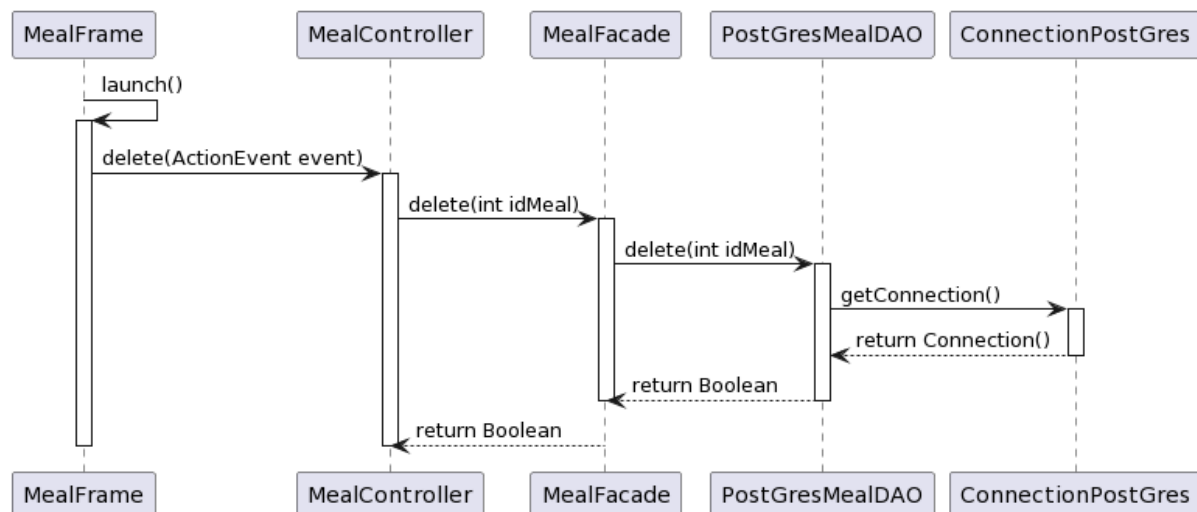
Le diagramme de classe pour **Meal**:



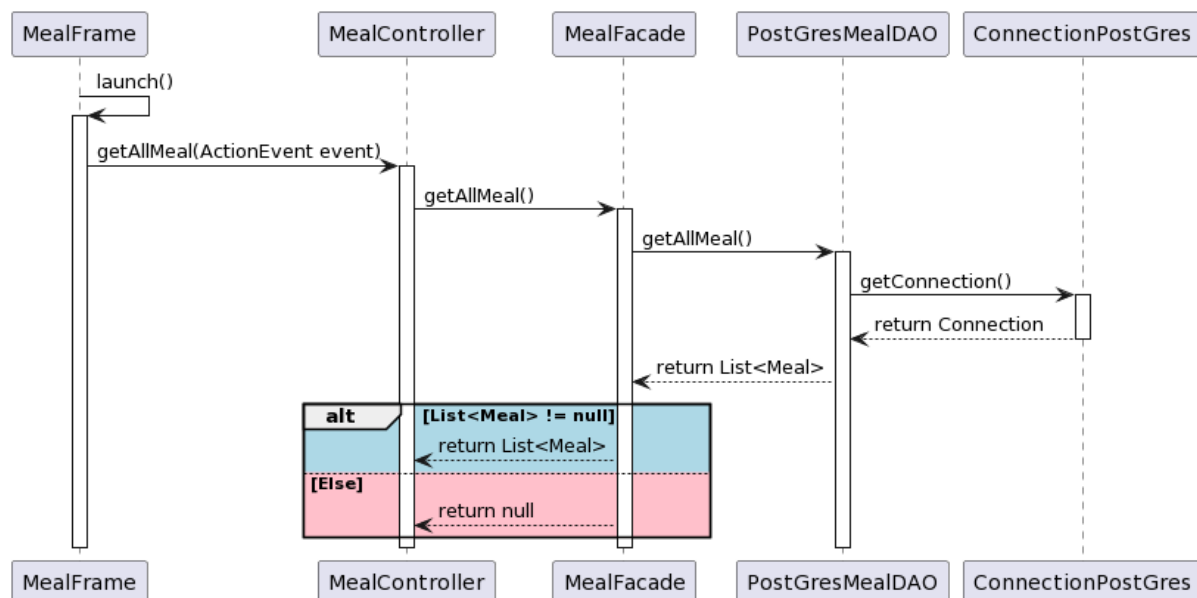
Le diagramme de séquence correspondant à la fonctionnalité “**create**” de Meal :



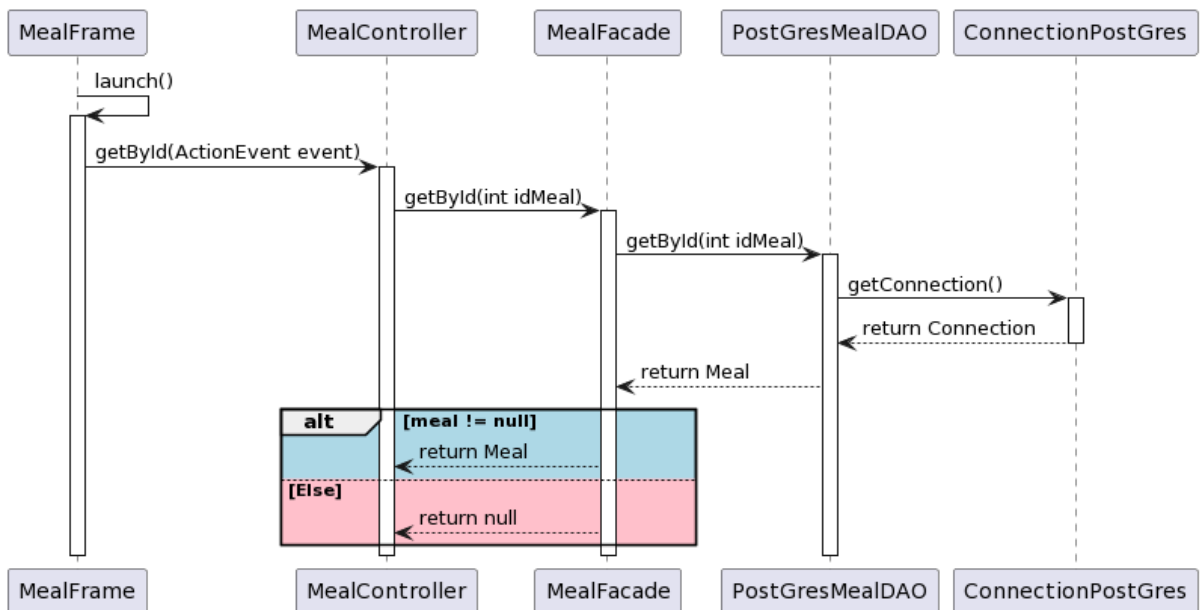
Le diagramme de séquence correspondant à la fonctionnalité “**delete**” de Meal :



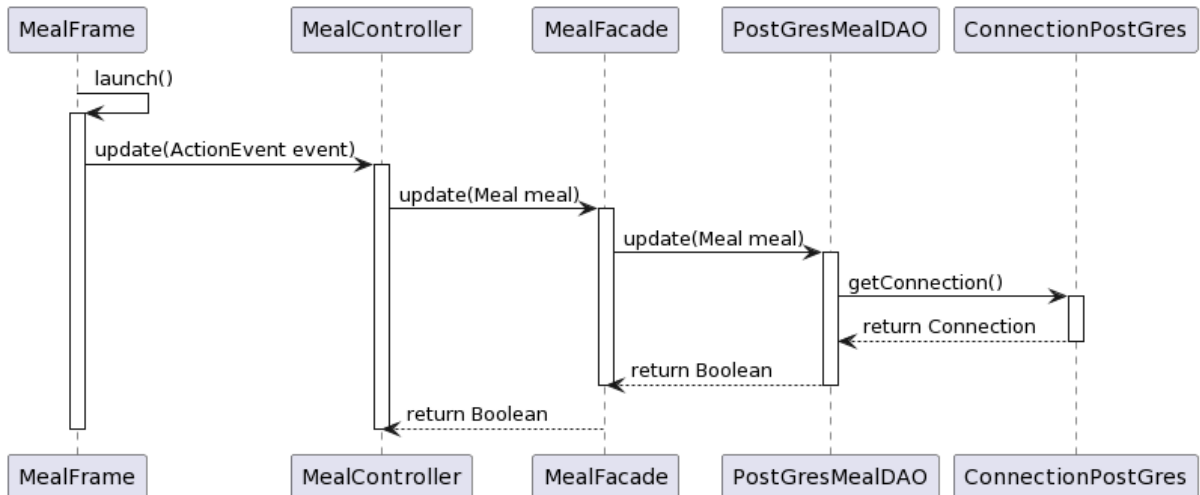
Le diagramme de séquence correspondant à la fonctionnalité “**getAllMeal**” de Meal :



Le diagramme de séquence correspondant à la fonctionnalité “**getById**” de Meal :

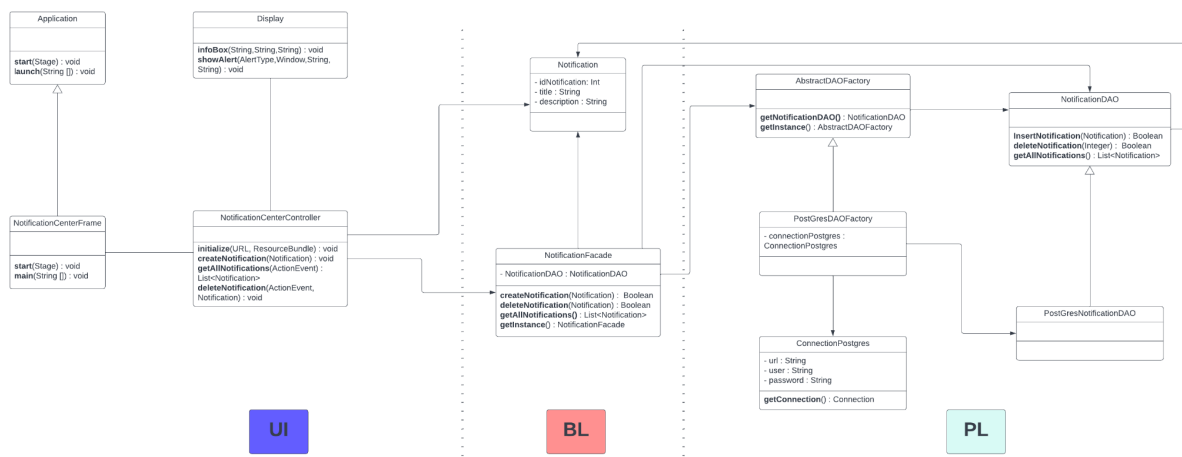


Le diagramme de séquence correspondant à la fonctionnalité **“update”** de Meal :

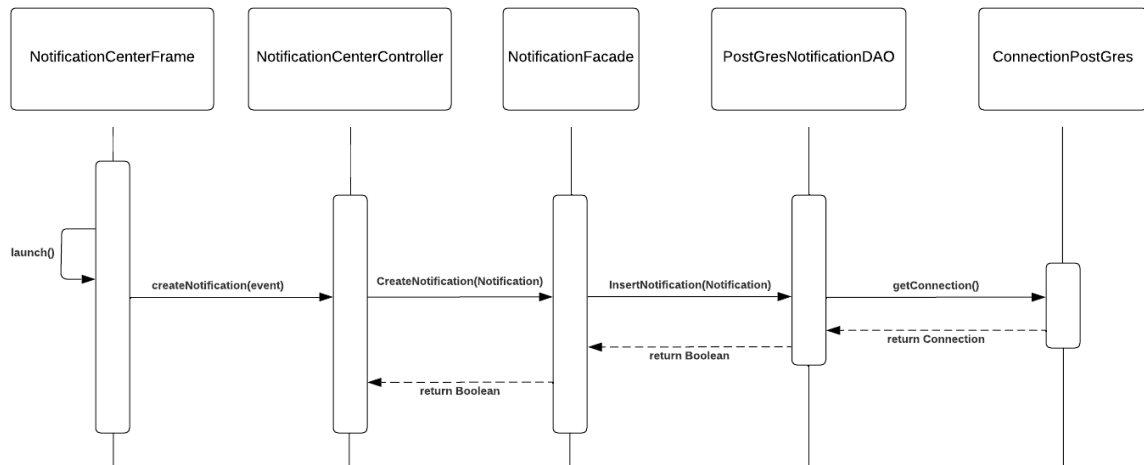


6.2.5 La notification

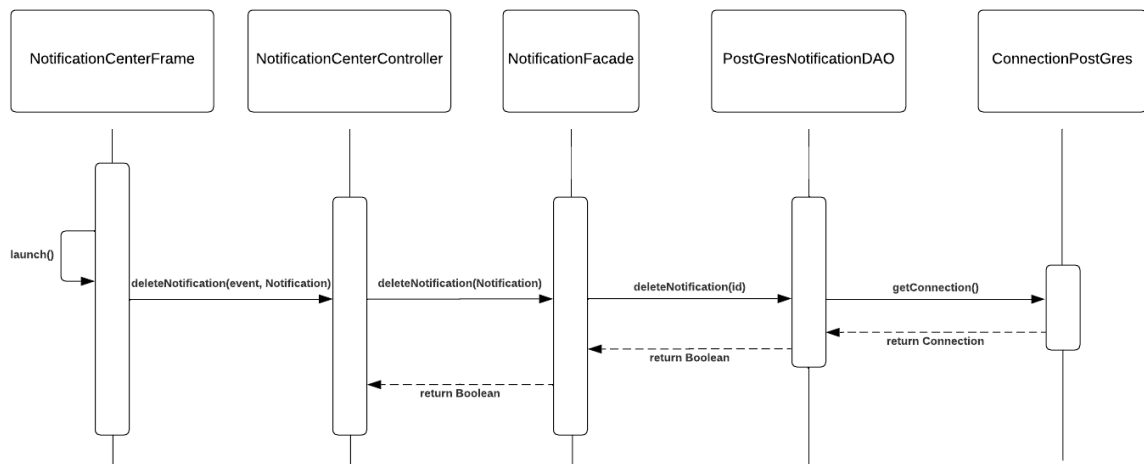
Le diagramme de classe pour **Notification**:



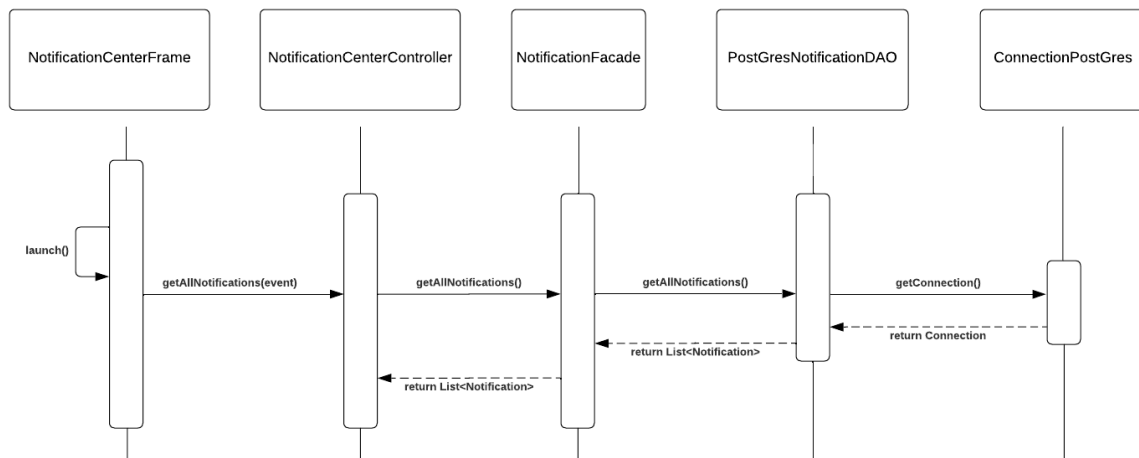
Le diagramme de séquence correspondant à la fonctionnalité “**createNotification**” de Notification :



Le diagramme de séquence correspondant à la fonctionnalité “**deleteNotification**” de Notification :

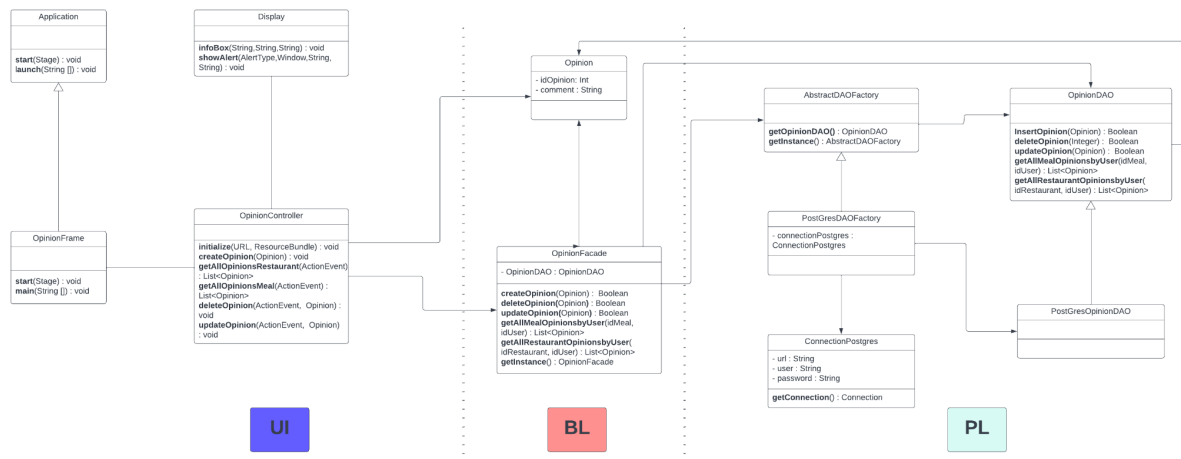


Le diagramme de séquence correspondant à la fonctionnalité “**getAllNotification**” de Notification :

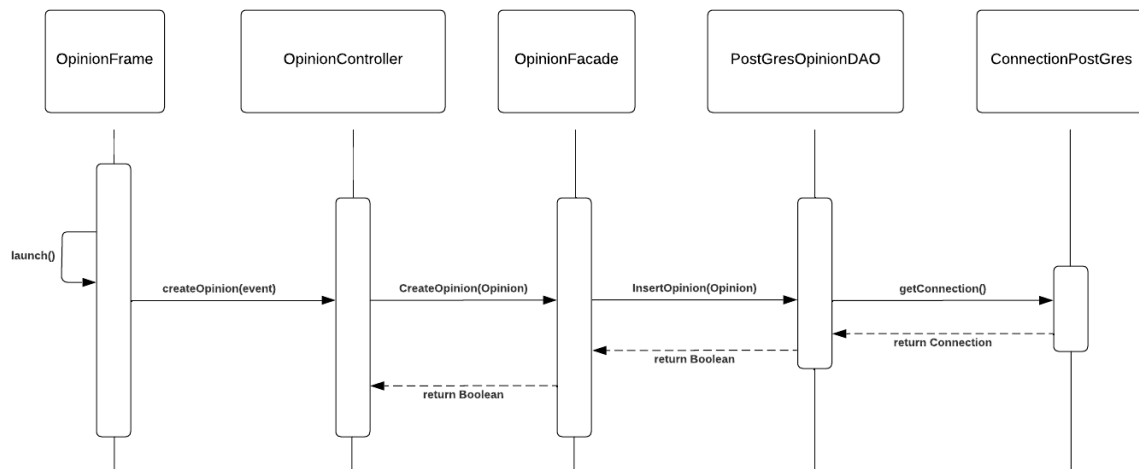


6.2.6 L'opinion

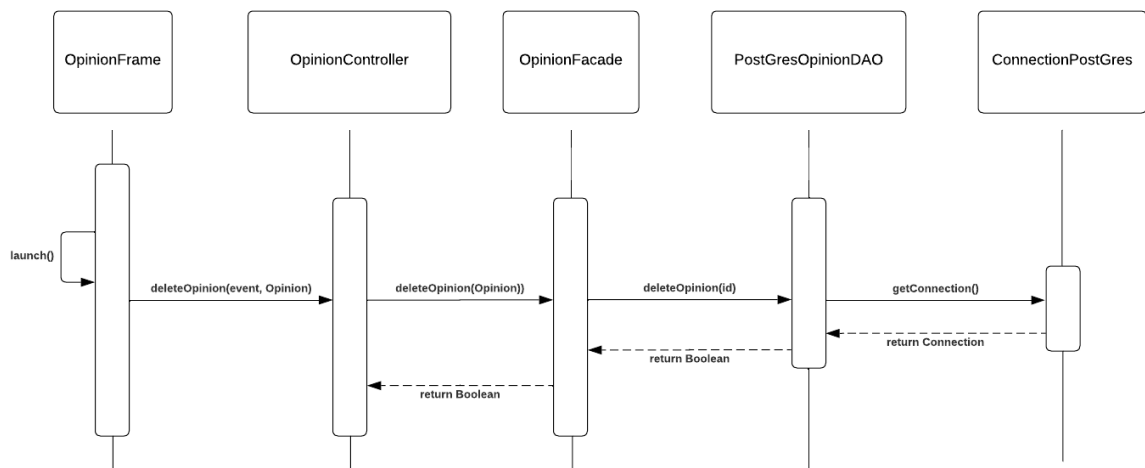
Le diagramme de classe pour **Opinion** :



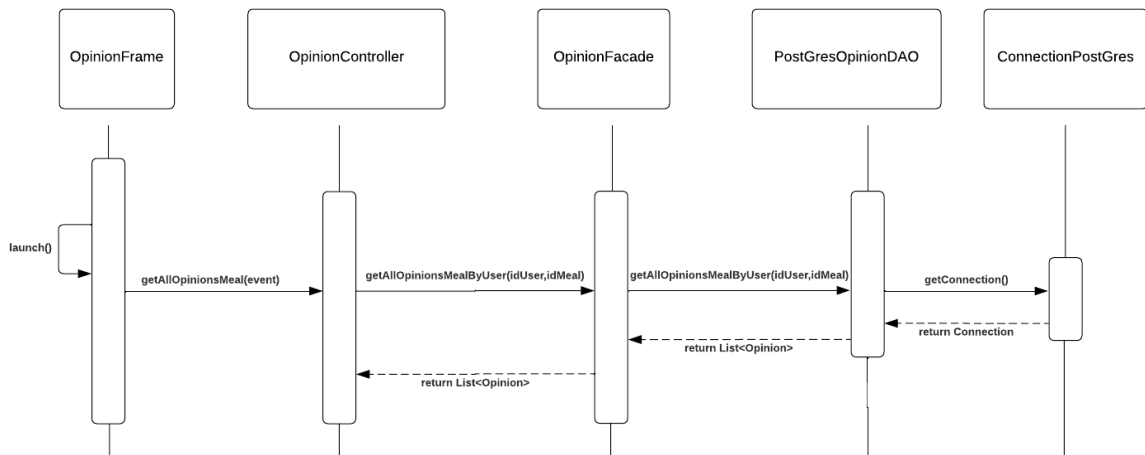
Le diagramme de séquence correspondant à la fonctionnalité “**createOpinion**” de Opinion:



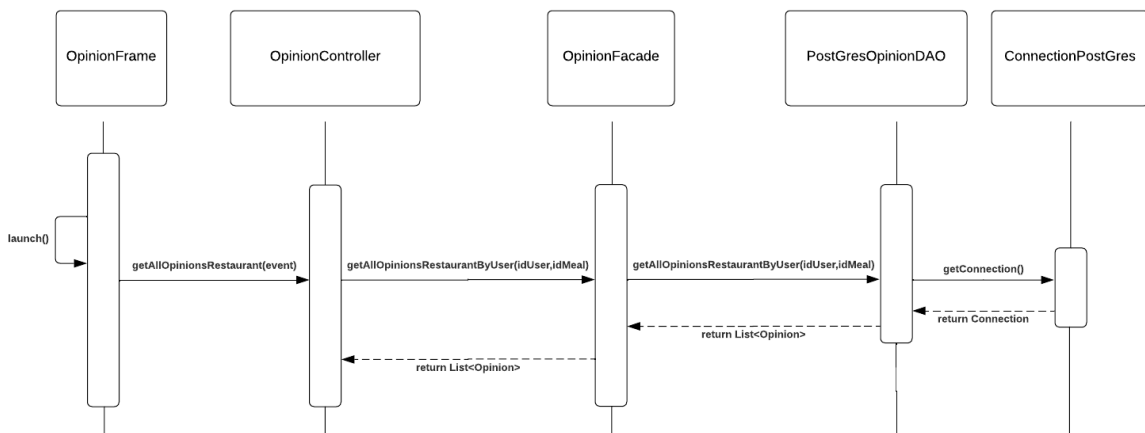
Le diagramme de séquence correspondant à la fonctionnalité “**deleteOpinion**” de Opinion:



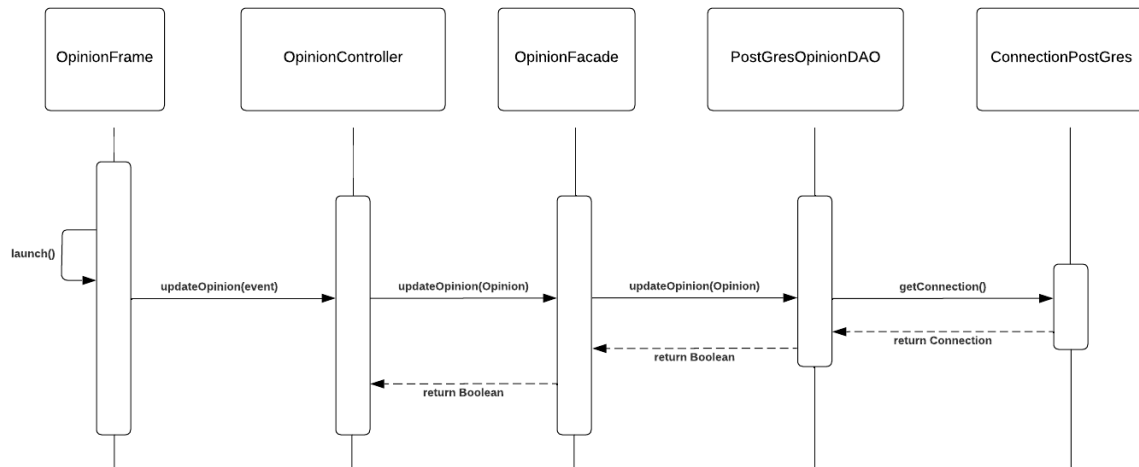
Le diagramme de séquence correspondant à la fonctionnalité “**getAllOpinionMeal**” de Opinion :



Le diagramme de séquence correspondant à la fonctionnalité “**getAllOpinionRestaurant**” de Opinion :

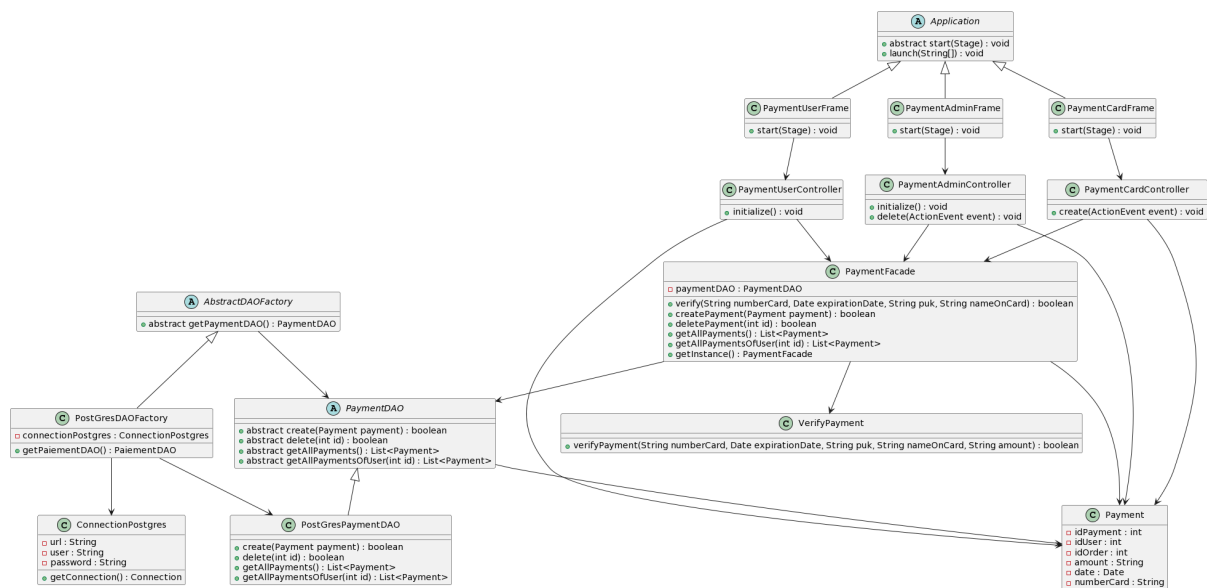


Le diagramme de séquence correspondant à la fonctionnalité “**updateOpinion**” de Opinion :

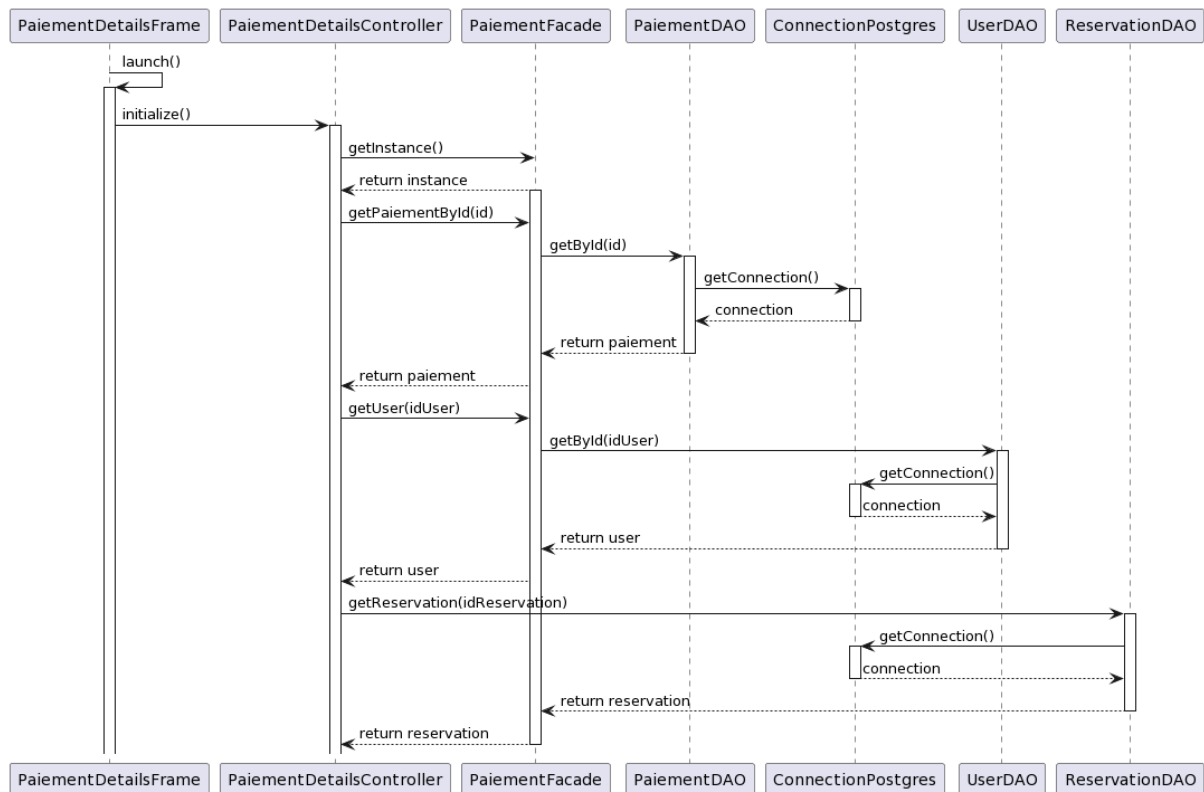


6.2.7 Le paiement

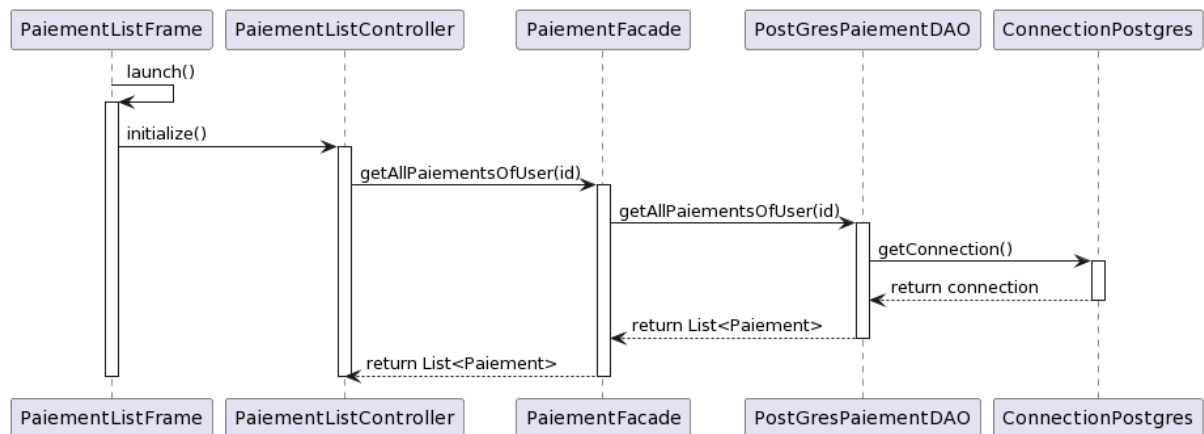
Le diagramme de classe pour **Paiement** :



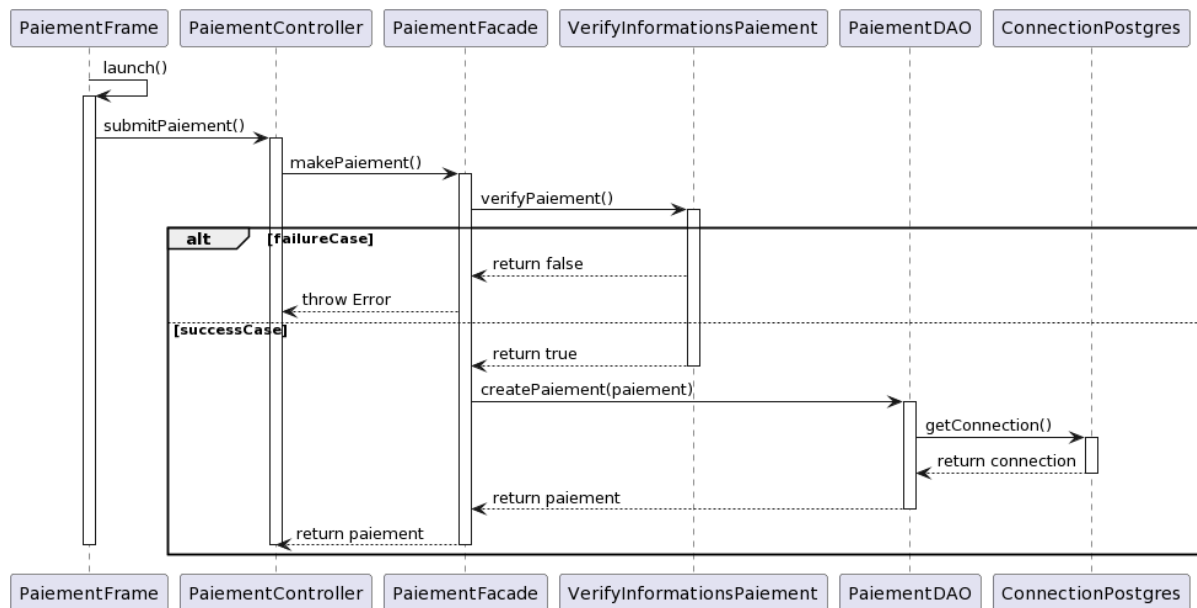
Le diagramme de séquence correspondant à la fonctionnalité “**getDetails**” de Paiement :



Le diagramme de séquence correspondant à la fonctionnalité “**getHistorique**” de Paiement :



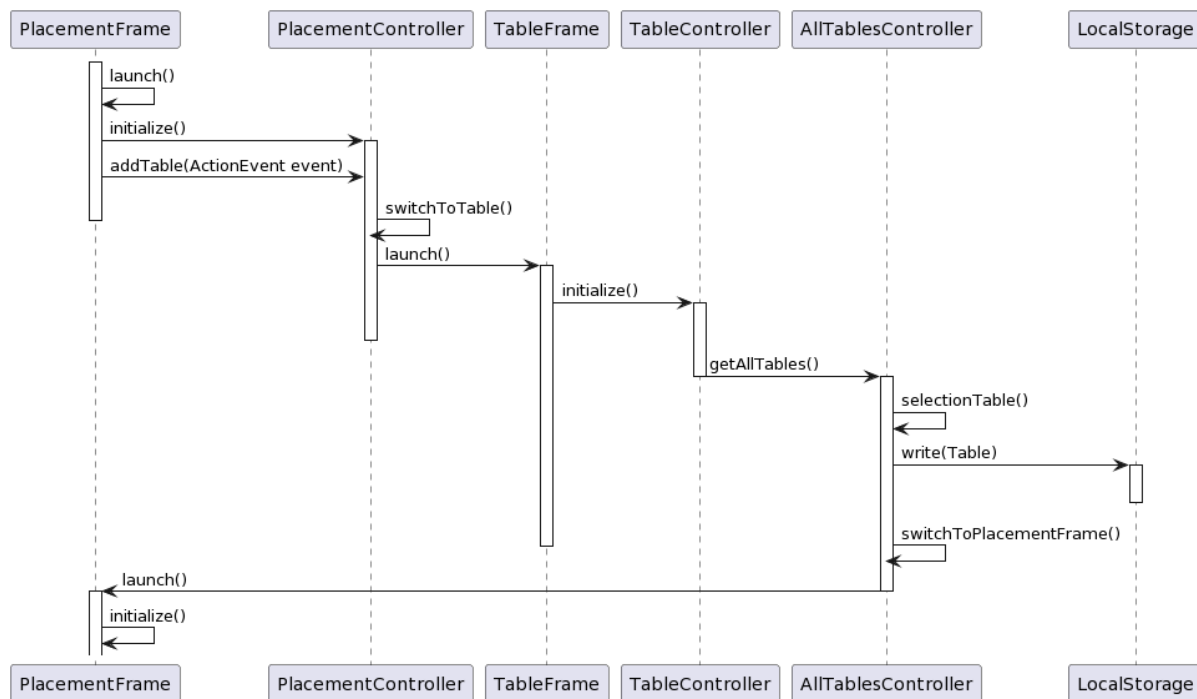
Le diagramme de séquence correspondant à la fonctionnalité “**makePaie**ment” de Paiement :



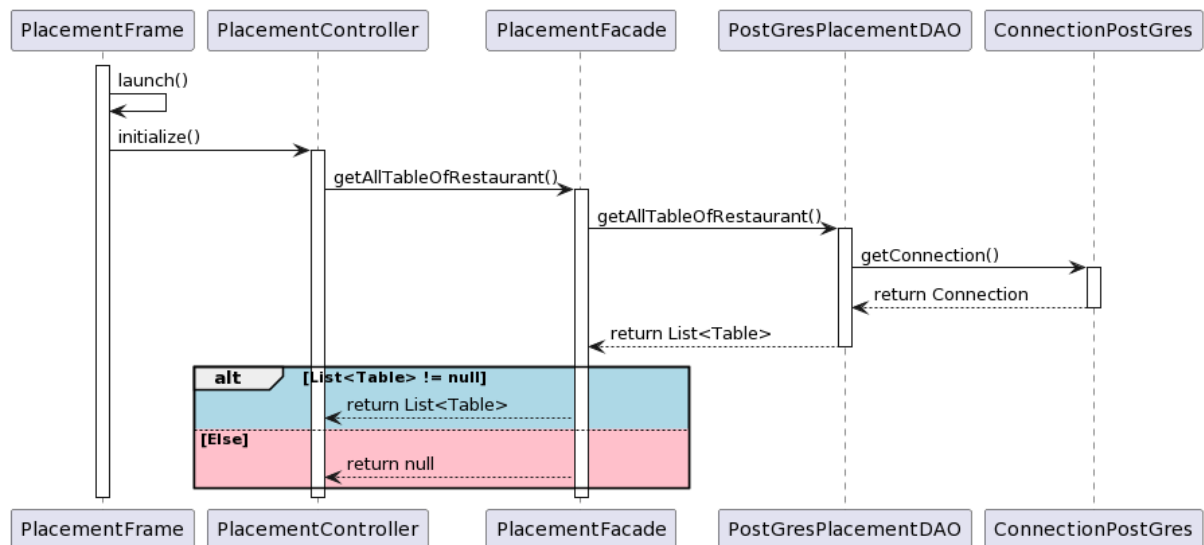
6.2.8 Le placement des tables

Le diagramme de classe pour **Placement** :

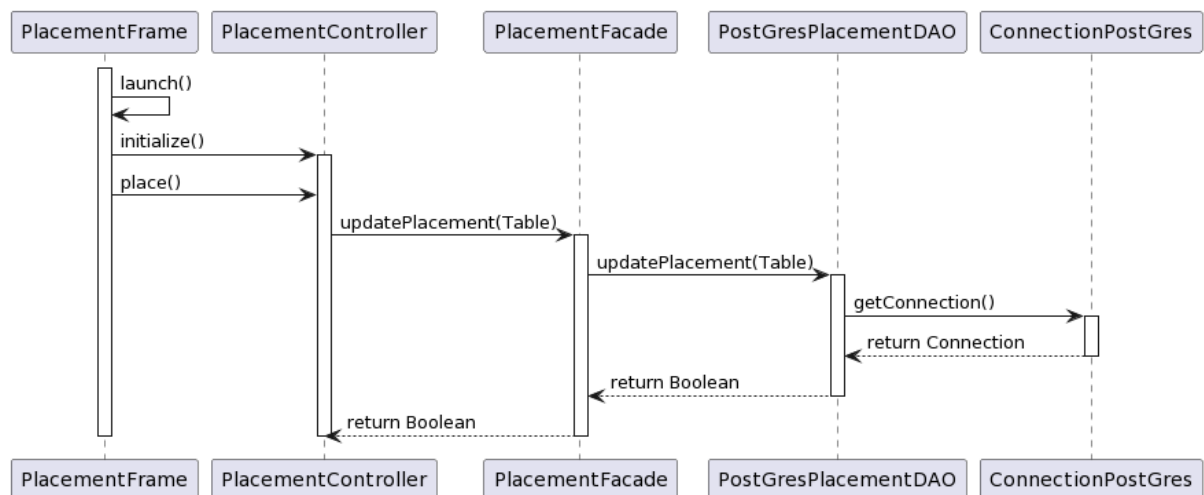
Le diagramme de séquence correspondant à la fonctionnalité “**addTable**” de Placement :



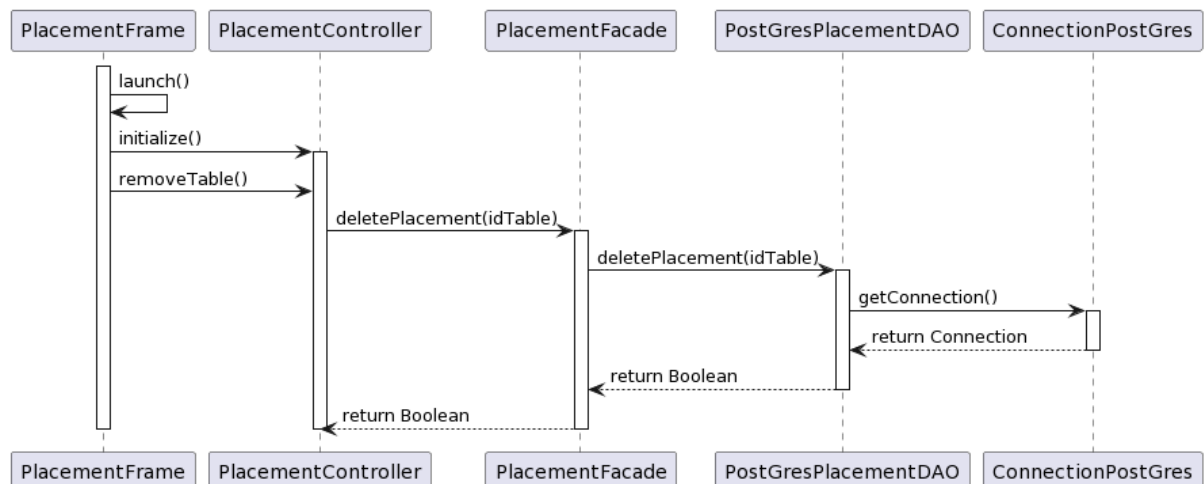
Le diagramme de séquence correspondant à la fonctionnalité “**initialize**” de Placement :



Le diagramme de séquence correspondant à la fonctionnalité “**place**” de Placement :

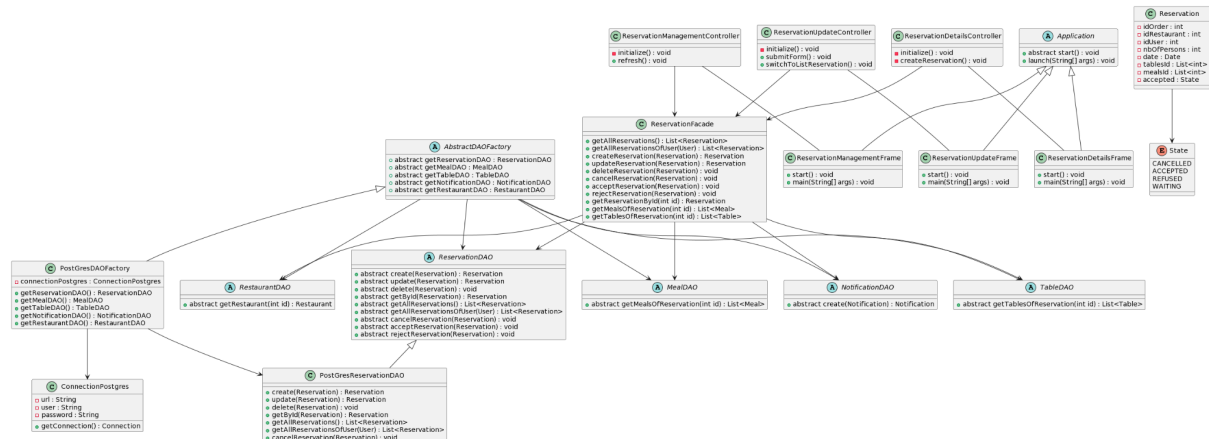


Le diagramme de séquence correspondant à la fonctionnalité “**removeTable**” de Placement :

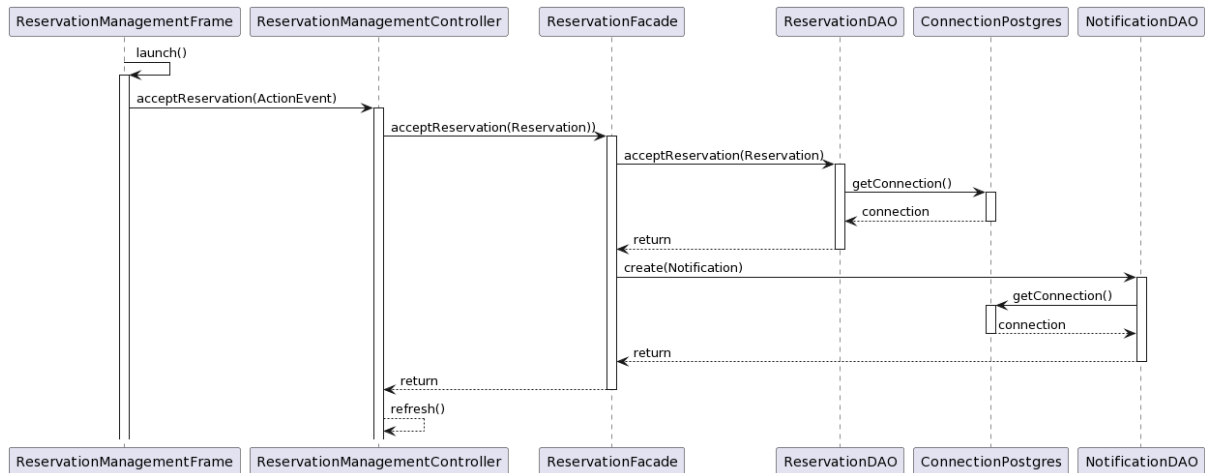


6.2.9 La réservation

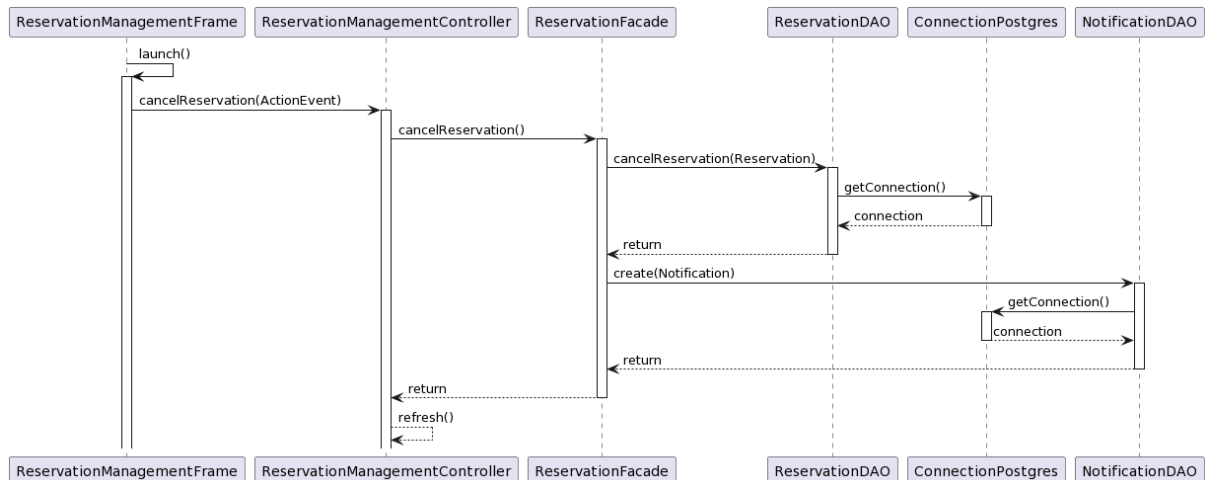
Le diagramme de classe pour **Reservation** :



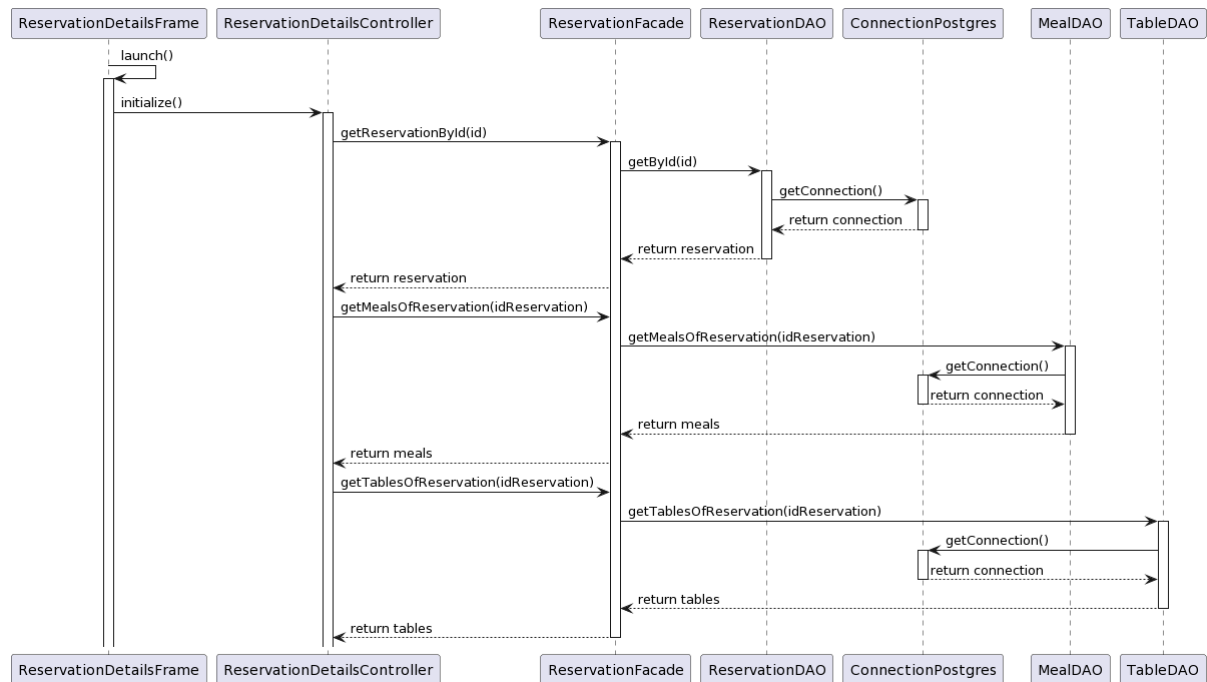
Le diagramme de séquence correspondant à la fonctionnalité “**acceptReservation**” de Reservation :



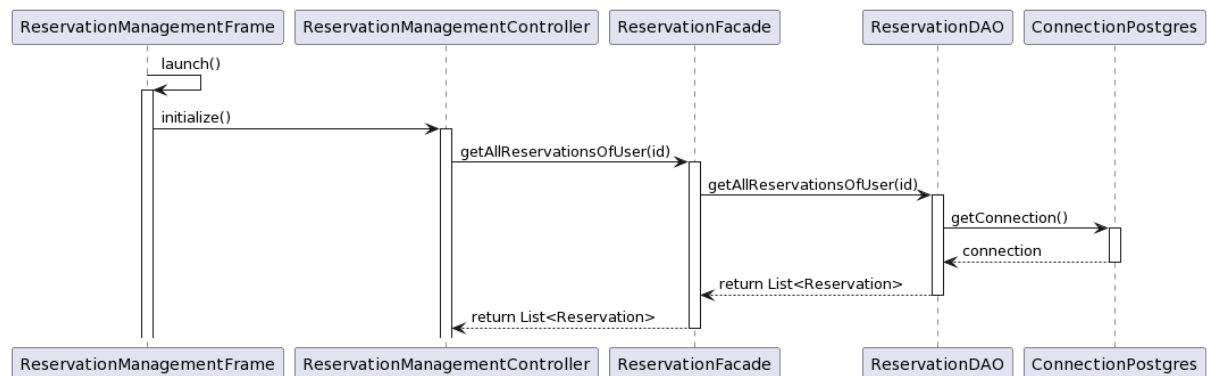
Le diagramme de séquence correspondant à la fonctionnalité “**cancelReservation**” de Reservation :



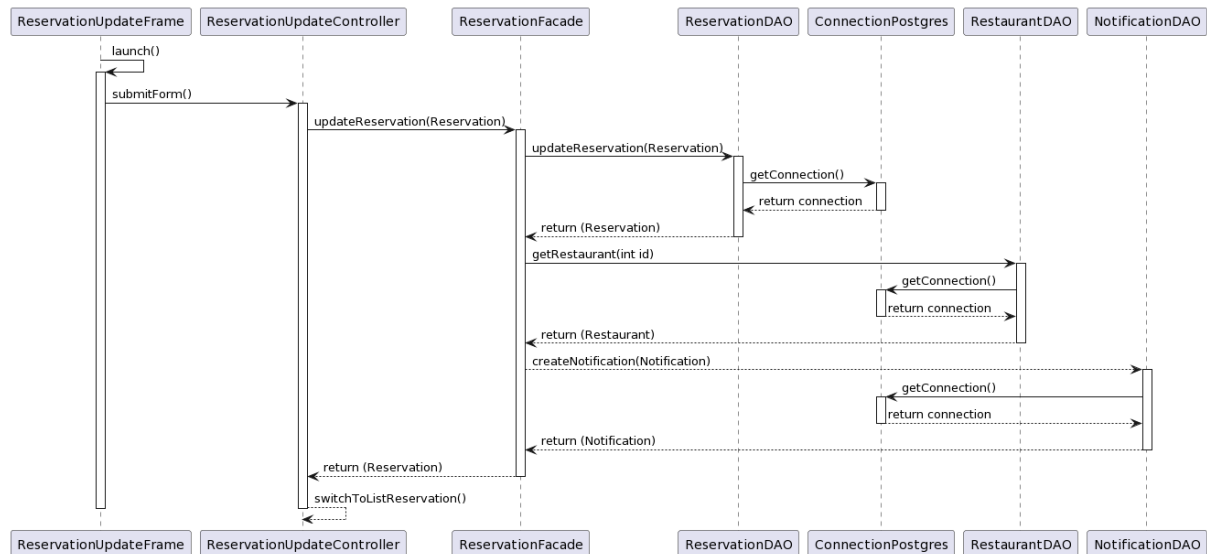
Le diagramme de séquence correspondant à la fonctionnalité “**getDetailsOfReservation**” de Reservation :



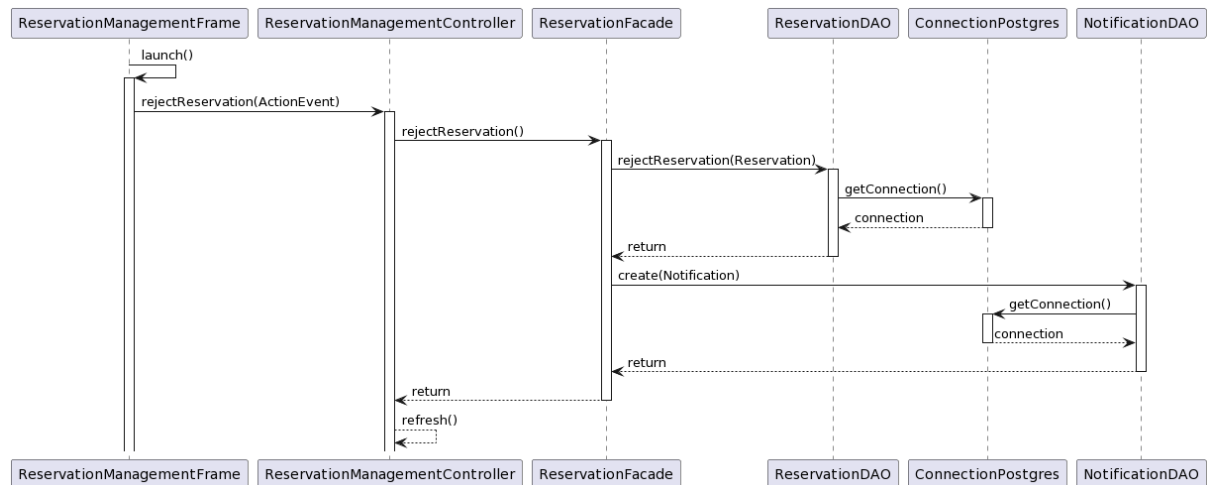
Le diagramme de séquence correspondant à la fonctionnalité “**getListReservation**” de Reservation :



Le diagramme de séquence correspondant à la fonctionnalité “**updateReservation**” de Reservation :

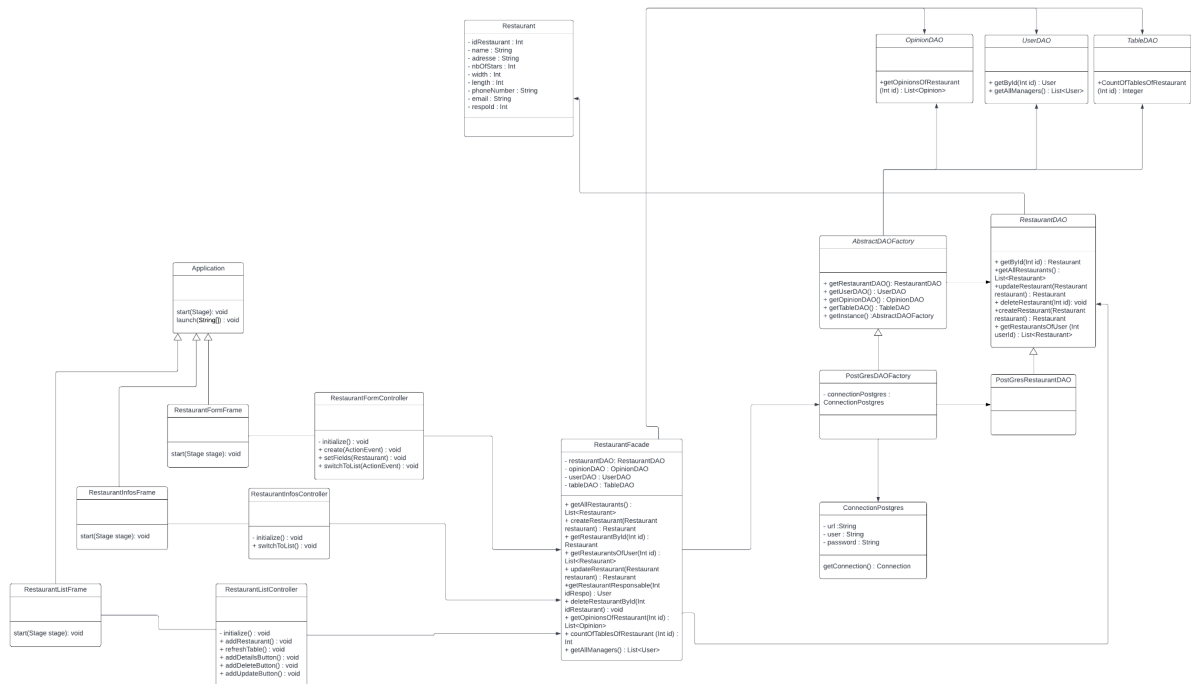


Le diagramme de séquence correspondant à la fonctionnalité “**deleteReservation**” de Reservation :

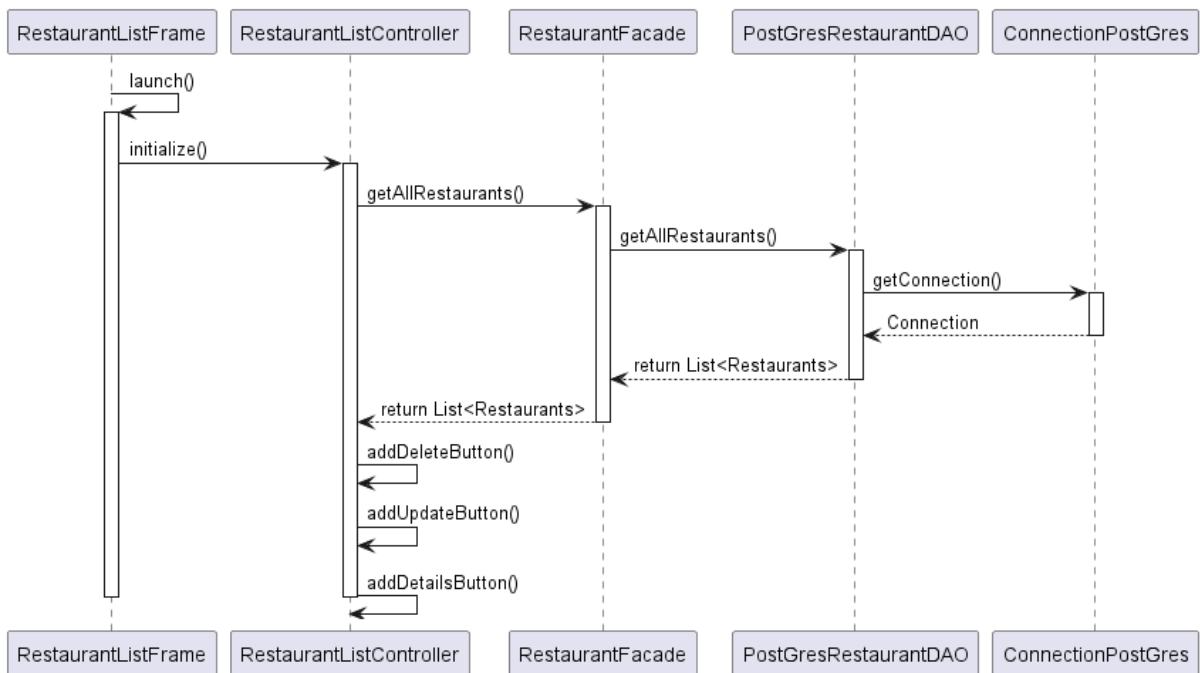


6.2.10 Le restaurant

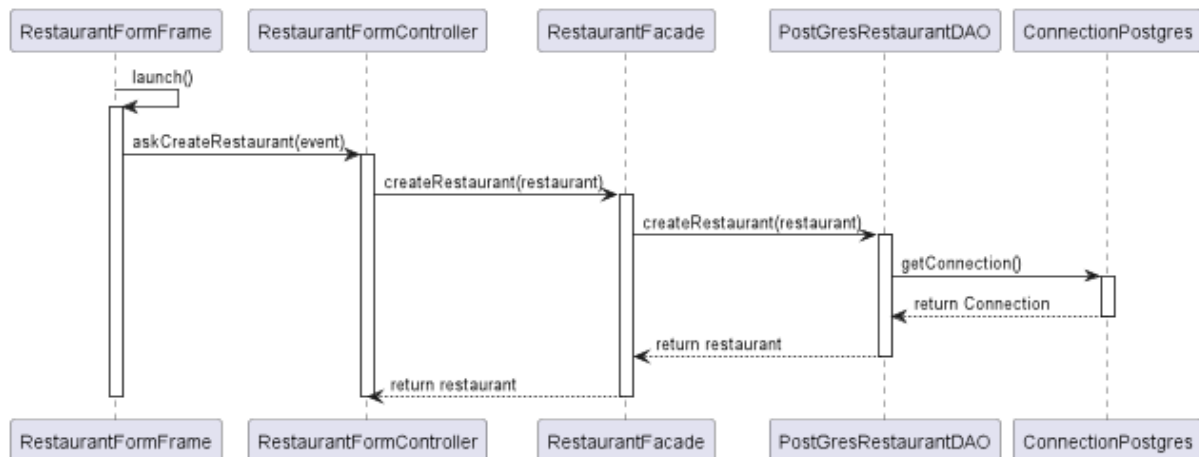
Le diagramme de classe pour **Restaurant** :



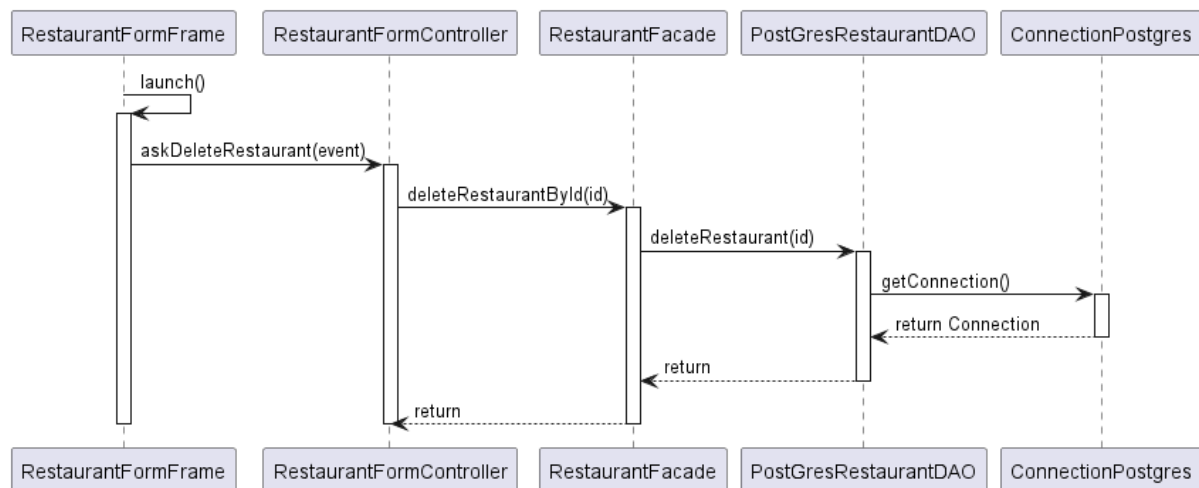
Le diagramme de séquence correspondant à la fonctionnalité **“consulterRestaurant”** de Restaurant :



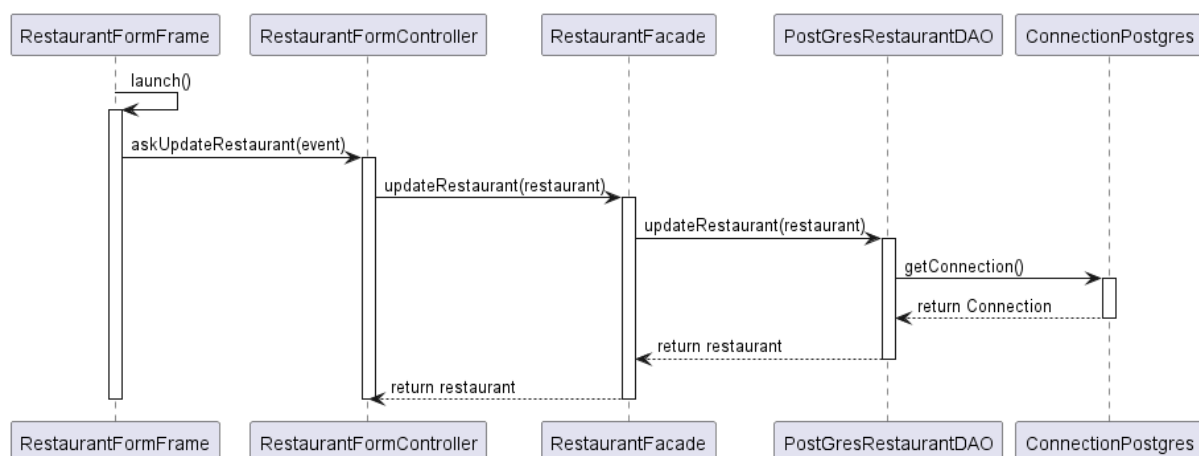
Le diagramme de séquence correspondant à la fonctionnalité “**createRestaurant**” de Restaurant :



Le diagramme de séquence correspondant à la fonctionnalité “**deleteRestaurant**” de Restaurant :

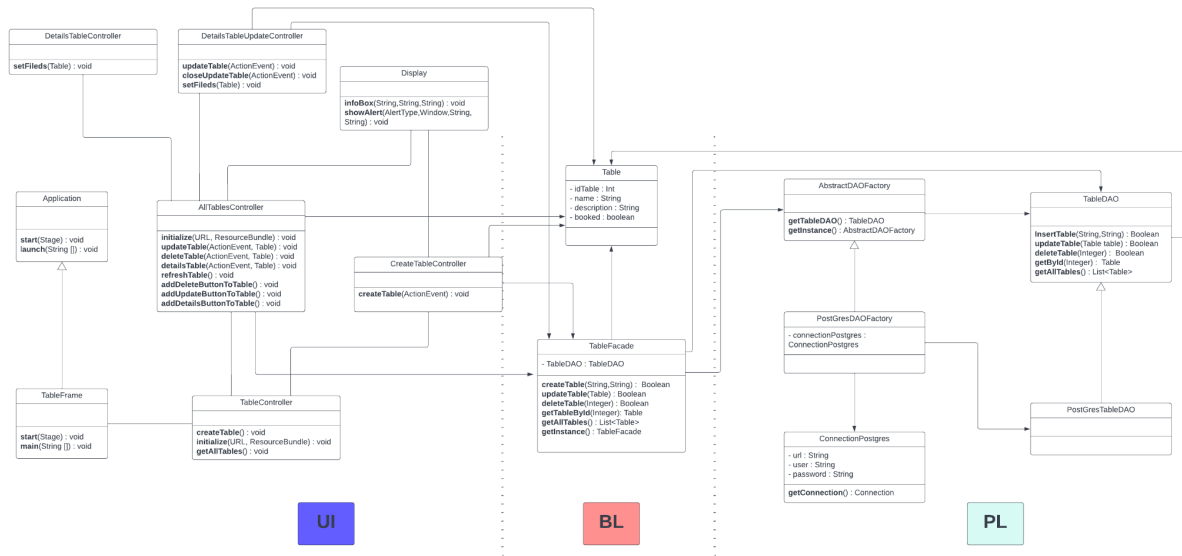


Le diagramme de séquence correspondant à la fonctionnalité “**updateRestaurant**” de Restaurant :

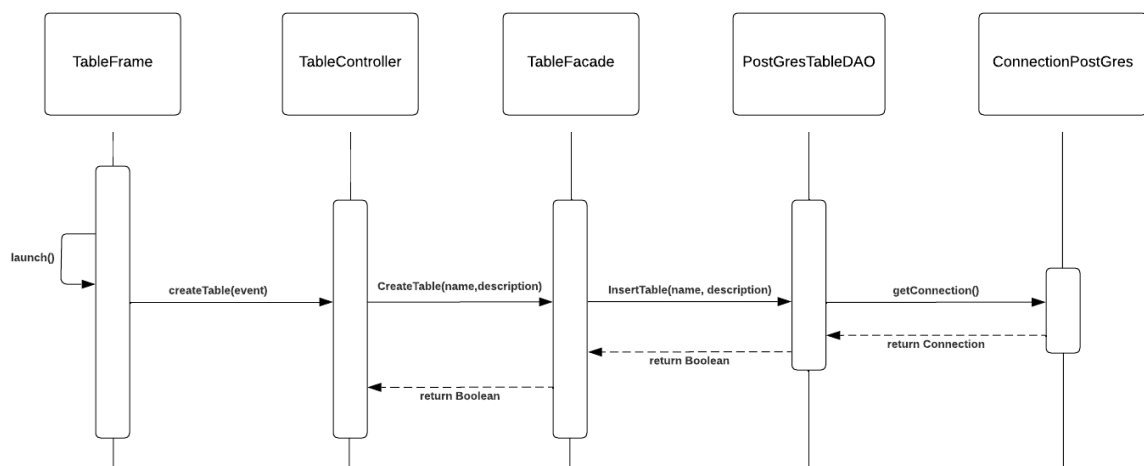


6.2.11 La table

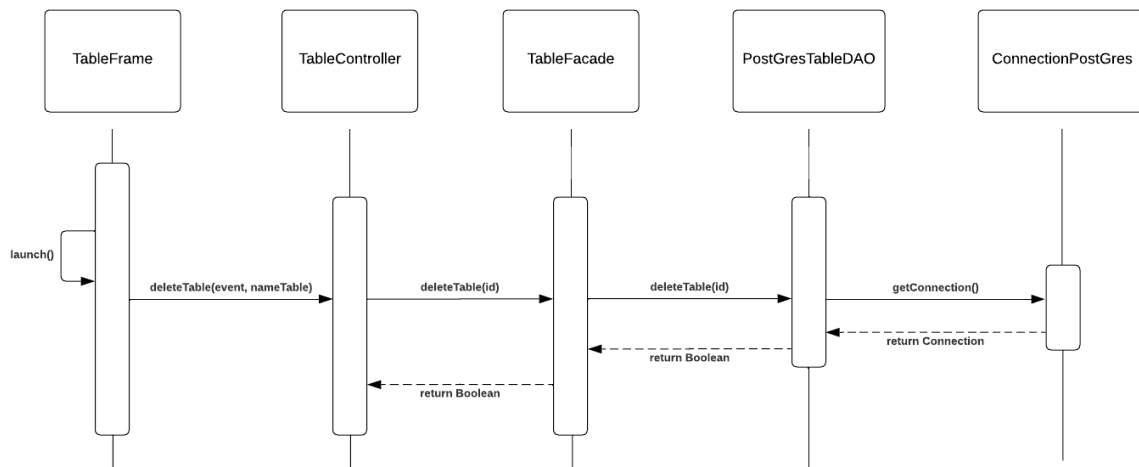
Le diagramme de classe pour **Table** :



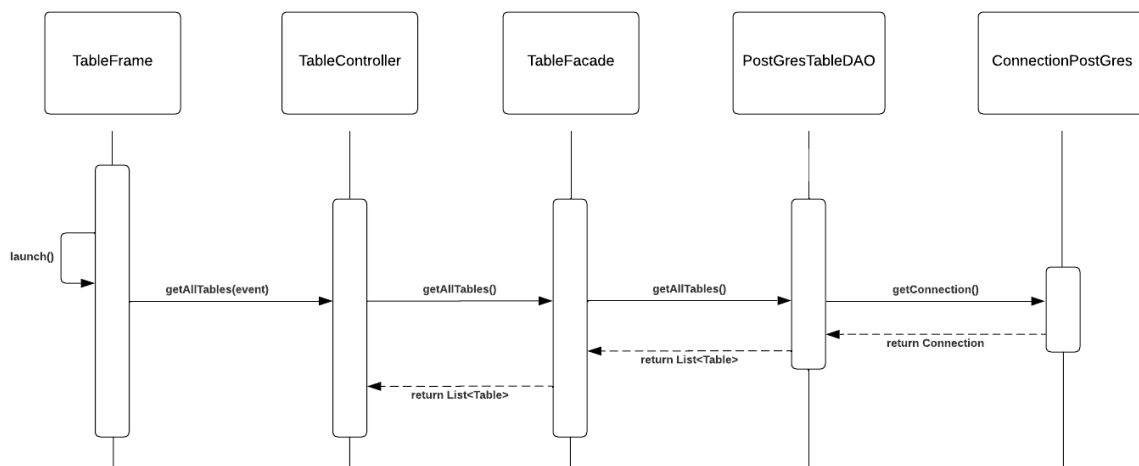
Le diagramme de séquence correspondant à la fonctionnalité “**createTable**” de **Table** :



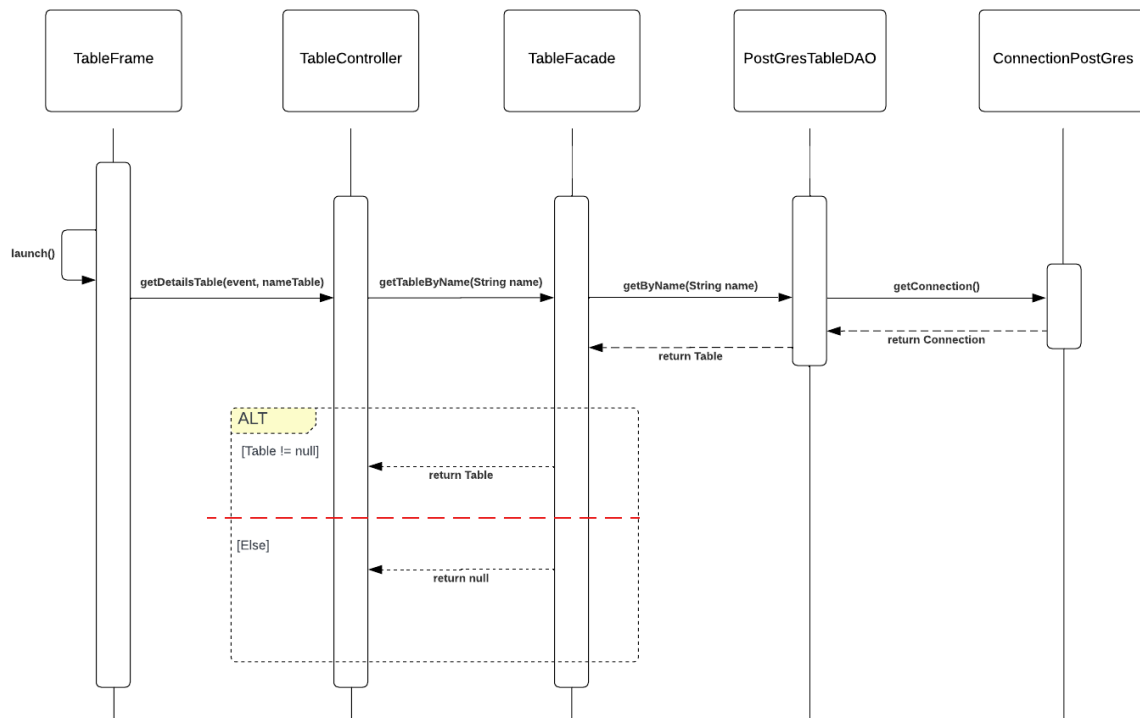
Le diagramme de séquence correspondant à la fonctionnalité “**deleteTable**” de Table :



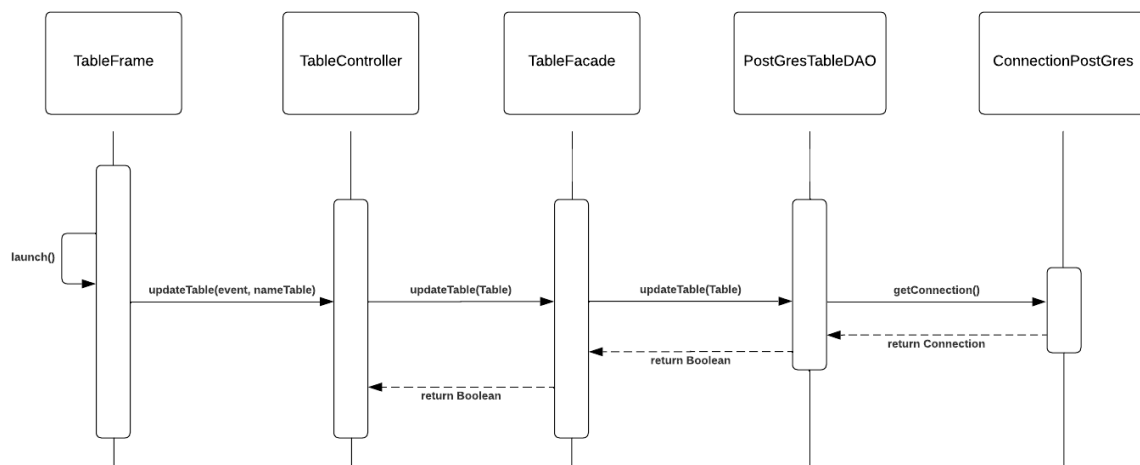
Le diagramme de séquence correspondant à la fonctionnalité “**getAllTable**” de Table :



Le diagramme de séquence correspondant à la fonctionnalité “**GetDetailsTable**” de Table :

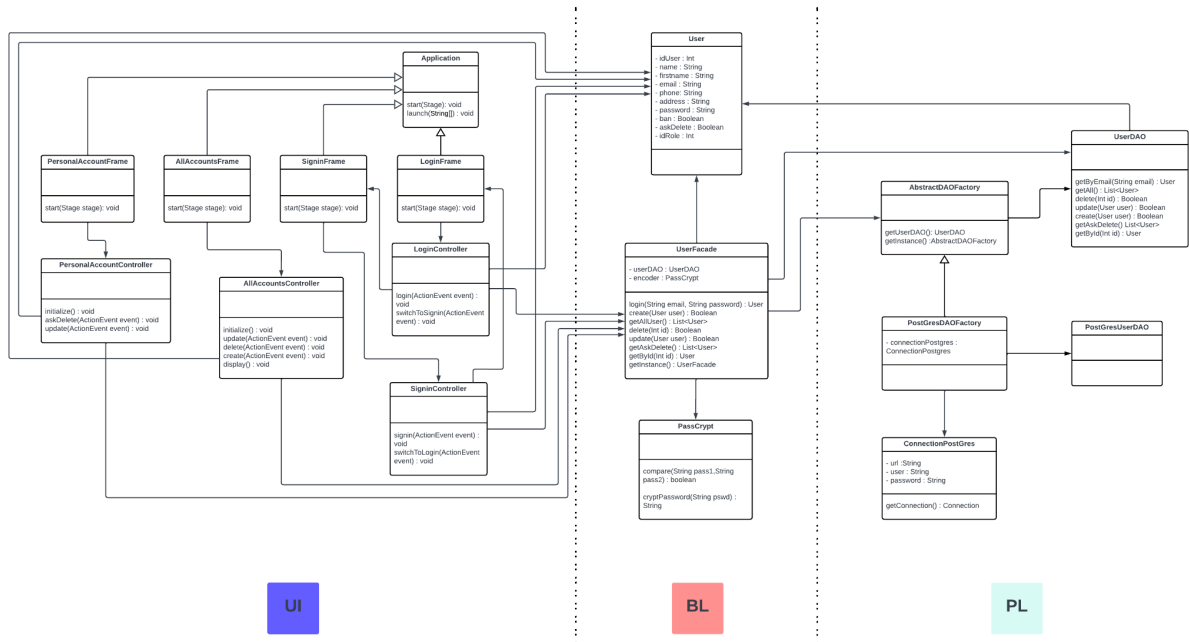


Le diagramme de séquence correspondant à la fonctionnalité “**updateTable**” de Table :

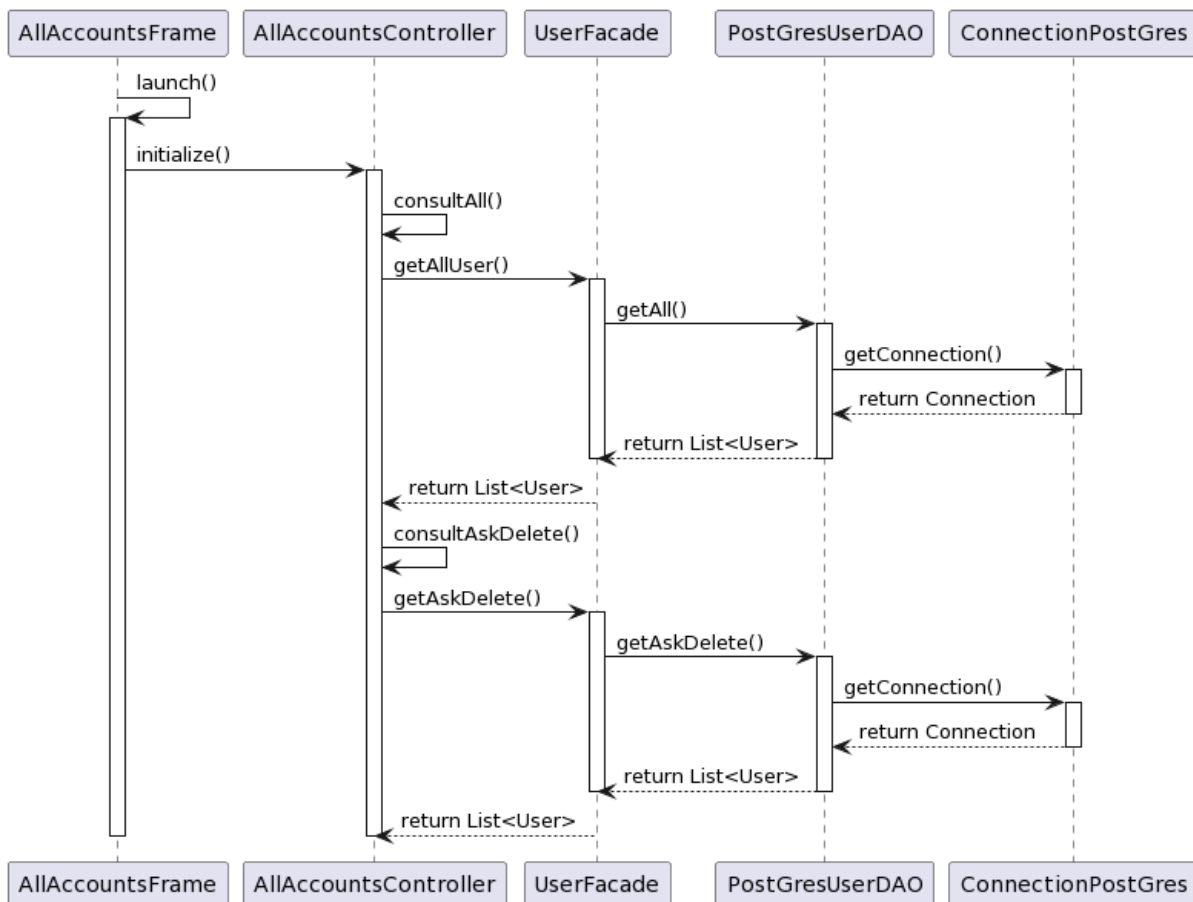


6.2.12 L'utilisateur

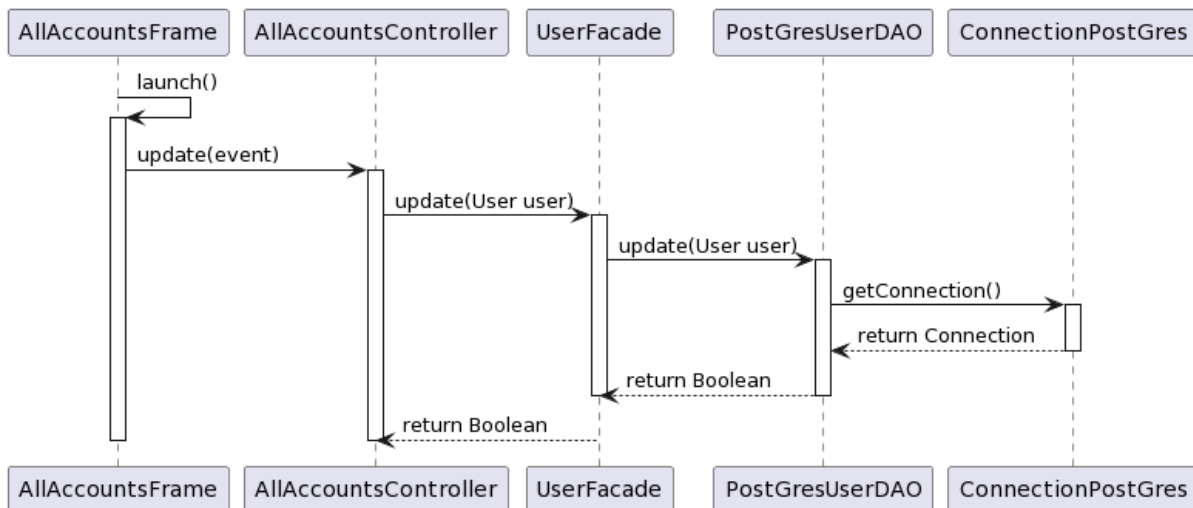
Le diagramme de classe pour **User** :



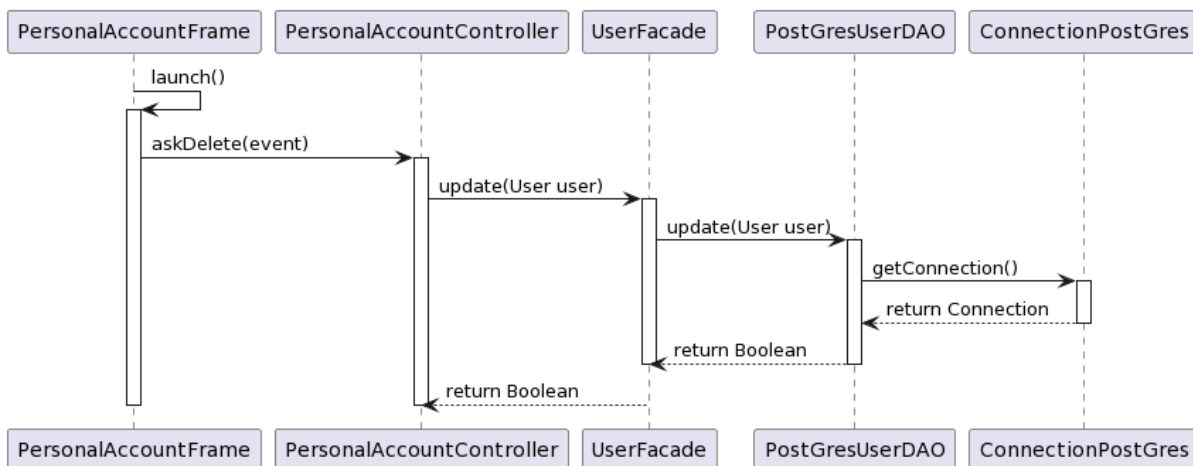
Le diagramme de séquence correspondant à la fonctionnalité “**allInitialize**” de User :



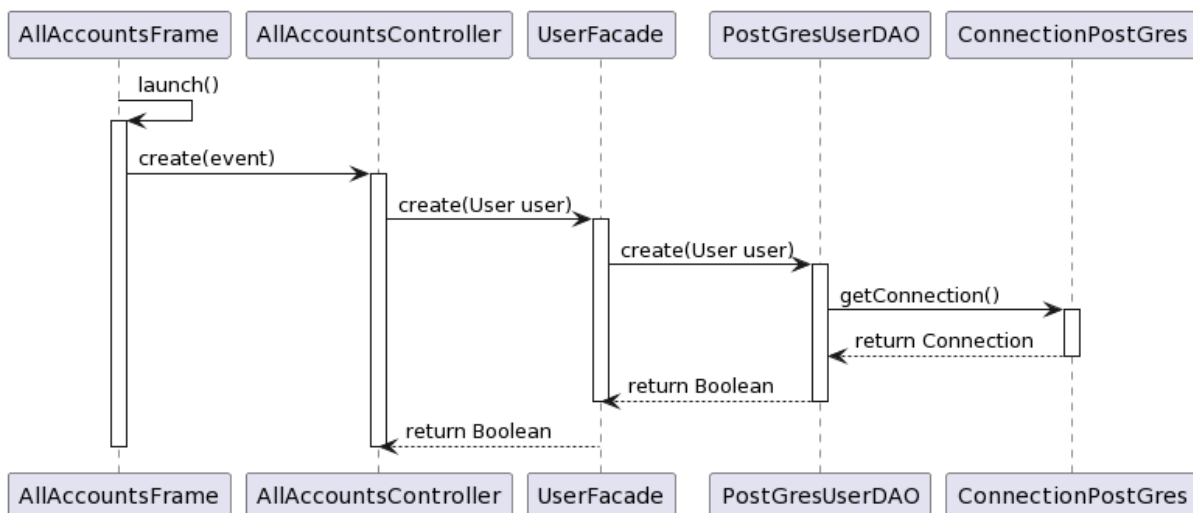
Le diagramme de séquence correspondant à la fonctionnalité “**all-update**” de User :



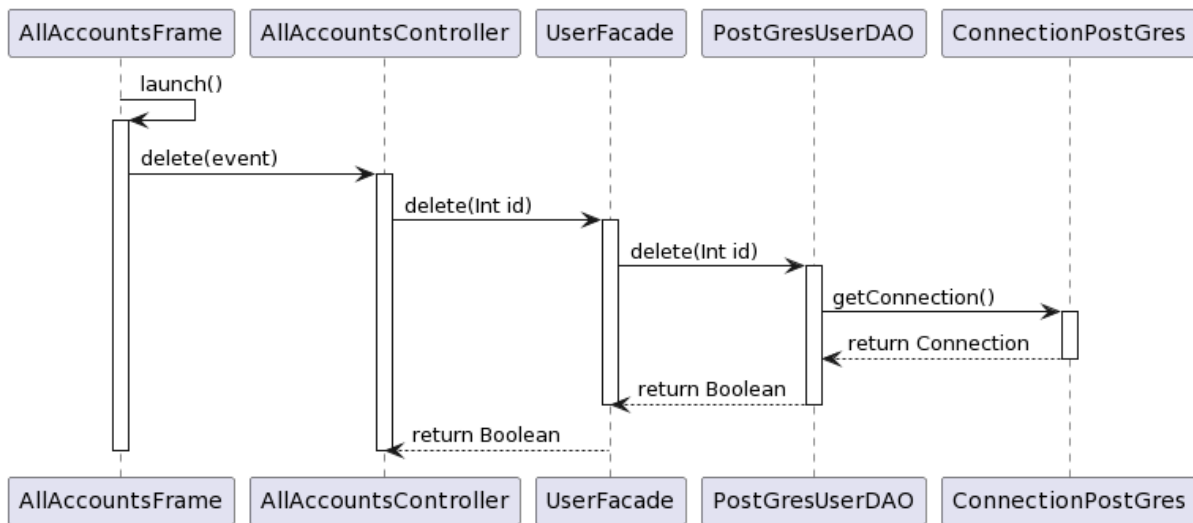
Le diagramme de séquence correspondant à la fonctionnalité “**askDelete**” de User :



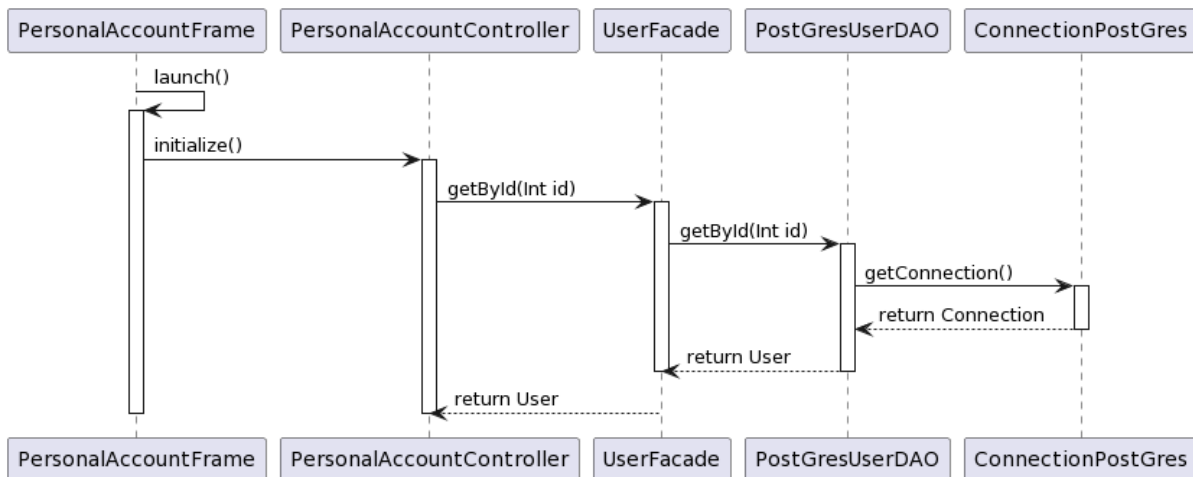
Le diagramme de séquence correspondant à la fonctionnalité “**create**” de User :



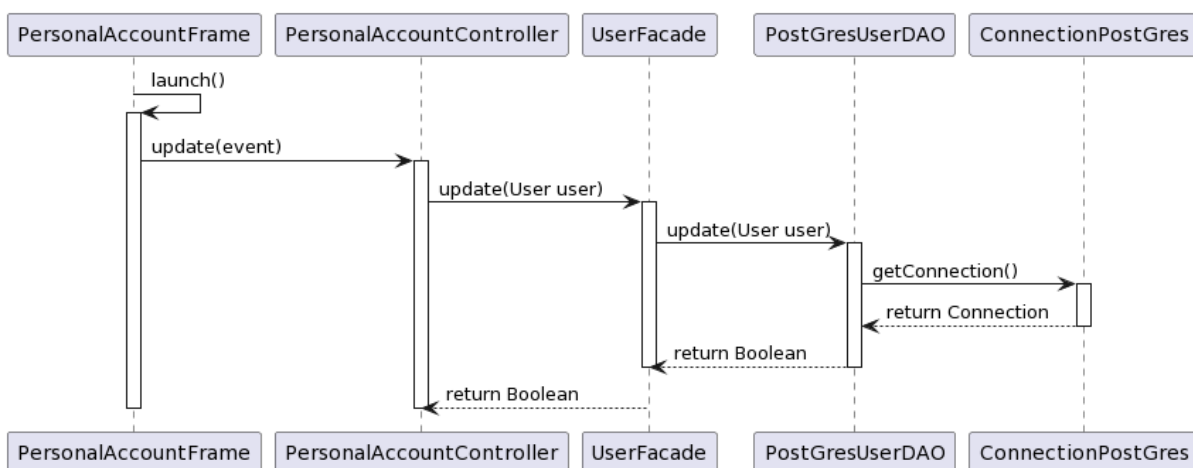
Le diagramme de séquence correspondant à la fonctionnalité “**delete**” de User :



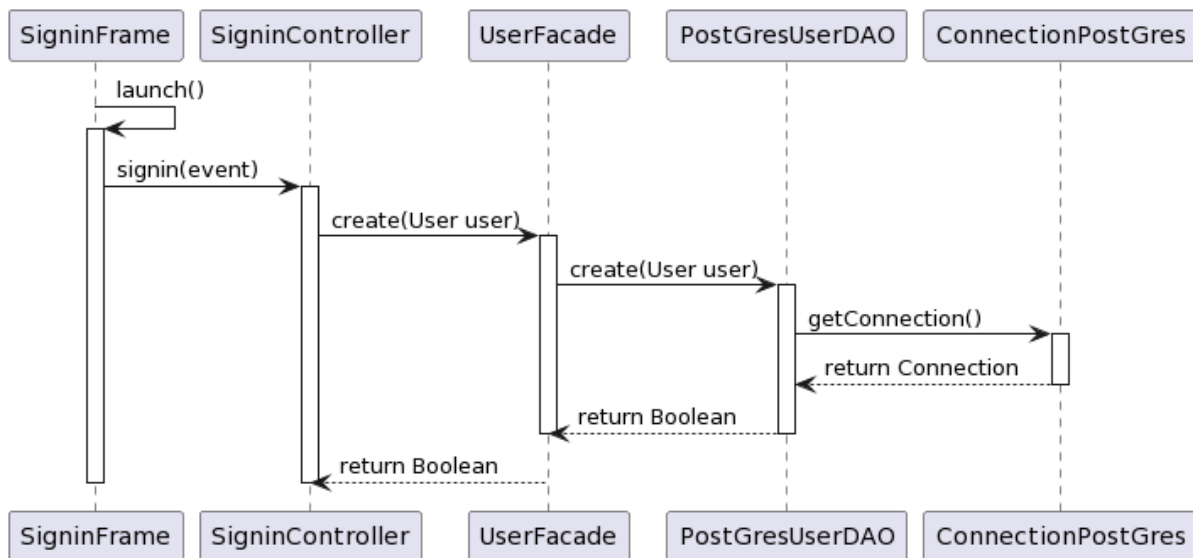
Le diagramme de séquence correspondant à la fonctionnalité “**personal-initialize**” de User :



Le diagramme de séquence correspondant à la fonctionnalité “**personal-update**” de User :



Le diagramme de séquence correspondant à la fonctionnalité “**signin**” de User :



Le diagramme de séquence correspondant à la fonctionnalité “**login**” de User :

