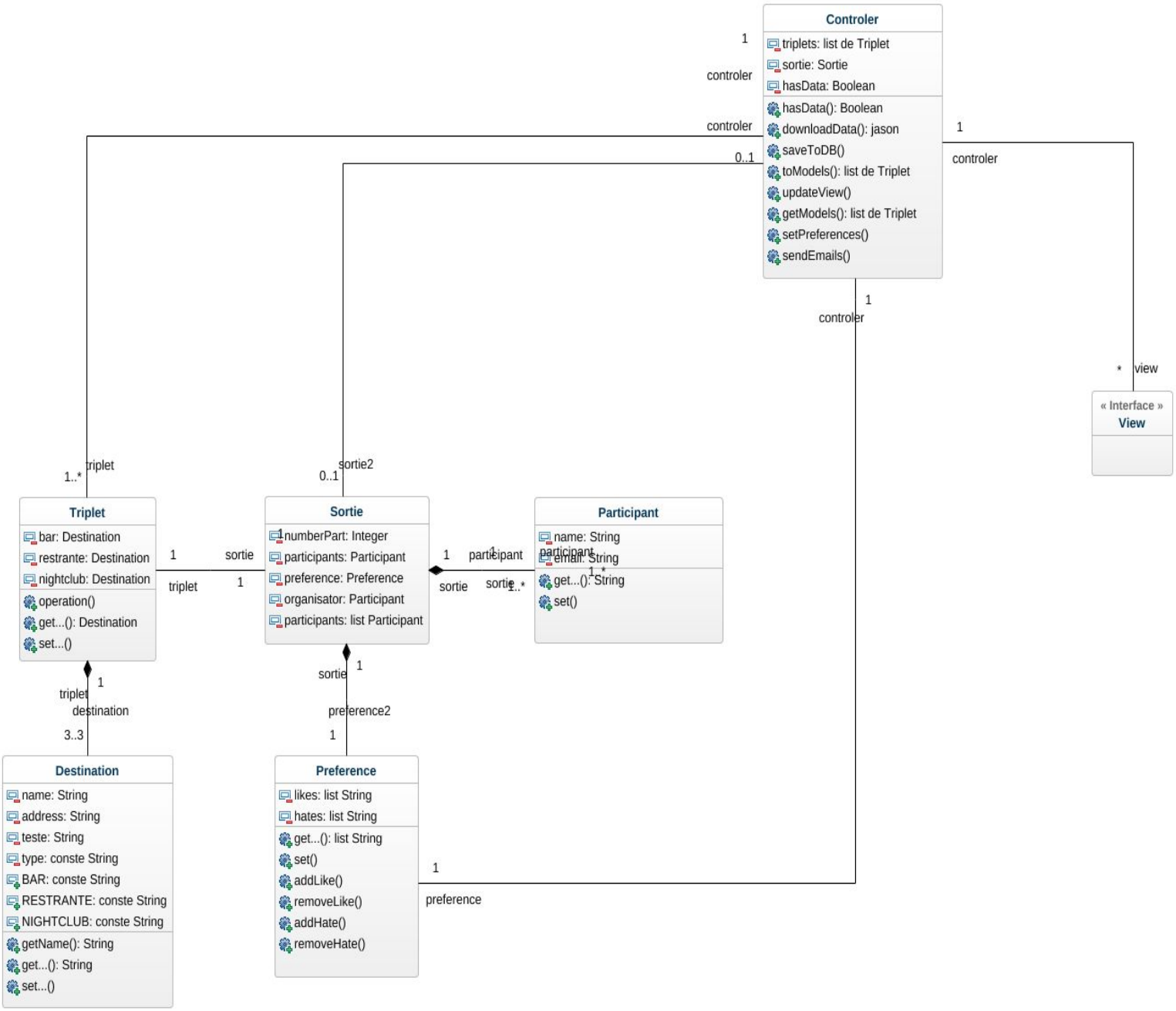
Sortir à Paris
Cahier des charges (version 3)
Conception

Diagramme de classe détaillé :

Pour la réalisation de notre projet nous avons opté pour modèle «MVC». Ce modèle de conception nous semblait approprié dans le cadre du développement d'une application web. L'utilisation du modèle MVC permet aux pages HTML, constituant la partie « vue » de contenir le moins possible de code serveur qui lui est géré par les deux autres parties de l'application.

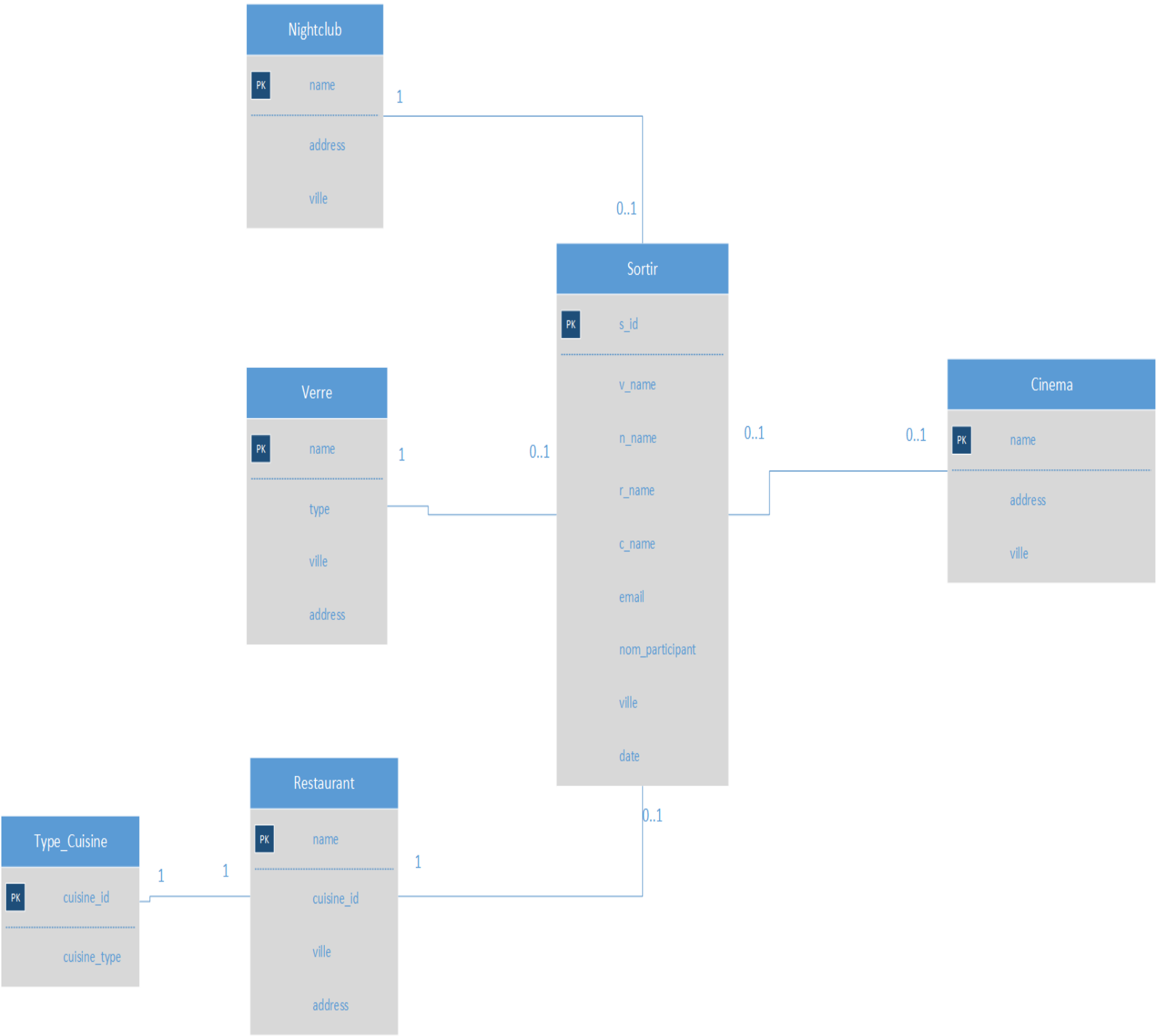

Controler
triplets: list de Triplet
sortie: Sortie
hasData: Boolean
hasData(): Boolean
downloadData(): jason
saveToDB()
toModels(): list de Triplet
updateView()
getModels(): list de Triplet
setPreferences()
sendEmails()
1
controler
controler
0..1
1
controler
1
controler
* view
« Interface »
View
1..*
triplet
0..1
sortie2
Triplet
bar: Destination
restrante: Destination
nightclub: Destination
operation()
get...(): Destination
set...()
1
sortie
1
triplet
Sortie
numberPart: Integer
participants: Participant
preference: Preference
organisator: Participant
participants: list Participant
1
participant
sortie
1..*
Participant
name: String
email: String
get...(): String
set()
1
triplet
destination
3..3
sortie
1
preference2
1
Destination
name: String
address: String
teste: String
type: conste String
BAR: conste String
RESTRANTE: conste String
NIGHTCLUB: conste String
getName(): String
get...(): String
set...()
Preference
likes: list String
hates: list String
get...(): list String
set()
addLike()
removeLike()
addHate()
removeHate()
1
preference

Description et explications :

Notre application est constitué d'un contrôleur principal qui va s'occuper de toutes les actions effectués par l'utilisateur, s'occuper de l'affichage des « vues » et communiquer avec l'api Google et la partie « données » de l'application. Les différentes « vues » de notre application gérées par des pages html sont représentées ici, pour simplifier, par une simple interface dont héritent les différentes vues. La partie « données » regroupe les différentes classes nécessaires aux stockages des diverses informations de la soirée. Nous avons fait également apparaître les fonctions/procédures et paramètres qui nous semblaient les plus importants pour plus de lisibilité.

Modèle de base de données :

l'application devra gérer une certaine quantité de donnée et de ce fait, il nous a semblé judicieux d'utiliser une base de données auxiliaire à notre application que nous allons administrer avec MySql. Mysql se combine très bien avec un autre langage de programmation (ici PHP). C'est quelque chose de très fréquent pour les sites web et nous l'avons pratiqué plusieurs fois en cours. C'est encore une fois le contrôleur qui va se charger d'interroger la base de données et de construire les modèles et les vues en fonction de celle-ci.


Nightclub
PK
name
address
ville
1
0..1
Sortir
PK
s_id
v_name
n_name
r_name
c_name
email
nom_participant
ville
date
Verre
PK
name
type
ville
address
1
0..1
0..1
0..1
Cinema
PK
name
address
ville
Restaurant
PK
name
cuisine_id
ville
address
0..1
1
Type_Cuisine
PK
cuisine_id
cuisine_type
1
1

Description et explications :

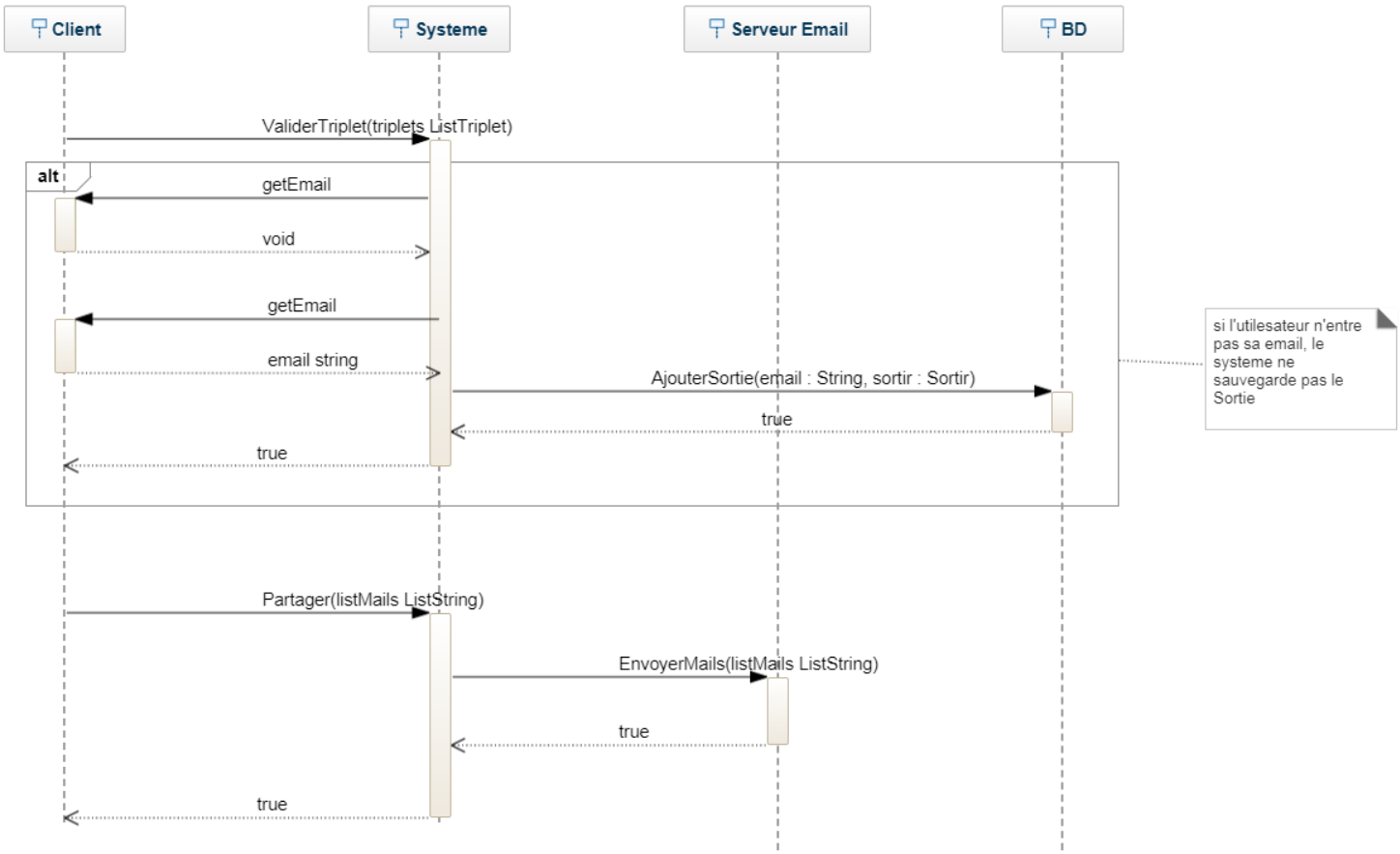
Nous retrouvons dans notre modèle, le schéma habituel, à savoir : une sortie est constitué d'un bar, d'une discothèque, d'un restaurant et possiblement, d'un cinéma. La gestion des donnés pour un restaurant est un peu plus complexe dans le sens où on l'on va stocker les goût spécifiés par l'utilisateur pour garder une trace des critères sur lesquels l'application a stocké les restaurants. Nous allons bien sûr garder en mémoire, les différents utilisateurs avec leur sortie affilié pour qu'ils puissent consulter facilement leurs sorties et les diverses informations après avoir fermé le site.

Au fur et à mesure ou des triplets seront créés, les restaurant/bar/nightclub vont s'accumuler dans la base de données. De ce fait, l'application va d'abord interroger la base de données avant de faire une demande sur l'api google au cas où l'information serait déjà stockée. Il sera également possible de garder les utilisateurs en mémoire (et peut être par la suite implémenter un système de comptes avancés). Ce modèle de base de donnée peut être provisoire et être amené à changer sur quelques détails par la suite.

Diagrammes de séquence :

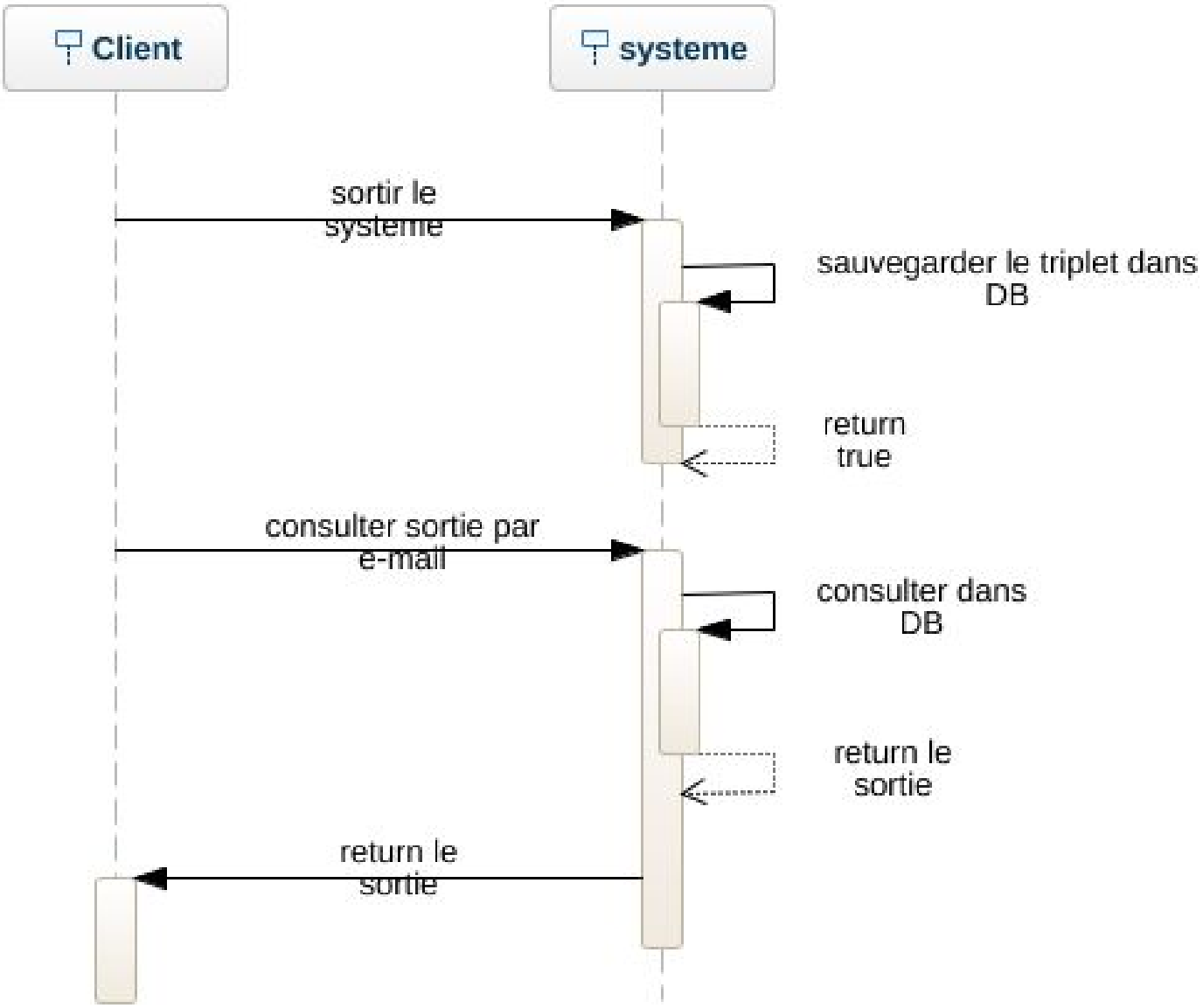
Supprimer les sortie passées :

Voici un diagramme de séquence présentant une action de maintenance pour un administrateur de l'application : supprimer les sorties ultérieures à la date du jour dans la base de données. Ce diagramme montre également qu'un administrateur peut directement récupérer l'id d'une sortie (ou d'autres informations) et supprimer directement des sorties avec les id récupérées. Quand une sortie est supprimée, le restaurant/bar/nightclub associé reste dans la base de données.



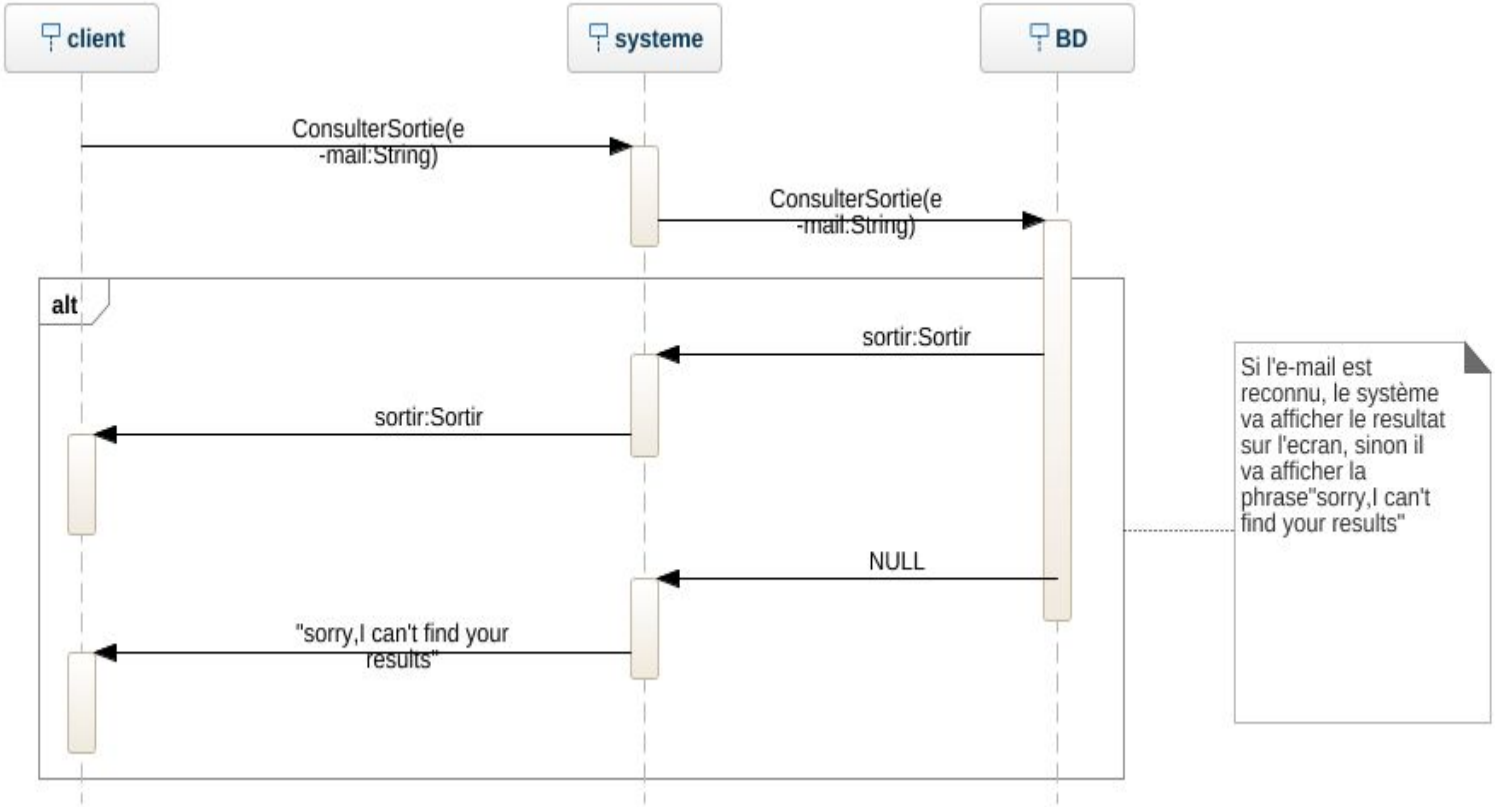
Déconnexion et consultation d'une sortie enregistrée :

Le diagramme suivant illustre les actions effectuées par le système quand un utilisateur quitte l'application ou souhaite re-consulter la sortie qu'il avait enregistré.



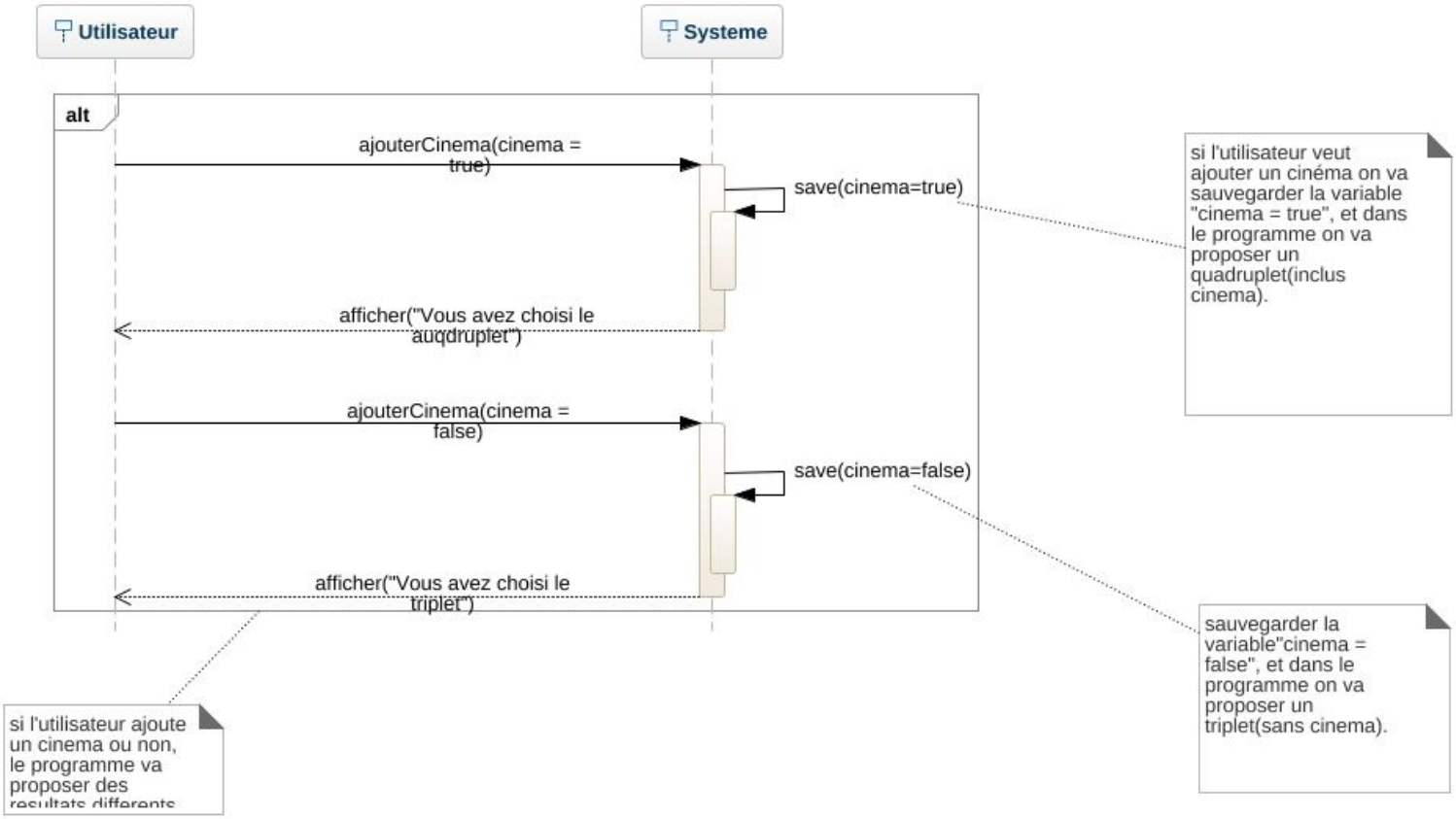
Consulter une sortie sauvegardée :

Voici un diagramme qui précise l'action de consulter une sortie sauvegardée grâce à l'email de l'utilisateur.



Choisir quadruplet :

Ce diagramme est une version modifiée du diagramme dans la partie analyse. L'utilisateur peut choisir de changer un triplet en quadruplet en ajoutant un cinéma à la sortie. On a donc deux cas possibles représentés ici.



Obtenir triplet :

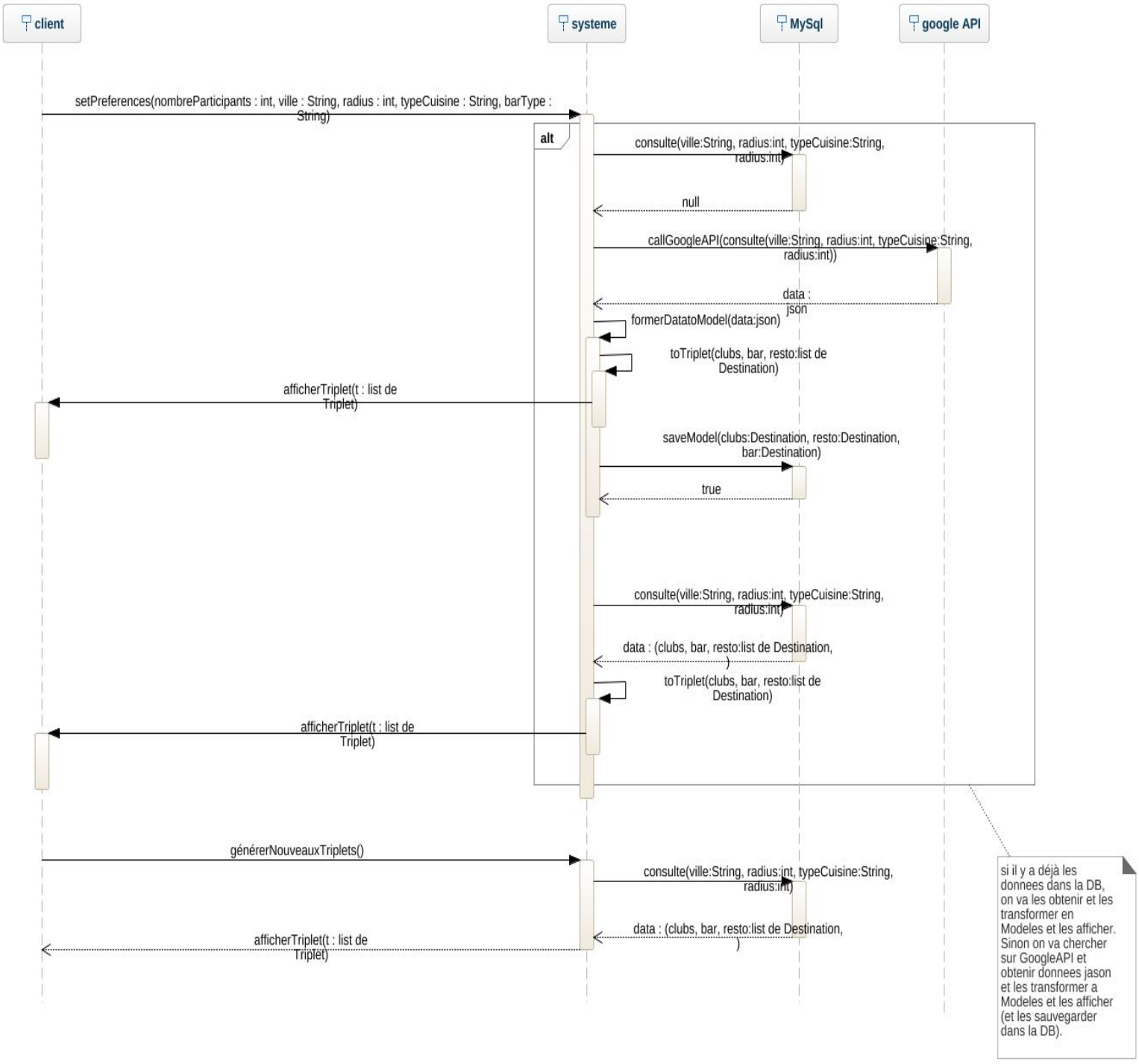
Ce diagramme reprend le digramme précédent de la partie analyse. Pour obtenir un triplet, l'utilisateur choisi ses préférences. A partir de là il y a deux cas possibles :

- Si les préférences sont déjà connues par la base de données, il est inutile d'aller interroger l'API Google. Les données nécessaires sont déjà stockées dans celle-ci et l'application va simplement utiliser sa base de donnée.

- Si les préférences sont inconnues, l'application va interroger l'api Google pour récupérer ce dont elle a besoin et le sauvegarder dans la base de données.

Dans les deux cas, à partir des données récupérées, l'application va générer les modèles utiles à l'élaboration des triplets à proposer.

L'application va proposer les triplets par cinq, et l'utilisateur peut demander la génération de cinq nouveaux triplets (obligatoirement avec une différence au moins) à tout moment.


client
systeme
MySql
google API
setPreferences(nombreParticipants : int, ville : String, radius : int, typeCuisine : String, barType : String)
alt
consulte(ville:String, radius:int, typeCuisine:String, radius:int)
null
callGoogleAPI(consulte(ville:String, radius:int, typeCuisine:String, radius:int))
data : json
formerDatatoModel(data:json)
toTriplet(clubs, bar, resto:list de Destination)
afficherTriplet(t : list de Triplet)
saveModel(clubs:Destination, resto:Destination, bar:Destination)
true
consulte(ville:String, radius:int, typeCuisine:String, radius:int)
data : (clubs, bar, resto:list de Destination,)
toTriplet(clubs, bar, resto:list de Destination)
afficherTriplet(t : list de Triplet)
générerNouveauxTriplets()
consulte(ville:String, radius:int, typeCuisine:String, radius:int)
data : (clubs, bar, resto:list de Destination,)
afficherTriplet(t : list de Triplet)
si il y a déjà les donnees dans la DB, on va les obtenir et les transformer en Modeles et les afficher. Sinon on va chercher sur GoogleAPI et obtenir donnees jason et les transformer a Modeles et les afficher (et les sauvegarder dans la DB).

Détailler triplet:

Encore une fois, nous avons repris le diagramme de la partie analyse. Quand l'utilisateur sélectionne un triplet, l'application doit afficher les détails du triplet avec notamment une carte de type google maps avec les destinations affichées et un trajet entre celles-ci. Quand une destination est sélectionnée, l'application affiche les détails de celle-ci avec son site internet si il existe. Nous n'avons pas encore décidé pour le moment comment générer le modèle et la vue à partir de l'API google. Nous mettrons ceic à jour par la suite.