Rapport Projet PolyEvent Team H

Florian Bourniquel
Alexandre Clément
Quentin Duret
Florian Lehmann

1.Contexte

Depuis ces dernières années, l'université de Sophia-Antipolis est confronté à un nombre grandissant de demandes de manifestation.

Dans le but de simplifier et d'améliorer l'efficacité du personnel pour l'organisation des manifestations, l'université nous a contacté pour réaliser un système informatique répondant à leurs besoins : Poly'Event.

Ce document a pour objectif de mettre en évidence l'architecture choisie pour Poly'Event. Nous allons dans un premier temps traiter le cas de la réservation d'une salle par un organisateur.

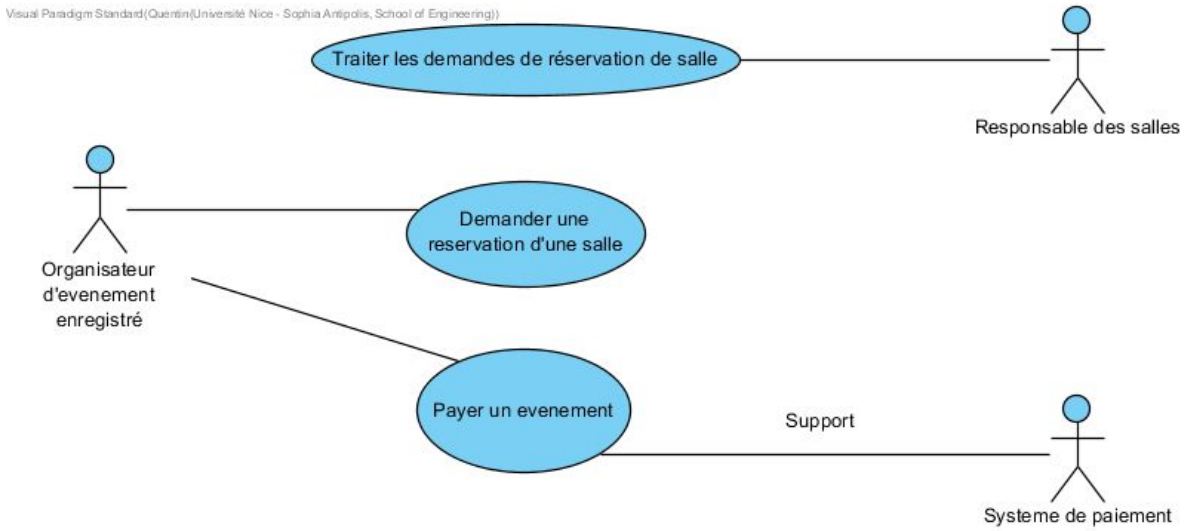
2.Vue fonctionnelle

2.1 Scénarios

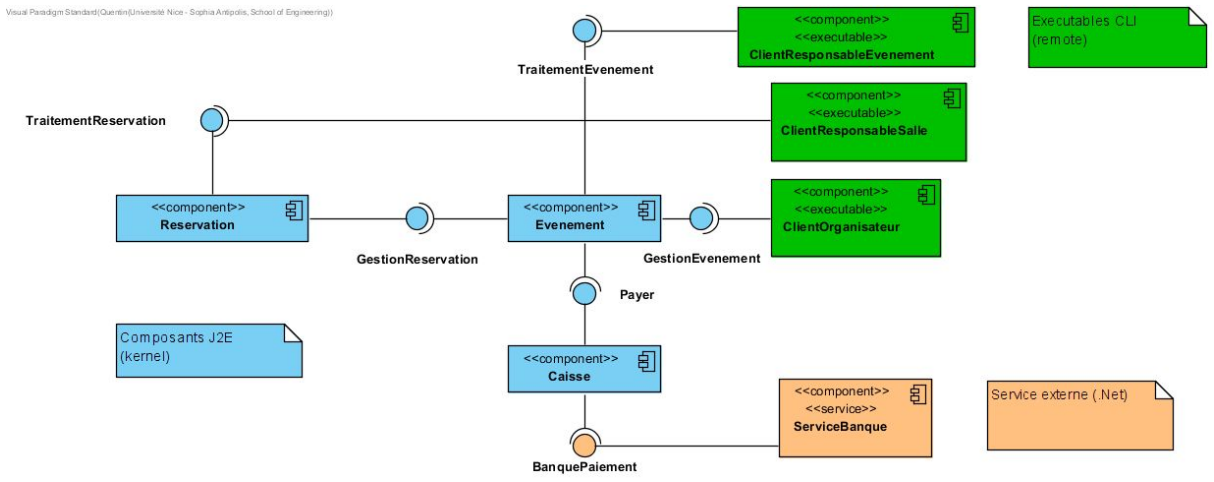
Scénario 1 : Réservation d'une salle par un organisateur

Cas d'utilisation : Reserver une salle **Acteur primaire** : Organisateur d'événement enregistré **Acteur support :** **Précondition** : L'organisateur est connecté à son compte et un événement a été créé Scénario primaire : 1. L'organisateur rentre les informations sur le types de salle qu'il souhaite réserver. (type de salle, taille, équipement nécessaire ...) 2. Le système d'information indique que la demande a été prise en compte. 3. Il reçoit une réponse positive pour sa demande. **Post condition :** L'opération s'est bien déroulée, une salle est réservée pour son événement/ L'opération ne s'est pas bien déroulée aucune salle n'a été réservée.	**Variantes :** 3a Il reçoit une réponse négative avec la raison de ce refus, ce qui met fin à l'opération.

2.2 Cas d'utilisations



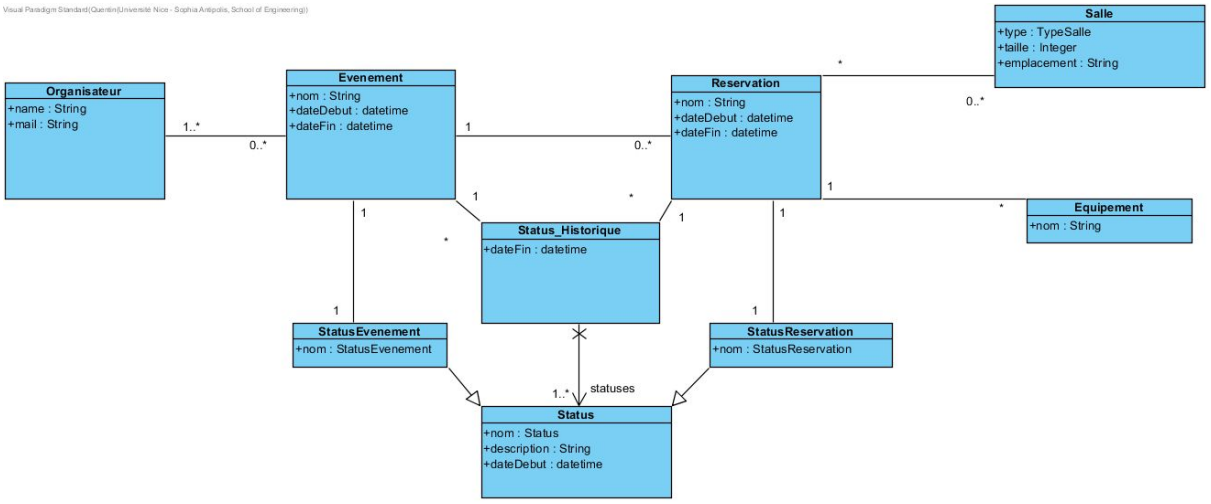
2.3 Diagramme de composants



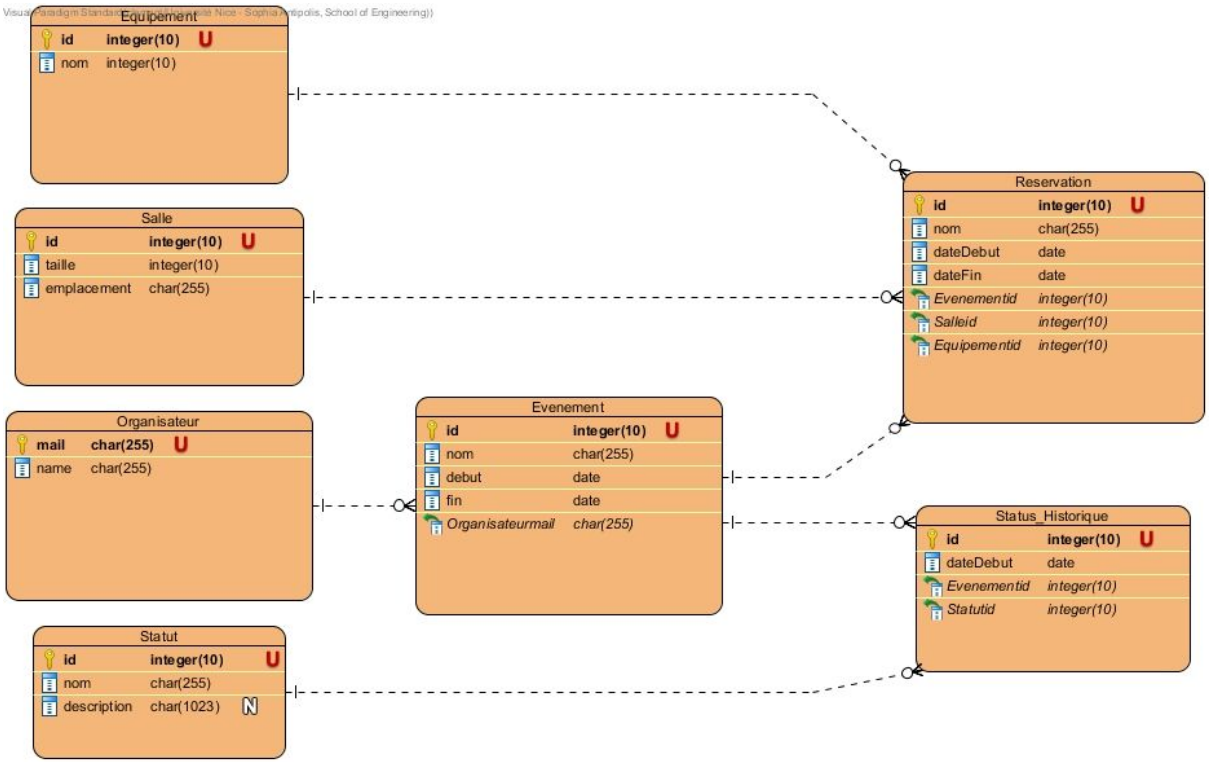
```
//Gestion de reservation
void demanderReservationSalle(Evenement,TypeSalle,Date,Date,Map<Equipement>)
void supprimerReservationSalle(Evenement,Reservation)
void modiferDateReservationSalle(Evenement,Reservation,Date,Date)
void modiferEquipementReservationSalle(Evenement,Reservation,Map<Equipement>)
void modiferTypeSalleReservationSalle(Evenement,Reservation,TypeSalle)

//Gestion d'evenement
void demanderCreationEvenement(Organisateur,String,Date,Date)
void supprimerEvenement(Evenement)
void modiferDateEvenement(Evenement,Date,Date)
void modiferNomEvenement(Evenement,String)

//Traitement reservation
void accepterReservation(Reservation)
void refuserReservation(Reservation,String)

//Traitement evenement
void accepterReservation(Evenement)
void refuserReservation(Evenement,String)

//Payer
String payerEvenement(Evenement)
```

3.Vue développement

3.1 Diagramme de classe



3.2 Modèle relationnel de stockage



Pour représenter les différents type de statut, nous avons décidé d'utiliser l'héritage sur une seule table. Cette solution a l'avantage de disposer de bonne performances par rapport aux autres solutions qui ont besoin d'effectuer de nombreux join et/ou union.
Par ailleurs, notre situation minimise le désavantage de celle-ci puisque nous n'aurons aucun champs null à cause du type dans notre table.

4.Vue déploiement