



Polytech Nice Sophia

ISA-Devops

Rapport d'architecture

réalisé par l'équipe

H



composée de

Dan NAKACHE, Sasha POURCELOT, Thomas DI GRANDE
& Vincent TUREL

Sommaire

Introduction	1
1 Cas d'utilisation	1
2 Diagramme de cas d'utilisation	2
3 Objets métiers	3
4 Interfaces	4
5 Composants	6
6 Scénarios MVP	8

Introduction

Le but de ce projet est de réaliser un logiciel pour gérer des événements sur un campus universitaire et mettre en relation les différents acteurs. L'application doit prendre en compte les multiples aspects liés à la préparation et à la mise en place de ces événements tels que la logistique, la facturation, la communication... De plus, le logiciel doit pouvoir être adapté à n'importe quel campus et non un campus spécifique tout en proposant les mêmes services.

Pour ce faire, nous avons posé les hypothèses suivantes :

- Chaque acteur a accès à une interface qui lui est bien spécifique,
- Il peut y avoir des modifications jusqu'au jour J,
- Les pancartes et moyens de communication sont gérés en interne,
- Laura gère les événements quand elle est contactée par un acteur externe,
- La préparation de l'événement commence seulement une fois que l'événement est validé par le commanditaire (facture envoyée).

1 Cas d'utilisation

On exploite les personnes suivantes :

- Jacques, le responsable de la logistique (s'occupe du bon fonctionnement du matériel ainsi que la mise en place de la signalisation),
- Pierre, l'organisateur d'événement entreprises-étudiants (va organiser des événements entre entreprises et étudiants de façon intéressante et amusante),
- Amaury, l'organisateur d'événement entreprises (va organiser des événement pour son entreprise tout en recrutant des prestataires pour certains services),
- Laura, la responsable de la communication (s'occupe de la diffusion de l'événement ainsi que de la signalisation sur le campus),
- Cécile, l'organisatrice des salles (s'occupe de la répartition des salles avec un emploi du temps pour chacune d'entre elle),
- Alison, la comptable (s'occupe du paiement et de la facturation),
- Agnès, la responsable du nettoyage (va prévoir les plannings de nettoyage sur le campus).

Les couples personna-scénario suivants sont des acteurs primaires :

- Cécile pour le renseignement et la validation des salles,
- Jacques pour la réservation des places de parking et la gestion du matériel,
- Pierre et Laura pour la création d'un événement,
- Alison pour la gestion du service comptable (facture, devis, ordre d'achat),
- Laura pour la diffusion de l'événement,
- Agnès pour la consultation du planning de ménage et la validation du nettoyage,
- Tout le monde pour la consultation du planning des salles,
- Tout le monde pour la création d'un ticket de support technique.

Les couples personna-scénario suivants sont des acteurs secondaires :

- Système de notification (avec Jacques en destinataire) pour la gestion des tickets de support technique,
- Système de notification (avec l'organisateur en destinataire) pour la transmission des factures et des devis,
- Système de notification (avec tous les intéressés) pour la diffusion de l'événement.

2 Diagramme de cas d'utilisation

On distingue les cas d'utilisation suivants :

- Cécile renseigne les salles disponibles,
- Cécile valide la distribution des salles,
- Tout le monde consulte le planning des salles,
- Tout le monde renseigne le matériel défectueux,
- Alison effectue la facturation et les devis et s'occupe des ordres d'achat,
- Laura et Pierre créent des événements (liste du matériel par salle, nombre de personnes...),
- Jacques réserve les places de parking,
- Jacques gère la liste du matériel
- Laura diffuse l'événement,
- Agnès consulte le planning de ménage,
- Agnès indique qu'une salle est propre.

Nous souhaitons que chaque acteur ait sa propre interface lui permettant de jouer son rôle. Cela permet d'éviter de noyer l'utilisateur sous une myriade d'éléments qui ne relève pas de sa responsabilité ni de son domaine d'expertise. Par exemple, Cécile, chargée de la gestion des salles, n'aura pas les mêmes fonctionnalités que Jacques qui s'occupe de la logistique. Certains éléments tels que le planning sont en revanche accessibles à tous.

La figure 1 montre le diagramme d'usage du PolyEvent.

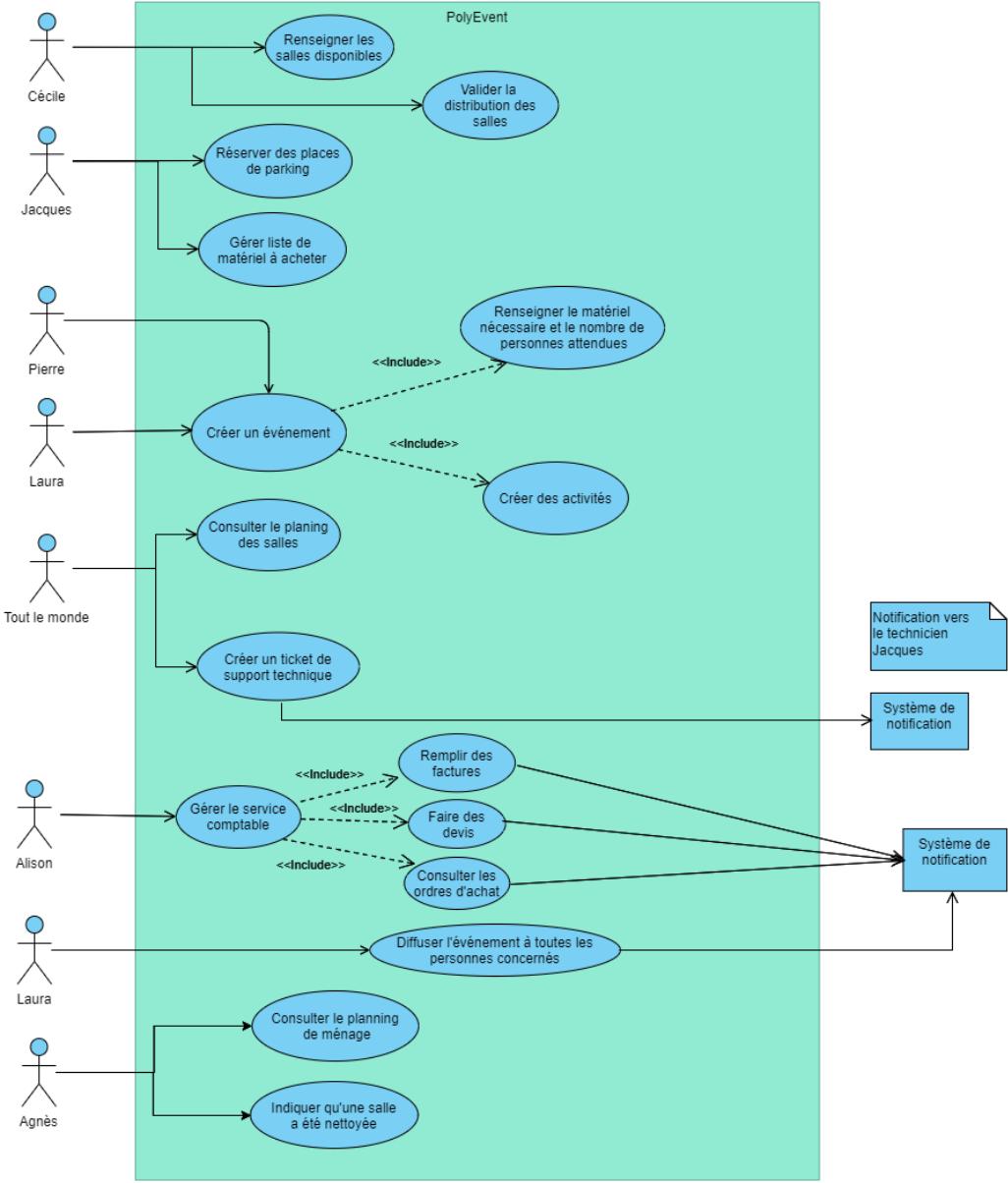


Figure 1: Use-case diagram de l'application PolyEvent

3 Objets métiers

Nous avons décidé d'utiliser le terme *event* pour parler des événements qu'on organise. La classe *Event* va donc représenter l'ensemble des informations relatives à l'événement. Un événement est composé de plusieurs activités représentées par la classe *Activity*. Les activités sont des regroupements de personnes dans une salle bien précise pendant une partie de l'événement. Les activités peuvent accueillir un nombre maximum de personnes qui doit être inférieur à la capacité des salles dans lesquelles elles se déroulent. Certaines activités requièrent du matériel particulier, représenté par la classe *Equipment*. On prend en compte le matériel déjà disponible car acquis par l'université ainsi que le matériel immobile, déjà présent dans les salles. Les tâches sont distribuées à chaque personne en fonction de son rôle, représentés respectivement par la classe *Task* et l'enum *Role*.

La figure 2 montre le diagramme de classes détaillé du projet.

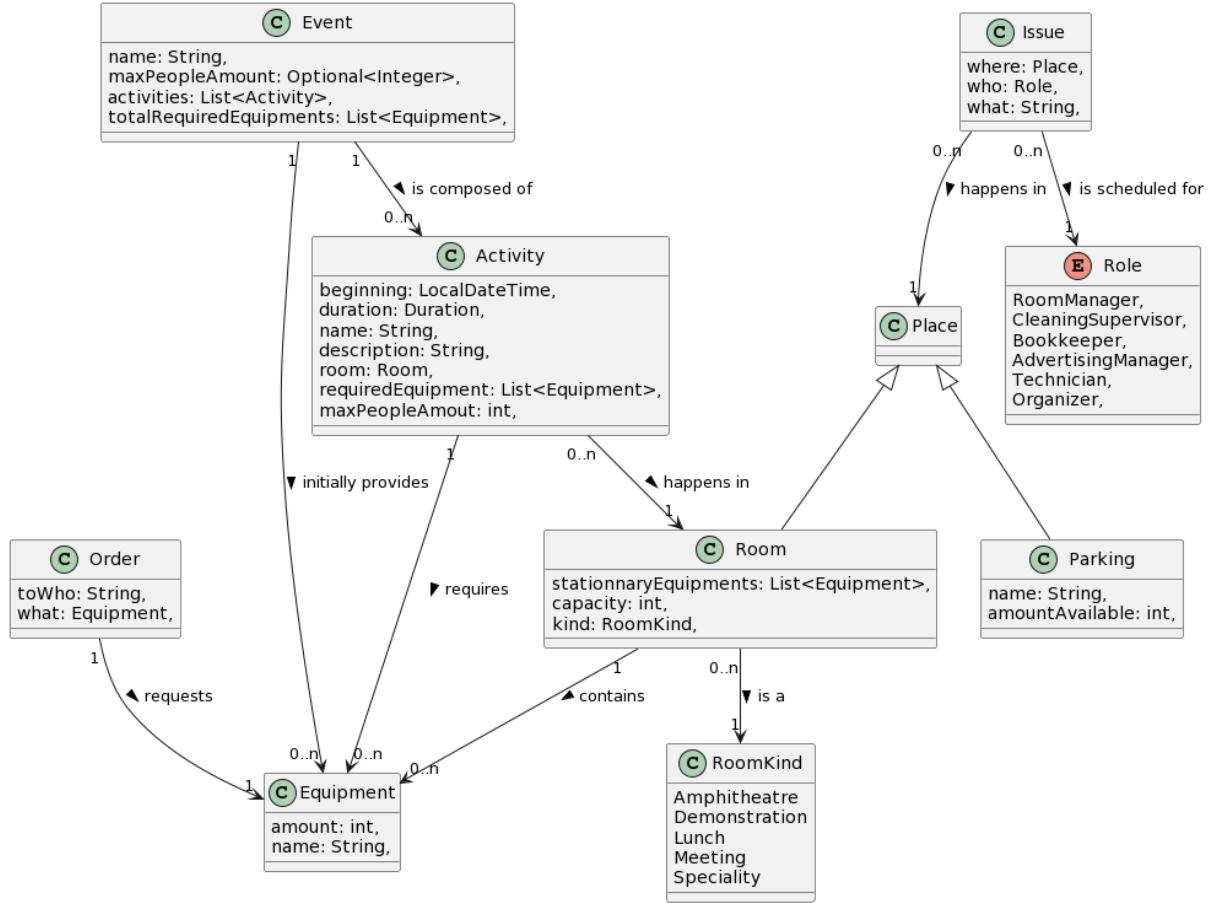


Figure 2: Diagramme de classes des objets métiers du PolyEvent

4 Interfaces

- Opérations pour modifier une activité (créer, supprimer, mettre à jour...),

```

public interface ActivityModifier {
    void create(LocalDateTime beginning, Duration duration, String name, String description, Room room,
               List<Equipment> requiredEquipment, int maxPeopleAmount);

    boolean modify(Activity activityToModify, Activity activityModified) throws ItemNotFoundException;

    boolean delete(Activity activityToDelete) throws ItemNotFoundException;
}

```

- Vérifie si l'école possède déjà le matériel nécessaire pour l'événement. Renvoie la liste des équipements à acheter,

```

public interface CheckMaterial {
    List<Equipment> check(List<Equipment> equipmentToCheck);
}

```

- Permet d'accéder au planning des salles à nettoyer,

```

public interface CleaningPlanningAccess {
    Planning<Room, LocalDateTime, Duration> getCleaningPlanning();
}

```

- Opérations pour modifier un événement (créer, supprimer, mettre à jour...),

```

public interface EventModifier {
    void create(String name, Room room, int maxPeopleAmount, List<Activity> activities, List<Equipment> requiredEquipment);

    boolean modify(Event eventToModify, Event eventModified) throws ItemNotFoundException;

    boolean delete(Event eventToDelete) throws ItemNotFoundException;
}

```

- Opérations pour envoyer une facture par mail,

```

public interface InvoiceQuoteTransmission {
    void sendNotification(Invoice invoiceToNotify);
}

```

- Opérations pour modifier la liste d'équipements (ajouter, retirer, mettre à jour...),

```

public interface MaterialModifier {
    boolean modify(Equipment equipmentToModify, Equipment equipmentModified) throws ItemNotFoundException;

    void add(Equipment equipmentToAdd);

    void delete(Equipment equipmentToRemove) throws ItemNotFoundException;
}

```

- Vérifie si un groupe de salles a besoin d'être nettoyé. Retire de la liste les salles déjà propres,

```

public interface NeedCleaningCheck {
    List<Room> check(List<Room> roomWhichMightNeedCleaning);
}

```

- Permet de réserver une place de parking ou de libérer une place déjà réservée,

```

public interface ParkingBooking {
    void bookSpace(Parking parking) throws RoomAlreadyBookedException;

    void freeSpace(Parking parking);
}

```

- Opérations pour modifier un parking (créer, supprimer, mettre à jour...),

```

public interface ParkingModifier {
    void create(String name, int amountAvailable);

    boolean modify(Parking parkingToModify, Parking parkingModified) throws ItemNotFoundException;

    boolean delete(Parking parkingToDelete) throws ItemNotFoundException;
}

```

- Génère un devis à destination de l'organisateur de l'événement pour le matériel que l'école ne possède pas déjà,

```

public interface ProcessOrder {
    void generateQuote(List<Equipment> equipmentsToBuy);
}

```

- Génère un devis à destination de l'organisateur de l'événement pour la location des salles de l'école,

```

public interface ProcessQuotes {
    void generateQuote(List<Room> roomToRent);
}

```

- Permet de réserver une salle ou de libérer une salle déjà réservée,

```

public interface RoomBooking {
    void bookRoom(Room room) throws RoomAlreadyBookedException;

    void freeRoom(Room roomToFree);
}

```

- Permet d'accéder au planning des salles,

```

public interface RoomExplorer {
    Planning<Room, LocalDateTime, Duration> exploreRooms();
}

```

- Opérations pour modifier la liste de pièces (ajouter, retirer, mettre à jour...),

```

public interface RoomModifier {
    void create(List<Equipment> equipments, int capacity, RoomKind kind);

    boolean modify(Room roomToModify, Room roomModified) throws ItemNotFoundException;

    boolean delete(Room roomToDelete) throws ItemNotFoundException;
}

```

- Permet d'accéder à la liste des tâches à faire,

```

public interface TaskExplorer {
    Set<Issue> exploreTasks();
}

```

- Opérations pour modifier la liste des tâches (ajouter, retirer, mettre à jour...),

```

public interface TaskModifier {
    void create(Place where, Role who, String what);

    boolean modify(Issue issueToModify, Issue issueModified) throws ItemNotFoundException;

    boolean delete(Issue issueToDelete) throws ItemNotFoundException;
}

```

- Opérations pour notifier qu'une tâche a été ajoutée,

```

public interface TaskTransmission {
    void sendNotification(Issue issueToNotify);
}

```

5 Composants

Nous avons identifié les composants suivants :

- EventManager : À la création d'un événement, toutes les informations vont être séparés en 2 à l'intérieur de ce composant pour être rediriger vers d'autres composants (tout ce qui est relatif à une activité, et tout ce qui a trait à l'organisation),
- Organisation : Dans ce composant, on va traiter les informations qui ne sont pas liées à la gestion des salles telles que la gestion du stock de nourriture ou des places de parking et les rediriger vers les composants qui s'occupent du traitement de ces informations,
- ActivityManager : À la création d'une activité, il faut trouver les salles correspondant au besoin ainsi que mettre à jour le matériel nécessaire,
- TaskList : Ce composant contient les différentes tâches à effectuer par les différents acteurs,
- ParkingManager : Ce composant est là pour gérer tout ce qui est lié au parking comme le nombre de places disponibles ainsi que la signalétique,
- MateriaInventory : Dans ce composant on va regarder le matériel nécessaire à l'événement et le comparer avec le matériel déjà présent pour pouvoir ensuite transmettre ce qu'il manque,
- RoomPlanning : Ce composant contient le planning des salles pour que chacun puisse le consulter et obtenir les informations liées aux salles telles que la répartition des salles pour un événement,
- RoomManager : Dans ce composant, on va compiler la liste des salles disponible avec le besoin de l'activité pour trouver les salles qui correspondent le mieux,
- Invoices And Quotes : Toutes les informations de l'événement arrivent dans ce composant pour être compilées et pouvoir générer facture ou devis.

La figure 3 montre le diagramme de composants du PolyEvent.

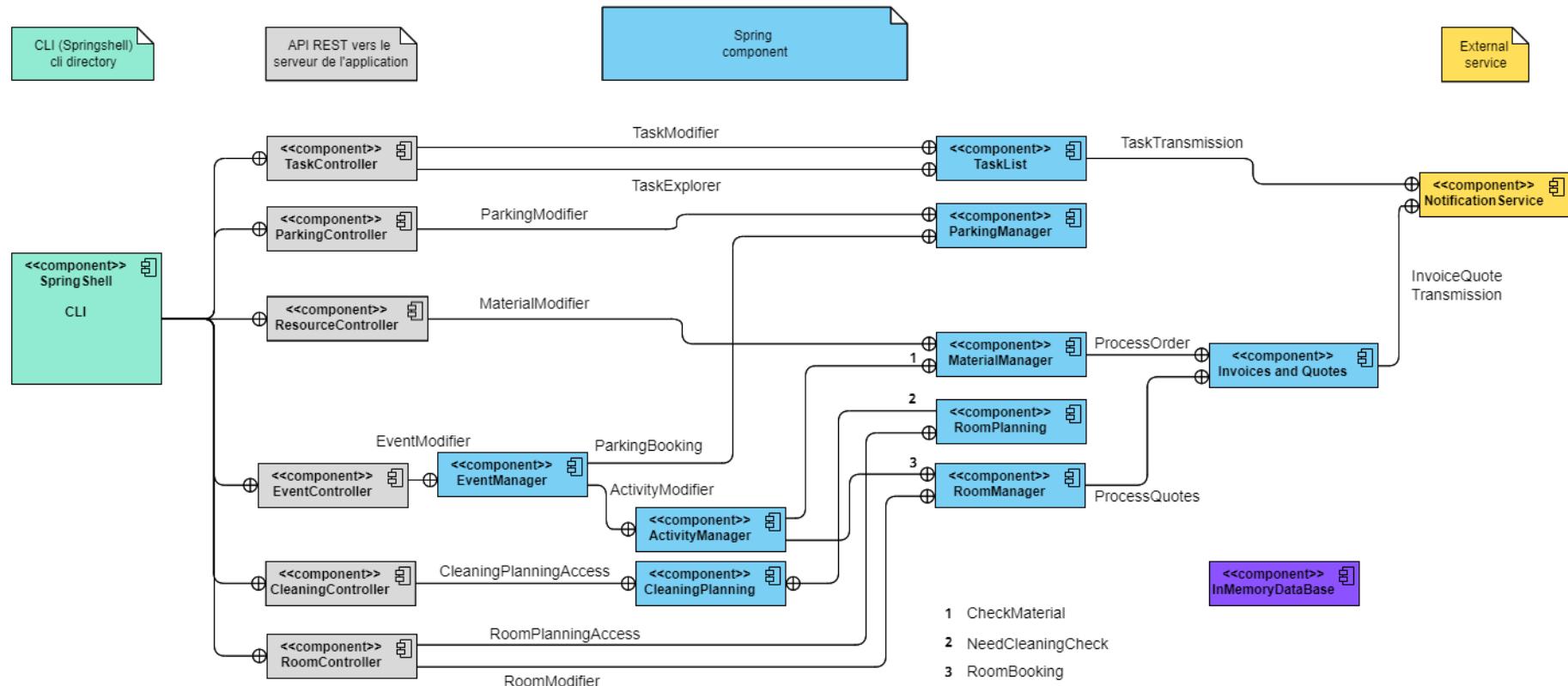


Figure 3: Diagramme de composants du PolyEvent

6 Scénarios MVP

Nous avons identifié les scénarios suivants :

- Créer un nouveau ticket d'assistance :

CLI → TaskController → TaskList → NotificationService,

- Voir les tickets ouverts :

CLI → TaskController → TaskList,

- Réserver un ou plusieurs emplacement(s) de parking supplémentaire pour l'événement :

CLI → ParkingController → ParkingManager

- Réserver du matériel supplémentaire pour l'événement :

CLI → ResourceController → MaterialManager

- Acheter du matériel supplémentaire non disponible :

CLI → ResourceController → MaterialManager → InvoicesAndQuotes → NotificationService

- Créer un événement complet sans parking ni matériel supplémentaire :

CLI → EventController → EventManager → ActivityManager → RoomManager → Invoices and Quotes → NotificationService

- Créer un événement complet avec parking :

CLI → EventController → EventManager :

 → ParkingManager.

 → ActivityManager → RoomManager → InvoicesAndQuotes → NotificationService.

- Créer un événement complet avec parking et matériel supplémentaire :

CLI → EventController :

 → ParkingManager.

 → ActivityManager :

 → RoomManager → InvoicesAndQuotes → NotificationService

 → CheckMaterials → MaterialManager → InvoiceAndQuotes → NotificationService

- Nettoyer les salles après l'événement :

CLI → CleaningControl → RoomPlanning (voir si une salle est libre mais a été utilisée par l'événement)

- Voir les salles disponibles :

CLI → RoomController → RoomPlanning

- Réserver une salle supplémentaire :

CLI → RoomController : →

 → RoomPlanningAccess → RoomPlanning

 → RoomManager → ProcessQuotes (tarif supplémentaire dû au rajout de la salle) → InvoicesAndQuotes

 → NotificationService