Maxime Flament
Théo Frasquet
Lucas Matteo
Jehan Milleret

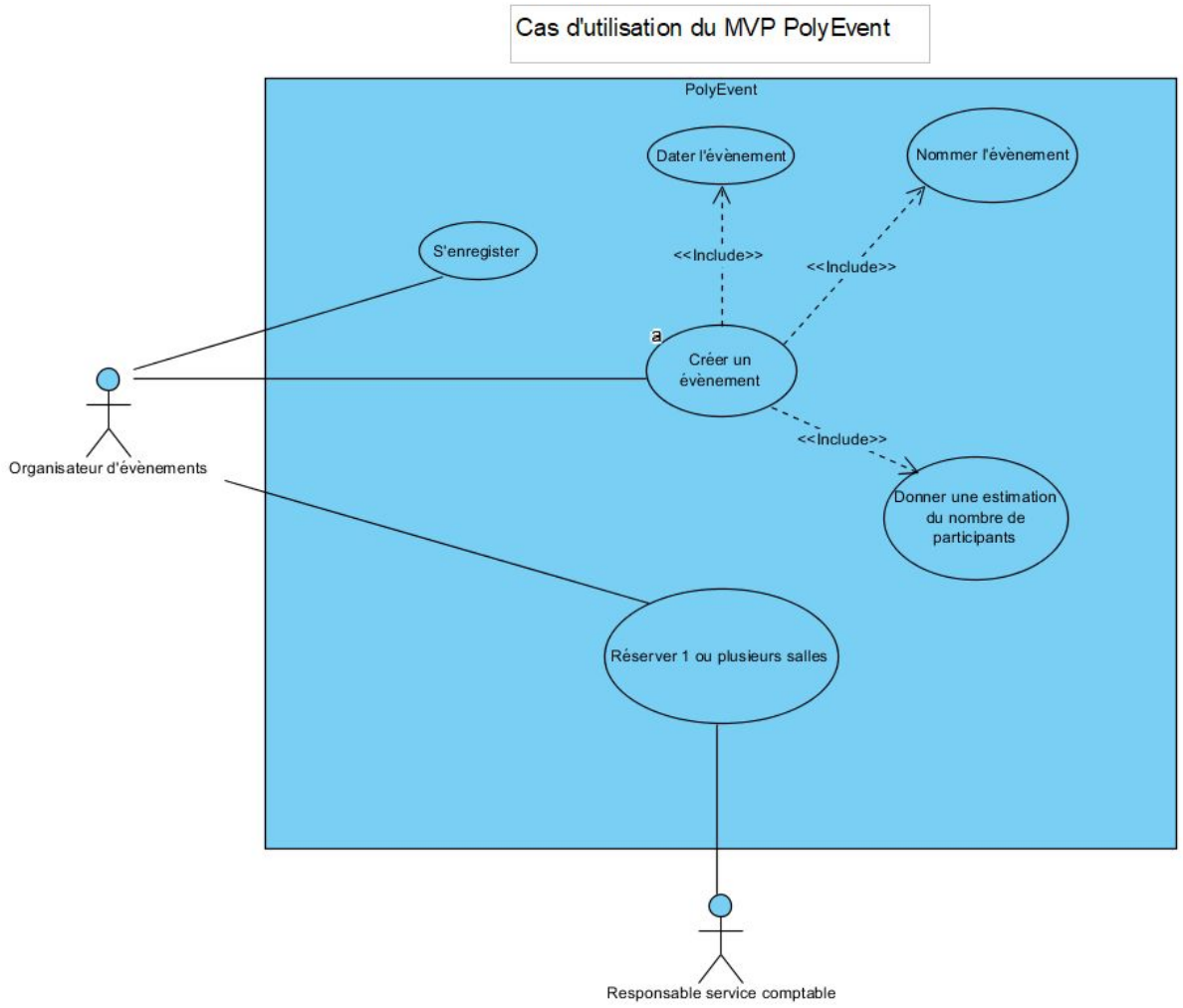
MVP PolyEvent

Equipe C

Contexte

Polytech a besoin d'une nouvelle structure afin de gérer au mieux les événements qui sont organisés en son sein. Nous proposons ici une première version de l'architecture du projet PolyEvent.

Cas d'utilisation de haut niveau



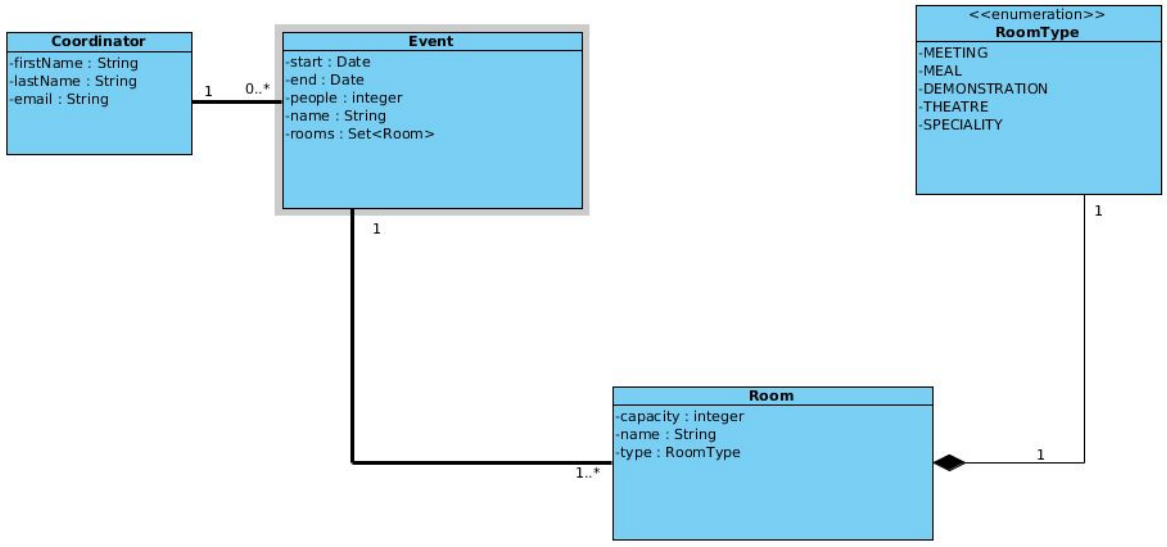
Nous avons choisi de nous concentrer sur la création simple d'un événement, pour que le processus de gestion d'événement puisse être réalisé du début à la fin, bien que minimal. Ainsi, l'organisateur peut s'enregistrer avec ses informations personnelles pour ensuite pouvoir créer un événement en spécifiant un nom, une date, le nombre de

participant. Si il a des salles particulières qu'il a l'habitude de réserver il peut les choisir en particulier, sinon les salles sont réservées automatiquement en fonction de leur capacités.

Une fois l'événement créé, le responsable du service comptable a accès aux salles réservées et peut réaliser son travail grâce à cette informations.

Les autres acteurs n'ont pour le moment pas d'accès à l'application, on suppose que leur implication se réalise encore grâce à l'organisateur qui va les contacter directement.

Diagramme de Classes Métier

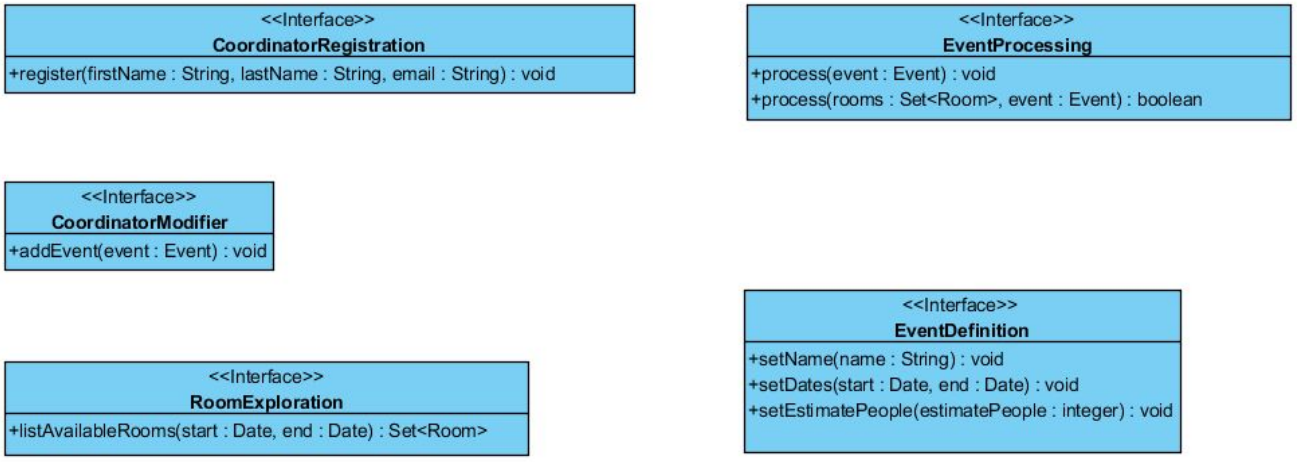


Le diagramme de classes précédent nous permet de réaliser les fonctionnalités décrites dans le Cas d'Utilisation de haut niveau.

Nous avons choisi de faire une Composition entre *Room* et *RoomType*. En effet, nous avons pensé logique qu'un type de salle ne puisse exister qu'avec une salle.

De plus, nous avons représenté l'Acteur *Organisateur d'événements* avec la classe *Coordinator*. En effet, nous avons besoin de ses informations afin de l'associer à une liste d'événements, pour pouvoir réaliser le Cas d'Utilisation *Créer un évènement*.

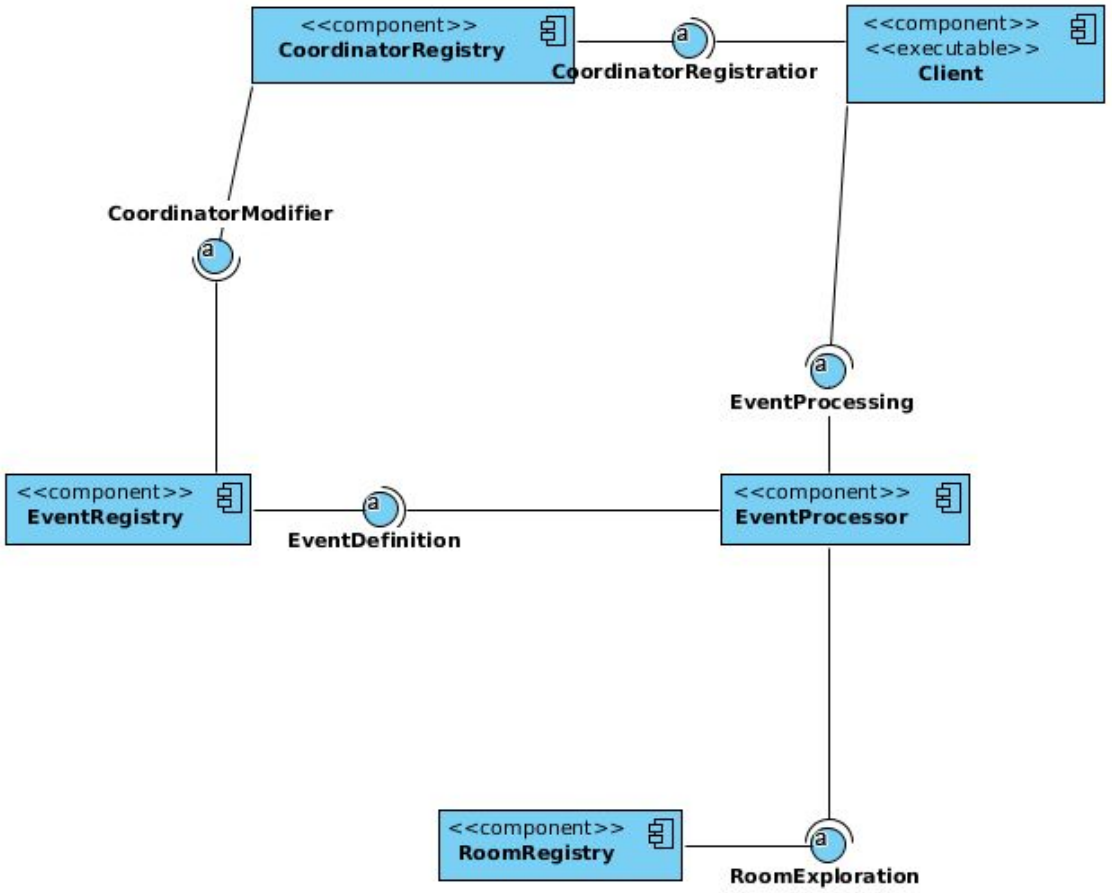
Diagramme d’interfaces



Nous fournissons la liste d’interfaces suivante afin de permettre à nos composants internes de communiquer entre eux :

- register permet à un utilisateur de s’enregistrer dans l’application
- addEvent permet à un utilisateur de finaliser la création d’un évenement
- listAvailableRooms nous permet de récupérer la liste des salles non occupées entre deux dates
- process(Event) permet de réserver directement une ou plusieurs salles en fonction de la date et de la capacité demandée
- process(Rooms, Event) permet de réserver les salles passées en paramètre, si cela est possible en fonction de la date et de la capacité demandée.
- Les interfaces de EventDefinition permettent enfin de créer l’événement.

Diagramme de Composants



Tout d'abord, on peut remarquer sur le diagramme qu'il existe un front-end *Client* (CLI, interface graphique…), qui interagit directement avec les composants J2E qui définissent la logique du MVP de l'application. De ce fait, il n'a pas été mis en place de Web Service entre le client et le reste du système puisque, dans le cadre du MVP, aucun de nos Cas d'Utilisation n'a besoin d'une API externe.

De plus, on peut noter que, dans la logique de l'application, les *Organisateurs d'événements* n'ont pas le rôle de créer les événements, fonctionnalité qui a été déléguée à un composant dédié, puisque ce type de données risque de devenir de plus en plus complexe avec les itérations.

Finalement, nous avons choisi de séparer les différents modèles de données afin de simplifier leurs opérations de persistance et de permettre leur évolution indépendamment du reste du système.

Diagramme de Déploiement

Nous n’avons pas eu le temps de réaliser ce diagramme. En effet, nous n’avons pas eu le temps de choisir un modèle d’ORM.

Etant donné que ce choix va grandement impacter notre conception et l’évolution de notre produit, nous préférons prendre plus de temps pour y réfléchir, plutôt que de faire des choix qui risquent de nous limiter par la suite.