Université de Nice - Sophia Antipolis

Polytech Nice Sophia

Sciences Informatiques 4ème année

Rapport
PolyDiploma

François DORE, Théo FORAY, Alexis SEGURA et Nathan STROBBE

Enseignants : Philippe COLLET, Guilhem MOLINES
et Anne-Marie DERY-PINNA

Table des matières

1 Vue fonctionnelle

1.1 Cas d'utilisations

Dans cette partie, nous avons étudié le sujet et défini le périmètre ainsi que les différents cas d'utilisations de notre projet.

1.1.1 Description des acteurs

1.1.1.1 Acteurs internes

Nous avons identifié plusieurs acteurs internes à partir des différents personnages évoqués dans le sujet :

- **Participant** : Cet acteur représente les personnes externes à l'université, impliquées dans l'évènement ou dans son organisation. Par exemple, la responsable de la salle des congrès ou encore une personne venant faire un discours le jour de la cérémonie. Ces personnes auront accès aux informations de la remise des diplômes.
- **Membre de l'université** : Cet acteur est abstrait. Il représente toutes les personnes de l'université prenant part à la cérémonie, que ce soit en tant qu'organisateur ou que participant. Ces membres auront la possibilité de se connecter via le service d'authentification de l'université.
- **Étudiant diplômé** : Cet acteur représente les étudiants qui seront diplômés lors de la cérémonie. Ces étudiants devront interagir avec le système afin de pouvoir confirmer leur présence et inscrire les personnes qu'ils souhaitent inviter.
- **Membre du service de communication** : Cet acteur représente les membres du service de communication de l'école. C'est ce service qui organise la remise des diplômes. Les membres doivent pouvoir effectuer diverses tâches administratives sur l'application afin de gérer l'organisation de la cérémonie.
- **Membre de la scolarité** : Les membres de la scolarité ont la responsabilité de renseigner les informations des étudiants diplômés. Cet acteur doit donc pouvoir effectuer la saisie sur l'application *PolyDiploma*.
- **Comptable** : Cet acteur représente le ou les comptable(s) de l'école. Ce dernier doit accéder à un certain nombre de documents (devis, facture, reçus...) et pouvoir en rajouter.
- **Responsable BDE** : Le BDE est la principale association étudiante de l'école et est impliqué dans l'organisation de l'évènement. Le responsable du BDE doit pouvoir proposer des intermèdes, consulter les différents documents comptables et gérer les invitations des étudiants.
- **Invité VIP** : Les invités VIP sont des participants dans le système, car ils peuvent intervenir lors de la cérémonie (par exemple : faire un discours). Ils doivent donc avoir accès aux informations de la RDD et pouvoir confirmer leur présence.

1.1.1.2 Acteurs externes

Nous avons identifié trois acteurs externes apportant divers services à notre solution :

- **Service mail de l'université** : Cet acteur permet l'envoi de mails par le système, que ce soit pour prévenir des changements liés à la cérémonie ou encore pour l'envoi des invitations.
- **Service de banque** : Cet acteur est impliqué dans le paiement lors de l'ajout de nouveaux invités par les diplômés (à partir de deux invités par personne).
- **Service de l'université** : Le service scolaire va permettre de récupérer le nom des étudiants diplômés et de fournir un système d'authentification permettant de différencier les utilisateurs.

1.1.2 Diagrammes de cas d'utilisation

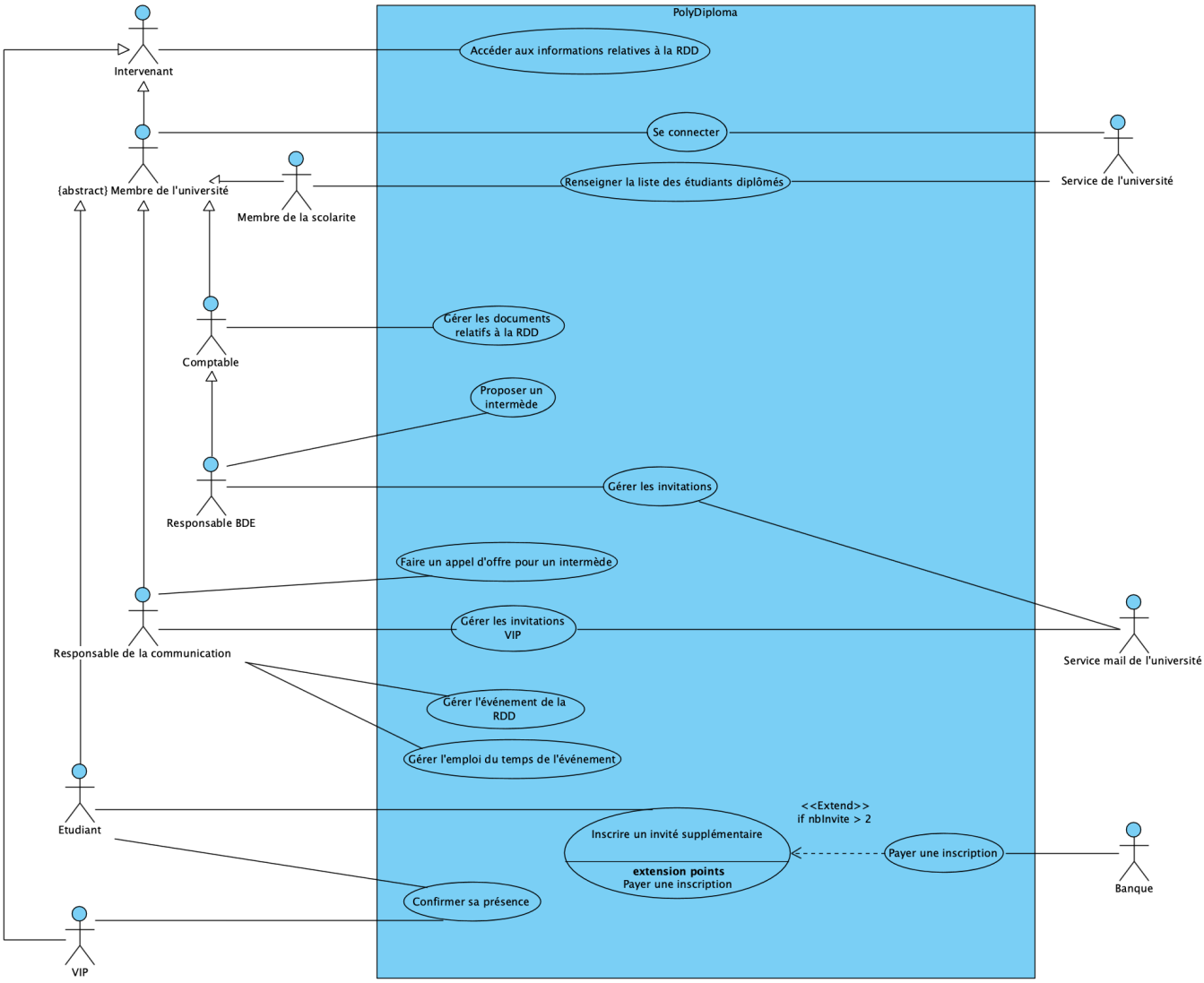

FIGURE 1 – Diagramme de cas d'utilisation.

1.1.3 Précisions concernant divers cas d'utilisation

1.1.3.1 Accéder aux informations relatives à la RDD

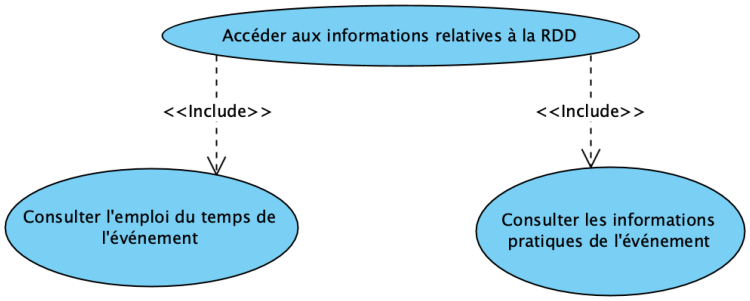

FIGURE 2 – Cas d'utilisation : Accéder aux informations relatives à la remise des diplômes.

Toute personne participant à la RDD pourra accéder aux informations de l'évènement. On distingue les informations concernant l'emploi du temps de la cérémonie (heure du repas, ordre de passage des promos, intermèdes ...) et les informations pratiques de l'évènement comme l'heure, le lieu, la date ou encore la date butoir des inscriptions.

1.1.3.2 Gérer les documents

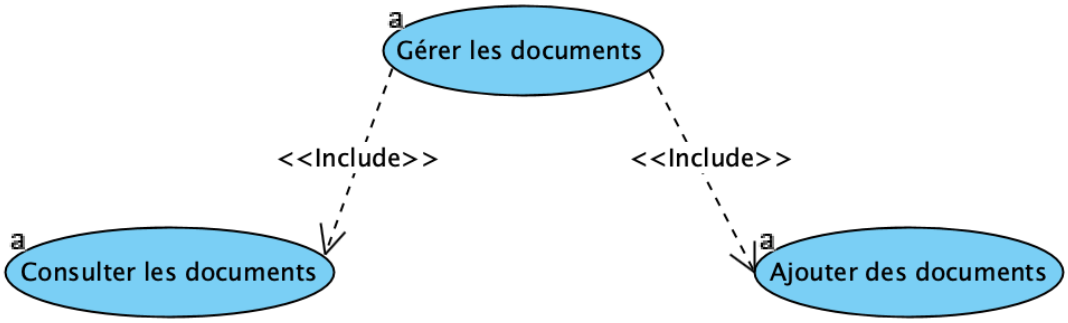

FIGURE 3 – Cas d'utilisation : Gérer les documents.

Le comptable de l'université va pouvoir consulter les différents documents, tels que les justificatifs, les factures, les reçus. Ces documents sont nombreux, cependant il va pouvoir facilement les trouver grâce à ce système. De plus, le comptable va également pouvoir ajouter des documents pour justifier des ordres d'achat pour la cérémonie.

1.1.3.3 Gérer les invitations VIP

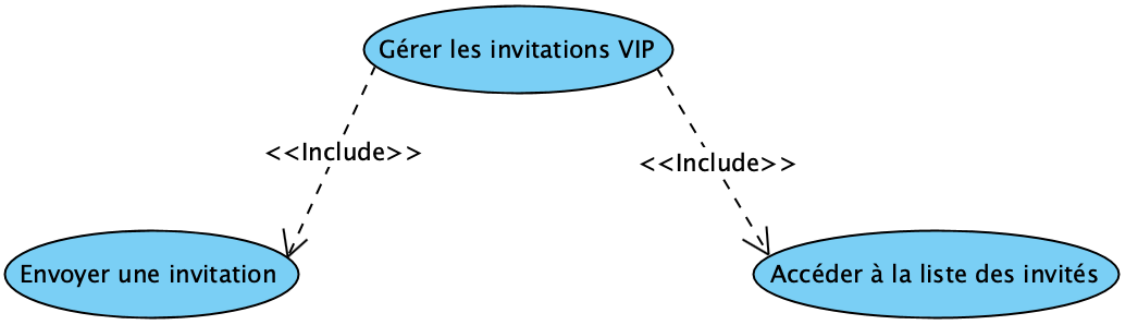

FIGURE 4 – Cas d'utilisation : Gérer les invitations VIP.

C'est le responsable de la communication qui va inviter des personnes externes à l'université en tant que VIP. Pendant l'organisation de la RDD, il va pouvoir envoyer des invitations et proposer de faire des interventions/discours. Cette invitation se fait grâce au service mail de l'université. Le responsable va donc également pouvoir accéder à la liste des invités participant à la RDD.

1.1.3.4 Gérer l'évènement de la RDD

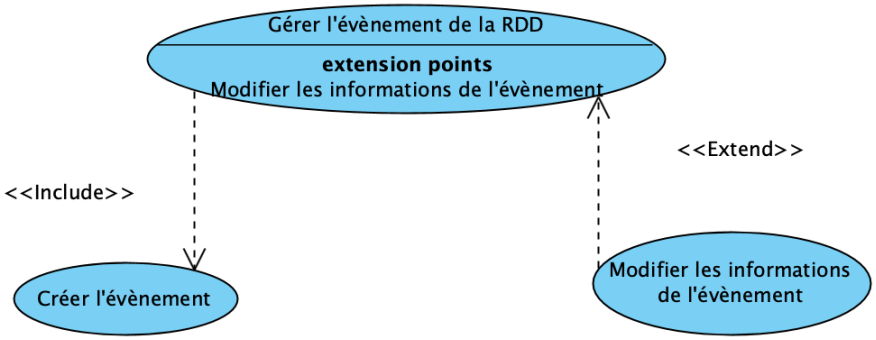

FIGURE 5 – Cas d'utilisation : Gérer l'évènement de la RDD.

L'acteur de ce cas d'utilisation est le responsable de la communication. Pour gérer l'évènement, il doit d'abord l'avoir créée, en renseignant entre autres la date butoir des inscriptions. La possibilité lui est alors laissée de modifier les informations de l'évènement après sa création.

1.1.3.5 Renseigner la liste des étudiants diplômés

Un membre de la scolarité pourra indiquer au système la liste des étudiants qui recevront leurs diplômes lors de la cérémonie. Il aura alors les droits pour accéder et éditer ces données. Il est important de noter que la décision de diplômer un élève ou non ne concerne pas notre système, nous sommes partis du principe que le membre de la scolarité aura reçu au préalable la liste desdits étudiants auprès d'un autre service de l'école. Enfin, cette fonctionnalité sera en lien avec un service externe de l'université contenant toutes les informations pertinentes quant à un élève.

1.2 Diagramme de composants

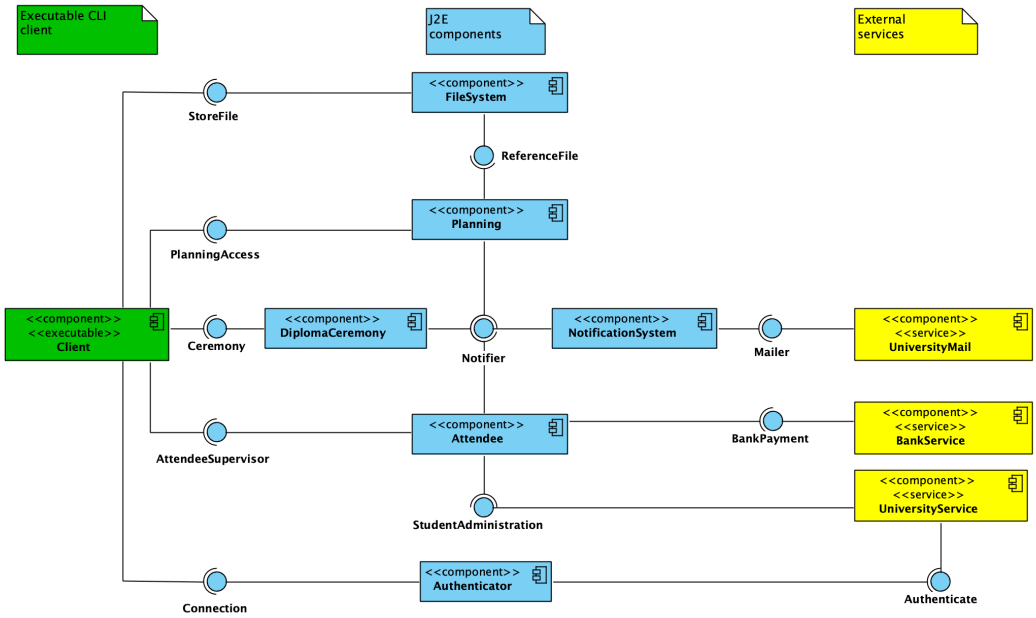

FIGURE 6 – Diagramme de composants.

Pour ce diagramme, nous avons responsabilisé chaque composant afin qu'il réponde aux besoins d'un concept métier. Au départ, nous souhaitions mettre des composants web services pour permettre le lien entre le client et les différents composants de la partie backend. Cependant, nous avons fait le choix de ne pas représenter cette partie, car elle n'apportait pas de comportement métier.

1.3 Interfaces

Notifier :

```
void notifyPlanningChange();
void notifyInformationChange();
void notifyVIPInvitations(VIPGuest[]);
void notifyStudentInvitations(Student[]);
```

L'interface offre la possibilité de notifier les personnes d'une modification des informations pratiques de la RDD ou du planning. L'interface permet aussi d'envoyer les invitations à une liste d'élèves ou d'invités VIP.

Mailer :

```
void sendMail(addressee: String, subject: String, content: String);
```

Cette interface permet l'envoi de mail via un service externe.

BankPayment :

```
boolean pay(price: int);
```

Cette interface permet le paiement de places supplémentaires via un service externe.

Connection :

```
boolean login(user: User);
```

L'interface **Connection** fait simplement le lien entre tout type d'**User** et la connexion au service de l'université (***UniversityService***). Cela permet de la modularité si de nouveaux utilisateurs étaient amenés à utiliser l'application.

InvitationSupervisor :

```
void sendStudentInvitations();
void sendVIPInvitations(guests: VIPGuest[]);
Attendee[] getConfirmedAttendees();
void addGraduatedStudent(student: Student);
void confirmAttendance(user: User);
```

Cette interface fait le lien entre le ***Client*** et le composant ***Attendee*** et régit tout le système d'invitation. Elle permet donc non seulement d'envoyer des invitations aux étudiants ou aux membres VIP, mais également de confirmer les invitations.

PlanningAccess :

```
boolean addEvent(event: Event);
boolean addInterlude(newInterlude: Interlude, interludeCallTender: InterludeCallTender);
boolean attributePerformers(interlude: Interlude, performerMails: String[]);
boolean linkDocument(documentsToLink: Document[], event: Event);
void addInterludeCallTender(newInterludeCallTender: InterludeCallTender);
void modifyEvent(event: Event);
Planning displayPlanning();
```

Cette interface permet au client d'accéder au planning de la cérémonie et de l'éditer. C'est une des interfaces majeures du système puisque la visualisation du planning est commune à tous les acteurs. Dans la méthode *attributePerformer*, on passe un tableau de string de mails et non des objets métiers, car les *performeurs* ne sont pas forcément des personnes enregistrées dans le système.

ReferenceFile :

```
void linkDocumentsToEvent(documentsToAdd: Document[]);
Document[] getDocuments(event: Event);
```

Cette interface va permettre de faire le lien entre le ***Planning*** et le ***FileSystem*** et essentiellement associer un ou plusieurs documents à un évènement du planning. En effet, la méthode *linkDocumentsToEvent* va permettre cette liaison entre ces objets métiers. De plus, on remarque que cette méthode est la même dans les interfaces **ReferenceFile** et **PlanningAccess**, en effet le ***Planning*** ne peut pas effectuer cette liaison sans la connaissance des objets **Documents**.

StoreFile :

```
boolean create(document: Document);
Document[] read();
Document[] read(filter: String);
boolean update(document: Document);
boolean delete(document: Document);
```

Cette interface va permettre une interaction totale entre tous les documents du système et le client. Cette interaction est basée sur le principe de persistance de données CRUD. Il fait donc en sorte de pouvoir créer, lire, mettre à jour et supprimer des documents. Celle-ci est essentielle au bon fonctionnement du système. L'interface permet également de lire certains documents à l'aide d'un filtre pour éviter une surcharge de données dans la lecture de tous les documents.

StudentAdministration :

```
Student[] getStudents();
```

Cette interface qui lie les composants ***Attendee*** et ***UniversityService*** permet d'obtenir les étudiants diplômés qui participeront à la RDD. C'est donc proposé par ***UniversityService*** et requis par ***Attendee***, pour faire entrer cette liste dans le système.

Ceremony :

```
boolean create(ceremony: DiplomaCeremony);
DiplomaCeremony readInfos();
boolean update(ceremony: DiplomaCeremony);
boolean delete(ceremony: DiplomaCeremony);
```

Cette interface est nécessaire pour la création initiale de l'évènement. De plus, elle permet d'éditer des informations non présentes dans les autres composants et importantes pour tous les participants comme le lieu ou la date de la cérémonie.

2 Vue développement

2.1 Diagramme de classes des Business objects

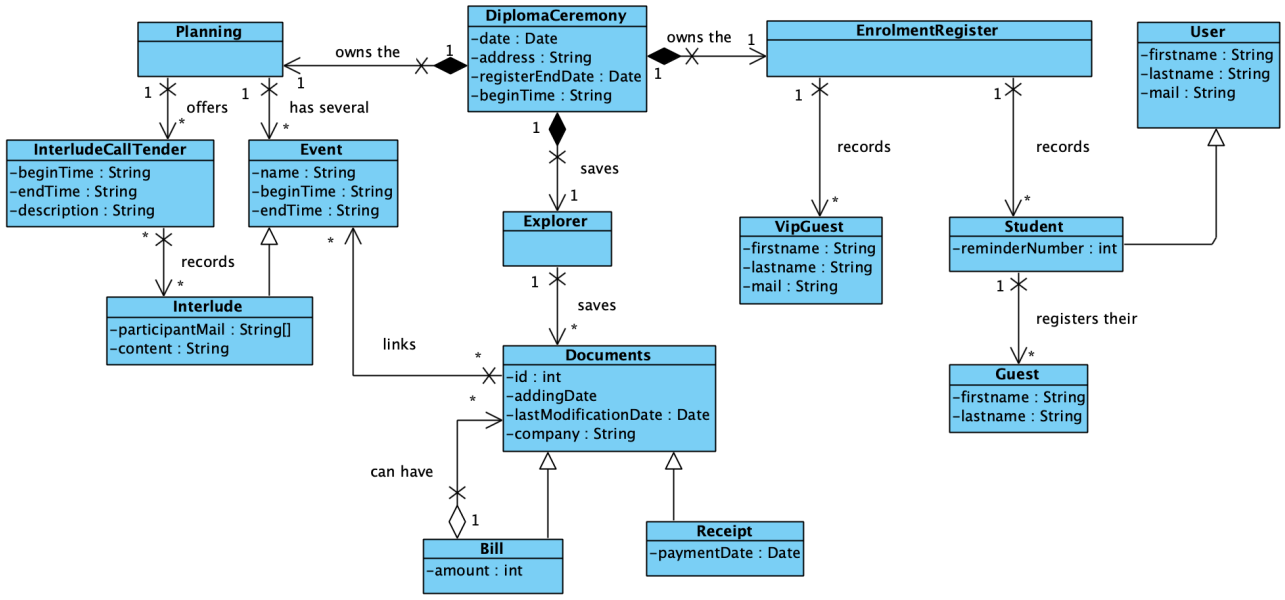

FIGURE 7 – Diagramme de cas de classes des objets métier.

Pour la partie inscription, tous les types de participants vont confirmer leur présence dans l'**EnrolmentRegister**. Nous avons une classe mère **User** dont **Student** va hériter, pour avoir une structure plus modulaire, dans le cas où de nouveaux types d'utilisateurs seraient ajoutés. Cependant les VIP sont séparés dans la classe **VipGuest**, qui n'hérite pas d'**User**, car nous ne les considérons pas comme des utilisateurs au même titre que les étudiants. En effet, ils sont extérieurs à l'école et seront plus passifs, par exemple ils ne peuvent pas inviter d'autres personnes.

Les **Documents** sont stockés dans un **Explorer**. Nous avons choisi le patron de conception *Composite Pattern* pour les documents. En effet, cela permet de stocker éventuellement plusieurs documents dans un document (par exemple, une facture avec son reçu de paiement). Actuellement, nous avons des factures et des reçus. Cette conception nous permet d'être plus flexibles et nous offre la possibilité d'ajout d'autre type de documents à l'avenir. De plus, il est possible d'associer un ou plusieurs documents à un **Event** (Cf. **ReferenceFile**)

Dans le planning, il est possible d'ajouter des **InterludeCallTender** (appels d'offres pour des intermèdes). Par la suite, des **Interlude** peuvent être ajoutés. Ces **Interlude** héritent d'**Event** car ces classes ont des propriétés communes et représentent toutes deux des créneaux horaires dans le **Planning**.

3 Conclusion

En rédigeant ce document, nous avons essayé d'avoir une vision globale de notre architecture grâce à de nouvelles notions comme les diagrammes de composants ou encore les objets métier. Cet exercice fut un véritable défi pour nous, car c'est la première fois que nous devons avoir une vision long terme aussi tôt dans un projet.
Ce document nous servira de référence et nous guidera dans le développement futur du projet.