Équipe 203

Polygram
Document d'architecture logicielle

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
2022-02-07	1.0	Rédaction initiale du document d'architecture logicielle	Équipe 203

Table des matières

1. Introduction 1

2. Objectifs et contraintes architecturaux 1

2.1. Objectifs 1

2.1.1. Réutilisabilité 1

2.1.2. Maintenabilité 1

2.2. Contraintes 1

2.2.1. Ressources humaines, temporelles et financières 1

2.2.2. Outils de développement 1

2.2.3. Langage de programmation et cadriciels 2

2.2.4. Portabilité 2

3. Vue des cas d'utilisation 2

Figure 1. Diagramme de cas d'utilisation pour l'authentification d'un utilisateur 2

Figure 2. Diagramme de cas d'utilisation pour les actions utilisateurs sur les albums 3

Figure 3. Diagramme de cas d'utilisation pour la configuration d'un compte utilisateur 3

Figure 4. Diagramme de cas d'utilisation pour la communication entre utilisateurs 4

Figure 5. Diagramme de cas d'utilisation pour l'éditeur de dessin du client lourd 5

Figure 6. Diagramme de cas d'utilisation pour l'éditeur de dessin du client léger 5

4. Vue logique 6

Figure 7. Diagramme de paquetage du système global 6

5. Vue des processus 9

Figure 8. Diagramme de séquence pour l'authentification d'un utilisateur 9

Figure 9. Diagramme de séquence pour l'obtention d'un nouveau mot de passe par le biais du bouton "Mot de passe oublié" 9

Figure 10. Diagramme de séquence pour l'inscription d'un nouvel utilisateur 10

Figure 11. Diagramme de séquence pour la recherche d'un dessin 10

Figure 12. Diagramme de séquence pour la modification d'un dessin lors d'une collaboration en synchronisation simple (les actions d'édition sont partagées à leur terme) 10

Figure 13. Diagramme de séquence pour l'envoi d'un message dans un canal 11

Figure 14. Diagramme de séquence pour la modification d'un profil utilisateur 11

6. Vue de déploiement 12

Figure 15. Diagramme de déploiement pour le système global 12

7. Taille et performance 13

Document d'architecture logicielle

1. Introduction

Ce document présente l'architecture logicielle de haut et bas niveau de l'application Polygram. Préalablement, les objectifs et contraintes architecturales seront discutés. Par la suite, l'architecture sera décomposée selon quatre (4) vues différentes afin de développer adéquatement chaque niveau d'abstraction : une vue des cas d'utilisation, une vue logique, une vue des processus ainsi qu'une vue de déploiement. Des diagrammes suivant le standard UML permettront de supporter le design architectural choisi pour chaque niveau. Respectivement, un diagramme de cas d'utilisation, un diagramme de paquetage, un diagramme de séquence ainsi qu'un diagramme de déploiement illustreront les vues décrites ci-dessus. S'en suivra par la suite une description des caractéristiques de taille et de performance qui pourraient venir impacter la conception de l'architecture logicielle de l'application.

2. Objectifs et contraintes architecturaux

2.1. Objectifs

2.1.1. Réutilisabilité

Afin de favoriser la réutilisabilité du code source, la structure logique de l'application doit, dans la mesure du possible, minimiser le couplage entre les classes et augmenter la cohésion interne de chaque classe. De plus, les méthodes et fonctions doivent être atomiques, c'est-à-dire doivent être écrites de façon à ce qu'elles occupent une seule responsabilité.

2.1.2. Maintenabilité

Afin de faciliter la maintenabilité du code source, celui-ci doit être écrit en suivant le principe KISS (*Keep It Simple, Stupid*) qui consiste à favoriser la simplicité au maximum et à proscrire la complexité non indispensable. Le code doit également suivre le principe de "self-documenting", c'est-à-dire un code source qui suit des conventions de nommage et d'écriture dictées préalablement (voir section 4.4 du SRS). Pour les parties du code source complexe (p. ex. des fonctions implémentant un algorithme ou une formule mathématique complexe), de courts commentaires doivent être présents. Également, l'architecture choisie doit être conçue de façon à bénéficier le plus possible des avantages de la programmation orientée objet ou orientée composante incluant l'héritage et le polymorphisme afin de faciliter la lisibilité du code et éviter au maximum la duplication du code. Des patrons de conception doivent également être utilisés dans les situations qui en bénéficient.

2.2. Contraintes

2.2.1. Ressources humaines, temporelles et financières

Le projet est développé dans un cadre scolaire, donc il est impératif d'être capable de l'achever dans les délais d'une session universitaire, soit 3 mois. De plus, notre équipe se limite à 5 étudiants et nous n'avons aucune aide financière pour réaliser ce projet. Par conséquent, la complexité de notre architecture doit prendre en considération ces faits.

2.2.2. Outils de développement

L'application doit être développée avec l'éditeur de code extensible Visual Studio Code dans le cas du

client lourd et du serveur. L'environnement de développement intégré (IDE) Android Studio doit être utilisé pour le développement du client léger. De plus, le logiciel de gestion de version utilisé doit être Git. La forge, se basant sur git, sera la plateforme gratuite GitLab.

2.2.3. Langage de programmation et cadriciels

L'application doit être programmée avec deux langages différents pour le client lourd et pour le client léger. Le langage de programmation TypeScript sera utilisé pour développer le client lourd et le serveur. Plus précisément, le client lourd utilisera le cadriciel Angular et Angular Material pour les composantes visuelles de l'application. Le serveur utilisera le service d'hébergement Heroku. Le client léger sera quant à lui programmé avec le langage de programmation Kotlin. Pour la persistance des données de l'application nous utiliserons MongoDB.

2.2.4. Portabilité

Le client lourd doit être compatible avec le système d'exploitation Windows 10, 64 bits. L'exécutable sera généré par l'environnement Electron. Le client léger doit être exécutable sur Android Pie.

3. Vue des cas d'utilisation

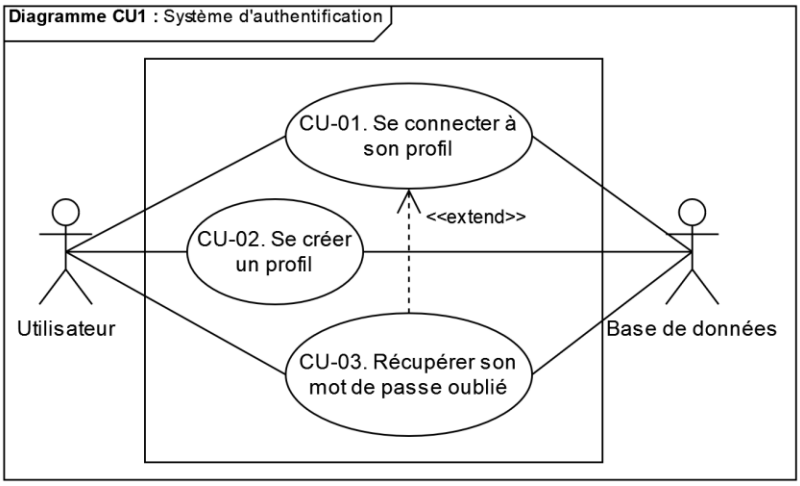

Figure 1. Diagramme de cas d'utilisation pour l'authentification d'un utilisateur

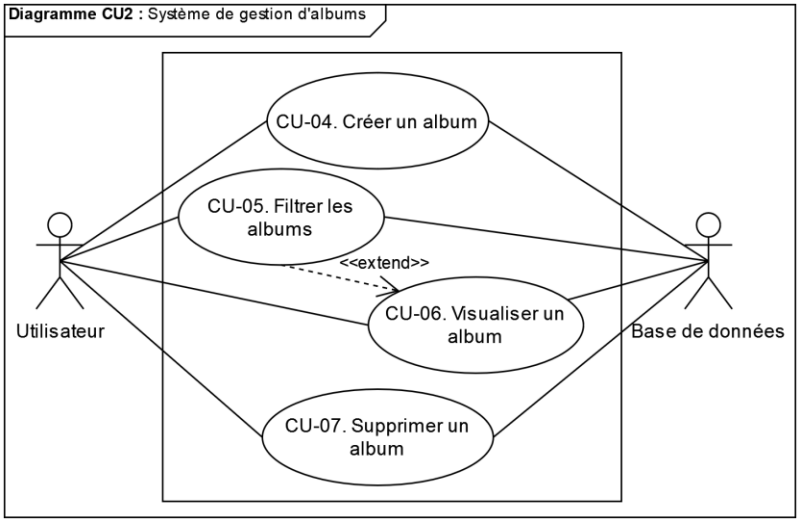

Figure 2. Diagramme de cas d'utilisation pour les actions utilisateurs sur les albums

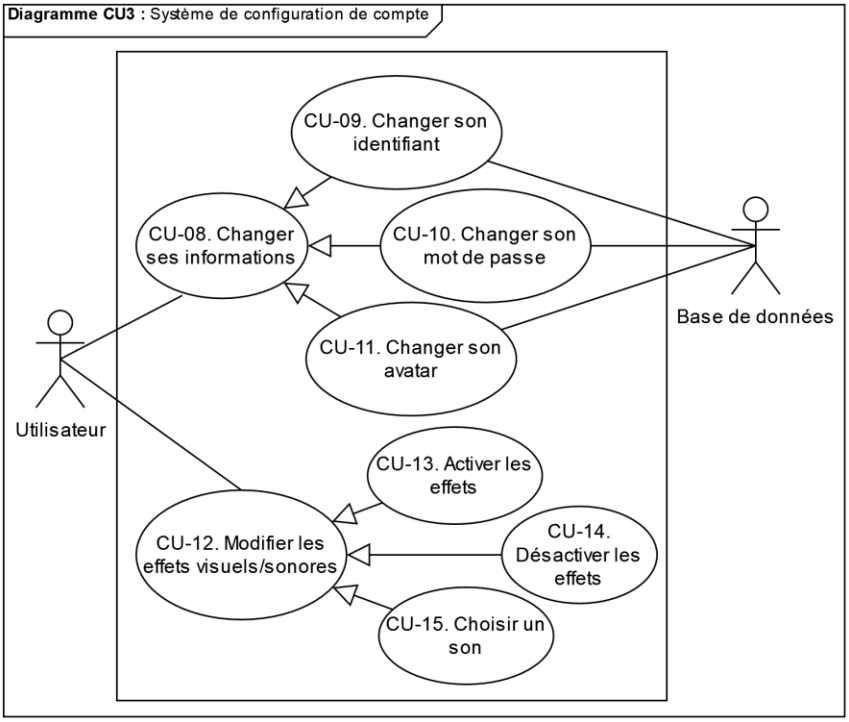

Figure 3. Diagramme de cas d'utilisation pour la configuration d'un compte utilisateur

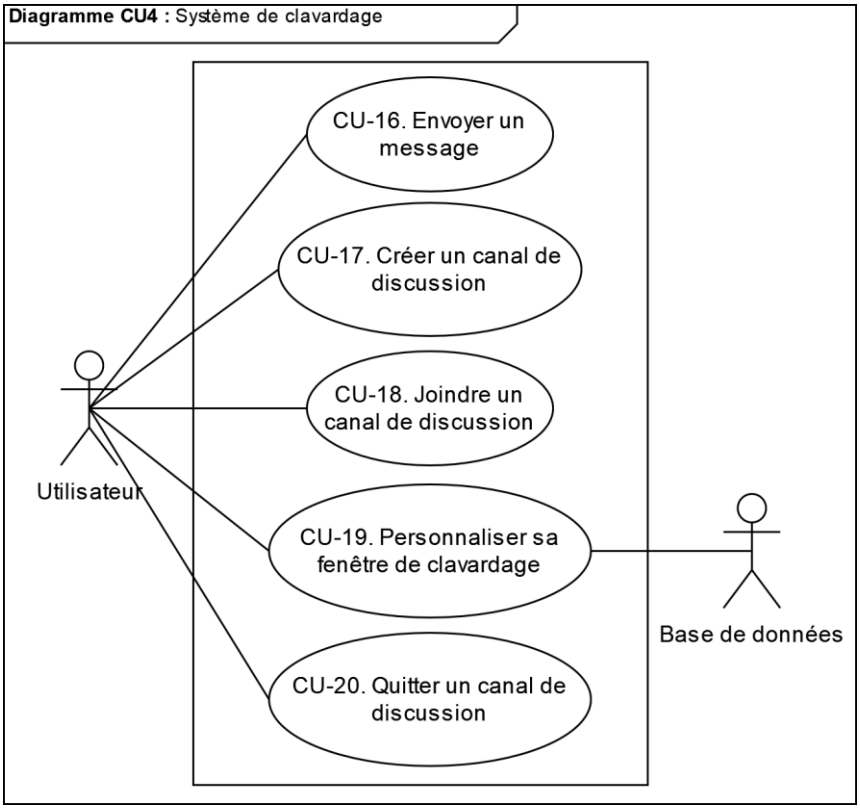

Figure 4. Diagramme de cas d'utilisation pour la communication entre utilisateurs

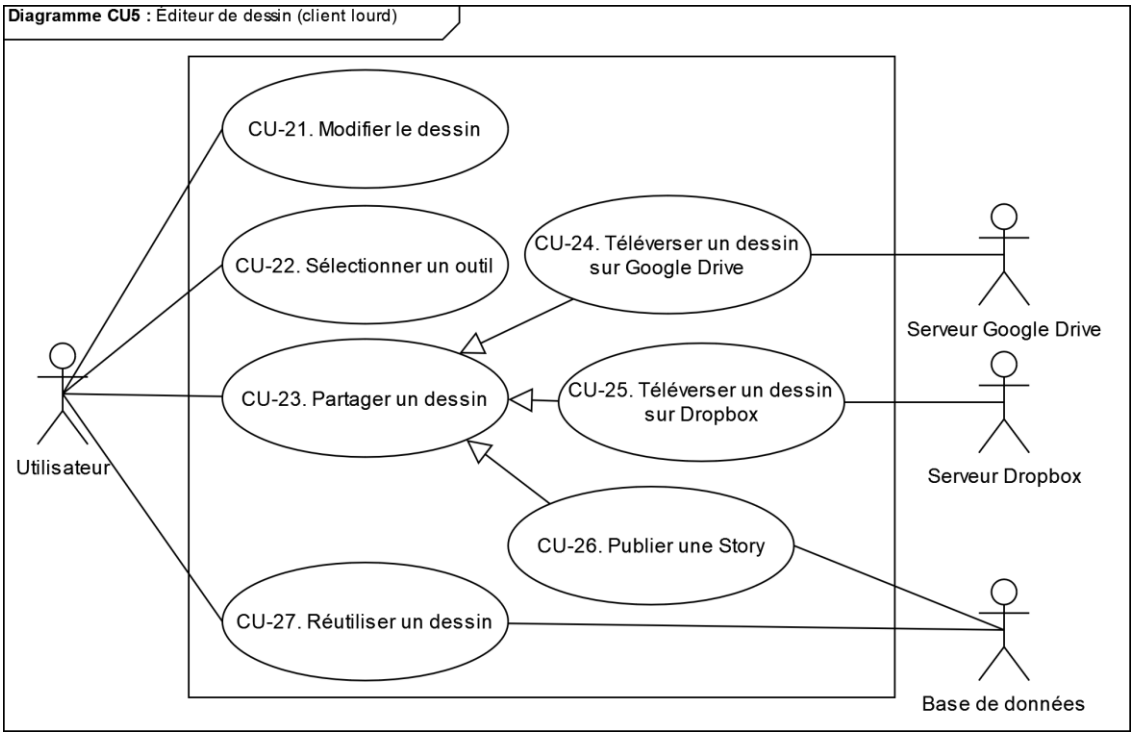

Figure 5. Diagramme de cas d'utilisation pour l'éditeur de dessin du client lourd

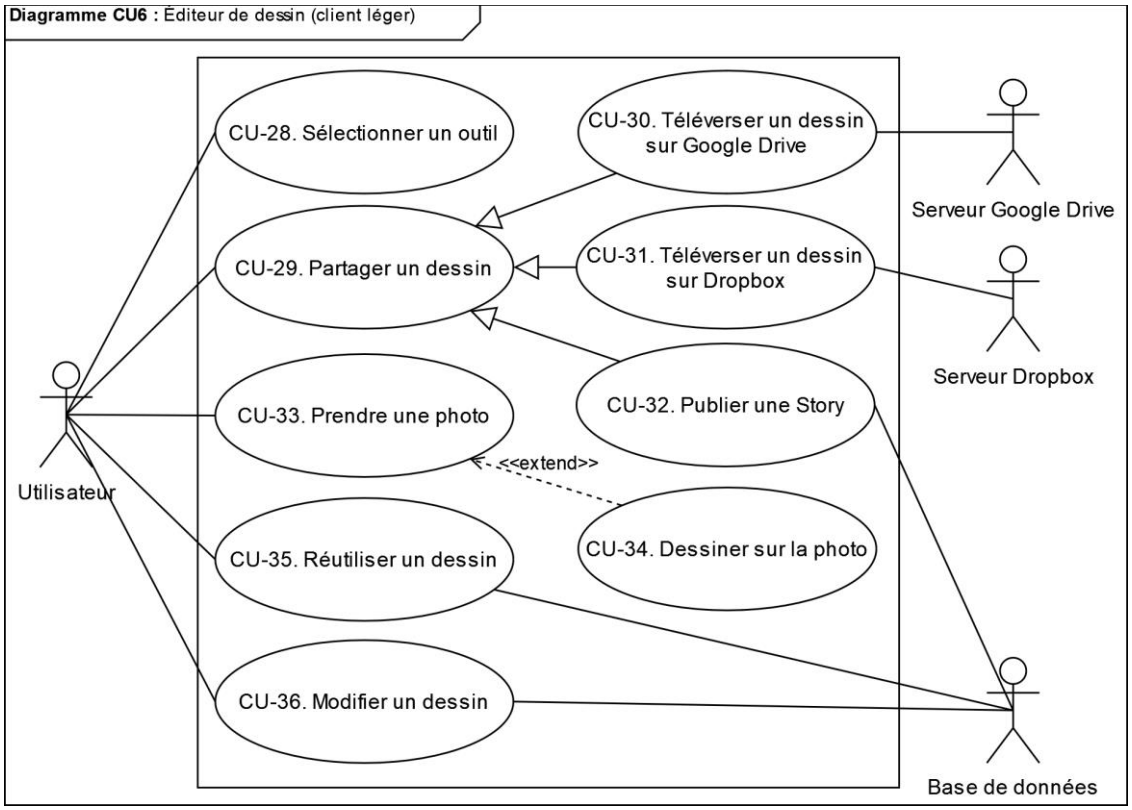

Figure 6. Diagramme de cas d'utilisation pour l'éditeur de dessin du client léger

4. Vue logique

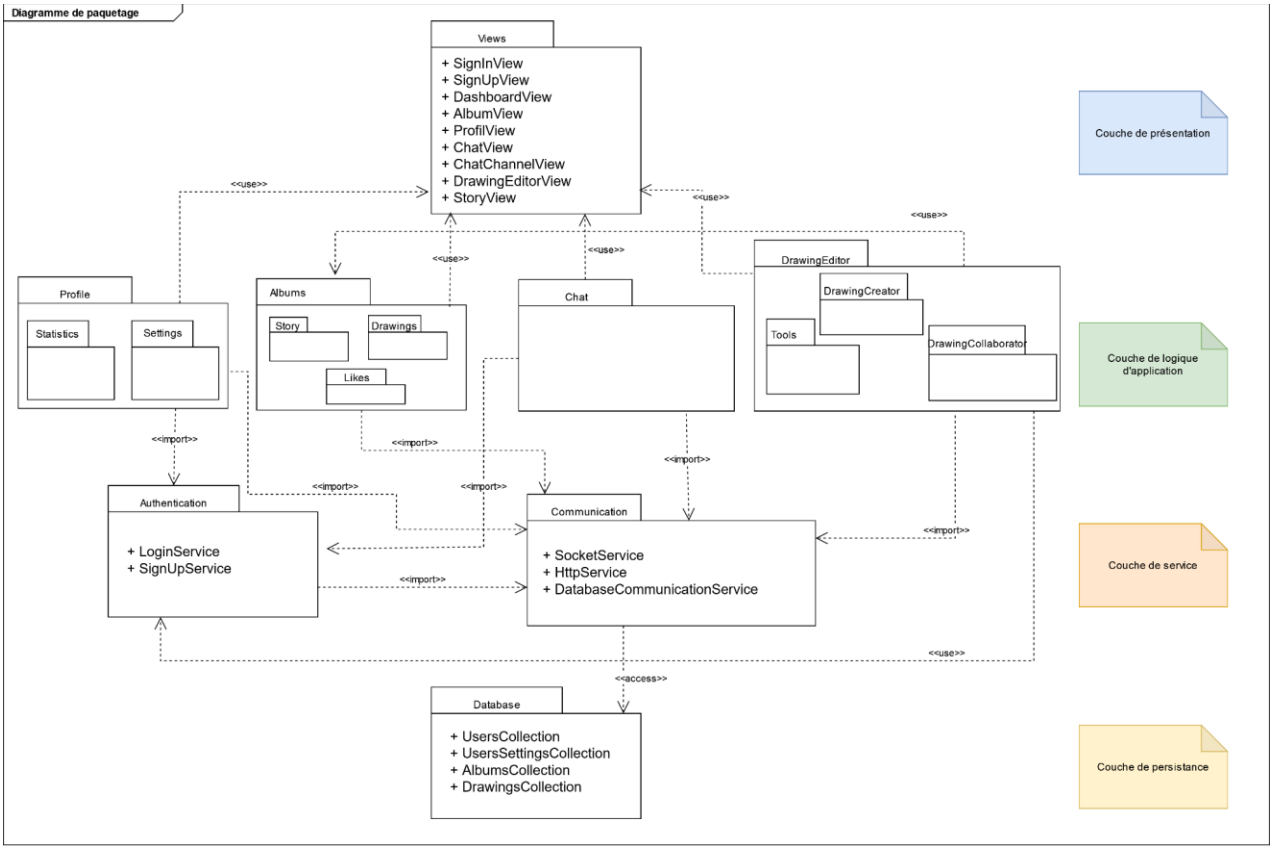

Figure 7. Diagramme de paquetage du système global

<View>
Ce paquetage regroupe toutes les différentes fenêtres et pages de l'application. Les composantes incluses dans le paquetage font partie de la couche de présentation et seront la partie visible par l'utilisateur.

<Profile>
Ce paquetage contient le paquetage <Statics> et le paquetage <Settings>. <Profile> va utiliser les évènements qui arrivent au niveau de la couche de présentation afin de les interpréter sous forme de statistique ou de paramètre qui apparaissent au niveau du profil. Ce paquetage contient la partie logique du profil utilisateur.

<Statistics>
Ce paquetage contient tous les modules reliés au calcul de statistique de l'utilisateur.

<Settings>
Ce paquetage contient les modules nécessaires afin de traiter les paramètres de l'utilisateur et assurer la persistance de ces paramètres durant la session de l'utilisateur.

<Albums>
Ce paquetage va permettre de réaliser toute la logique derrière le système d'album public et privé, va nous assurer la vue des *story* grâce au paquetage <Story> et va nous permettre de gérer les dessins par le biais du paquetage <Drawings>. Le paquetage utilise la communication par une dépendance de *import* afin d'avoir les informations sur les utilisateurs pour interpréter l'impact de leurs interactions.

<Story>
Ce paquetage va permettre de réaliser la logique du système de story dans l'application. Une story est assimilée à un album de dessin, mais avec une visualisation différente et quelques fonctionnalités de plus. Le paquetage est lié par une relation de dépendance de type "use" avec le drawing editor, car les story proviennent de l'espace de dessin.

<Drawings>
Ce paquetage va permettre de réaliser toute la logique derrière le système d'album public et privé, va nous assurer la vue des story grâce au paquetage <Stroy> et va nous permettre de gérer les dessins par le biais du paquetage <Drawings>.

<Like>
Ce paquetage gère les mentions "j'aime" à travers notre application. Il fait partie du paquetage Albums car c'est ce dernier qui nous assure de voir toutes les mentions j'aime laissée sur les différents dessins dans les albums auxquels l'utilisateur a accès. Le paquetage Like va utiliser les mécanismes de communication qui sont dans le paquetage correspondant d'où la relation import. Il faut qu'un utilisateur soit correctement identifié pour laisser des mentions j'aime ou "like" d'où le lien import avec ce paquetage.

<Chat>
Ce paquetage contient tous les modules nécessaires pour le clavardage, le fenêtrage du clavardage, le système de notification et le système de canaux pour le chat. Il utilise le paquetage <Authentication>, car l'utilisateur doit se connecter afin de pouvoir clavarder. Il *import* le paquetage de communication, car le clavardage et le système de canaux sont possibles grâce au service de socket.

<DrawingEditor>
Ce paquetage va nous permettre de gérer tout l'aspect de dessin de notre application. Cela inclut les outils de dessin à travers le paquetage <Tools>, la création de dessin, les interactions avec la zone de dessin à travers le paquetage <DrawingCreator>, et le côté dessin collaboratif entre les utilisateurs par le biais de <DrawingCollection>. Pour la collaboration nous avons une dépendance avec le paquetage communication de type "import". (Cela est expliqué dans le document protocole de communication)

<DrawingCreator>
Ce paquetage gère la création de dessins et la zone de dessin.

<Tools>
Ce paquetage contient tous les outils de dessin nécessaires.

<DrawingCollaborator>
Ce paquetage gère le mode de dessin collaboratif.

<Communication>
Ce paquetage gère tout le volet communication et constitue notre volet serveur pour l'application. C'est la couche qui va permettre la transmission des divers paquets qui circulent dans notre système pour la majorité de nos fonctionnalités.

<Authentication>
Contient les services de connexion et de création de comptes en étant relié à la base de données afin d'envoyer et vérifier les données entrées par l'utilisateur et ce à travers un service de base de données, ce qui explique la dépendance de type <import> avec le paquetage de communication qui lui à son tour est relié par une une dépendance <access> à la base de données.

<Database>
Ce paquetage va inclure les différentes collections dont nous aurons besoin pour les différents modules de notre système.

5. Vue des processus

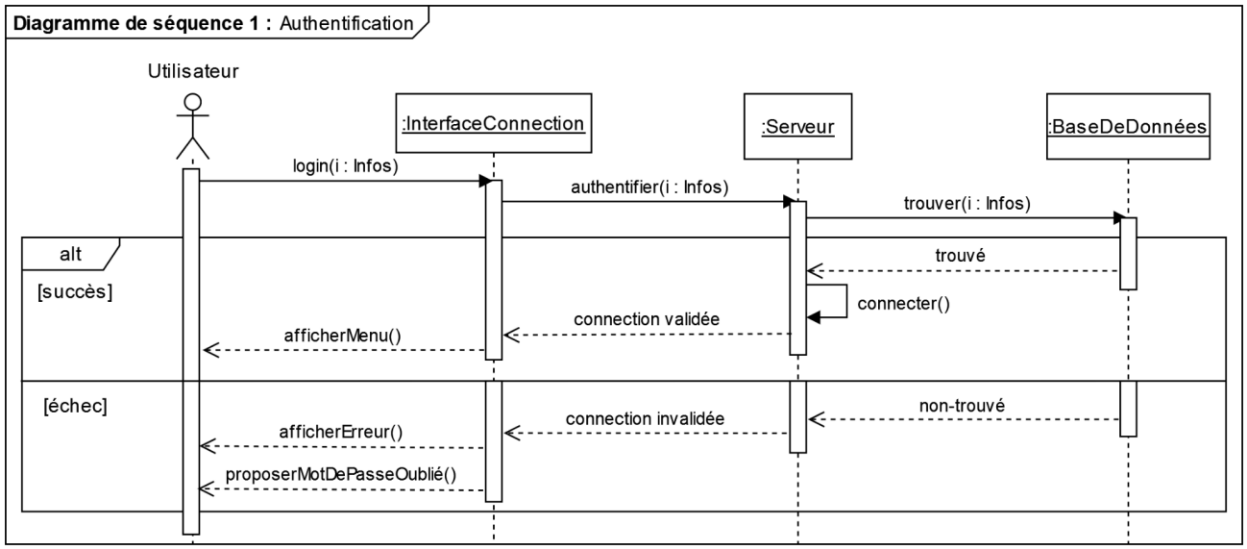

Figure 8. Diagramme de séquence pour l'authentification d'un utilisateur

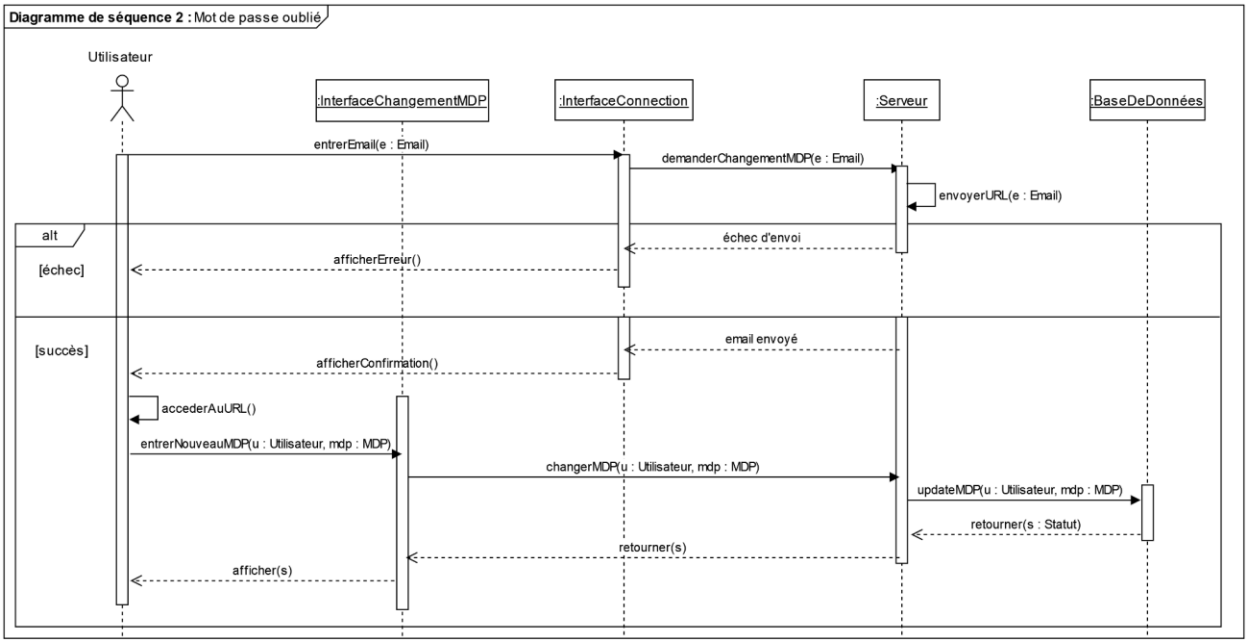

Figure 9. Diagramme de séquence pour l'obtention d'un nouveau mot de passe par le biais du bouton "Mot de passe oublié"

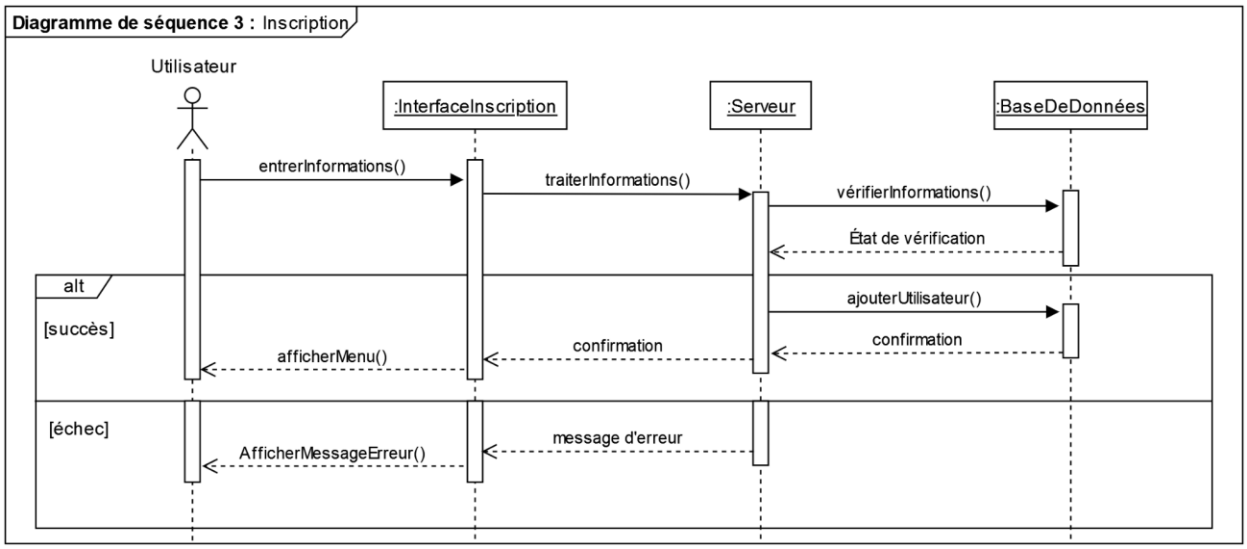

Figure 10. Diagramme de séquence pour l'inscription d'un nouvel utilisateur

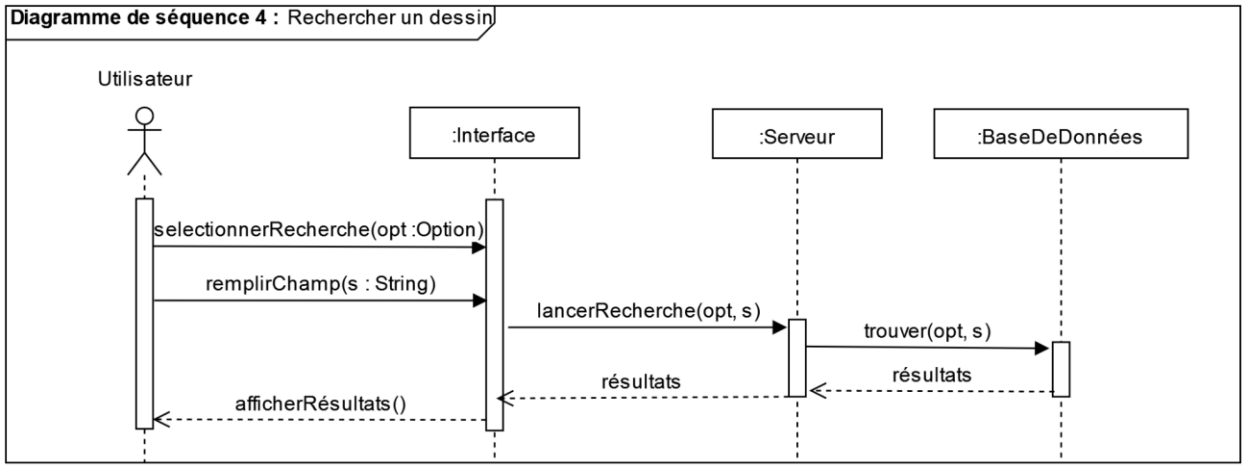

Figure 11. Diagramme de séquence pour la recherche d'un dessin

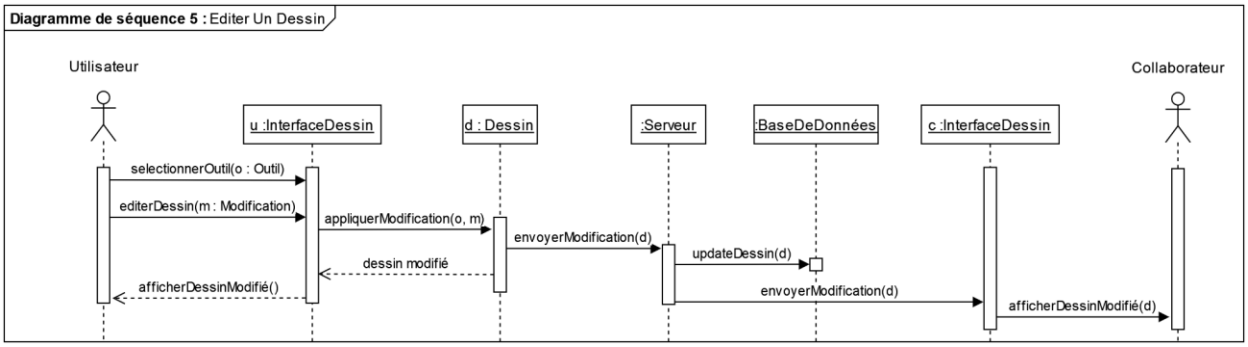

Figure 12. Diagramme de séquence pour la modification d'un dessin lors d'une collaboration en synchronisation simple (les actions d'édition sont partagées à leur terme)

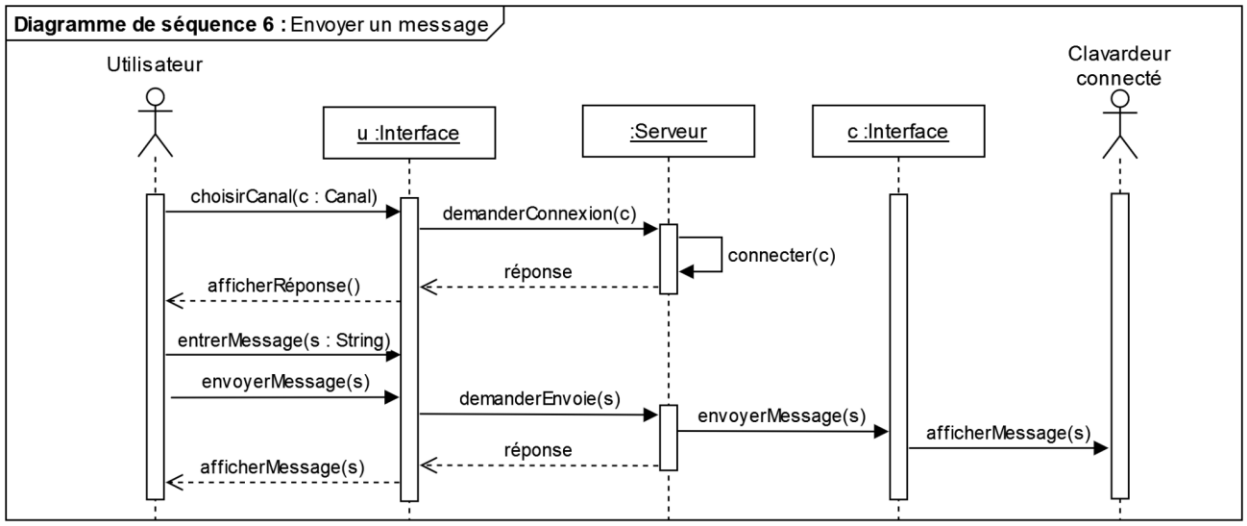

Figure 13. Diagramme de séquence pour l'envoi d'un message dans un canal

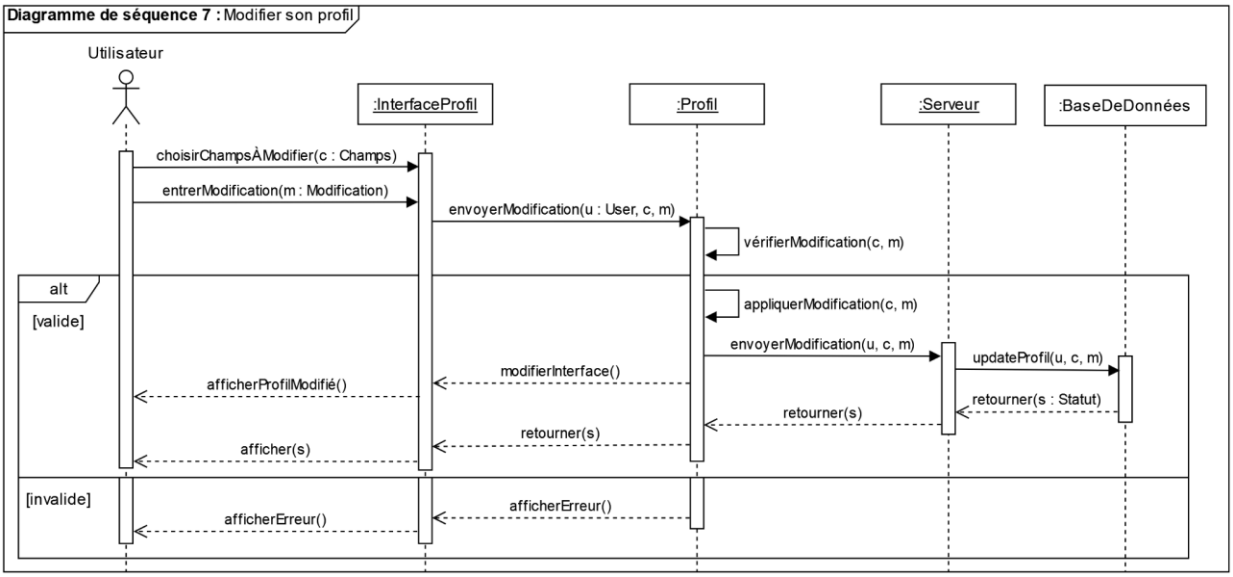

Figure 14. Diagramme de séquence pour la modification d'un profil utilisateur

6. Vue de déploiement

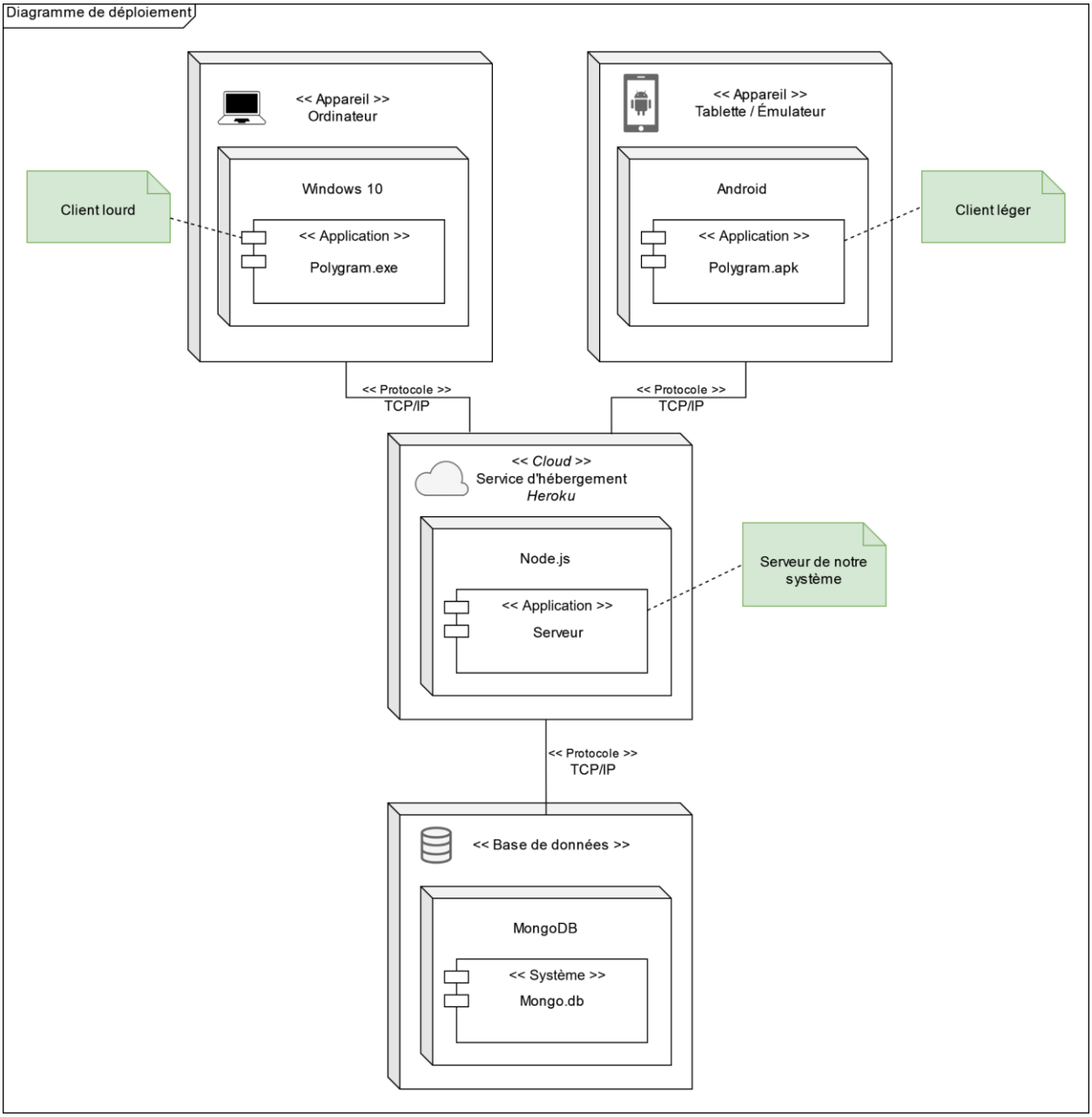

Figure 15. Diagramme de déploiement pour le système global

7. Taille et performance

Le client lourd devra nécessiter au plus 1 Go d'espace sur le disque et au plus 500 Mo de mémoire vive. Le client léger devra quant à lui nécessiter au plus 500 Mo d'espace mémoire sur la tablette et au plus 300 Mo de mémoire vive. En ce qui concerne les performances graphiques, le taux de rafraîchissement du client lourd et léger devra minimalement être de 60 et 30 images par seconde respectivement. En ce qui concerne la performance du côté réseau, celui-ci devra avoir une capacité minimale de 10 Mb/s et un temps de latence maximale de 100 ms afin d'assurer une fluidité lors des sessions de dessins collaboratifs. Du côté de la base de données, le temps de réponse des requêtes vers celle-ci devra au plus être de deux secondes. Finalement, le serveur devra être en mesure de supporter jusqu'à 16 sessions de collaboration simultanées, dans laquelle chaque session peut accueillir au plus 4 collaborateurs.