Poly Paint Pro

Document d'architecture logicielle

Version 1.07

Historique des révisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Auteur** |
| 2019-01-29 | 1.00 | Cas d’utilisation primaires + clavardage. Diagramme de déploiement | Maxime Gosselin |
| 2019-01-30 | 1.01 | Cas d’utilisation Création + Protection + Édition. Les 3 diagrammes de processus, première ébauche. | Maxime Gosselin |
| 2019-02-05 | 1.02 | Diagramme de paquetage et de classe client léger. | Maxime Gosselin, Mathieu Giroux-Huppé |
| 2019-02-05 | 1.03 | Diagramme de paquetage client lourd | Maxime Gosselin, Antoine D-D |
| 2019-02-05 | 1.04 | Diagramme de paquetage du serveur | Maxime Gosselin, Jérémie Huppé |
| 2019-02-07 | 1.05 | Révision des sections 1 à 7 et ajouts de commentaires | Geneviève Bock, Antoine D-D, Jérémie Huppé |
| 2019-02-07 | 1.06 | Diagramme de classe client lourd et du serveur + Taille et performance. | Maxime Gosselin |
| 2019-02-08 | 1.07 | Révision de la section 7 | Geneviève Bock |

Table des matières

[**1. Introduction**](#_gjdgxs) **4**

[**2. Objectifs et contraintes architecturaux**](#_30j0zll) **4**

[**3. Vue des cas d’utilisation**](#_3znysh7) **4**

[**4. Vue logique**](#_2et92p0) **4**

[**5. Vue des processus**](#_tyjcwt) **4**

[**6. Vue de déploiement**](#_3dy6vkm) **4**

[**7. Taille et performance**](#_1t3h5sf) **4**

Document d'architecture logicielle

# 1. Introduction

Ce document a pour but d’expliciter l’architecture choisie pour le Poly Paint Pro. L’architecture sera exposée à l’aide de diagrammes et de textes.

Le présent document est divisé en 7 sections. La section 2 décrit les contraintes pouvant affecter l’architecture de Poly Paint Pro. La section 3 illustre plusieurs cas d’utilisation. Par la suite, les diagrammes de classes et de paquetages seront présentés à la section 4. Dans la section 5, trois processus seront détaillés mettant en lumière différentes parties du système. La section 6 a pour but de présenter une vue de très haut niveau des interactions entre les composantes matérielles du système. Finalement, la section 7, aborde les problèmes par rapport à la taille de l’application et les performances réduites qui pourraient en découler.

# 2. Objectifs et contraintes architecturaux

Dans cette section, les facteurs externes qui ont influencés l’architecture du système seront présentés. Il y a notamment les facteurs de portabilité, l'échéancier, les outils de développement ainsi que les langages de programmation.

Le facteur portabilité est très important, puisque nous développerons un client léger et un client lourd. Il est alors essentielle de garder en tête que ces deux plateformes sont différentes et ils doivent communiquer ensemble. De plus, nous travaillerons avec beaucoup de matériels différents, alors il est impératif que les applications soient portables.

L’échéancier est aussi un facteur important. En effet, nous avons un temps fixe pour développer le système, sans possibilité de retarder la livraison du produit final. Nous avons donc dû faire des choix architecturaux qui accélèrent le processus de développement.

Finalement, les outils de développement et les langages de programmation ont aussi contribués à certaine décisions. Le client léger sera développé sous Android Studio 3.3 à l’aide du langage de programmation Kotlin, le client lourd sous Visual Studio avec le langage de programmation C#.

# 

# 3. Vue des cas d’utilisation

|  |
| --- |
| Diagramme de cas d’utilisation - primaire |
| *Figure 1 :* Diagramme de cas d’utilisation primaire |

|  |
| --- |
| Diagramme de cas d’utilisation - Clavardage |
| *Figure 2 :* Diagramme de cas d’utilisation - Clavardage |

# 

|  |
| --- |
| Diagramme de cas d’utilisation - Création d’une image |
| *Figure 3 :* Diagramme de cas d’utilisation - Création d’une image |

# 

|  |
| --- |
| Diagramme de cas d’utilisation - Protection d’une image |
| *Figure 4 :* Diagramme de cas d’utilisation - Protection d’une image |

# 

|  |
| --- |
| Diagramme de cas d’utilisation - Édition d’une image |
| *Figure 4 :* Diagramme de cas d’utilisation - Édition d’une image |

# 

# 4. Vue logique

## 4.1 Client Lourd

|  |
| --- |
| **Relation entre les paquetages** |
| Relation haut niveau entre les différents paquetages.    *Figure 5 :* Diagramme de paquetage - client lourd |

# 

|  |
| --- |
| **View** |
| Contient toutes les vues. Une vue est un composant contenant tous les éléments graphiques    *Figure 6 :* Diagramme de classe - client lourd - paquetage View |

|  |
| --- |
| **ViewModel** |
| Contient toutes les ViewModel. Un ViewModel est un composant permettant de faire le lien entre une vue et un modèle. Il est responsable de la communication entre ceux-ci.    *Figure 7 :* Diagramme de classe - client lourd - paquetage ViewModel |

|  |
| --- |
| **Model** |
| Contient tous les modèles. Un modèle est une représentation d’un objet personnalisé. Ces objets ont des propriétés et des méthodes spécifiques.    *Figure 8 :* Diagramme de classe - client lourd - paquetage Model |

|  |
| --- |
| **Service** |
| Contient tous les services. Un service est en charge de la communication entre un ViewModel et le serveur.    *Figure 9 :* Diagramme de classe - client lourd - paquetage ViewModel |

## 

## 

## 4.2 Client Léger

|  |
| --- |
| **Relation entre les paquetages** |
| Relation haut niveau entre les différents paquetages.    *Figure 10 :* Diagramme de paquetage - client léger |

|  |
| --- |
| **Activity** |
| Contient toutes les activités. Une activité est une vue haut niveau pouvant contenir ou non des fragments.    *Figure 11 :* Diagramme de classe - client léger - paquetage Activity |

|  |
| --- |
| **Fragment** |
| Contient tous les fragments. Un fragment est un composant qui sera intégré dans une activité.    *Figure 12 :* Diagramme de classe - client léger - paquetage Fragment |

|  |
| --- |
| **Adapter** |
| Contient tous les adaptateurs. Un adaptateur est introduit dans un fragment ou une activité afin de mieux gérer une liste de composants présents dans le paquetage Layout.    *Figure 13 :* Diagramme de classe - client léger - paquetage Adapter |

|  |
| --- |
| **Holder** |
| Contient tous les « Holder » du projet. Un holder est un objet contenant de l’information réutilisable dans plusieurs activités, fragments et adaptateurs.    *Figure 14 :* Diagramme de classe - client léger - paquetage Holder |

|  |
| --- |
| **Application** |
| Contient la classe servant de point d’entrée dans l’application.    *Figure 15 :* Diagramme de classe - client léger - paquetage Application |

## 

|  |
| --- |
| **Model** |
| Contient tous les objets complexes spécifiques au projet.    *Figure 16 :* Diagramme de classe - client léger - paquetage Model |

## 

## 

## 4.3 Serveur

|  |
| --- |
| **Relation entre les paquetages** |
| Relation haut niveau entre les différents paquetages.    *Figure 17 :* Diagramme de paquetage - serveur |

# 

|  |
| --- |
| **Authentification** |
| Ce paquetage regroupe les fonctions de Socket.IO et tous les objets nécessaires au processus d’authentification d’un usager.    *Figure 18 :* Diagramme de classe - serveur - paquetage Authentification |

|  |
| --- |
| **Chat** |
| Ce paquetage regroupe les fonctions de Socket.IO et tous les objets nécessaires au processus de chat de l’application.    *Figure 19 :* Diagramme de classe - serveur - paquetage Chat |

|  |
| --- |
| **DrawingElement** |
| Ce paquetage regroupe les fonctions de Socket.IO et tous les objets nécessaires à la création d’un élément de dessin.    *Figure 20 :* Diagramme de classe - serveur - paquetage DrawingElement |

|  |
| --- |
| **DrawingTransformation** |
| Ce paquetage regroupe les fonctions de Socket.IO et tous les objets nécessaires à l’application d’une transformation sur un élément de dessin.    *Figure 21 :* Diagramme de classe - serveur - paquetage DrawingTransformation |

|  |
| --- |
| **DrawingManagement** |
| Ce paquetage regroupe les fonctions de Socket.IO et tous les objets nécessaires à l’administration d’une image et la communication avec la base de données.    *Figure 22 :* Diagramme de classe - serveur - paquetage DrawingManagement |

|  |
| --- |
| **Drawing** |
| Ce paquetage regroupe les fonctions de Socket.IO et tous les objets nécessaires à la représentation d’un dessin individuel.    *Figure 23 :* Diagramme de classe - serveur - paquetage Drawing |

# 

# 

# 5. Vue des processus

|  |
| --- |
| **Diagramme de processus - Insertion d’une forme en réseau** |
| *Figure 24 :* Diagramme de processus - Insertion d’une forme en réseau |

# 

# 

|  |
| --- |
| **Diagramme de processus - Création d’une nouvelle image publique** |
| *Figure 25 :* Diagramme de processus - Création d’une nouvelle image publique |

# 

# 

|  |
| --- |
| **Diagramme de processus - Envoyer un message dans un canal de discussion** |
| *Figure 26 :* Diagramme de processus - Envoyer un message dans un canal de discussion |

# 

# 6. Vue de déploiement

|  |
| --- |
| Diagramme de Déploiement |
| Figure 27 : Diagramme de déploiement |

# 

# 7. Taille et performance

Plusieurs contraintes liées à la taille et la performance de l’application ont influencé l’architecture de la solution proposée.

Premièrement, la fréquence élevée d’envoi de données par chaque client au serveur a dû être considéré lors de la construction de l’architecture logicielle. La façon la plus simple de synchroniser les dessins des clients entre eux serait d’envoyer, à chaque modification effectuée par un client, tout le contenu du dessin à tous les autres clients. Cela signifie que le client ayant effectué une modification doit sérialiser le dessin en JSON, puis envoyer le JSON en entier. Ensuite, les autres clients doivent désérialiser le dessin et le reconstruire localement, ce qui nécessite plus de traitement. Plus le nombre d’éléments dans le dessin est grand, plus il faut de traitement. Plus il y a de traitement, plus la performance de l’application serait diminuée. Somme toute, ce processus aurait été très lourd au niveau du réseau et niveau de la performance de l’application. Puisque nous visons un taux de rafraîchissement minimal de 24 images par seconde pour le client léger et de 30 par images par seconde pour le client lourd, nous avons opté pour l’envoi de la modification apportée au dessin au serveur. De cette façon, le traitement sera moindre, ce qui garantit une performance accrue des applications.

Deuxièmement, nous avons porté une attention particulière à garder la taille des applications aussi petite que possible. Ceci se traduit dans la façon dont nous gérons le stockage d’informations. La quasi-totalité des informations nécessaires au fonctionnement des applications seront stockées dans une base de données distante et commune aux deux clients. Le serveur distant connecté à une base de données permet donc d’éviter l’enregistrement de l’information sur les clients. Les seules données qui devront être enregistrées sur les clients sont des données que l’usager désire enregistrer volontairement. Il aura donc le plein contrôle de la taille des données enregistré son sa machine. Il est important à noter que cette fonctionnalité sera disponible que sur le client lourd. .

Troisièmement, nous avons décidé de représenter un diagramme par l’ensemble des éléments qui le compose; c’est ainsi que nous l’enregistrons. Ceci nous évite la complexité de la manipulation d’un diagramme représenté par une image, sous forme de pixels. Plus précisément, nous allons seulement enregistrer le JSON regroupant l’ensemble des éléments permettant de reconstruire l’image. Par ce moyen, nous nous assurons de la compatibilité avec nos clients en plus de limiter les opérations de traitement entre la sauvegarde et l’affichage d’un diagramme.

Finalement, un mécanisme de sauvegarde automatique sur le serveur est prévu. Afin de ne pas surcharger la base de données et le serveur, les sauvegardes automatiques se produiront aux 30 secondes.

En conclusion, nous avons fait des choix d’architecture judicieux qui tiennent compte de la performance de l’application, de la taille des clients ainsi que de la taille des communications entre les clients et le serveur.