Équipe 12

Poly Paint
Document d'architecture logicielle

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
2019-02-01	1.0	Rédaction initiale	Chelsy Binet, Olivier Lauzon, Alexis Loiselle, Sébastien Cadorette, Sébastien Labine et William Sévigny

Table des matières

Document d'architecture logicielle

1. Introduction

Le document d'architecture logicielle décrit le la structure de haut niveau d'une application. Des précisions sur les objectifs et contraintes ainsi que leur impact architectural sont présentés. La structure de l'application est décrite selon quatre points de vue: la vue des cas d'utilisation, la vue logique, la vue des processus et la vue de déploiement. Enfin, les caractéristiques de taille et de performance de l'application qui pourraient affecter l'architecture sont exposées.

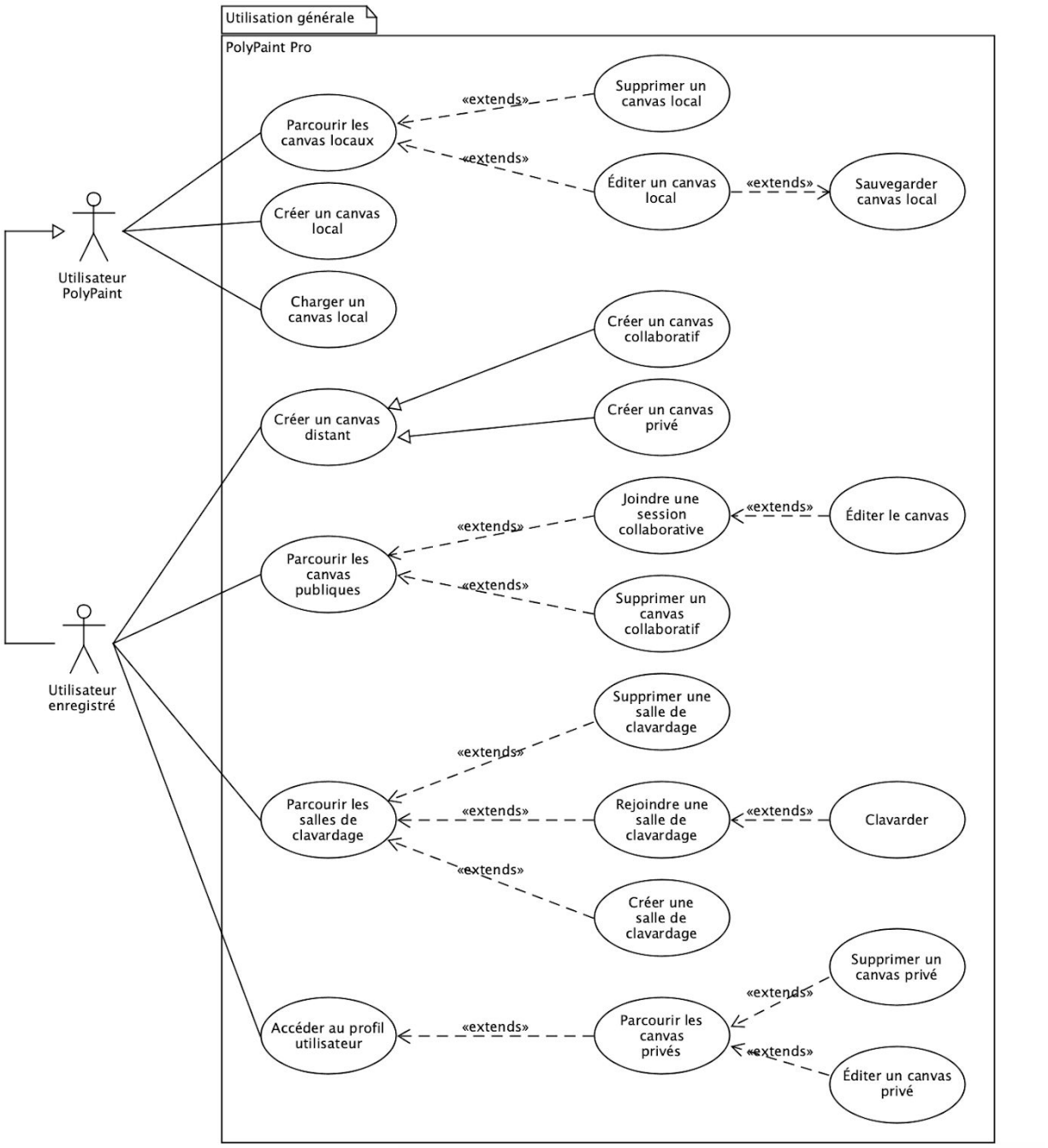
2. Objectifs et contraintes architecturaux

La plateforme PolyPaint pro doit avoir deux clients: le premier dans un environnement Windows et le second dans un environnement iOS. Le serveur doit être réutilisable et multiplateforme pour faciliter la mise en marché. L'architecture des comptes utilisateurs doit être sécure avec encryption pour assurer la sécurité des données personnelles des utilisateurs. Le code doit être simple et facilement maintenable pour respecter le budget et l'échéancier. L'architecture de la plateforme doit être évolutive pour permettre l'ajout de nouvelles fonctionnalités dans l'avenir et doit maximiser la réutilisabilité de code libre de droits pour accélérer le développement.

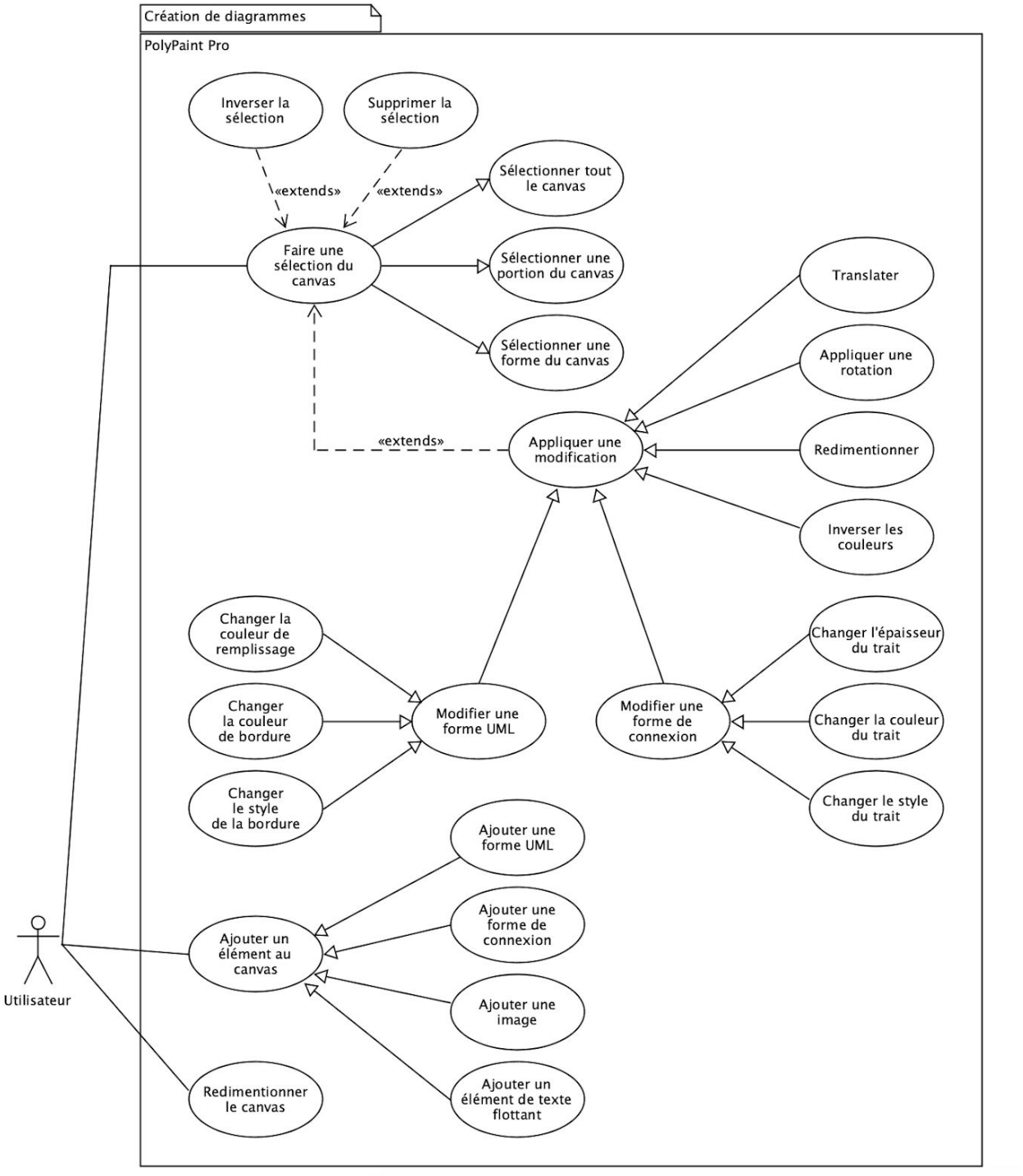
3. Vue des cas d'utilisation

Afin de bien décrire l'utilisation de la plateforme PolyPaint Pro, nous avons identifié les principaux cas d'utilisation.

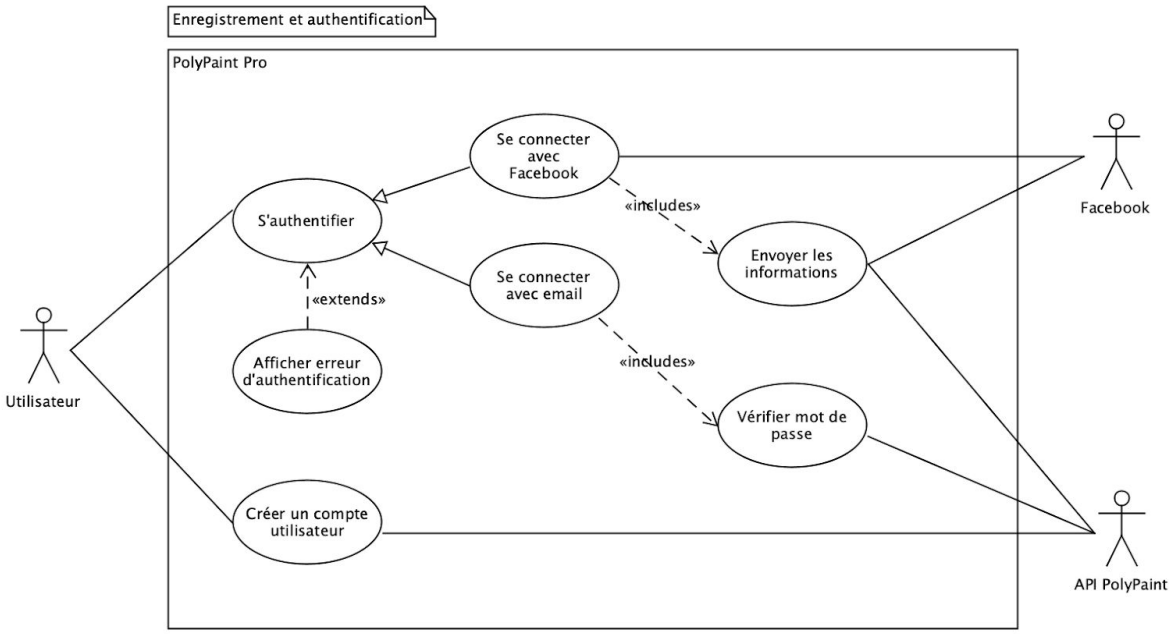
Le premier cas d'utilisation l'utilisation générale de l'application.



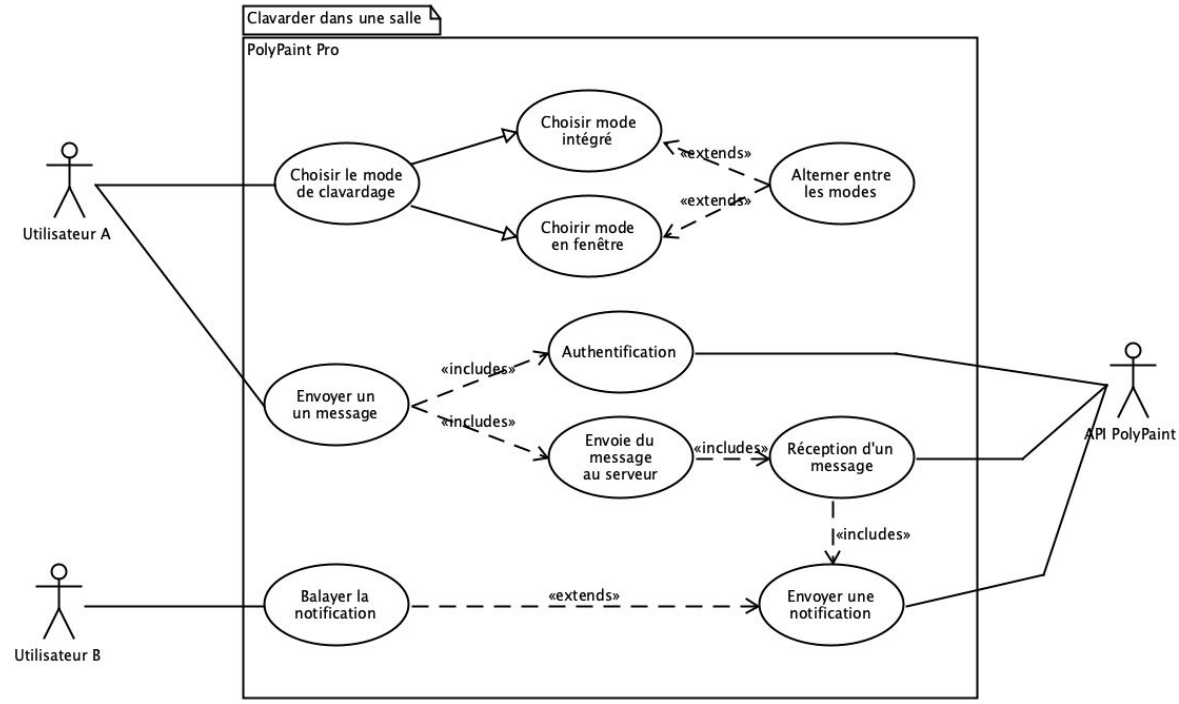
Le second cas d'utilisation est la création de diagrammes avec l'éditeur PolyPaint Pro.



Le troisième cas d'utilisation est l'enregistrement et la connexion à la plateforme.



Le dernier cas d'utilisation est clavarder avec d'autres utilisateurs.

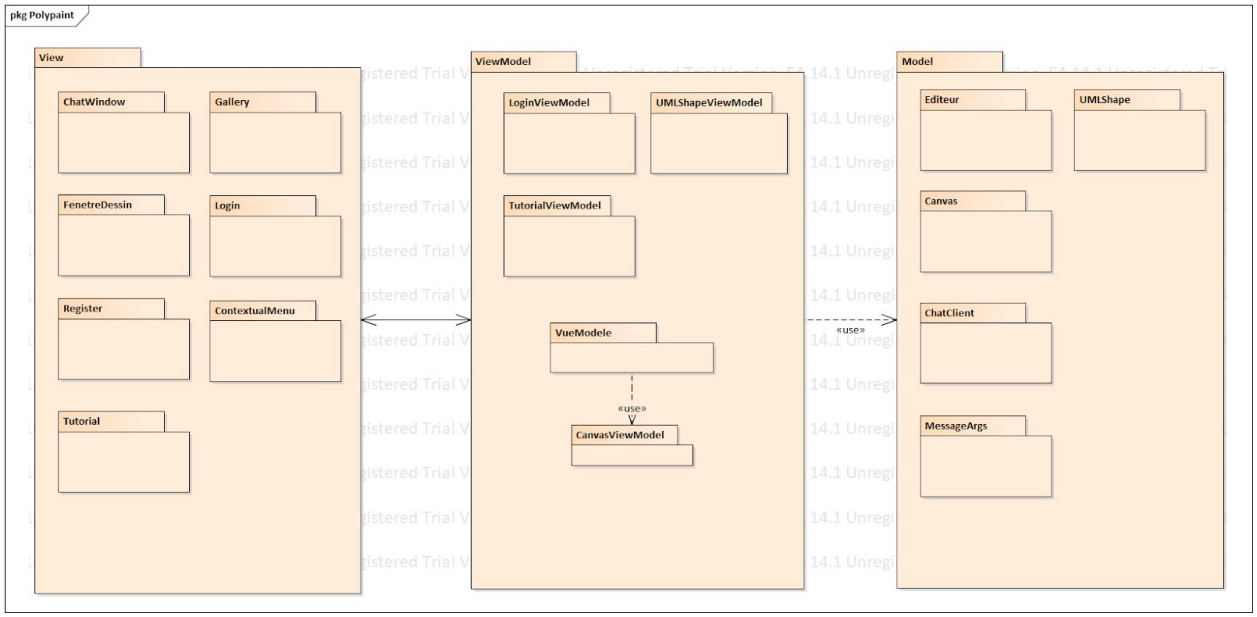


4. Vue logique

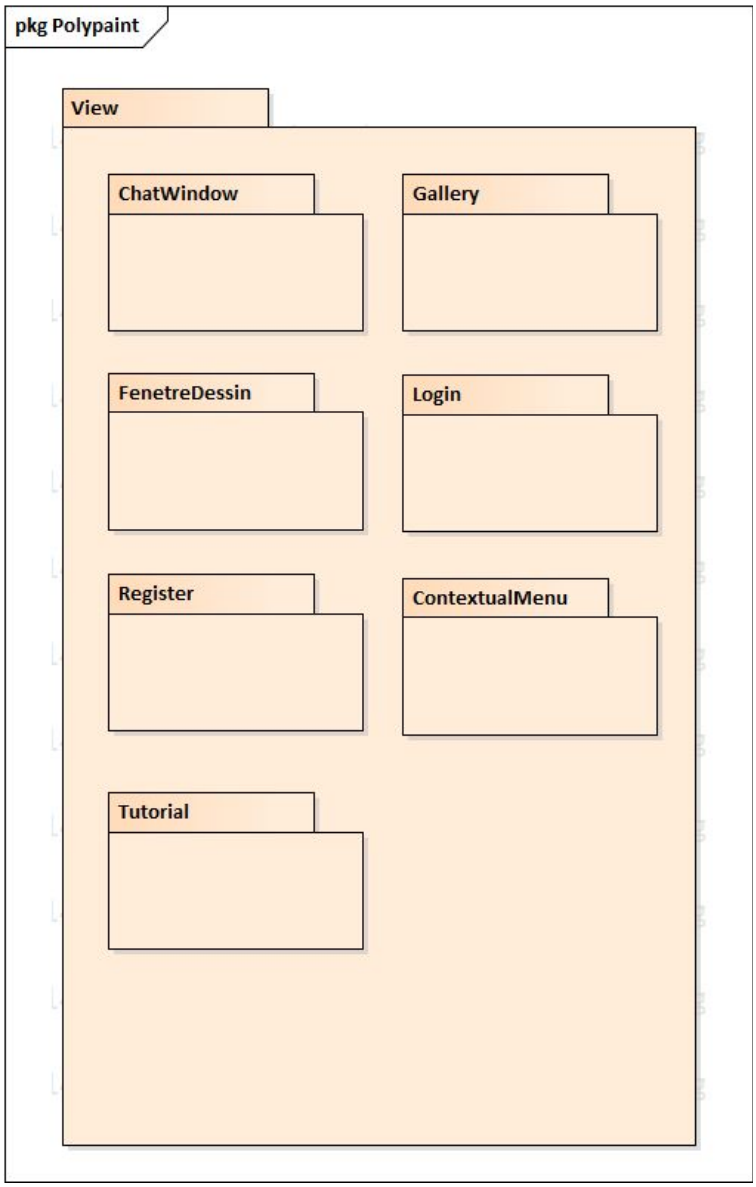
[Cette section présente les parties architecturalement significatives du modèle de design. Pour chaque paquetage, remplir le tableau suivant et présenter un diagramme de paquetages (ou diagramme de classes, selon le contexte).]

Afin de présenter la vue logique de la partie WPF, nous avons décomposé PolyPaint en trois paquetages principaux, qui contiennent d'autres paquetages. Les trois principaux suivent le modèle architectural du projet, soit MVVM (Modèle, Vue et Vue-Modèle).

Sur l'image ci-dessous, on retrouve, dans l'ordre, View, ViewModel et Model. Chaque paquetage principal est décrit individuellement plus bas, avec des images individuelles afin de mieux voir. L'image suivante sert seulement d'idée générale et qui montre les liens entre chaque paquetage principal.

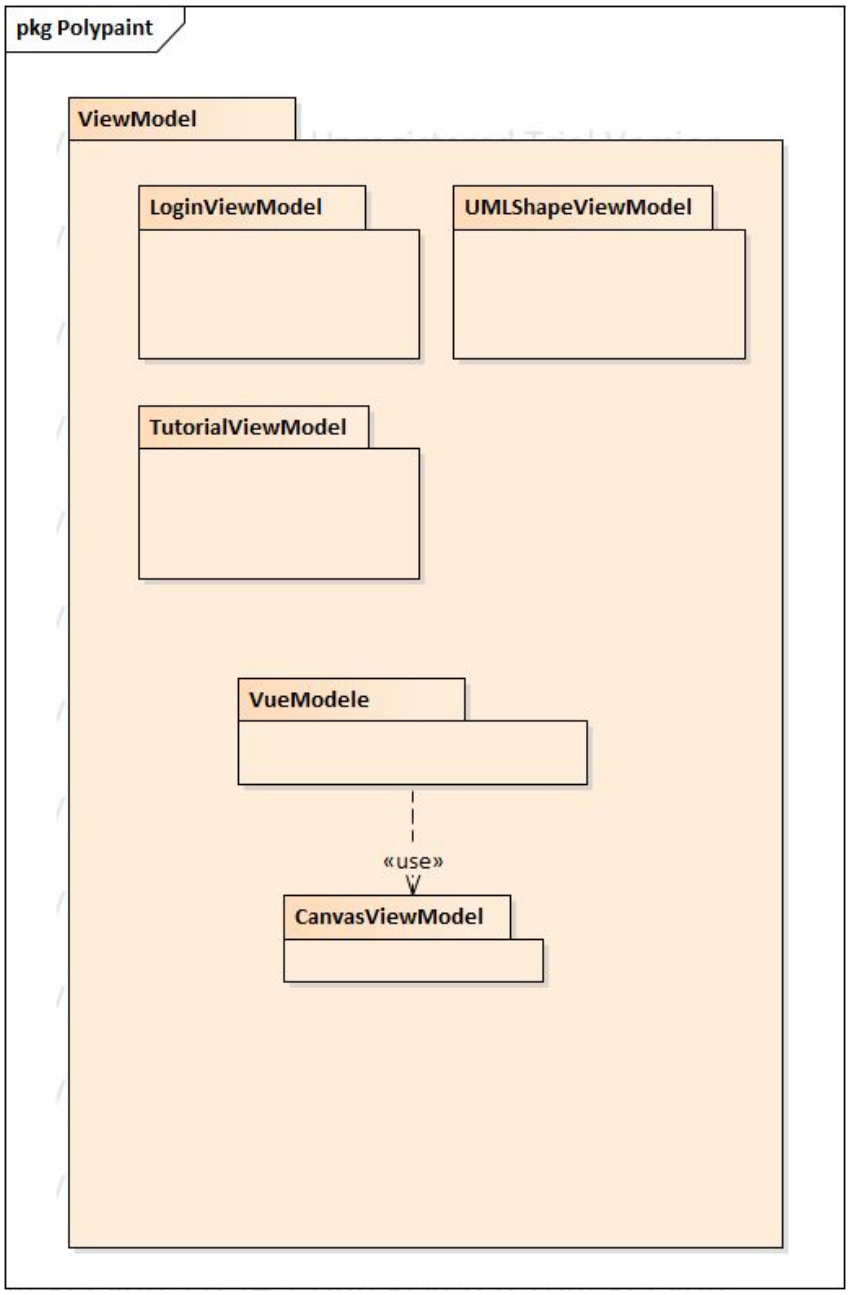


Le paquetage View regroupe tous les paquetages de vue pour le projet, c'est donc un regroupement de tout ce qui doit être affiché dans le projet.



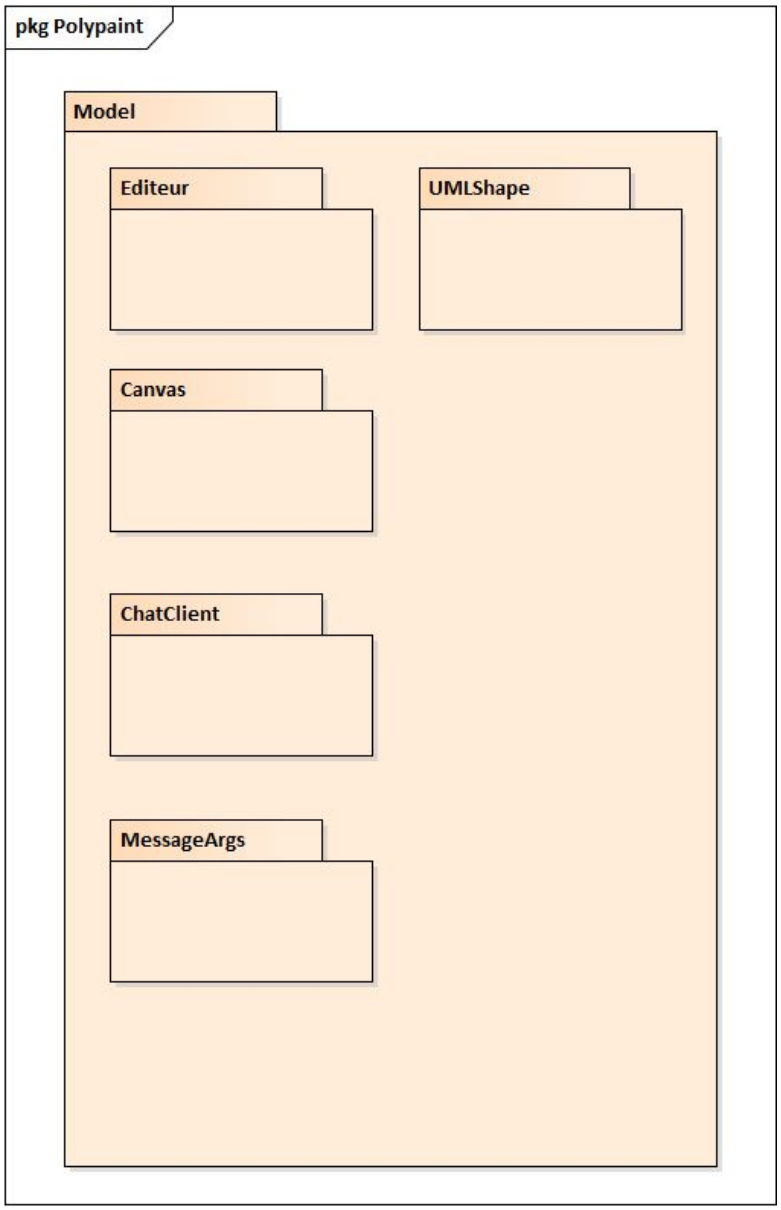
View
ChatWindow
La vue du ChatWindow est la représentation de la fenêtre de clavardage en mode fenêtré.
FenetreDessin
La classe FenetreDessin est la classe responsable de l'affichage de la fenêtre principale, qui contient le canevas et les outils de dessin. De plus, elle affiche la fenêtre de clavardage en mode intégré.
Register
Cette classe représente la vue de la fenêtre pour enregistrer un compte sur la base de données.
Login
Cette classe représente la vue de la fenêtre permettant de se connecter avec un compte utilisateur.
Gallery
Cette classe représente la vue de la galerie d'image qui est hébergée localement ou sur le serveur distant.
ContextualMenu
ContextualMenu représente la vue du menu contextuel qui est au haut de la fenêtre de dessin principale.
Tutorial
La vue du Tutorial représente la fenêtre qui permet d'afficher la séquence d'images formant le tutoriel.

Le second paquetage principal est le ViewModel. Ce dernier regroupe tous les paquetages qui servent à faire le lien entre les données des modèles, et la vue.



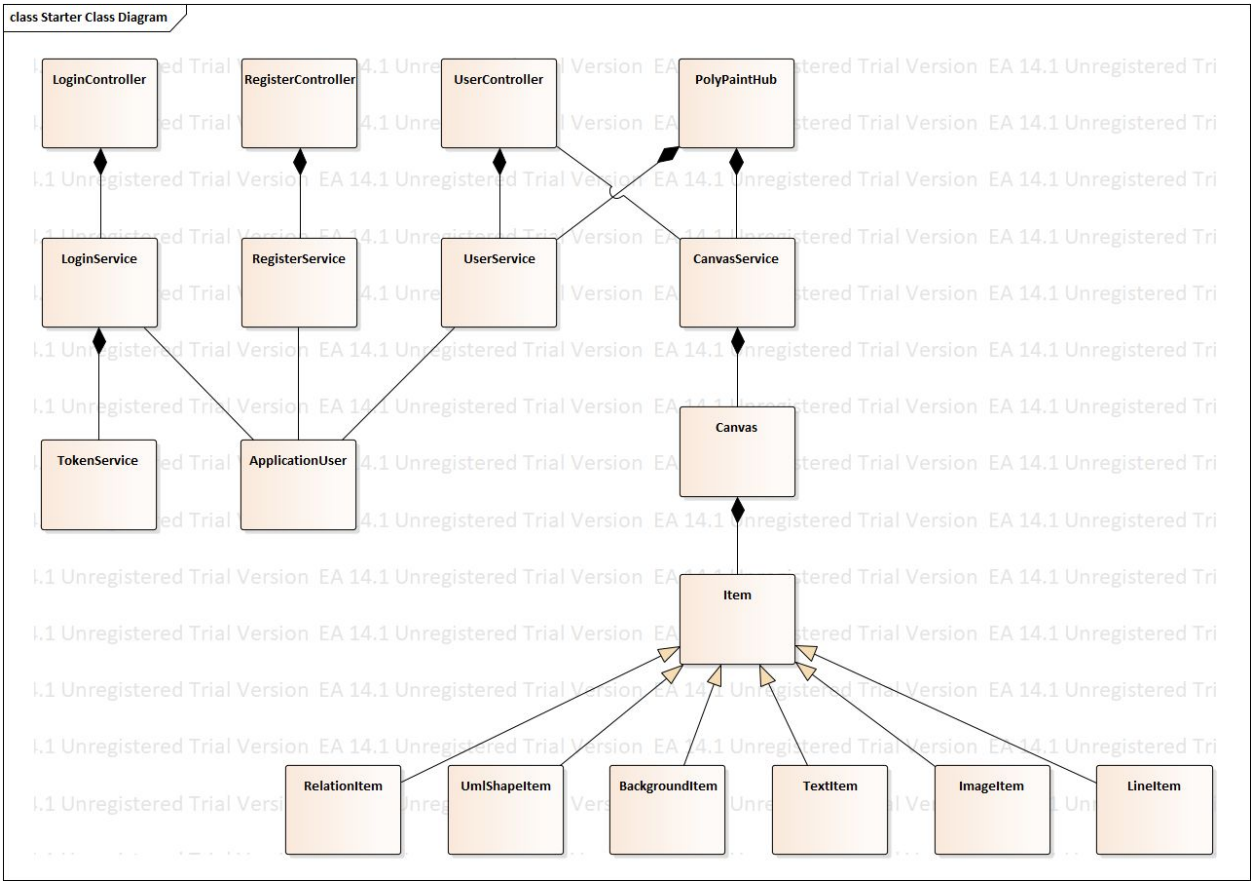
ViewModel
LoginViewModel
Cette classe sert à utiliser l'information qui permet de se connecter.
VueModele
Cette classe contrôle l'information de la fenêtre principale, c'est-à-dire la fenêtre de dessin.
CanvasViewModel
Cette classe utilise l'information qu'elle récupère du modèle du canevas afin de faire différents affichages par rapport à ceux-ci.
UMLShapeViewModel
Ce modèle-vue sert à prendre l'information fournie par l'utilisateur pour la création de formes UML et la traiter pour l'afficher correctement sur le canevas.
TutorialViewModel
Ce modèle-vue sert à gérer l'utilisation du tutoriel par l'utilisateur et de savoir si l'utilisateur a déjà suivi le tutoriel ou non.

Le troisième paquetage principal est celui de Model. Ce paquetage regroupe toutes les classes qui nécessitent nt un Model afin de transmettre de l'information provenant d'un API.



Model
Editeur
Cette classe permet de retenir de l'information par rapport à la fenêtre de dessin principale.
Canvas
Le Canvas sert à retenir l'information de chaque canevas.
ChatClient
Sert à initialiser et retenir de l'information de connexion de chaque client.
MessageArgs
Cette classe permet de retenir les arguments des messages de clavardage (tels que le message, le nom d'utilisateur et le timestamp).
UMLShape
Cette classe sert à retenir de l'information pour créer des formes UML.

Le diagramme de classes suivant représente la vue logique du serveur.



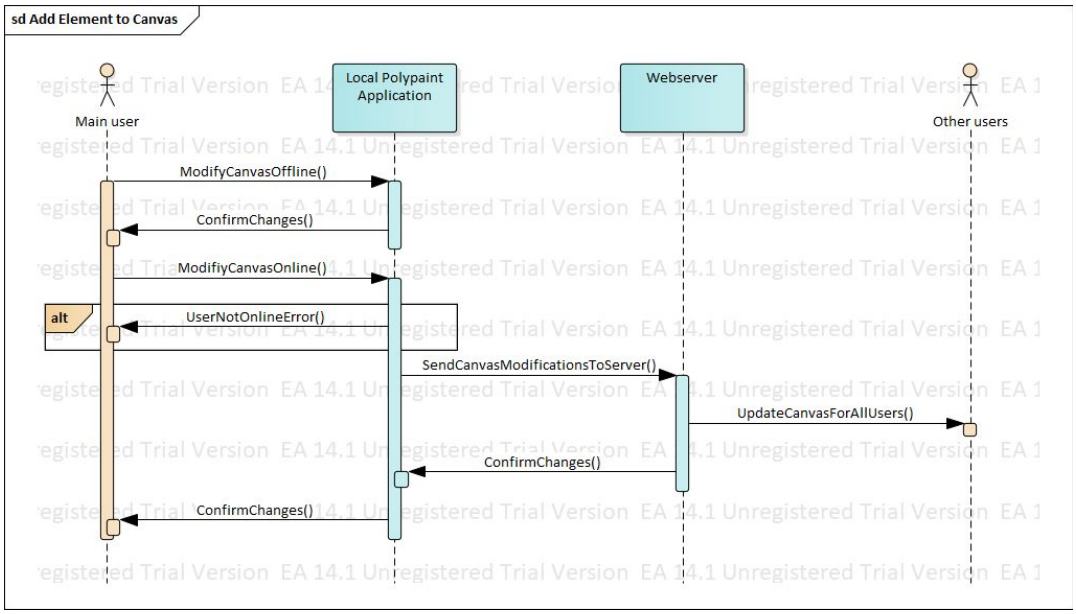
LoginController
Cette classe sert à faire les requêtes HTTP liées au service de connexion d'utilisateurs.
LoginService
Cette classe est un service permettant de gérer la connexion d'un utilisateur.
TokenService
Cette classe permet de gérer la génération de jetons pour les utilisateurs.
RegisterController
Cette classe sert à faire les requêtes HTTP liées au service de création de nouveaux comptes usagers.
RegisterService
Cette classe est un service qui gère la création de nouveaux comptes usagers.
UserController
Cette classe sert à faire les requêtes HTTP liées au service qui gère l'information des utilisateurs.
UserService
Cette classe est un service qui permet de gérer toute l'information liée à un utilisateur.
ApplicationUser
Classe qui sert à représenter un utilisateur du côté serveur.
PolyPaintHub
Classe qui sert à gérer les requêtes de SignalR.
CanvasService
Cette classe représente le service qui gère les canevas sur le serveur.
Canvas
Représente un canevas. Un canevas est représenté par une liste d'Item.
Item
Item est un classe abstraite qui représente un objet quelconque présent dans un canvas.
RelationItem
Classe héritant de Item représentant un objet de relation UML.
UmlShapeItem
Classe héritant de Item représentant un objet de forme UML.
BackgroundItem
Classe héritant de Item représentant le fond d'écran du canevas.

TextItem
Classe héritant de Item représentant un élément de texte sur le canevas.
ImageItem
Classe héritant de Item représentant une image qu'on ajoute au canevas.
LineItem
Classe héritant de Item représentant une ligne qu'on ajoute au canevas.

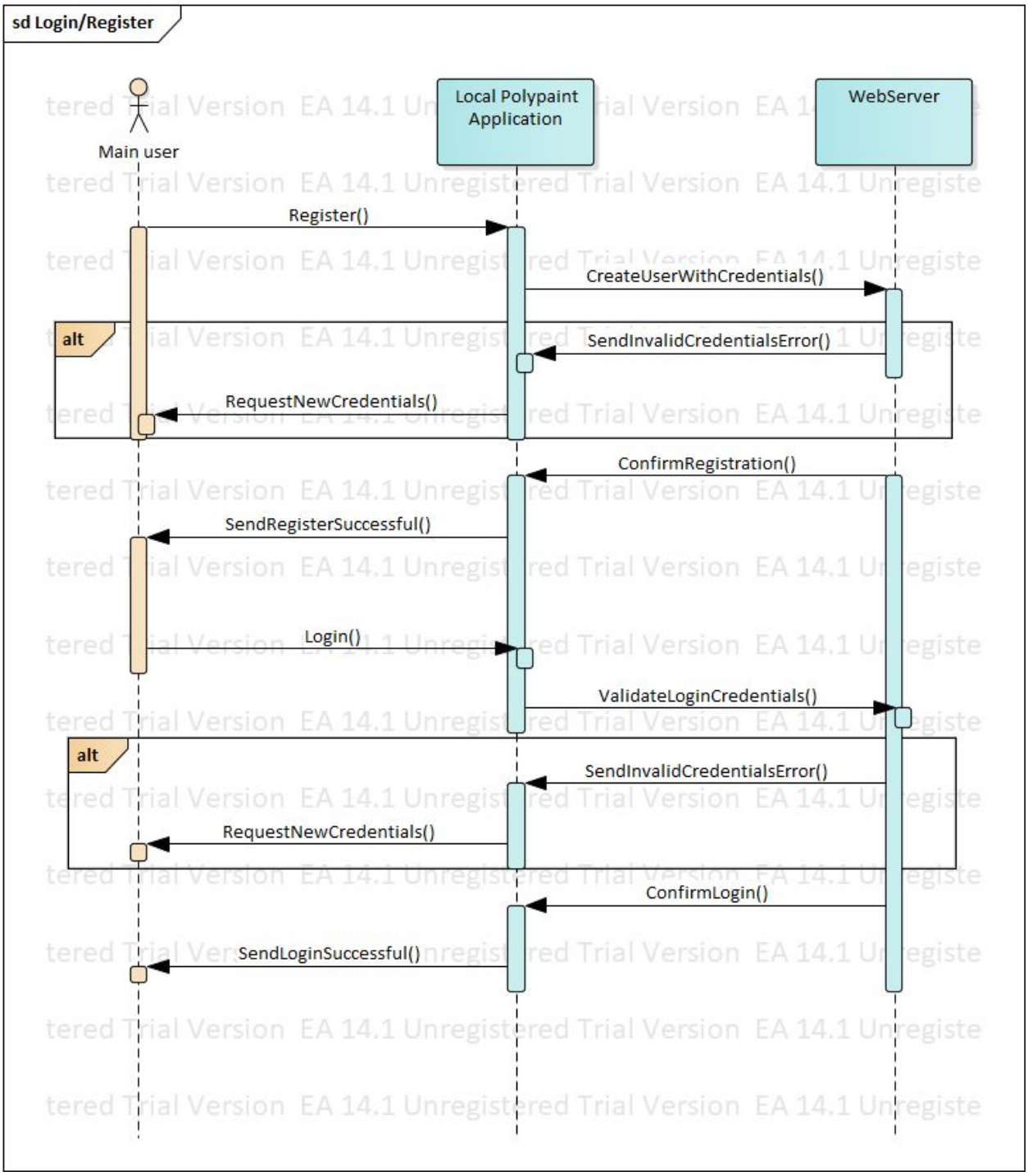
5. Vue des processus

Afin de décrire le système en termes d'interactions entre les différents processus significations, nous avons déterminé les tâches principales, et avons construit un diagramme de séquence pour chacune de ses tâches. Sur ces diagrammes, nous retrouvons les interactions entre un utilisateur quelconque, son application locale, le serveur distant ainsi que les autres utilisateurs en ligne.

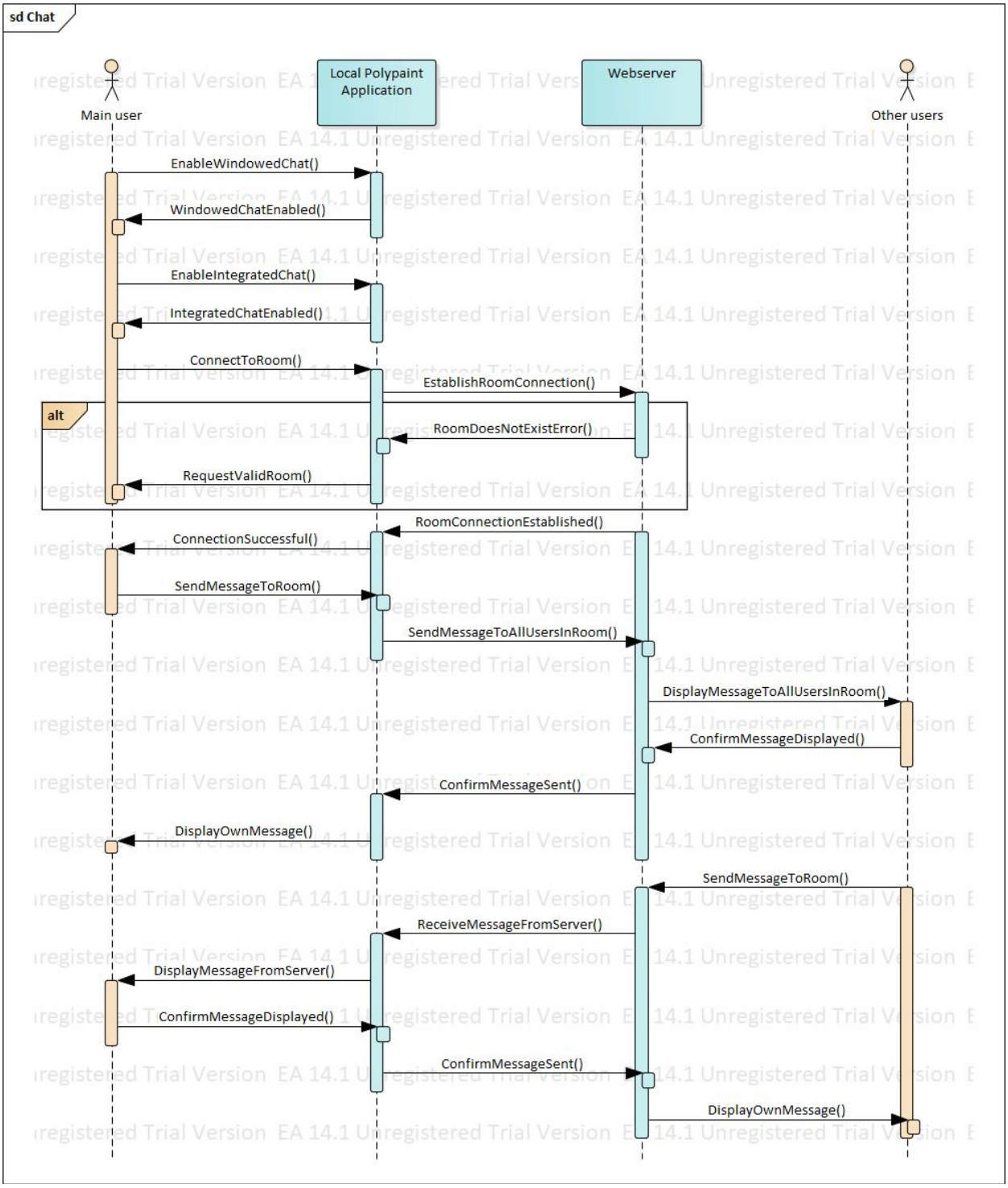
Le premier processus représenter par un diagramme de séquence est l'ajout d'un élément sur le canvas (en ligne ou hors-ligne).



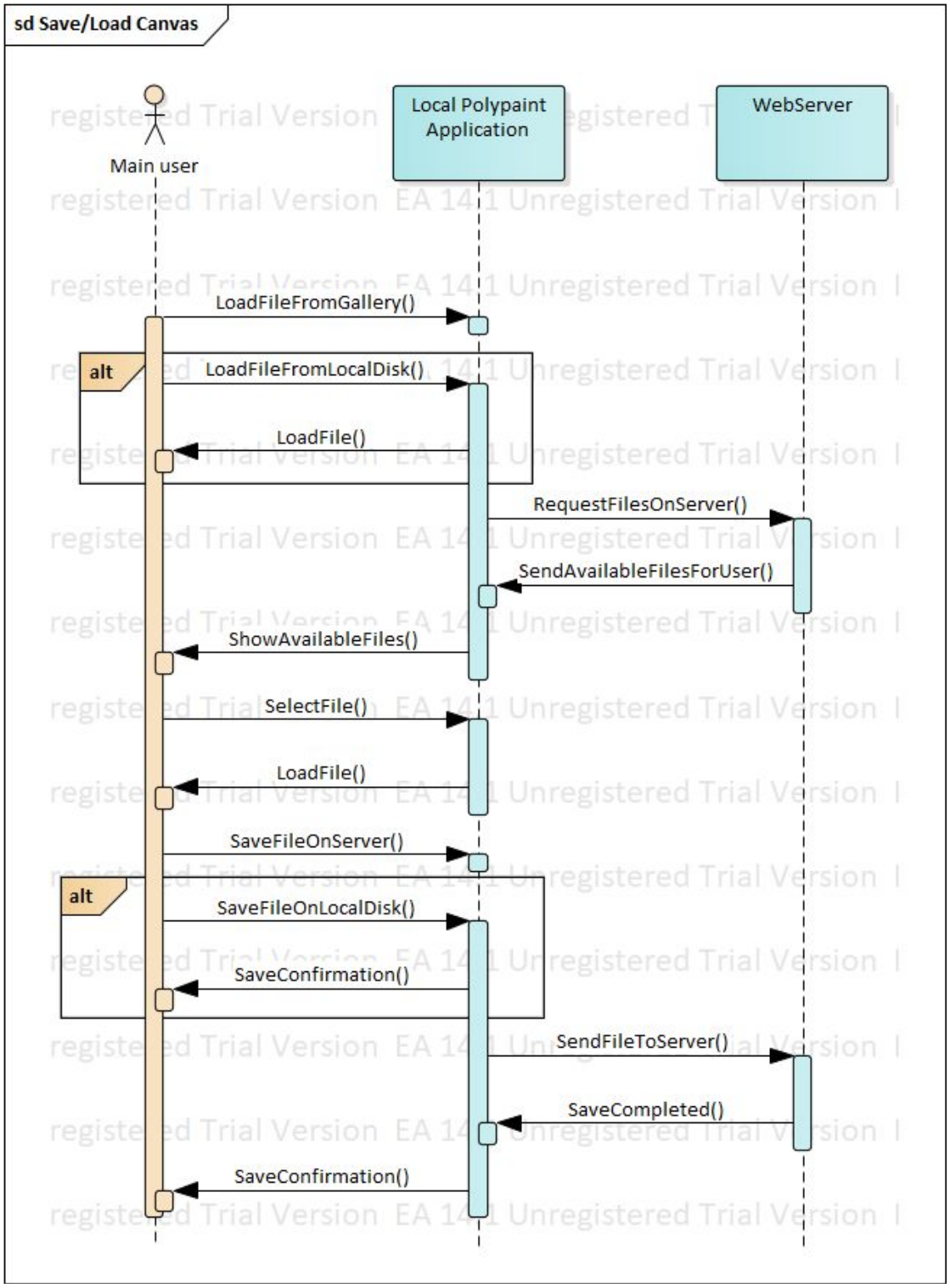
Le second processus important est l'authentification sur le serveur et la connexion grâce à ce dernier.



Le troisième processus que nous avons jugé pertinent à présenter est le processus de clavardage.

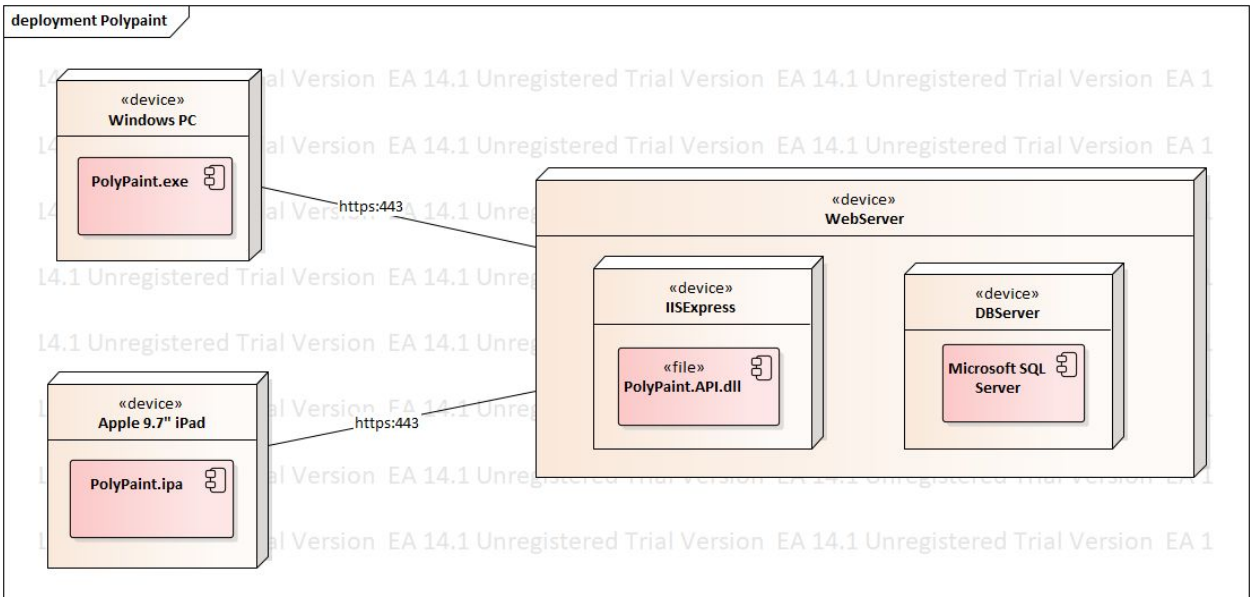


Le quatrième processus important est la sauvegarde et le chargement de fichier (de manière locale et distante).



6. Vue de déploiement

Cette section décrit une configuration de matériel physique complète pour le déploiement de Poly Paint.



7. Taille et performance

Tout d'abord, la performance doit respecter les critères énoncés dans le SRS. Ces critères sont que le délai d'authentification doit être inférieur à 2 secondes. De plus, les messages échangés dans la messagerie instantanée doivent avoir un délai inférieur à 1 seconde, alors que le temps réel des dessins devrait avoir un délai inférieur à 2 secondes. La dernière exigence de performance pour le SRS est que l'ouverture d'un dessin lors du chargement devrait être inférieur à 5 secondes. Ces exigences mises toutes à minimiser les délais afin de réduire la latence entre les utilisateurs, et de rendre l'expérience utilisateur plus agréable de manière globale.

Pour ce qui est des ressources matérielles, le client lourd doit minimiser celles-ci afin que les tâches en arrière-plan sur l'appareil soient en mesure de continuer leur déroulement normal (on ne s'attend pas à ce que l'ordinateur s'exécute parfaitement lors de l'exécution de PolyPaint s'il y avait des difficultés avant le lancement de l'application). Ces conditions s'appliquent de manière encore plus importantes sur le client léger, puisque ce dernier possède moins de mémoire vive que la majorité des ordinateurs de nos jours. Même si le stockage n'est plus vraiment un problème de nos jours, on souhaite minimiser la taille de nos applications, puisqu'il s'agit d'une application relativement simple, on devrait avoir des fichiers de tailles très raisonnables.