Équipe 12

Poly Paint
Protocole de communication

Version 1.2

Historique des révisions

Date	Version	Description	Auteur
2019-01-30	1.0	Jet initial	Sébastien Labine, Alexis Loiselle, Olivier Lauzon, Sébastien Cadorette, Chelsy Binet, William Sévigny
2019-02-06	1.1	Continuation de la partie des paquets	Sébastien Labine, Alexis Loiselle
2019-02-08	1.2	Correction des fautes à l'aide d'antidote	Sébastien Labine

Table des matières

Protocole de communication

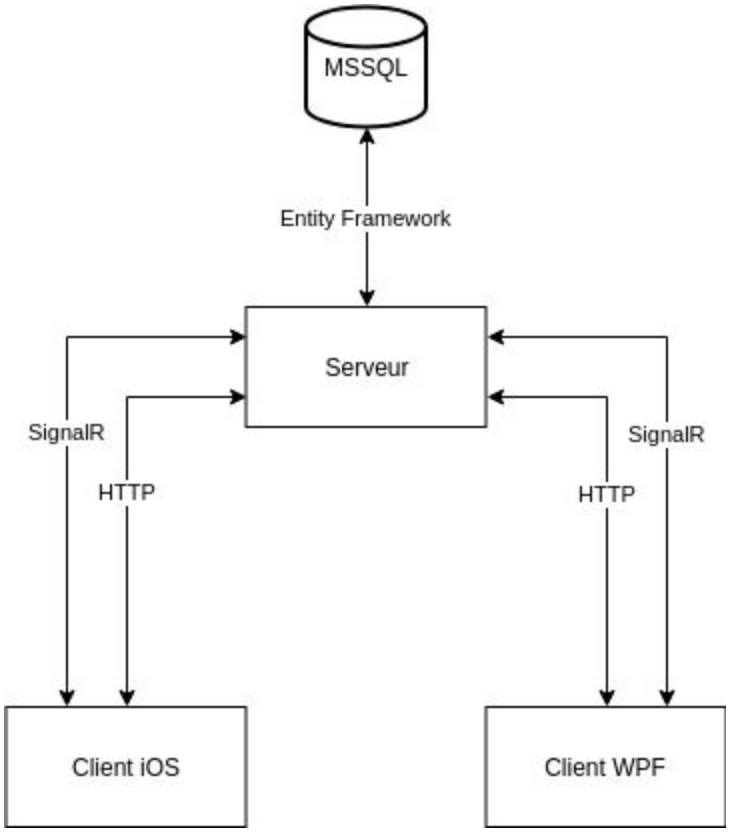
1. Introduction

Le document courant sert à décrire et à expliquer le fonctionnement des communications entre les différents éléments composantsle logiciel PolyPaint. La section 2 présente et décrit les technologies choisies pour la communication au sein du logiciel. La section 3 décrit en détailschaque parcelle de communication faite entre les clients et le serveur. La structure de chaque message est présentée profondément.

2. Communication client-serveur

2.1 Schéma

Dans l'application PolyPaint, on retrouve différentes façons de communiquer entre les composants du logiciel. Voici un schéma montrant l'architecture de communication :



2.2 Choix du framework serveur

La première chose à noter, c'est que nous n'avons pas réinventé la roue. C'est-à-dire que nous sommes partis de protocoles déjà existants. En fait, tout est parti de notre choix du serveur : Asp.Net Core. Nous avons choisi cette technologie étant donné que la plupart de l'équipe était familière préalablement au projet et que cette technologie est multi-plate-forme. Nous avons par la suite choisi les protocoles de communication en fonction de notre choix de Asp.Net Core.

2.3 Communication Serveur - Base de données

Tout d'abord, le framework Asp.Net Core détient une librairie du nom de Entity Framework qui permet de faire le lien entre le serveur et la base de données. EF permet de traduire la structure des modèles présents dans le serveur en tables SQL. EF permet aussi de traduire les requêtes du C# en requêtes SQL faites sous le capot. La connexion à la base de données est effectuée de façon locale puisque l'API et la base de données se retrouvent sur le même serveur. La connexion se fera via le protocole TCP/IP sur localhost au port 1432.

2.4 Communication Serveur - Clients

Ensuite, pour la communication entre les clients et le serveur, 2 protocoles sont utilisés: HTTPS et WebSocket. En ce qui concerne le protocole websocket, celui-ci est implémenté par le framework SignalR, que nous avons utilisé pour ce projet.

2.4.1 HTTPS

Nous avons choisi d'utiliser HTTPS pour les connexions et les gestions de fichiers étant donné que c'est la meilleure pratique pour ces cas précis. En effet, pour qu'une connexion soit effectuée, le client doit faire une requête au serveur qui lui renvoie un jeton en tant que réponse. C'est la manière la plus simple de gérer cette logique, donc, nous nous sommes penchés vers ce chemin. La même logique est utilisée pour la sauvegarde des fichiers. De plus, nous voulions faire une séparation entre la partie individuelle et la partie collaborative. Par exemple, si quelqu'un se connecte, il ne se connecte pas nécessairement au serveur de collaboration ou de chat. Ainsi, le serveur requiert une seconde connexion avec le protocole SignalR.

2.4.2 SignalR

Ceci nous amène au dernier protocole choisi : SignalR. Celui-ci fonctionne essentiellement de la même façon que les WebSockets, c'est-à-dire qu'il offre une connexion continue entre les clients et le serveur. À vrai dire, SignalR utilise les WebSockets sous le capot. C'est la direction que nous avons choisi e tant donné,e,encore une fois, que SignalR est plus supporté pour le framework Asp.Net Core. En effet, il offre tout ce que les WebSockets offrent, mais enlève la complexité d'implémentation des WebSockets. Ce type de communication est très utilisé dans l'industrie lorsqu'on veut communiquer en temps réel. Il offre la possibilité au serveur d'envoyer un message à plus qu'un client en même temps, ce qui est très utile pour un logiciel comme PolyPaint où la collaboration et la messagerie en temps réel sont des fonctionnalités requises. Le principal obstacle était le support de SignalR sur les clients, précisément iOS. Une fois avoir trouvé un client de SignalR supporté avec Swift, nous avons décidé de s'en aller dans cette direction.

3. Description des paquets

La section suivante expliquera précisément les deux protocoles décrits plus haut. Nous présenterons en détails le fonctionnement de chacun et exposerons les différentes fonctions disponibles dans les différents protocoles de communication.

3.1 Protocole HTTPS

Le protocole HTTPS est au coeur des applications modernes. En effet, c'est un des protocoles les plus utilisés dans le domaine lorsqu'une application doit communiquer avec une interface de communication distante. Dans le cas de PolyPaint Pro, nous allons avoir à communiquer avec une interface de communication hébergée sur les serveurs de Google, ce qui va nécessiter une communication par ce protocole.

3.1.1 Sécurité

Le protocole de communication HTTPS sera utilisé afin de s'assurer que tout échange d'information à travers le protocole HTTP soit encrypté. Afin de limiter l'accès à certaines ressources de notre interface de programmation, certains points d'accès de notre serveur devront être limités aux personnes autorisées seulement. Pour ce faire, nous allons utiliser les « Json Web Token », une technologie permettant de façon simple et sécuritaire d'identifier un utilisateur. Les clients devront inclure l'entête suivante dans leurs requêtes HTTP: "Authorization": "Bearer {token}". Le jeton est obtenu lorsque l'utilisateur s'authentifie avec succès à notre serveur (voir 3.1.1.2).

3.1.2 Cloudflare

Chacune des requêtes faite à travers le protocole HTTP ou HTTPS sera interceptée et gérée par l'intermédiaire Cloudflare avant d'être acheminée à notre serveur. Cet intermédiaire aura pour but premier de sécuriser notre application contre les différentes attaques Internet telles que les attaques par déni de service. Aussi, Cloudflare permettra automatiquement de rediriger le trafic HTTP vers une adresse sécurisée HTTPS, ce qui assure des communications sécurisées ainsi que d'obtenir une panoplie de statistiques sur les performances de notre serveur.

3.1.3 Utilisation du paque

Dans l'éventuelle possibilité que des requêtes invalides ne contenant pas les données attendues soient envoyées à notre interface de programmation, l'intergiciel (Middleware) *Microsoft.AspNetCore.Mvc*, signalera une erreur de validation puis enverra à l'utilisateur une réponse avec le statut 400 afin de l'informer des erreurs qu'il a commises. En effet, cet intergiciel permet de blinder notre interface de programmation afin de ne recevoir que les données que nous nous attendons, minimisant les erreurs internes du serveur.

3.1.4 Routes

La prochaine section aborde les différentes routes disponibles dans notre serveur qui devront être utilisées par les clients. Il est à noter que toutes les requêtes devront être effectuées à l'adresse web https://polypaint.me/, adresse où sera héberger notre serveur. De plus, il est important de spécifier que le corps de chacune des requêtes nécessitant l'envoi de données devra être sous forme de JSON.

3.1.5 Enregistrement

Afin de s'enregistrer, les différents clients devront effectuer une requête de type POST à la route /register. Cette dernière demandera plusieurs champs obligatoires:

Champs	Catégorie	Type	Description
firstName	Body	string	Prénom de l'utilisateur
lastName	Body	string	Nom de famille de l'utilisateur
email	Body	string	Adresse courriel de l'utilisateur
username	Body	string	Nom d'utilisateur
password	Body	string	Mot de passe

Tableau #1: Champs requis dans le corps de la requête pour enregistrer un nouveau compte

Lors d'un enregistrement réussi, le serveur enverra une réponse avec le status 200. Cependant, en cas d'échec un code de statut 400 sera retourné à l'utilisateur.

HTTP Code	Description	Corps de la réponse
200	Connexion réussie	"Votre compte à été enregistré! Vous pouvez maintenant vous connecter..."
400	Échec de la connexion	« Erreurs: » Descriptions des erreurs liées à l'enregistrement

Tableau #2: Différentes réponses possibles suite à une requête d'enregistrement

3.1.6 Authentification

Afin de s'authentifier auprès de l'application, les différents clients devront faire une requête de type POST à la route /api/login. Il est important de noter qu'une seule connexion simultanée par compte sera permise. Il sera donc impossible de se connecter sur plusieurs clients en simultané avec les même identifiants de connexions.

Champs	Catégorie	Type	Description
username	Body	string	Nom d'utilisateur
password	Body	string	Mot de passe

Tableau #3: Champs requis dans le corps de la requête pour authentifier un compte existant

Lors d'une authentification réussie, le serveur enverra une réponse avec le statut 200 contenant un jeton (JWT) dans le corps de la réponse de la requête. Cependant, en cas d'échec de connexion, le statut 400 sera envoyé avec un message d'erreur.

HTTP Code	Description	Corps de la réponse
200	Connexion réussie	{ Jeton de connexion }
400	Échec de la connexion	« Identifiants de connexion non valides »

Tableau #4: Différentes réponses possibles suite à une tentative de connexion

Pour se connecter à l'aide de l'authentification Facebook, les clients devront faire une requête de type GET à la route /api/login/facebook, qui les redirigeront par la suite vers l'interface de connexion Facebook. Sur cette page, ils seront priés de rentrer leur nom d'utilisateur de leur compte Facebook. Lorsque Facebook acceptera la connexion, les informations du compte Facebook de l'utilisateur seront envoyées au serveur à la route /api/login/fb-callback, ce qui permettra d'identifier l'utilisateur à l'interne et de lui envoyer un jeton de connexion.

3.1.7 Déconnexion

Lorsqu'un utilisateur devra se déconnecter, une requête de type GET à la route /api/user/logout devra être effectuée. Cette requête nécessite le jeton qui aura été fourni lors de la requête afin de pouvoir identifier l'utilisateur et sécuriser l'appel de cette fonction.

HTTP Code	Description	Corps de la réponse

200	Déconnexion réussie	« Déconnexion réussie »
400	Échec de la déconnexion	« Échec lors de la déconnexion »

Tableau #5: Différentes réponses possibles suite à une tentative de déconnexion

3.1.8 Galerie d'images

3.1.8.1 Sauvegarde et modification

Un utilisateur désirant sauvegarder son image sur un serveur distant pourra le faire en interpellant à l'aide d'une requête POST à la route /api/user/canvas. Un id sera automatiquement créé pour chacune des images enregistrées.

Champs	Type	Description
name	string	Nom d'utilisateur
public	boolean	Confidentialité de l'image
image	byte[]	L'image sous forme de bytes
password	string (optionnel)	Mot de passe protégeant l'image

Tableau #6: Champs requis dans le corps de la requête pour enregistrer ou modifier image

3.1.8.1 Chargement

Lorsqu'un utilisateur voudra charger une image dans l'application depuis le serveur distant, celui-ci devra faire une requête POST.

Champs	Type	Description
id	string	Identificateur de l'utilisateur
password	string (optionnel)	Mot de passe protégeant l'image.

Tableau #7: Champs requis dans le corps de la requête pour charger une image

3.1.8.4 Images publiques et privées

Les images du domaine public devront être accessibles par tous les utilisateurs. Pour ce faire, les clients devront faire une requête de type GET à la route /api/canvas afin d'obtenir tous les canevas du domaine public. En fournissant le jeton de connexion, l'utilisateur pourra aussi obtenir toutes les images privées. La réponse est un statut 200 contenant une liste d'images composées des mêmes champs que ceux décrits dans le volet 3.1.8.1.

3.1.9 Tutoriel

Afin de déterminer si le tutoriel doit être présenté à l'utilisateur, une requête GET à la route /api/user/tutorial permettra de recevoir un booléen indiquant si le tutoriel à été complété par celui-ci ou non. Encore une fois, puisque nous devons être en mesure d'identifier l'utilisateur, celui-ci devra fournir son jeton de connexion lors de la requête.

3.2 Protocole SignalR

SignalR sera utilisé pour les communications en temps réel entre les clients et le serveur. Il sera donc utilisé lors de la messagerie instantanée et lors de sessions de collaboration sur un dessin. Cette partie portera sur la description des messages qui transigerons entre les clients et le serveur.

La structure générale d'un message comportera un nom de fonction appelée, et un argument. Certaines fonctions peuvent être appelées et reçues, d'autres seulement appelées. Lorsqu'une fonction est reçue, c'est le serveur qui l'envoie au client. Le client doit donc gérer l'information reçue.

3.2.1 Liste de fonctions

Voici la liste des messages possibles :

Nom de la fonction	**Type d'argument**
SendMessage	ChatMessage
ConnectToCanvas	ConnectionMessage
DisconnectFromCanvas	ConnectionMessage
FetchChannels	ChannelsMessage
CreateChannel	ChannelMessage
ConnectToChannel	ConnectionMessage
DisconnectFromChannel	ConnectionMessage
FetchCanvas	ItemsMessage
AddItem	ItemMessage
ChangeItems	ItemsMessage
UnauthorizedAction	ErrorMessage
RemoveItems	ItemsMessage
SelectItem	ItemMessage
SelectStyle	StyleMessage
ResetCanvas	-
ResizeCanvas	SizeMessage
RedoChange	ItemsMessage
UndoChange	ItemsMessage
Duplicate	ItemsMessage
Cut	ItemsMessage
Copy	ItemsMessage
ActivateProtection	-

DeactivateProtection	-
SelectAll	ItemsMessage
InvertColors	ItemsMessage
InvertSelection	ItemsMessage

3.2.2 Liste de types d'arguments

Voici la structure des différents types d'arguments:

3.2.2.1 ChatMessage

Attributs	**Type**	**Valeur**
username	string	Identité du messager
message	string	Valeur du message
channelId	string	Identification du canal du message
timestamp	string	Estampille de l'envoi

3.2.2.2 ConnectionMessage

Attributs	**Type**	**Valeur**
username	string	Identité de l'usager
canvasId	string	Identificateur du canevas de collaboration
channelId	string	Identificateur du canal de communication

3.2.2.3 ChannelMessage

Attributs	**Type**	**Valeur à la réception**
channel	Channel	Canal de communication

3.2.2.4 ChannelsMessage

Attributs	**Type**	**Valeur à la réception**
channels	List<Channel>	Canaux de communication

3.2.2.5 ItemsMessage

Attributs	**Type**	**Valeur**
canvasId	string	Identification du canevas de collaboration
username	string	Identité de l'utilisateur
items	List<Item>	Liste d'items dans le canevas

3.2.2.6 ItemMessage

Attributs	**Type**	**Valeur**
username	string	Identité de l'utilisateur
item	Item	Item à changer, enlever, ajouter, etc...

3.2.2.7 ErrorMessage

Attributs	**Type**	**Valeur**
message	string	Message d'erreur

3.2.2.8 StyleMessage

Attributs	**Type**	**Valeur**
style	Style	Style voulu

3.2.2.9 SizeMessage

Attributs	**Type**	**Valeur**
size	Point	Dimensions

3.2.2.10 Classes utilisées dans les messages

3.2.2.10.1 Channel

Attributs	**Type**	**Description**
id	string	Identificateur du canal de communiation
name	string	Nom du canal de communication

3.2.2.10.2 abstract Item

Attributs	Type	Description
id	string	Identificateur de l'item
name	string	Identification du type d'item

3.2.2.10.3 TextItem : Item

Attributs	Type	Description
id	string	Identificateur de l'item
name	string	Identification du type d'item
position	Point	Position de l'item
text	string	Text contenu dans l'item
size	Point	Dimensions de l'item
angle	double	Angle de l'item en radians

3.2.2.10.4 ImageItem : Item

Attributs	Type	Description
id	string	Identificateur de l'item
name	string	Identification du type d'item
position	Point	Position de l'item
data	byte[]	Données de l'image
size	Point	Dimensions de l'image
angle	double	Angle de l'item en radians

3.2.2.10.5 BackgroundItem : Item

Attributs	Type	Description
id	string	Identificateur de l'item
name	string	Identification du type d'item
data	byte[]	Données de l'arrière-plan
size	Point	Dimensions de l'arrière-plan

3.2.2.10.6 UmlShapeItem : Item

Attributs	Type	Description

id	string	Identificateur de l'item
name	string	Identification du type d'item
position	Point	Position de l'item
type	string	Type de forme UML
label	string	Libellé de l'item
methods	List<string>	Liste des méthodes (seulement applicable dans une forme UML de classe)
attributes	List<string>	Liste des attributs (seulement applicable dans une forme UML de classe)
size	Point	Dimensions de l'image
style	Style	Style de la forme UML
angle	double	Angle de l'item en radians

3.2.2.10.7 RelationItem : Item

Attributs	**Type**	**Description**
id	string	Identificateur de l'item
name	string	Identification du type d'item
type	string	Type de relation
fromId	string	Identificateur de l'item source
toId	string	Identificateur de l'item destination
style	Style	Style de la relation

3.2.2.10.8 LineItem : Item

Attributs	**Type**	**Description**
id	string	Identificateur de l'item
name	string	Identification du type d'item
type	string	Type de trait
fromPosition	Point	Position source
toPosition	Point	Position destination

style	Style	Style du trait

3.2.2.10.9 Point

Attributs	**Type**	**Description**
x	double	Valeur en x
y	double	Valeur en y

3.2.2.10.10 Style

Attributs	**Type**	**Description**
thickness	double	Épaisseur
color	string	Couleur
type	string	Type (pointillé, full, etc…)

3.2.1 SendMessage

Cette fonction sera appelée lorsqu'un utilisateur envoiera un message. L'argument à envoyer est un ChatMessage avec les attributs message et channelId définis. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un ChatMessage sera fourni. Les attributs username, message, channelId et timestamp seront définis.

ChatMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
username	-	✓
message	✓	✓
channelId	✓	✓
timestamp	-	✓

3.2.2 ConnectToCanvas

Cette fonction sera appelée lorsqu'un utilisateur souhaitera se connecter à une session de collaboration. L'argument à envoyer est un ConnectionMessage avec l'attribut canvasId défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un ConnectionMessage sera fourni. Les attributs username et canvasId seront définis.

ConnectionMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
username	-	✓
canvasId	✓	✓

channelId	-	-

3.2.3 DisconnectFromCanvas

Cette fonction sera appelée lorsqu'un utilisateur souhaitera se déconnecter d'une session de collaboration. L'argument à envoyer est un ConnectionMessage avec l'attribut canvasId défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un ConnectionMessage sera fourni. Les attributs username et canvasId seront définis.

ConnectionMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
username	-	✓
canvasId	✓	✓
channelId	-	-

3.2.4 FetchChannels

Cette fonction sera appelée lorsqu'un utilisateur souhaitera accéder à la liste de canaux de communication. Il n'y aura pas d'argument à envoyer. Lors de la réception de ce message, un ChannelsMessage sera fourni. L'attribut channels sera défini.

ChannelsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
channels	-	✓

3.2.5 CreateChannel

Cette fonction sera appelée lorsqu'un utilisateur souhaitera créer un canal de communication. L'argument à envoyer est un ChannelMessage avec l'attribut channel défini décrivant le canal à créer. Un message d'erreur sera envoyé si le format de l'argument fourni est invalide. Lors de la réception de ce message, un ChannelMessage sera fourni. L'attribut channel sera défini décrivant le canal nouvellement créé.

ChannelMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
channel	✓	✓

3.2.6 ConnectToChannel

Cette fonction sera appelée lorsqu'un utilisateur souhaitera se connecter à un canal de communication. L'argument à envoyer est un ConnectionMessage avec l'attribut channelId défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un ConnectionMessage sera fourni. Les attributs username et channelId seront définis.

ConnectionMessage

Attributs	Présence à l'appel	Présence à la réception
username	-	✓
canvasId	-	-
channelId	✓	✓

3.2.7 DisconnectFromChannel

Cette fonction sera appelée lorsqu'un utilisateur souhaitera se déconnecter d'un canal de communication. L'argument à envoyer est un ConnectionMessage avec l'attribut channelId défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un ConnectionMessage sera fourni. Les attributs username et channelId seront définis.

ConnectionMessage

Attributs	Présence à l'appel	Présence à la réception
username	-	✓
canvasId	-	-
channelId	✓	✓

3.2.8 FetchCanvas

Cette fonction sera appelée lorsqu'un utilisateur souhaitera accéder à un canevas. L'argument à envoyer est un ItemsMessage avec l'attribut canvasId défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un ItemsMessage sera fourni. L'attribut items sera défini.

ItemsMessage

Attributs	Présence à l'appel	Présence à la réception
canvasId	✓	-
username	-	-
items	-	✓

3.2.9 AddItem

Cette fonction sera appelée lorsqu'un utilisateur souhaitera ajouter un item au canevas. L'argument à envoyer est un ItemMessage avec l'attribut item défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un ItemMessage sera fourni. L'attribut item sera défini.

ItemMessage

Attributs	Présence à l'appel	Présence à la réception
username	-	-

item	✓	✓

3.2.10 ChangeItems

Cette fonction sera appelée lorsqu'un utilisateur souhaitera modifier les items du canevas. L'argument à envoyer est un ItemsMessage avec l'attribut items défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un ItemsMessage sera fourni. Les attributs items et username seront définis.

ItemsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
canvasId	-	-
username	-	✓
items	✓	✓

3.2.11 UnauthorizedAction

Cette fonction ne pourra pas être appelée. Lors de la réception de ce message, un ErrorMessage sera fourni. L'attribut message sera défini.

ErrorMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
message	-	✓

3.2.12 RemoveItems

Cette fonction sera appelée lorsqu'un utilisateur souhaitera supprimer les items du canevas sélectionnés. Il n'y aura pas d'argument à envoyer. Lors de la réception de ce message, un ItemsMessage sera fourni. L'attribut items sera défini.

ItemsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
canvasId	-	-
username	-	-
items	-	✓

3.2.13 SelectItem

Cette fonction sera appelée lorsqu'un utilisateur souhaitera sélectionner un item du canevas. L'argument à envoyer est un ItemMessage avec l'attribut item défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un ItemMessage sera fourni. Les attributs item et username seront définis.

ItemMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
username	-	✓
item	✓	✓

3.2.14 SelectStyle

Cette fonction sera appelée lorsqu'un utilisateur souhaitera sélectionner un style. L'argument à envoyer est un StyleMessage avec l'attribut style défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un StyleMessage sera fourni. L'attribut style sera défini.

StyleMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
style	✓	✓

3.2.15 ResetCanvas

Cette fonction sera appelée lorsqu'un utilisateur souhaitera réinitialiser le canevas. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, aucun argument sera envoyé.

3.2.16 ResizeCanvas

Cette fonction sera appelée lorsqu'un utilisateur souhaitera modifier la taille du canvas. L'argument à envoyer est un SizeMessage avec l'attribut size défini. Un message d'erreur sera envoyé si le format des arguments fourni est invalide. Lors de la réception de ce message, un SizeMessage sera fourni. L'attribut size sera défini.

SizeMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
size	✓	✓

3.2.17 RedoChange

Cette fonction sera appelée lorsqu'un utilisateur souhaitera empiler un changement. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, un ItemsMessage sera fourni. Les attributs items et username seront définis.

ItemsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
canvasId	-	-
username	-	✓
items	-	✓

3.2.18 UndoChange

Cette fonction sera appelée lorsqu'un utilisateur souhaitera dépiler un changement. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, un ItemsMessage sera fourni. Les attributs items et username seront définis.

ItemsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
canvasId	-	-
username	-	✓
items	-	✓

3.2.19 Duplicate

Cette fonction sera appelée lorsqu'un utilisateur souhaitera dupliquer des items du canevas. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, un ItemsMessage sera fourni. Les attributs items et username seront définis.

ItemsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
canvasId	-	-
username	-	✓
items	-	✓

3.2.20 Cut

Cette fonction sera appelée lorsqu'un utilisateur souhaitera couper les items du canevas sélectionnés. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, un ItemsMessage sera fourni. L'attribut items sera défini.

ItemsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
canvasId	-	-
username	-	-
items	-	✓

3.2.21 Copy

Cette fonction sera appelée lorsqu'un utilisateur souhaitera copier les items du canevas sélectionnés. Il n'y a pas d'argument à envoyer. Cette fonction ne peut pas être reçue.

3.2.22 ActivateProtection

Cette fonction sera appelée lorsqu'un utilisateur souhaitera activer la protection d'un canevas. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, aucun argument n'est fourni.

3.2.23 DeactivateProtection

Cette fonction sera appelée lorsqu'un utilisateur souhaitera désactiver la protection d'un canevas. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, aucun argument n'est fourni.

3.2.24 SelectAll

Cette fonction sera appelée lorsqu'un utilisateur souhaitera sélectionner tous les items du canevas. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, un ItemsMessage sera fourni. Les attributs items et username seront définis.

ItemsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
canvasId	-	-
username	-	✓
items	-	✓

3.2.25 InvertColors

Cette fonction sera appelée lorsqu'un utilisateur souhaitera inverser les couleurs des items du canevas sélectionnés. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, un ItemsMessage sera fourni. Les attributs items et username seront définis.

ItemsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**
canvasId	-	-
username	-	✓
items	-	✓

3.2.26 InvertSelection

Cette fonction sera appelée lorsqu'un utilisateur souhaitera inverser la sélection courant. Il n'y a pas d'argument à envoyer. Lors de la réception de ce message, un ItemsMessage sera fourni. Les attributs items et username seront définis.

ItemsMessage

Attributs	**Présence à l'appel**	**Présence à la réception**

canvasId	-	-
username	-	✓
items	-	✓