Équipe 11

Projet Intégrateur de 3e Année Poly Paint
Protocole de communication

Version 4.0

Historique des révisions

Date	Version	Description	Auteur
2019-01-29	1.1	Proposition de protocole	Nicolas Coutu
2019-02-06	1.2	Description des paquets	Nicolas Coutu, Tom Gautier
2019-02-07	1.3	Communication client-serveur	Tom Gautier
2019-02-09	2.0	Revue finale	Tom Gautier
2019-03-22	3.0	Modifications des événements pour le collaboratif	Tom Gautier
2019-03-28	3.1	Mise à jour fonctionnalités non implémentées	Tom Gautier
2019-04-07	3.2	Ajout de nouveaux événements	Tom Gautier
2019-04-08	4.0	Relecture finale	Tom Gautier

Table des matières

Protocole de communication

1. Introduction

Ce document présente les décisions prises en ce qui concerne la partie réseau de la solution Poly Paint. Tout d'abord, une brève explication des différents moyens utilisés ainsi que de leur utilisation dans le programme sera faite.
Par la suite, la description des paquets nécessaires à la communication dans l'application sera détaillée à travers plusieurs tableaux, associés en paire Requête-Réponse.

2. Communication client-serveur

La majorité de la communication à travers l'application se fait en temps réel et avec une grande fluidité. En effet, que ce soit dans le clavardage ou lors d'une session collaborative, on veut établir une communication avec le moins de latence possible, permettant ainsi une expérience utilisateur agréable.
Par conséquent, nous avons choisi d'utiliser les websockets pour ces applications. Cette technologie possède des avantages qui nous intéressent particulièrement dans le cadre du projet PolyPaint. Parmi les principales caractéristiques, elle permet:

- Un temps de réaction très rapide: étant donné que les WebSockets n'ont pas besoin du "overhead" classique des requêtes/réponses HTTP pour chaque message reçu et envoyé, la communication devient plus efficiente;
- De gérer le "long-polling" d'HTTP d'une manière élégante afin que les clients n'aient pas à faire de requêtes régulières au serveur pour rafraîchir les informations dont ils ont besoin. Celles-ci leur seront directement communiquées lors d'un changement ou d'une modification;
- Le support pour les "salles" est intégré. Cela facilitera grandement le développement du serveur afin de gérer de multiples canaux de discussion et sessions collaboratives. Cela se fait de manière très efficace et avec une latence faible.

Néanmoins, l'usage du protocole HTTP sera justifiée dans plusieurs fonctionnalités de l'application, notamment celles dont les mises-à-jour ne seront pas aussi fréquentes. En effet, ce protocole reconnu pour sa stabilité est très utile pour :

- Demander de l'information à la base de données que l'on n'aura pas besoin de mettre à jour le plus rapidement possible;
- Gérer des requêtes qui auront possiblement des erreurs, comme l'authentification à un logiciel par exemple.

3. Description des paquets

Tous les tableaux de cette section respecteront la nomenclature:

Raison de la requête	Protocole	Événement	Arguments

Tous les arguments des requêtes seront communiqués en format JSON.

3.1. Connection au serveur socket

Requête au serveur	Websockets	Connect	-

3.1. Déconnexion du serveur socket

Requête au serveur	Websockets	Disconnect	-

3.2. Authentification

Login:

Requête au serveur	HTTP	POST connection/login/	HEADER username: string, password: string
Réponse du serveur	HTTP	SEND	HEADER sessionId: string

Signup:

Requête au serveur	HTTP	POST connection/signup/	HEADER username: string, password: string
Réponse du serveur	HTTP	SEND	HEADER sessionId: string

3.2. Mot de passe oublié

Requête au serveur	HTTP	POST connection/forgot/:username/:email	HEADER: username: string, email: string
Réponse du serveur	Websockets	SEND	message: string

3.3. Réinitialiser mot de passe

Requête au serveur	HTTP	POST connection/reset/:username/:password/:newPassword	HEADER: username: string, email: string, newPassword: string
Réponse du serveur	Websockets	SEND	sessionId: string

3.3. Chat - Envoyer un message

Requête au serveur	Websockets	MessageSent	author: string, message: string, conversationId: string
Réponse du serveur	Websockets	MessageReceived	date: string, author: string, message: string, conversationId: string

3.5. Chat - Joindre un canal de discussion

Requête au serveur	Websockets	UserJoinedConversation	sessionId: string, username: string conversationId: string
Réponse du serveur	Websockets	UserJoinedConversation	conversationId: string

Réponse du serveur	Websockets	ChannelAlreadyJoined	conversationId: string

Réponse du serveur	Websockets	ChannelCouldntBeJoined	conversationId: string

3.5. Chat - Quitter un canal de discussion

Requête au serveur	Websockets	UserLeftConversation	sessionId: string, username: string conversationId: string
Réponse du serveur	Websockets	UserLeftConversation	conversationId: string

Réponse du serveur	Websockets	ChannelAlreadyLeft	conversationId: string

Réponse du serveur	Websockets	ChannelCouldntBeLeft	conversationId: string

3.6. Chat - Créer un canal de discussion

Requête au serveur	HTTP	POST /api/chat/:sessionId/: username/:conversati onName	HEADER sessionId: string, username: string, conversationName: string
Réponse du serveur	HTTP	SEND	conversationName: string

3.7. Chat - Recevoir liste des canaux de discussion

Requête au serveur	HTTP	GET /api/chat/:sessionId/: username	HEADER sessionId: string, username: string
Réponse du serveur	HTTP	SEND	conversations: Conversation[] *

3.8. Chat - Inviter un usager à un canal de discussion

Requête au serveur	Websockets	InviteToConversation	sessionId: string, username: string, invitedUsername: string, conversationId: string
Réponse du serveur	Websockets	InvitedToConversation	username: string, invitedUsername: string, conversationId: string
Réponse du serveur	Websockets	UserIsNotConnected	

3.9. Chat - Répondre à une invitation à un canal de discussion

Requête au serveur	Websockets	RespondToConversation Invite	sessionId: string, username: string, invitedUsername: string, imageId: string
Réponse du serveur	Websockets	RespondedToConversati onInvite	username: string, invitedUsername: string, imageId: string
Réponse du serveur	Websockets	UserIsNotConnected	

3.10. Galerie - Joindre une session collaborative

Requête au serveur	Websockets	JoinDrawingSession	sessionId: string, username: string, imageId: string, password (opt.) : string
Réponse du serveur	Websockets	JoinedDrawingSessio n	imageId: string
Requête du serveur	HTTP	GET api/images/single /:sessionId/:username /:imageId	HEADER sessionId: string, username: string, imageId: string
Réponse du serveur	HTTP	SEND	image: Image
Requête du serveur	HTTP	GET api/shapes/:sessionId/ :username/:imageId	HEADER sessionId: string, username: string, imageId: string
Réponse du serveur	HTTP	SEND	shapes: Shape[]

3.11. Chat - Inviter un usager à une session de dessin collaboratif

Requête au serveur	Websockets	InviteToDrawing	sessionId: string, username: string, invitedUsername: string, imageId: string
Réponse du serveur	Websockets	InvitedToDrawing	username: string, invitedUsername: string, imageId: string
Réponse du serveur	Websockets	UserIsNotConnected	

3.12. Chat - Répondre à une invitation à une session de dessin collaboratif

Requête au serveur	Websockets	RespondToDrawingInvite	sessionId: string, username: string, invitedUsername: string, imageId: string
Réponse du serveur	Websockets	RespondedToDrawingInvite	username: string, invitedUsername: string, imageId: string
Réponse du serveur	Websockets	UserIsNotConnected	

3.13. Galerie - Créer une nouvelle image

Requête au serveur	Websockets	CreateDrawingSession	name: string, protection: Protection, visibility: Visibility, image: Image
Réponse du serveur	Websockets	CreatedDrawingSession	imageId: string, success : bool

3.14. Galerie - Quitter une session collaborative

Requête au serveur	Websockets	LeaveDrawingSession	sessionId: string, username: string, imageId: string
Réponse du serveur	Websockets	LeftDrawingSession to room “imageId”	users: string[],

3.15. Galerie - Obtenir les images publiques des galeries

Requête au serveur	HTTP	GET /api/images/:sessionId/:username/public	HEADER sessionId: string, username: string
Réponse du serveur	HTTP	SEND	images: Image[] 400: Mauvaise requête 403: Accès refusé 520: Unknown error

3.16. Galerie - Obtenir les images privées des galeries

Requête au serveur	HTTP	GET /api/images/:sessionId/:username/private	HEADER sessionId: string, username: string
Réponse du serveur	HTTP	SEND	images: Image[] 400: Mauvaise requête 403: Accès refusé 520: Unknown error

3.17. Galerie - Créer une nouvelle image

Requête au serveur	HTTP	POST /api/images/:sessionId/:username	HEADER sessionId: string, username: string BODY author: string, imageId: string, visibility: string, protection: string, canvasX: number, canvasY: number
Réponse du serveur	HTTP	SEND	image: Image 400: Mauvaise requête 403: Accès refusé 520: Unknown error

3.18. Galerie - Envoyer les formes dessinées hors-ligne

Requête au serveur	HTTP	POST /api/images/offline/:sessionId/:username	HEADER sessionId: string, username: string, imageId: string BODY shapes: Shape[]
Réponse du serveur	HTTP	SEND	image: Image 400: Mauvaise requête 403: Accès refusé 520: Unknown error

3.19. Galerie - Obtenir les formes associées à une image

Requête au serveur	HTTP	GET /api/shapes/:sessionId/:username/:imageId	HEADER sessionId: string, username: string, imageId: string
Réponse du serveur	HTTP	SEND	images: Shapes[] 400: Mauvaise requête 403: Accès refusé 520: Unknown error

3.20. Galerie - Mettre à jour une miniature

Requête au serveur	HTTP	POST /api/images/thumbnail /:sessionId/:username/ :imageId	HEADER sessionId: string, username: string, imageId: string BODY thumbnail: string, thumbnailTimestamp: number
Réponse du serveur	HTTP	SEND	image: Image[] 400: Mauvaise requête 403: Accès refusé 520: Unknown error

3.21. Accessibilité des images - Édition de l'accès

Requête au serveur	HTTP	SEND	HEADER imageId: string, protection: string, visibility: string
Réponse du serveur	HTTP	SEND	imageId: string, protection: string

3.22. Liste d'utilisateur - Notification de la connexion d'un nouvel utilisateur

Réponse du serveur	Websockets	UserJoinedChat	username: string

3.23. Liste d'utilisateur - Obtenir tous les utilisateurs

Requête au serveur	HTTP	GET /api/user/	HEADER sessionId: string, username: string
Réponse du serveur	HTTP	SEND	[{username: string, connected: boolean}]

3.24. Session collaborative - Checks before any action

Réponse du serveur	Websockets	UserIsNotLogged	objectId: string

Réponse du serveur	Websockets	ObjectIsntSelected	objectId: string

Réponse du serveur	Websockets	ObjectSelectedByOtherUser	objectId: string

3.25. Session collaborative - Ajouter un élément

Requête au serveur	Websockets	AddElement	HEADER sessionId: string, username: string, BODY shape: Shape
Réponse du serveur	Websockets	AddedElement	BODY shape: Shape

3.26. Session collaborative - Dupliquer un élément

Requête au serveur	Websockets	DuplicateElements	HEADER sessionId: string, username: string, BODY shapes: Shape[]
Réponse du serveur	Websockets	DuplicatedElements	BODY shapes: Shape[]

3.29. Session collaborative - Dupliquer un élément lorsqu'il n'y a pas de sélection

Requête au serveur	Websockets	DuplicateCutElements	HEADER sessionId: string, username: string, BODY shapes: Shape[]
Réponse du serveur	Websockets	DuplicatedCutElements	BODY shapes: Shape[]

3.30. Session collaborative - Supprimer un ou des éléments

Requête au serveur	Websockets	DeleteElements	HEADER sessionId: string, username: string, BODY ids: string[]
Réponse du serveur	Websockets	DeletedElements	BODY ids: string[]

3.31. Session collaborative - Couper un ou des éléments

Requête au serveur	Websockets	CutElements	HEADER sessionId: string, username: string, BODY ids: string[]
Réponse du serveur	Websockets	CutedElements	BODY ids: string[]

3.32. Session collaborative - Modifier un élément

Requête au serveur	Websockets	ChangeElement	HEADER sessionId: string, username: string, BODY shapes: Shape[]
Réponse du serveur	Websockets	ChangedElement	BODY shapes: Shape[]

3.33. Session collaborative - Empiler

Requête au serveur	Websockets	Stack	HEADER sessionId: string, username: string, BODY ids: string[]
Réponse du serveur	Websockets	ChangedElement	BODY ids: string[]

3.34. Session collaborative - Dépiler

Requête au serveur	Websockets	Unstack	HEADER sessionId: string, username: string, BODY shapes: Shape[]
Réponse du serveur	Websockets	ChangedElement	BODY shapes: Shape[]

3.35. Session collaborative - Sélectionner/déselectionner un ou des éléments

Requête au serveur	Websockets	SelectObjects	HEADER sessionId: string, username: string, BODY oldElementIds: string[], newElementIds: string[]
Réponse du serveur	Websockets	SelectedObjects	BODY oldElementIds: string[], newElementIds: string[]

3.36. Session collaborative - Redimensionner le canevas

Requête au serveur	Websockets	ResizeCanvas	HEADER: sessionId: string, username: string, BODY newCanvasDimensions: { x: int, y: int }
Réponse du serveur	Websockets	ResizedCanvas	BODY newCanvasDimensions: { x: int, y: int }

3.37. Session collaborative - Reset le canevas

Requête au serveur	Websockets	ResetCanvas	HEADER: sessionId: string, username: string, BODY: null
Réponse du serveur	Websockets	CanvasReset	null

* L'objet "Shape" est un JSON défini comme suit:
{

id: string,

imageId: string,

author: string,

properties: { **type**: string, **fillingColor**: string, **borderColor**: string, **middlePointCoord**: int[2], **height**: int, **width**: int, **rotation**: int }

}

Le **id** de Shape doit être produit du côté client sous la forme "**username_newObjectId**" où le **newObjectId** doit être un timestamp.

* L'objet "Conversation" est un JSON défini comme suit:
{

name: string,

participants: string[]

}