Équipe 10

Poly Paint Pro
Protocole de communication

Version 1.02

Historique des révisions

Date	Version	Description	Auteur
2019-01-26	1.00	Première ébauche	Alexandre Rault
2019-02-03	1.01	Sections 1, 3.3 et 3.4	Antoine Daigneault-Demers
2019-02-05	1.02	Section 3	Alexandre Rault Antoine Daigneault-Demers

Table des matières

1. Introduction **4**

2. Communication client-serveur **4**

3. Description des paquets **4**

3.1. Gestion de l'utilisateur 4

3.2. Clavardage 6

3.3. Galerie d'images 8

3.4. Édition de base collaborative 10

3.5. Sauvegarde d'images 16

3.6. Définitions supplémentaires 17

Protocole de communication

1. Introduction

Le protocole de communication décrit comment sera organisée la communication entre les clients et le serveur de l'application PolyPaint.

Il explique le choix du moyen de communication et décrit les paquets utilisés afin de transmettre les informations.

2. Communication client-serveur

La communication client-serveur passe par des sockets utilisant la librairie socket io. Socket IO est excellent pour faire de la communication basée sur des évènements, ce qui convient bien au logiciel PolyPaint.

3. Description des paquets

3.1. Gestion de l'utilisateur

<table>
<tr><th colspan="3">Créer un utilisateur</th></tr>
<tr><th></th><th>Format</th><th>Description</th></tr>
<tr><td>Requête</td><td>socket.on("createUser", function {…});</td><td rowspan="2">Le client peut envoyer ce message au serveur pour ajouter un usager sur la base de donnée.</td></tr>
<tr><td>Paramètre requête</td><td>"{
"username": string,
"password": string
}";</td></tr>
<tr><td>Réponse</td><td>socket.emit("createUserResponse", data);</td><td rowspan="2">Le serveur renvoie un message au client en spécifiant si la création de l'usager a été un succès ou un échec.</td></tr>
<tr><td>Paramètre réponse</td><td>"{
"isUserCreated": bool
}";</td></tr>
</table>

Connecter un utilisateur		
	Format	Description
Requête	socket.on(“loginUser”, function {…});	Le client peut envoyer ce message au serveur pour vérifier la connexion avec le mot de passe entré.
Paramètre requête	“{ “username”: string, “password”: string }”;	
Réponse	socket.emit(“loginUserResponse”, data);	Le serveur renvoie un message au client en spécifiant si la connexion de l’usager a été un succès ou un échec.
Paramètre réponse	“{ “isLoginSuccessful”: bool }”;	

3.2. Clavardage

Créer un canal de discussion		
	Format	Description
Requête	socket.on(“createChatroom”, function {…});	Le client peut envoyer ce message au serveur pour créer un canal de discussion.
Paramètre requête	“{“roomName”: string}”;	
Réponse	socket.emit(“createChatroomResponse”, data);	Le serveur renvoie un message au client contenant le nom du canal de discussion et le succès ou l’échec de sa création.
Paramètre réponse	“{ “roomName”: string, “isCreated”: bool }”;	

Joindre un canal de discussion		
	Format	Description
Requête	socket.on(“joinChatroom”, function {…});	Le client peut envoyer ce message au serveur pour joindre un canal de discussion.
Paramètre requête	“{ “roomName”: string }”;	
Réponse	socket.emit(“joinChatroomResponse”, data);	Le serveur renvoie un message au client contenant le nom du canal de discussion et le succès ou l’échec de la tentative de le joindre.
Paramètre réponse	“{ “roomName”: string, “isJoined”: bool, “history”: Message[] }”;	

Quitter un canal de discussion		
	Format	Description
Requête	socket.on(“leaveChatroom”, function {…});	Le client peut envoyer ce message au serveur pour quitter un canal de discussion.
Paramètre requête	“{“roomName”: string}”;	
Réponse	socket.emit(“leaveChatroomResponse”, data);	Le serveur renvoie un message au client contenant le nom du canal de discussion et le succès ou l’échec de la tentative de le quitter.
Paramètre réponse	“{ “roomName”: string, “isLeft”: bool }”;	

Envoyer la liste des canaux de discussion		
	Format	Description
Requête	socket.on(“getChatrooms”, function {…});	Le client peut envoyer ce message au serveur pour demander la liste des canaux de discussion.
Paramètre requête	“{ “roomNames”: string[] }”;	
Réponse	socket.emit(“getChatroomsResponse”, data);	Le serveur renvoie un message au client contenant la liste des canaux de discussion.
Paramètre réponse	“{ “roomNames”: string[] }”;	

Envoyer un message		
	Format	Description
Requête	socket.on(“sendMessage”, function {…});	Le client peut envoyer ce message au serveur pour transmettre un message dans le canal de discussion.
Paramètre requête	“{ message: Message }”;	
Réponse	socket.emit(“messageSent”, data);	Le serveur renvoie un message aux clients dans le canal de discussion contenant le message.
Paramètre réponse	“{ message: Message }”;	

3.3. Galerie d'images

Création d'un canevas		
	Format	Description
Requête	socket.on("createCanevas", function {…});	Le client peut envoyer ce message au serveur pour indiquer qu'il a créé un canevas. Le message contient l'auteur ainsi que le nom du canevas, s'il est public, et sa protection.
Paramètre requête	"{ "author": string, "canevasName": string, "isPublic": bool, "isProtected": bool, "password": string }";	
Réponse	socket.emit("createCanevasResponse", data);	Le serveur renvoie un message au client indiquant si le canevas a bien été créé.
Paramètre réponse	"{"isCreationSuccessful": bool}";	

Afficher les canevas publiques		
	Format	Description
Requête	socket.on("publicCanevas", function {…});	Le client peut envoyer ce message au serveur pour avoir accès aux canevas publics.
Paramètre requête	"{ }";	
Réponse	socket.emit("publicCanevasResponse", data);	Le serveur renvoie un message au client indiquant tous les canevas publics disponibles.
Paramètre réponse	"{"canevasList": string[]}";	

Afficher les canevas privés		
	Format	Description
Requête	socket.on("privateCanevas", function {…});	Le client peut envoyer ce message au serveur pour avoir accès à ses canevas privés.
Paramètre requête	"{ "username": string }";	
Réponse	socket.emit("privateCanevasResponse", data);	Le serveur renvoie un message au client indiquant tous les canevas privés de l'usager spécifié.
Paramètre réponse	"{ "canevasList": string[] }";	

Accès à un canevas non protégé		
	Format	Description
Requête	socket.on(“canevasAccess”, function {…});	Le client peut envoyer ce message au serveur pour demander l’accès à un canevas non protégé via le nom de ce dernier.
Paramètre requête	“{ “canevasName”: string }”;	
Réponse	socket.emit(“canevasAccessResponse”, data);	Le serveur renvoie un message au client indiquant si le client a pu se connecter au canevas.
Paramètre réponse	“{ “isRequestSuccessul”: bool }”;	

Accès à un canevas protégé		
	Format	Description
Requête	socket.on(“protectedCanevasAccess”, function {…});	Le client peut envoyer ce message au serveur pour demander l’accès à un canevas protégé via le nom de ce dernier et un mot de passe.
Paramètre requête	“{ “imageName”: string, “password”: string }”;	
Réponse	socket.emit(“protectedCanevasAccessResponse”, data);	Le serveur renvoie un message au client indiquant si le client a pu se connecter au canevas.
Paramètre réponse	“{ “isRequestSuccessul”: bool }”;	

3.4. Édition de base collaborative

Sélection de formes		
	Format	Description
Requête	socket.on(“selectForm”, function {…});	Le client peut envoyer ce message au serveur pour indiquer qu’il a sélectionné une ou des formes.
Paramètre requête	“{ “username”: string, “formsId”: string[] }”;	
Réponse	socket.emit(“selectFormResponse”, data);	Le serveur renvoie un message aux clients connectés sur l’image qui spécifie quelles formes ont été sélectionnées et par quel utilisateur.
Paramètre réponse	“{ “username”: string, “formsId”: string[] }”;	

Déplacement de formes		
	Format	Description
Requête	socket.on(“moveForm”, function {…});	Le client peut envoyer ce message au serveur pour indiquer qu’il a déplacé une ou des formes.
Paramètre requête	“{ “coordinates”: double[][], “formsId”: string[] }”;	
Réponse	socket.emit(“moveFormResponse”, data);	Le serveur renvoie un message aux clients connectés sur l’image qui spécifie quelles formes ont été déplacées et à quel endroit.
Paramètre réponse	“{ “coordinates”: double[][], “formsId”: string[] }”;	

Ajout de formes		
	Format	Description
Requête	socket.on(“createForm”, function {…});	Le client peut envoyer ce message au serveur pour indiquer qu’il a créé une nouvelle forme ainsi que la position de celle-ci.
Paramètre requête	“{ “formType”: int, “coordinates”: double[] }”;	
Réponse	socket.emit(“createFormResponse”, data);	Le serveur renvoie un message aux clients connectés sur l’image qui spécifie quelle forme a été créée et la position de celle-ci.
Paramètre réponse	“{ “formKind”: int “coordinates”: double[] }”;	

Ajout d’une image au canevas		
	Format	Description
Requête	socket.on(“createImage”, function {…});	Le client peut envoyer ce message au serveur pour indiquer qu’il a ajouté une image au canevas, la position de celle-ci et son contenu.
Paramètre requête	“{ “formKind”: int, “coordinates”: double[], “image”: arraybuffer }”;	
Réponse	socket.emit(“createImageResponse”, data);	Le serveur renvoie un message aux clients connectés sur le canevas qui spécifie l’image ajoutée et sa position.
Paramètre réponse	“{ “formKind”: int “coordinates”: double[], “image”: arraybuffer }”;	

Suppression de formes		
	Format	Description
Requête	socket.on(“deleteForm”, function {…});	Le client peut envoyer ce message au serveur pour indiquer qu’il a retiré une forme du canevas.
Paramètre requête	“{ “formsId”: string[] }”;	
Réponse	socket.emit(“deleteFormResponse”, data);	Le serveur renvoie un message aux clients connectés sur le canevas qui spécifie quelle forme a été supprimée.
Paramètre réponse	“{ “formsId”: string[] }”;	

Redimension de formes		
	Format	Description
Requête	socket.on(“resizeForm”, function {…});	Le client peut envoyer ce message au serveur pour indiquer les nouvelles dimensions d’une forme.
Paramètre requête	“{ “formId”: string, “formSize”: double[], “coordinates”: double[] }”;	
Réponse	socket.emit(“resizeFormResponse”, data);	Le serveur renvoie un message aux clients connectés sur le canevas qui spécifie quelle forme a été modifiée et ses nouvelles dimensions et coordonnées.
Paramètre réponse	“{ “formId”: string, “formSize”: double[], “coordinates”: double[] }”;	

Rotation de formes		
	Format	Description
Requête	socket.on(“rotateForm”, function {…});	Le client peut envoyer ce message au serveur pour indiquer la rotation d’une forme.
Paramètre requête	“{ “formId”: string, “formAngle”: double }”;	
Réponse	socket.emit(“rotateFormResponse”, data);	Le serveur renvoie un message aux clients connectés sur le canevas qui spécifie la rotation d’une forme.
Paramètre réponse	“{ “formId”: string, “formAngle”: double }”;	

Modification de texte		
	Format	Description
Requête	socket.on(“floatingTextModified”, function {…});	Le client peut envoyer ce message au serveur pour indiquer qu’il a modifié un texte.
Paramètre requête	“{ “formId”: string, “text” : string, “textNumber”: int }”;	
Réponse	socket.emit(“floatingTextModifiedResponse”, data);	Le serveur renvoie un message aux clients connectés sur le canevas qui spécifie quel texte a été modifié et quel est le nouveau contenu.
Paramètre réponse	“{ “formId”: string, “text”: string, “textNumber”: int }”;	

Réinitialisation du canevas		
	Format	Description
Requête	socket.on(“canevasReinitialized”, function {…});	Le client peut envoyer ce message au serveur pour indiquer qu’il a réinitialisé le canevas.
Paramètre requête	“{ }”;	
Réponse	socket.emit(“canevasReinitializedResponse”, data);	Le serveur renvoie un message aux clients connectés sur le canevas qui spécifie quel texte flottant a été modifié et quel est le nouveau contenu.
Paramètre réponse	“{ }”;	

Redimension du canevas		
	Format	Description
Requête	socket.on(“canevasResized”, function {…});	Le client peut envoyer ce message au serveur pour indiquer les nouvelles dimensions du canevas.
Paramètre requête	“{ “dimensions”: double[] }”;	
Réponse	socket.emit(“canevasResizedResponse”, data);	Le serveur renvoie un message aux clients connectés sur le canevas qui spécifie les nouvelles dimensions de ce dernier.
Paramètre réponse	“{ “dimensions”: double[] }”;	

Changer la couleur de bordure d'une forme		
	Format	Description
Requête	socket.on("changeOutlineColor", function {…});	Le client peut envoyer ce message au serveur pour indiquer un changement de couleur pour la bordure d'une ou plusieurs formes.
Paramètre requête	"{ "formId": string[], "color": string }";	
Réponse	socket.emit("changeOutlineColorResponse", data);	Le serveur renvoie un message aux clients connectés sur le canevas qui informe d'un changement de couleur pour la bordure d'une ou plusieurs formes.
Paramètre réponse	"{ "formId": string[], "color": string }";	

Changer le style de bordure d'une forme		
	Format	Description
Requête	socket.on("changeOutlineStyle", function {…});	Le client peut envoyer ce message au serveur pour indiquer un changement de style pour la bordure d'une ou plusieurs formes.
Paramètre requête	"{ "formId": string[], "style": int }";	
Réponse	socket.emit("changeOutlineStyleResponse", data);	Le serveur renvoie un message aux clients connectés sur le canevas qui informe d'un changement de style pour la bordure d'une ou plusieurs formes.
Paramètre réponse	"{ "formId": string[], "style": int }";	

Changer la couleur de remplissage d'une forme		
	Format	Description
Requête	socket.on("changeColorFill", function {…});	Le client peut envoyer ce message au serveur pour indiquer un changement de couleur pour le remplissage d'une ou plusieurs formes.
Paramètre requête	"{ "formId": string[], "color": int }";	
Réponse	socket.emit("changeColorFillResponse", data);	Le serveur renvoie un message aux clients connectés sur le canevas qui informe d'un changement de couleur pour le remplissage d'une ou plusieurs formes.
Paramètre réponse	"{ "formId": string[], "color": int }";	

Changer la couleur de remplissage d'une forme		
	Format	Description
Requête	socket.on("changeBorderWeight", function {…});	Le client peut envoyer ce message au serveur pour indiquer un changement d'épaisseur pour un ou plusieurs liens.
Paramètre requête	"{ "formId": string[], "weight": int }";	
Réponse	socket.emit("changeBorderWeightResponse", data);	Le serveur renvoie un message aux clients connectés sur le canevas qui informe d'un changement de d'épaisseur pour un ou plusieurs liens.
Paramètre réponse	"{ "formId": string[], "weight": int }";	

3.5. Sauvegarde d'images

Importation d'un canevas		
	Format	Description
Requête	socket.on("requestCanevas", function {…});	Le client peut envoyer ce message au serveur pour demander d'importer un canevas.
Paramètre requête	"{ "canevasName": String }";	
Réponse	socket.emit("requestCanevasResponse", data);	Le serveur renvoie un message au client indiquant si le client a pu importer le canevas ainsi que le canevas dans le cas d'un succès.
Paramètre réponse	"{ "isRequestSuccessul": bool, "canevas": Drawing }";	

Sauvegarder le canevas		
	Format	Description
Requête	socket.on("saveCanevas", function {…});	Le client peut envoyer ce message au serveur pour indiquer qu'il souhaite créer une sauvegarde.
Paramètre requête	"{ }";	
Réponse	socket.emit("saveCanevasResponse", data);	Le serveur renvoie un message au client pour l'informer si la sauvegarde s'est bien déroulée.
Paramètre réponse	"{ "saveSucessful": bool }";	

3.6. Définitions supplémentaires

Objet DrawingElement:

{id: string, type: int, coordinates: double[], texts: string[], borderColor: string, borderStyle: int, borderWeight: double}

Objet Drawing:

{id: string, dimensions: double[], forms: Form[], isProtected: bool, isPublic: bool}

Objet Message:

{username: string, message: string, createdAt: timestamp}