Équipe 102

Colorimage
Protocole de communication

Version 3.0

Historique des révisions

Date	Version	Description	Auteur
2021-09-25	1.0	Rédaction initial du protocole de communication	Équipe 102
2021-10-23	2.0	Correction du protocole de communication	Équipe 102
2021-12-05	3.0	Mise à jour final du protocole de communication	Équipe 102

Table des matières

Protocole de communication

1. Introduction

Ce document a pour objectif de définir les différentes façons que les clients communiqueront avec le serveur pour l'application ColorImage. Tout d'abord, ce document définit les deux types de communications utilisées pour l'échange de données entre les clients (lourd et léger) et le serveur, soit par des requêtes HTTP définies par le REST API de l'application ou bien par des websockets. Ensuite, on définit une liste de requêtes acceptées par le REST API, et une liste des différents événements émis et écoutés par le serveur et le client. Finalement, on définit une liste d'interfaces qui seront utilisées pour l'échange de données lors de la communication client-serveur.

2. Communication client-serveur

La communication client-serveur est gérée par deux types de communications: les requêtes HTTP et les sockets.

Requêtes HTTP

La communication traditionnelle, requête HTTP, suit une structure fixe où le client fait une requête au serveur et le serveur lui retourne une réponse. Cette communication est unidirectionnelle puisque le serveur est limité. En effet, il peut seulement communiquer avec le client lorsque celui-ci lui fait une requête. Ainsi, nous utiliserons ce genre de technologie pour des communications où le serveur n'a pas besoin d'entamer une communication vers le client tels qu'une demande d'information par un client, une demande pour se connecter en tant qu'utilisateur sur la plateforme, etc.

Gestion d'un utilisateur

Les fonctionnalités reliées à la gestion d'un utilisateur utilisent le REST API. La gestion d'un utilisateur regroupe entre autres la création d'un nouvel utilisateur, la modification des paramètres d'un utilisateur, la demande d'une liste des utilisateurs de la plateforme, la connexion d'un utilisateur existant sur la plateforme, la déconnexion d'un utilisateur de la plateforme, la demande pour obtenir les données liées à un utilisateur.

Gestion d'un canal

Certaines fonctionnalités reliées à la gestion des canaux utilisent le REST API. La gestion d'un canal regroupe entre autres la création d'un nouveau canal, l'ajout d'un utilisateur au canal, retirer un utilisateur du canal, la demande d'informations relatives à un canal, et la demande d'une liste des canaux existants.

Gestion d'une équipe de collaboration

Certaines fonctionnalités reliées à la gestion d'une équipe de collaboration utilisent le REST API. La gestion d'une équipe de collaboration regroupe entre autres la création d'une nouvelle équipe de collaboration, la demande pour quitter une équipe de collaboration, la demande pour se joindre à une équipe de collaboration, la demande pour supprimer une équipe de collaboration, la demande d'une liste d'équipe de collaboration existant sur la plateforme, la demande d'informations relatives à une équipe de collaboration, et la demande pour retourner une liste d'utilisateur faisant partie d'une équipe de collaboration.

Gestion d'un dessin collaboratif

Certaines fonctionnalités reliées à la gestion d'un dessin collaboratif utilisent le REST API. La gestion d'un dessin collaboratif regroupe entre autres la demande pour obtenir la liste de dessins existants sur la plateforme filtrée selon les paramètres spécifiés, la création d'un nouveau dessin, la demande pour obtenir les données reliées à un dessin, et le changement des paramètres d'un dessin collaboratif.

WebSocket

Le deuxième type de communication utilisé est par socket en utilisant la librairie SocketIO. Contrairement à la requête HTTP, SocketIO permet d'établir une communication bidirectionnelle qui permet aux deux parties (serveur et client) d'enclencher une communication. Ce type de communication est nécessaire pour faire des échanges d'information entre le serveur et le client en temps réel. Ainsi, cette communication est priorisée pour tout ce qui est lié au clavardage entre utilisateur dans un canal et aussi pour l'édition collaborative d'un dessin.

Clavardage

Les fonctionnalités liées au clavardage utilisent le WebSocket. Le clavardage regroupe les fonctionnalités suivantes: envoyer un message, rejoindre un canal, quitter un canal, signaler qu'un nouveau canal a été créé, signaler qu'un canal a été supprimé.

Équipe de collaboration

Certaines fonctionnalités liées à l'équipe de collaboration utilisent le WebSocket comme moyen de communication entre le serveur et le client. Ces fonctionnalités regroupent : rejoindre une équipe de collaboration, quitter une équipe de collaboration, signaler qu'une nouvelle équipe de collaboration a été créée, signaler qu'une équipe de collaboration a été supprimée.

Édition collaborative

Les fonctionnalités liées à l'édition collaborative utilisent le WebSocket. L'édition collaborative regroupe les fonctionnalités suivantes: dessiner un trait, insérer un rectangle, insérer une ellipse, sélectionner un trait, déplacer un trait, redimensionner un trait, supprimer un trait, pivoter un trait, insérer du texte, empiler une action, dépiler une action, rejoindre une édition collaborative, et quitter une édition collaborative.

Type de données transférées

Le transfert de données sera encapsulé dans un format JSON avec l'information à transférer sectionné par un titre dans le JSON. Le type de données transférées à l'intérieur du JSON peut prendre plusieurs types de bases (ex: string, number) ou bien des types prédéfinis par des interfaces. Une liste définie d'interface est présentée à la section 3.

Ex: { "message" : "Le message à transférer." }

3. Description des paquets

REST API

L'adresse universelle de base (URL) dépend du système sur lequel est hébergé le serveur. Comme le serveur sera hébergé sur Heroku, l'adresse universelle pour faire des requêtes est la suivante :

https://log3900-102.herokuapp.com/

Toutefois, il est aussi possible d'héberger le serveur localement. Ainsi, l'URL de base pour les requêtes au serveur sera par défaut localhost:3000. À noter que le port par défaut est le port 3000 s'il n'est pas utilisé.

La structure de notre REST API est décomposée en quatre routes parentes, c'est-à-dire une route qui contient d'autres sous routes, soit la route « /user », la route « /room », la route « /team », et la route « /drawing ».

Route « /user »

L'URL de base pour la route « /user » est :

https://log3900-102.herokuapp.com/user

Route	Méthode HTTP	Paramètres	Réponse	Description
/validate	GET	{ token: string }	{ response: Response }	Requête permettant de valider si un jeton de connexion disponible chez le client est encore valide (non expiré).
/register	POST	{ user: User }	{ response: Response }	Requête permettant de créer un nouvel utilisateur sur la plateforme.
/login	POST	{ email: string, password: string }	{ response: Response, token: string, username: string }	Requête permettant d'identifier un utilisateur et de le connecter au service à l'aide de son adresse courriel et de son mot de passe. Elle génère et renvoie au client un jeton de sécurité valide pendant 12h, qui permet la connexion.
/logout	POST	{ nickname : string }	{ response: Response }	Requête permettant de se déconnecter de la plateforme. Le jeton de l'utilisateur est alors invalidé.
/get_user_data	GET	{ nickname: string user: string }	{ response: Response, user: User }	Requête permettant d'obtenir les données relatives à un utilisateur. Elle prend un paramètre le nom de l'utilisateur effectuant la requête d'obtention de données, et le nom de l'utilisateur cible.
/update_username	PUT	{ user: string, newUser: string }	{ response: Response }	Requête permettant de modifier le nom d'un utilisateur
/update_avatar	PUT	{ user: string, avatar: string|Array<number>	{ response: Response }	Requête permettant de modifier l'avatar d'un utilisateur

		}		
/update_password	PUT	{ user: string, password: string }	{ response: Response }	Requête permettant de modifier le mot de passe d'un utilisateur
/update_fullname_privacy	PUT	{ user: string, isNamePublic: bool }	{ response: Response }	Requête permettant de modifier la confidentialité du nom d'un utilisateur
/update_email_privacy	UPDATE PUT	{ user: string, isEmailPublic: bool }	{ response: Response }	Requête permettant de modifier la confidentialité du courriel d'un utilisateur

Route « /room »

L'URL de base pour la route « /chat » est :

https://log3900-102.herokuapp.com/chat

Route	méthode HTTP	Paramètres	Réponse	Description
/join_room	POST	{ room: string, user: string }	{ response: Response }	Requête permettant de rejoindre un canal existant sur la plateforme.
/leave_room	POST	{ room: string, user: string }	{ response: Response }	Requête permettant de quitter un canal existant sur la plateforme.
/delete_room	POST	{ room: string, user: string }	{ response: Response }	Requête permettant de supprimer un canal existant sur la plateforme.
/create_room	POST	{ room: string, user: string }	{ response: Response }	Requête permettant de créer un nouveau canal de clavardage sur la plateforme.
/room_data	GET	{ room: string }	{ response: Response room : Room }	Requête permettant d'obtenir les données liées à un canal de clavardage de la plateforme.
/all_rooms	GET	-	{ response: Response, rooms: Array<String> }	Requête permettant d'obtenir une liste de tous les canaux existants.
/user_joined_rooms	GET	{ user: string }	{ rooms: Array<Room> }	Requête permettant d'obtenir une liste de tous les canaux ou se trouve l'utilisateur spécifié.
/user_unjoined_rooms	GET	{ user: string }	{ rooms: Array<Room> }	Requête permettant d'obtenir une liste de tous les canaux ou ne se trouve pas l'utilisateur spécifié.

Route « /team »

L'URL de base pour la route « /team » est :

https://log3900-102.herokuapp.com/team

Route	méthode HTTP	Paramètres	Réponse	Description
/create	POST	{ name: string, users:Array<string> }	{ response: Response }	Requête permettant de créer une nouvelle équipe de collaboration sur la plateforme.
/join_team	POST	{ name: string, user : string }	{ response: Response }	Requête permettant de joindre une équipe de collaboration existante sur la plateforme.
/leave_team	POST	{ name: string, user : string }	{ response: Response }	Requête permettant de quitter une équipe de collaboration dont l'utilisateur fait partie sur la plateforme.
/delete	DELETE	{ name: string, user : string }	{ response: Response }	Requête permettant de supprimer une équipe de collaboration existante sur la plateforme.
/get_all_teams	GET	-	{ teams: Array<Team> }	Requête permettant d'obtenir une liste de toutes les équipes existantes.

Route « /drawing »

L'URL de base pour la route « /drawing » est :

https://log3900-102.herokuapp.com/drawing

Route	Méthode HTTP	Paramètres	Réponse	Description
/create	POST	{ user: string, drawing: Drawing }	{ response: Response }	Requête permettant de créer un nouveau dessin de collaboration vide.
/update	PUT	{ name: string, preview: Array<number> }	{ response: Response }	Requête permettant de mettre à jour l'aperçu d'un dessin.
/save	POST	{ drawing: string, user: string }	{ response: Response }	Requête permettant de sauvegarder un dessin collaboratif existant.
/new_version	POST	{ drawing: string, user: string }	{ response: Response }	Requête permettant de créer une nouvelle version d'un dessin collaboratif existant sur la plateforme.
/get_drawing	GET	{ user: string, drawing: string }	{ response: Response, drawing: Drawing }	Requête permettant d'obtenir les informations liées à un dessin de collaboration.

/get_all_drawings	GET	{ user: string, drawings : Array<Drawing> }	{ response:Response, Array<Drawing> }	Requête permettant d'obtenir une liste de dessins existants sur la plateforme.
/get_drawing_data	GET	{ drawing: string }	{ drawing: Drawing }	Requête permettant d'obtenir les informations d'un dessin collaboratif.
/join_drawing	POST	{ drawing: string, user: string }	{ response: Response, objects: Array<VectorObject>, version: number, versions: number }	Requête permettant à un usager de se joindre à un dessin collaboratif existant sur la plateforme.
/leave_drawing	POST	{ drawing: string, user: string, preview: Array<number> }	{ response: Response }	Requête permettant à un usager de quitter un dessin collaboratif.
/swap_version	POST	{ drawing: string, version: number }	{ response: Response }	Requête permettant de changer la version d'un dessin collaboratif existant sur la plateforme.

WebSocket

Comme le socket est une communication bidirectionnelle entre le client et le serveur, il faut ainsi que le serveur ait défini des événements qu'il devra émettre au client et des événements qu'il devra écouter provenant du client. Même chose pour le client, il va devoir écouter à des événements envoyés par le serveur et émettre des événements au serveur.

Événements - Serveur

Événement	Paramètres	Description
send_message	{ room: string, message: Message }	Événements permettant de transmettre un message à tous les utilisateurs actifs du canal spécifié.
create_line	{ name: string, user: string, line: Line }	Événement qui notifie le serveur qu'un client actif a ajouté un sur le dessin.
end_drawing	{ name: string, user: string, id: string}	Événement qui notifie le serveur qu'un client a terminé de dessiner un trait.
draw_line	{ user: string, point: Point, id: string }	Événement qui notifie le serveur qu'un nouveau point a été ajouté à un trait existant, identifié par son id.
create_rectangle	{ name: string, user: string, rectangle: Shape}	Événement qui notifie le serveur qu'un nouveau rectangle a été ajouté , identifié par son id.
draw_rectangle	{ name: string, point: Point, id: string }	Événement qui notifie le serveur qu'un nouveau rectangle a été modifié, identifié par son id

create_ellipse	{ name: string, user: string, ellipse: Shape}	Événement qui notifie le serveur qu'une nouvelle ellipse a été ajoutée , identifiée par son id.
draw_ellipse	{ name: string, point: Point, id: string }	Événement qui notifie le serveur qu'une nouvelle ellipse a été modifiée, identifiée par son id.
select	{ name: string, id : string}	Événement qui notifie le serveur qu'un trait est sélectionné, identifié par son id.
unselect	{ name: string, id : string}	Événement qui notifie le serveur qu'un trait est désélectionné, identifié par son id.
edit	{ name: string, id: string, object: VectorObject }	Événement qui notifie le serveur qu'un client a modifié un trait (modification ponctuelle, ex: changement de couleur) sur le dessin, identifié par son id .
start_edit	{ name: string, id: string, object: VectorObject }	Événement qui notifie le serveur qu'un client a commencé la modification (modification soutenue, ex: translation/rotation) d'un trait sur le dessin, identifié par son id.
end_edit	{ name: string, id: string }	Événement qui notifie le serveur qu'un client a terminé la modification soutenue d'un trait sur le dessin, identifié par son id.
delete	{ user: string, id : string}	Événement qui notifie le serveur qu'un client du dessin qu'un trait a été supprimé.
undo	{ name: string, user: string}	Événement qui notifie le serveur qu'un client du dessin a annulé (empilé) sa dernière action.
redo	{ name: string, user: string}	Événement qui notifie le serveur qu'un client du dessin a refait sa dernière action.
send_to_back	{ name: string, id: string }	Événement qui notifie le serveur qu'un client a refait sa dernière action.
bring_to_front	{ name: string, id: string }	Événement qui permet de notifier le serveur qu'un client du dessin qu'un trait est envoyé à l'avant, identifié par son id.
backward	{ name: string, id: string }	Événement qui permet de notifier le serveur qu'un client du dessin qu'un trait est reculé d'une position, identifié par son id.
forward	{ name: string, id: string }	Événement qui permet de notifier le serveur qu'un client du dessin qu'un trait est avancé d'une position, identifié par son id.

Événements - Client

Événement	Paramètres	Description

send_message	{ room: string, message: Message }	Événement permettant d'envoyer au client le message qu'un utilisateur essaie d'envoyer.
connected	{ user: string }	Événement permettant de notifier le client de la connexion d'un utilisateur.
change_status	{ user: string, status: string }	Événement permettant de notifier les clients du changement de statuts d'un utilisateur.
disconnected	{ user: string }	Événement permettant de notifier le client de la déconnection d'un utilisateur.
join_room	{ room: string, user: string }	Événement permettant de signaler au client qu'un usager s'est joint au canal de discussion.
leave_room	{ room: string, user: string }	Événement permettant de signaler au client qu'un usager a quitté le canal de discussion.
create_room	{ room: string }	Événement permettant de signaler au client qu'un canal de discussion a été créé.
delete_room	{ room: string }	Événement permettant de signaler au client qu'un canal de discussion a été supprimé.
new_drawing	{}	Événement permettant de signaler au client qu'un nouveau dessin collaboratif a été créé.
join_drawing	{ drawing: string, user: string }	Événement permettant de signaler au client qu'un usager s'est joint au dessin collaboratif.
leave_drawing	{ drawing: string, user: string }	Événement permettant de signaler au client qu'un usager a quitté le dessin collaboratif.
new_version	{ versions: number }	Événement permettant de signaler au client qu'une nouvelle version du dessin collaboratif a été créée.
changed_version	{}	Événement permettant de signaler au client que la version du dessin collaboratif a changé.
create_line	{ line: VectorObject, user: string }	Événement permettant de signaler au client qu'un trait a été créé.
edit_line	{ name: string, user: string, id: string, line: VectorObject }	Événement permettant de signaler au client qu'un trait a été mis à jour.
edit_rectangle	{ name: string, user: string, id: string, rectangle: VectorObject }	Événement permettant de signaler au client qu'un rectangle a été mis à jour.
edit_ellipse	{ name: string, user: string, id: string, ellipse: VectorObject }	Événement permettant de signaler au client qu'une ellipse a été mise à jour.
edit_z	{ objects: Map<id: string, z: number> }	Événement permettant de signaler au client qu'une liste d'objets changent de strate (z).

create_rectangle	{ user: string, rectangle : Shape }	Événement permettant de signaler au client qu'un nouveau rectangle a été ajouté , identifié par son id.
draw_rectangle	{ name: string, point: Point, id: string }	Événement permettant de signaler au client qu'un nouveau rectangle a été modifié, identifié par son id
create_ellipse	{ user: string, ellipse: Shape }	Événement permettant de signaler au client qu'une nouvelle ellipse a été ajoutée , identifiée par son id.
draw_ellipse	{ name: string, point: Point, id: string }	Événement permettant de signaler au client qu'une nouvelle ellipse a été modifiée, identifiée par son id.
select	{ name: string, id : string }	Événement permettant de signaler au client qu'un trait est sélectionné, identifié par son id.
unselect	{name: string, id : string }	Événement permettant de signaler au client actif du dessin qu'un trait est désélectionné, identifié par son id.
delete	{ name: string, user: string, id : string }	Événement permettant de signaler au client qu'un trait a été supprimé.
join_team	{ user: string, status: string, team: string }	Événement permettant de signaler au client qu'un utilisateur tente de se joindre à une équipe de collaboration.
leave_team	{ user: string, team: string }	Événement permettant de signaler au client qu'un utilisateur tente de quitter une équipe de collaboration.

Interfaces définies

Response

Définition:

```
{
    title : string,
    message : string
}
```

Description:

Champ	**Description**
title	Si la requête est autorisée, le champ contient "Authorized", sinon retourne "Unauthorized".
message	message indiquant que la requête a été traitée avec succès ou retourne un message d'erreur.

User

Définition:

```
{
    email : string,
    password : string,
    firstname : string,
    lastname : string,
    username : string,
    avatar : string,
    lastConnection : number,
    LastDeconnecion : number,
    historiqueEdition : Array<string>,
    collaboration : Array<string>,
    ownership : Array<string>,
    teams : Array<string>,
    totalTimeCollab : number,
    isNamePublic : bool,
    isEmailPublic : bool
    lastAction : Array<Action>
}
```

Description:

Champ	**Description**
email	adresse courriel de l'usager.
password	mot de passe de l'usager.
firstname	prénom de l'usager.
lastname	nom de famille de l'usager.
username	nom d'utilisateur de l'usager.
avatar	lien pointant vers l'avatar de l'usager.
lastConnection	horodatage de la dernière connexion de l'usager.

LastDeconnecion	horodatage de la dernière déconnexion de l'usager.
historiqueEdition	liste d'historiques d'éditions de dessins de l'usager.
collaboration	liste de dessins collaboratifs dont l'usager fait partie.
ownership	liste de dessins collaboratifs créés par l'usager.
teams	liste d'équipes de collaboration dont l'usager fait partie.
totalTimeCollab	temps total que l'usager a passé en mode édition de dessins.
isNamePublic	nom et prénom peuvent être affichés au public si True. Sinon, le nom et prénom sont privés.
isEmailPublic	courriel peut être affiché au public si True. Sinon, le courriel est privé.
lastAction	liste d'actions faites par l'usager sur l'application.

Room

Définition:

```
{
    name  : string,
    owner : string,
    password : string,
    history : Array<Message>,
    users : Array<string>,
    connectedUsers :   Array<string>
}
```

Description:

Champ	**Description**
name	nom du canal. Ce champ doit être unique.
owner	propriétaire du canal.
password	mot de passe du canal. Ce champ est optionnel.
history	liste des messages du canal.

users	liste d’utilisateurs qui font partie du canal de discussion.
connectedUsers	liste d’utilisateurs en ligne qui font partie du canal de discussion.

Message

Définition:

{

content : string,

sender : string,

timestamp : number

}

Description:

Champ	**Description**
content	contenu du message.
sender	nom d’utilisateur de l’expéditeur.
timestamp	horodatage du message.

Team

Définition:

{

name : string,

owner : string,

users : Array<User>,

drawings : Array<Drawing>,

password : string,

status : Map<string, string>,

maxUsers : number,

bio : string

}

Description:

Champ	Description
name	nom de l'équipe de collaboration.
owner	nom d'utilisateur du propriétaire de l'équipe de collaboration.
users	noms d'utilisateurs faisant partie de l'équipe de collaboration.
drawings	liste de dessins collaboratifs dont le propriétaire est l'équipe de collaboration.
password	mot de passe pour rejoindre l'équipe de collaboration.
status	liste des statuts des membres de l'équipe de collaboration.
maxUsers	nombre d'utilisateurs maximum de l'équipe de collaboration.
bio	description de l'équipe de collaboration.

Drawing

Définition:

```
{
    name : string,
    owner : string,
    creationTimestamp : number,
    objects : Array<VectorObject>,
    privacy : enum { public, protected, private },
    password : string,
    preview : string,
    team : string | undefined,
    version : number,
    versions : number,
    z : number
}
```

Description:

Champ	Description
name	nom du dessin collaboratif.
owner	nom d'utilisateur du propriétaire du dessin collaboratif.
creationTimestamp	horodatage de la date de création du dessin collaboratif.

objects	Liste de traits du dessin collaboratif.

privacy	niveau de visibilité du dessin collaboratif. Ce champ peut être privé, protégé ou publique.
	
	enum privacy {
	public = 0
	protected = 1
	private = 2
	}

password	Mot de passe du dessin. Ce champ est seulement obligatoire lorsque le niveau de visibilité du dessin est défini à « protégé ».
preview	aperçu du dessin collaboratif.
team	l'équipe de collaboration si le dessin collaboratif a été créé par une équipe de collaboration, sinon null.
version	version courante du dessin collaboratif.
versions	nombre de versions existantes du dessin collaboratif.
z	strate du dernier élément existant du dessin collaboratif.

Point

Définition:

{

x : number,

y : number

}

Description:

Champ	**Description**
x	valeur sur l'axe des abscisses ou le déplacement sur l'axe des abscisses.
y	valeur sur l'axe des ordonnées ou le déplacement sur l'axe des ordonnées.

VectorObject

Définition:

{

id : string,

z : number,

color : string,

isSelected : bool,

strokeWidth : number,

matrix : Matrix

}

Description:

Champ	**Description**
id	identifiant unique de l'objet.
z	strate de l'objet.
color	couleur de l'objet.
isSelected	True si l'objet est sélectionnée, Sinon False.
strokeWidth	épaisseur de la bordure de l'objet.
matrix	matrice de transformation de l'objet.

Shape extends VectorObject

Définition:

{

id : string,

z : number,

color : string,

isSelected : bool,

strokeWidth : number,

matrix : Matrix,

strokeColor : string,

initialPoint : Point,

finalPoint : Point,

text : string,

textColor : string

}

Description:

Champ	Description
id	identifiant unique de l'objet.
z	strate de la forme.
color	couleur de fond de la forme.
isSelected	True si la forme est sélectionnée, Sinon False.
strokeWidth	épaisseur de la bordure de la forme.
matrix	Matrice de transformation de la forme.
strokeColor	couleur de la bordure de la forme.
initialPoint	Ce champ représente le point initial du rectangle encadrant la forme.
finalPoint	Ce champ représente le point final du rectangle encadrant la forme.
text	texte contenu dans la forme.
textColor	couleur du texte contenu dans la forme.

Line extends VectorObject

Définition:

{

id : string,

z: number,

color : string,

isSelected: bool,

strokeWidth: number,

matrix: Matrix

points : Array<Point>

}

Description:

Champ	Description
id	identifiant unique du trait.
z	strate du trait.
color	couleur du trait.
isSelected	True si le trait est sélectionné, Sinon False.
strokeWidth	épaisseur du trait.
matrix	matrice de transformation du trait.
points	liste de points du trait.

TypedShape extends Shape

Définition:

```
{
    id : string,
    z : number,
    color : string,
    isSelected : bool,
    strokeWidth : number,
    matrix : Matrix,
    strokeColor : string,
    initialPoint : Point,
    finalPoint : Point,
    text : string,
    textColor : string,
    type : string
}
```

Description:

Champ	Description
id	identifiant unique de l'objet.
z	strate de la forme.
color	couleur de la forme.
isSelected	True si la forme est sélectionnée, Sinon False.
strokeWidth	épaisseur de la bordure de la forme.
matrix	Matrice de transformation de la forme.
strokeColor	couleur de la bordure de la forme.
initialPoint	point initial de la forme encadrant la forme.
finalPoint	point final de la forme encadrant la forme.
text	texte contenu dans la forme.
textColor	couleur du texte contenu dans la forme.
type	type de la forme, soit rectangle ou ellipse.

Matrix

Définition:

```
{
    a : number,
    b : number,
    c : number,
    d : number,
    e : number,
    f : number
}
```

Description:

Champ	Description
a	première valeur dans la matrice de transformation de l'objet.
b	deuxième valeur dans la matrice de transformation de l'objet.
c	troisième valeur dans la matrice de transformation de l'objet.

d	quatrième valeur dans la matrice de transformation de l'objet.
e	cinquième valeur dans la matrice de transformation de l'objet.
f	sixième valeur dans la matrice de transformation de l'objet.

Action

Définition:

```
{
    action : string,
    timestamp : number,
    drawing : string | null
}
```

Description:

Champ	**Description**
action	désigne l'action faite par l'utilisateur, soit : connexion, déconnexion, ou éditer un dessin.
timestamp	horodatage de l'action.
drawing	le nom du dessin si l'action est *éditer un dessin*, sinon il est laissé par défaut à null.