Documentation API http rest de Yag

Version de ce document

Généralités
- Pourquoi un serveur web ?
- REST
- But du document

Serveur web

Architecture globale
- Sans le serveur web
- Avec le serveur web
  - Demande d'informations locales
  - Demande d'informations à distance
- URL publique et locale

Yag et Javascript
- Les objets JSON de l'API
- Requêtes AJAX
- Fonction javascript type
- Chaîne de traitement d'une requête
- Encodage URL
  - Exemple manuel

API REST
- Aide-mémoire
- Les pions
  - /yagapi/pawn/get/guid
  - /yagapi/pawn/get/id
  - /yagapi/pawn/get/selected/firstvisible
  - /yagapi/pawn/get/selected/firstfromcurrentcs
  - /yagapi/pawn/set/ids
  - /yagapi/pawn/set/current
  - /yagapi/pawn/attack
  - /yagapi/pawn/get/selected/list
  - /yagapi/pawn/select/nextincharactersheet
  - /yagapi/pawn/select/nextvisible
  - /yagapi/pawn/select/id
  - /yagapi/pawn/preview/id
  - /yagapi/pawn/preview/firstselected
  - /yagapi/pawn/goto/id
  - /yagapi/pawn/goto/firstselected

- Les feuilles de personnage
  - /yagapi/charactersheet/select/guid
  - /yagapi/charactersheet/get/list
  - /yagapi/charactersheet/get/current
  - /yagapi/charactersheet/set/current
  - /yagapi/charactersheet/duplicate/guid
  - /yagapi/charactersheet/create
- Les dés
  - /yagapi/dice/clear
  - /yagapi/dice/roll/panel
  - /yagapi/dice/roll/init/all
  - /yagapi/dice/roll/init/current
  - /yagapi/dice/roll/line/current
  - /yagapi/dice/load
- Le journal
  - /yagapi/journal/get/all
  - /yagapi/journal/get/last
  - /yagapi/journal/get/at
  - /yagapi/journal/send/public
  - /yagapi/journal/send/local
- Le tour
  - /yagapi/initiative/get/list
  - /yagapi/initiative/down
  - /yagapi/initiative/up
  - /yagapi/initiative/reset

Version de ce document

Si vous avez trouvé ce document dans votre installation de Yag, vous aurez peut être besoin de récupérer la dernière version ici:
http://yagame.fr/documentation/

Généralités

Yag dispose d'un serveur http interne qu'on appellera par la suite indistinctement "serveur web", "serveur http", ou même simplement "Yag" quand le contexte est évident.
On dira donc "Yag traite la requête" pour dire exactement "Le serveur http interne de Yag traite la requête".

Ce serveur est complet et peut donc gérer un site web.

Pourquoi un serveur web ?

Intégrer un serveur web complet offre 2 possibilités importantes:

- On peut créer n'importe quel site
- On peut interfacer ce serveur directement avec Yag

La première possibilité permet d'utiliser les technologies standard du web (html / css / javascript…) ce qui enrichit grandement les possibilités d'automatisation de Yag.

La deuxième possibilité permet d'interroger Yag pour modifier/récupérer des données internes du jeu comme les pions, les feuilles de personnage, les dés, etc.

Ces deux possibilités combinées permettent de piloter certaines fonctionnalités de Yag avec du Javascript.

REST

L'interrogation de Yag doit bien sûr respecter un format.
Le format standard pour les requêtes http est le REST (REpresentational State Transfert).
Dans notre contexte, en pratique, ça signifie simplement que les URL des requêtes doivent être organisées et lisibles sous la forme d'une arborescence.
Par exemple, toutes les requêtes s'adressant à l'API commenceront par /yagapi/
On spécifiera ensuite le contexte général, par exemple:

- Les requêtes concernant les pions commenceront par /yagapi/pawn
- Les requêtes concernant les dés commenceront par /yagapi/dice

Plus on avance dans la requête plus on est précis.

Par exemple la requête /yagapi/dice/roll/init/current signifie:

- Qu'on s'adresse à l'API Yag (yagapi)
- Qu'on s'intéresse aux dés (dice)
- Qu'on va demander un lancer (roll)
- Qu'on va lancer l'initiative (init)

But du document

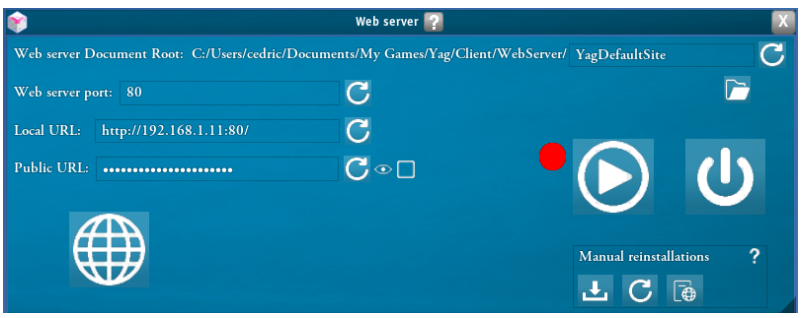
Ce document décrit:

- l'utilisation du serveur web de Yag
- les spécifications complètes de l'API REST de Yag

Serveur web

Ce serveur est complet et peut donc gérer un site web.
Il s'agit d'une intégration du serveur web mongoose.

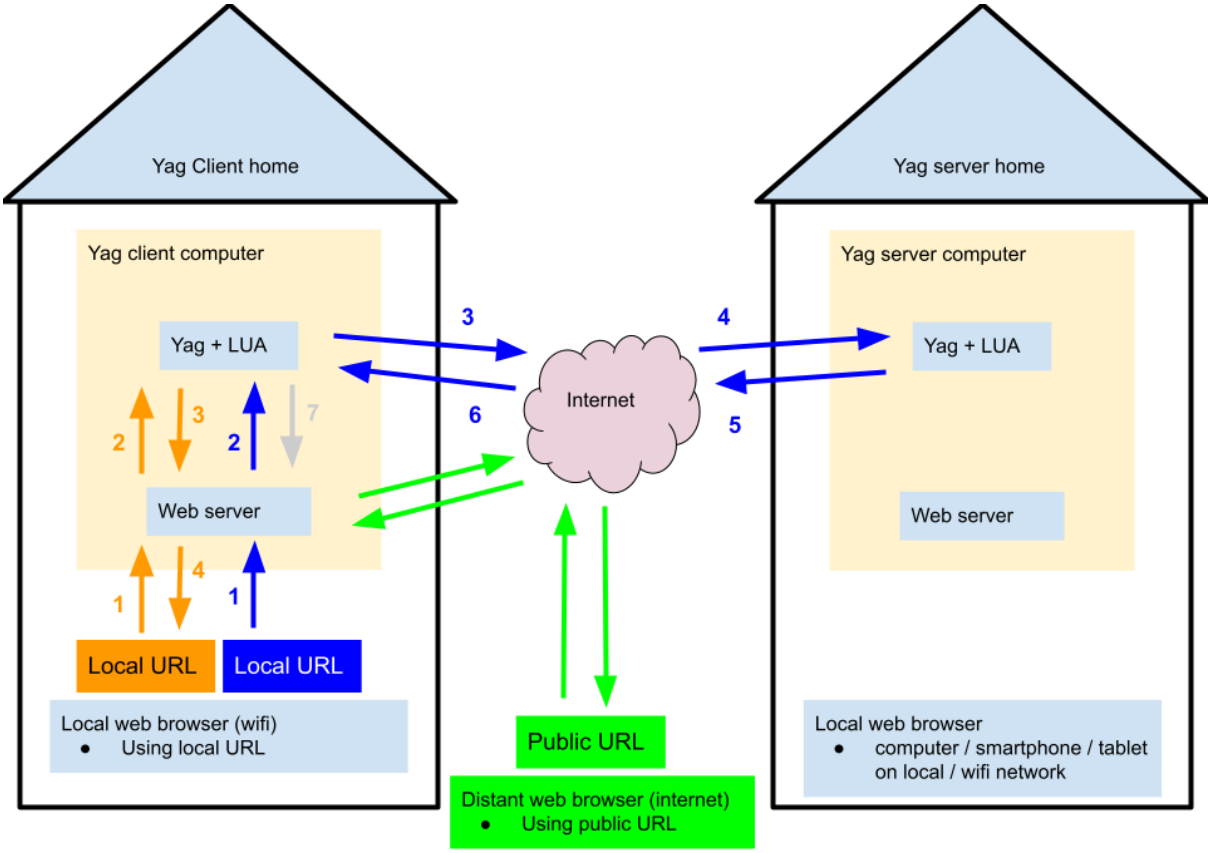
Il est géré dans la fenêtre "web server":



Consulter le pdf de documentation de cette fenêtre pour les détails de fonctionnement et d'utilisation: https://yagame.fr/documentation/

Architecture globale

Il peut être utile de comprendre l'architecture globale du fonctionnement de Yag en termes de réseau.
Cette section entre un peu dans les détails techniques et, même s'il est utile de comprendre ce qui se passe en interne, elle peut être ignorée sans problème si vous n'êtes pas trop intéressé par la technique.



Sans le serveur web

L'essentiel à comprendre est qu'un client Yag ne stocke pas toutes les informations sur le jeu.
En particulier, toutes les feuilles de personnage sont stockées sur le serveur et le client ne peut en obtenir qu'une à la fois.
En effet, le volume de données dans les feuilles de perso est trop important pour être partagé entièrement (cela conduit à des crashes lorsqu'il y a trop de feuilles de personnage).
Ainsi lorsqu'un client Yag a besoin d'une feuille de perso, il envoie une requête interne au serveur Yag qui renvoie la feuille au client (flèches bleues 3, 4, 5, 6 sur l'image).
Tout cela est transparent pour l'utilisateur lorsqu'il n'utilise pas le serveur Web : Yag fait tranquillement sa cuisine interne et l'utilisateur ne s'en rend jamais compte.

Avec le serveur web

Les choses se compliquent un peu lors de l'utilisation du serveur Web.
C'est parce que http est un protocole sans état (stateless) et ne peut pas maintenir une connexion.

Demande d'informations locales

Lorsque le navigateur demande des données locales, il n'y a pas de problème.
Ceci est illustré par les flèches oranges 1 à 4.
Dans ce qui suit, O1 signifie « flèche orange 1 », etc.
O1- Le navigateur envoie une requête API REST au serveur Web
O2- Le serveur Web demande les informations à Yag
O3- Yag peut répondre instantanément car les informations sont stockées localement
O4- Le navigateur obtient sa réponse dans la même connexion
C'est pourquoi les requêtes API REST interrogeant des informations locales peuvent renvoyer leurs résultats dans la même requête (et les résultats peuvent être traités directement dans la fonction callback de la requête)

Ainsi, en pratique, la demande de données locales ne nécessite qu'une seule requête Web.

Demande d'informations à distance

Lorsque le navigateur demande des données distantes, les choses sont plus compliquées.
Ceci est illustré par les flèches bleues 1 à 6
Dans ce qui suit, B1 signifie « flèche bleue 1 », etc.
B1- le navigateur envoie une requête API REST au serveur Web
B2- le serveur Web demande les informations à Yag
B3- Yag n'a pas les informations et envoie une requête au serveur Yag
Parce que B3 traverse le net, de nombreuses frames de jeu peuvent passer et la connexion B1 est perdue : B8 n'existe pas et B7 est inutile (donc grisé).
B4, B5, B6- Le serveur Yag reçoit la demande et renvoie les données au client Yag.

À ce stade, le client Yag a les informations localement mais ne peut pas les renvoyer au navigateur car B8 n'existe plus.

Et parce que Yag ne peut pas initier une connexion au navigateur Web, le navigateur Web doit créer une nouvelle demande pour obtenir les informations (désormais) locales via un cycle local (flèches oranges).

Ainsi, en pratique, demander des données distantes prend deux requêtes Web.

C'est pourquoi lorsque javascript a besoin d'une feuille de personnage, deux requêtes sont nécessaires :

- /yagapi/charactersheet/select/guid pour sélectionner la feuille cible et récupérer ses données depuis le serveur (flèches bleues)

- /yagapi/charactersheet/get/current pour obtenir la feuille (désormais) courante du client (flèches oranges)

Voir l'exemple javascript pour voir comment cela fonctionne dans la pratique.

URL publique et locale

Accessoirement, le schéma ci dessus illustre également la différence d'utilisation entre l'URL locale et l'URL publique dans la fenêtre du serveur web:

L'URL locale peut être utilisée sur le réseau interne de la maison (wifi, etc.), ce sont les flèches oranges et les les flèches B1 et B2.
L'URL publique doit être utilisée depuis internet, ce sont les flèches vertes.

En dehors du fait qu'il faut autoriser la connexion entrante sur l'ordinateur (NAT, port forwarding, pare-feu, etc.), les flèches vertes sont exactement équivalentes aux flèches oranges: tant que rien n'est demandé au serveur Yag la connexion verte n'est pas perdue et fonctionne comme une connexion locale.

Yag et Javascript

Afin de garantir l'application des scripts LUA définis, Yag appelle les fonctions du LUA en interne.
Les structures utilisées pour l'API Rest Javascript sont donc calquées sur celles du LUA.

Les objets JSON de l'API

Les structures utilisées pour le Javascript sont identiques à celles utilisées pour le LUA, mais au format JSON.
Par exemple, la structure ps est définie ainsi dans la documentation sur l'API LUA:

```
{
  health = 10,
  initiative = 14,
  round = true,
  righthand = true,
  lefthand = true
}
```

Dans l'API Javascript elle se présentera donc comme un objet JSON contenant les mêmes champs:

```
{
  "health" : 10,
  "initiative" : 14,
  "round" : true,
  "righthand" : true,
  "lefthand" : true
}
```

Et identiquement pour toutes les structures utilisées pour l'API Javascript.
Consultez la documentation de l'API LUA pour la description complète des structures disponibles:
http://yagame.fr/wp-content/uploads/2019/02/yag_lua_api_fr.pdf

Requêtes AJAX

L'interrogation de Yag depuis le Javascript se fait de façon asynchrone, avec des requêtes AJAX (Asynchronous Javascript And Xml).

Il n'y a donc pas d'état connecté: chaque requête crée une nouvelle connexion qui disparaît après le traitement de la requête.

Ce fonctionnement impose une contrainte importante: Yag ne peut pas créer une connexion vers le navigateur, c'est toujours le navigateur qui doit initier une requête, et attendre la réponse de Yag sur la connexion créée pour cette requête.

Yag ne peut donc pas signaler au navigateur de se mettre à jour en dehors du contexte de la connexion créée par une requête.

Par exemple pour le journal, Yag ne peut pas signaler au navigateur qu'il a été mis à jour, c'est au navigateur de créer des requêtes périodiques (par exemple toutes les secondes) pour vérifier s'il doit mettre à jour son affichage du journal.

Autre exemple: si vous sélectionnez un pion en cliquant dessus dans Yag, vous n'avez aucun moyen de mettre automatiquement à jour l'affichage dans le navigateur, il faudra soit prévoir un bouton de mise à jour soit faire une interrogation périodique, par exemple toutes les secondes, pour que le navigateur sache toujours quel est le pion actuellement sélectionné et éventuellement tienne son affichage à jour.

Lorsque vous modifiez un résultat, il faut appeler la fonction de mise à jour de l'affichage dans le script. Par exemple, quand on lance l'initiative en javascript, il faut prévoir la mise à jour du pion dans la callback:

```
function DiceRollCSInit() {

    // nouvelle requête ajax
    var xhr = new XMLHttpRequest();

    // préparation de la callback
    xhr.onreadystatechange = function () {
        if (xhr.readyState === 4 && xhr.status === 200) {
            PawnSheetGetById(document.getElementById("CSPawnId").value)
        }
    };

    // lancement de la requête ajax
    xhr.open('POST', '/yagapi/dice/roll/init/current');
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded", "charset=UTF-8");
    var myFormData = "Guid=" + encode(document.getElementById("CSGuid").value);
    xhr.send(myFormData);
}
```

C’est aussi la raison pour laquelle l’affichage des feuilles de perso se fait en 2 temps. Il est techniquement impossible de faire une requête unique, comme celle qui suit, pour récupérer les fiches:

- /yagapi/charactersheet/get/guid

La récupération de la feuille de perso se fait sur le serveur Yag, et c’est aussi une requête asynchrone entre le client Yag et le serveur Yag. Lorsque le client demande la fiche au serveur, la connexion http se termine et le client Yag ne peut plus renvoyer au navigateur la fiche qu’il vient de récupérer.
On a donc 2 requêtes différentes:

- /yagapi/charactersheet/select/guid
- /yagapi/charactersheet/get/current

La première sélectionne la fiche (en coulisse, le client Yag la demande au serveur Yag).
La deuxième renvoie la fiche courante (elle est maintenant présente localement sur le client Yag qui n’a plus besoin de la demander au serveur).

En bref, pour tout ce qui peut changer en cours de jeu (liste des feuilles de perso, journal, etc.), il faut prévoir des mises à jour manuelles (avec un bouton dédié) ou périodiques (automatisées en Javascript) car Yag n’a aucun moyen de contacter le navigateur, il ne peut que répondre.

Fonction javascript type

Voici un exemple de fonction javascript typique utilisant l’API

La fonction prend en paramètre l’identifiant du pion (pid), génère une requête AJAX avec l’API REST Yag correspondante (/yagapi/pawn/get/id), et met à jour l’affichage via la mise en place d’une fonction callback.

```
function PawnSheetGetById(pid) {

   // création de la requête AJAX
   var xhr = new XMLHttpRequest();

   // Préparation de la fonction callback
   xhr.onreadystatechange = function () {
      if (xhr.readyState === 4 && xhr.status === 200) {
         // Décodage et analyse de la réponse
         var DecodedResponse = decode(xhr.responseText);
         var PawnInfoJson = JSON.parse(DecodedResponse);
         var PSJson = PawnInfoJson["ps"];

         // Remplissage des champs de la page HTML
         document.getElementById("CSRound").checked = PSJson.round;
         document.getElementById("CSCurrentHealth").value = PSJson.health;
         document.getElementById("CSInitResult").value = PSJson.initiative;
         document.getElementById("CSShowLeftHand").checked = PSJson.lefthand;
         document.getElementById("CSShowRightHand").checked = PSJson.righthand;

         document.getElementById("CSPawnId").value = pid;
      }
   };

   // Préparation et envoi de la requête
```

```
    var myFormData = "id=" + encode(pid);
    xhr.open('GET', '/yagapi/pawn/get/id?' + myFormData);
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded", "charset=UTF-8");
    xhr.send();
}
```

Chaîne de traitement d'une requête

Avec la fonction ci dessus, le traitement complet d'une requête de l'API est le suivant:

1. *Navigateur (javascript):* envoi de la requête API Rest /yagapi/pawn/get/id?id=#3
2. *Serveur http interne de Yag (c++)*: réception de la requête et transfert vers le code Yag (c++ et API LUA)
3. *Code Yag (c++ et API LUA)*: traitement de la requête
    a. récupération de la ps du pion #3
    b. envoi du résultat au serveur http interne
4. *Serveur http interne de Yag (c++)*: transformation du résultat en objet JSON et envoi du résultat au navigateur
5. *Navigateur (javascript)*: réception, traitement et affichage du résultat

Encodage URL

Afin d'être capable de gérer les caractères spéciaux dans les requêtes, Yag ne travaille qu'avec des données URL-encodées.
Il faudra donc prendre soin d'encoder les données avant de les envoyer, et de les décoder après réception.
Dans la fonction ci dessus:

- Encodage
    - var myFormData = "id=" + **encode**(pid);
- Décodage
    - var DecodedResponse = **decode**(xhr.responseText);

C'est aussi la raison pour laquelle les requêtes AJAX devront toujours signaler dans leur header que leur contenu est encodé. La ligne suivante devra donc systématiquement être utilisée:
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded", "charset=UTF-8");

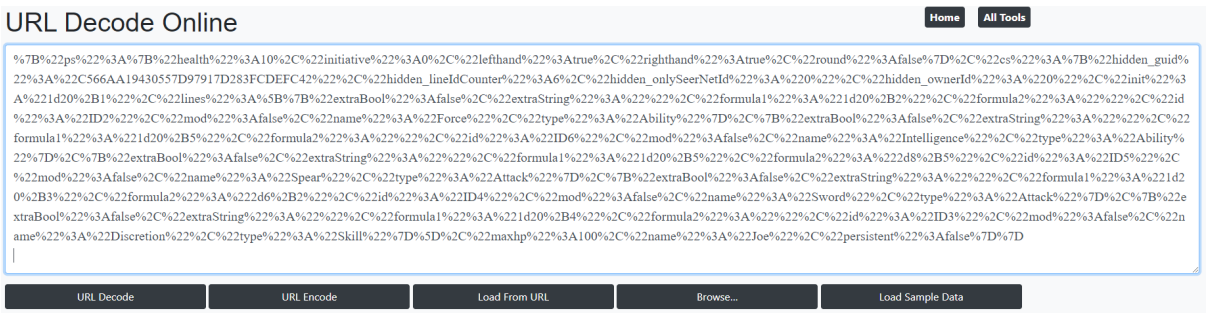
Exemple manuel

Par exemple pour récupérer la ps du pion identifié par #3, il faudrait en principe envoyer à Yag la requête suivante:
http://localhost/yagapi/pawn/get/id?id=#3
Mais pour gérer le caractère spécial #, les données doivent être URL-encodés.

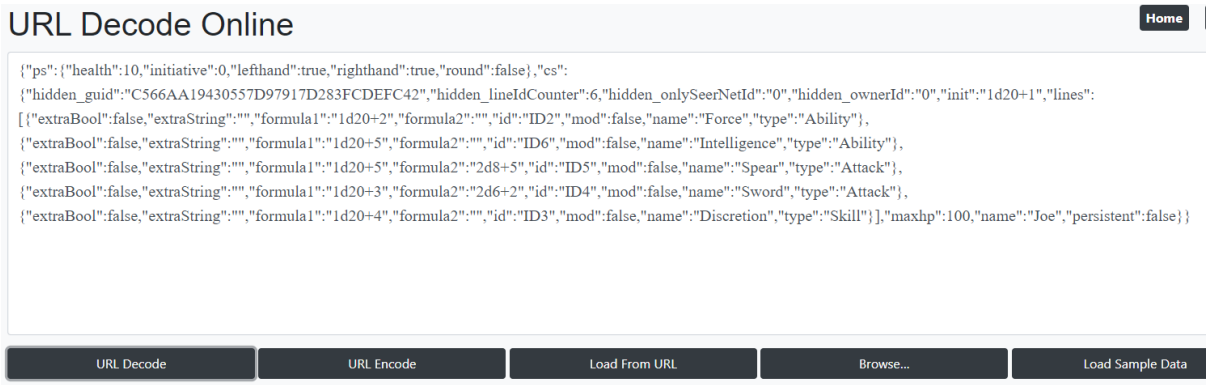
- non URL-encodé: #3
- URL-encodé: %233

Si l'on veut interroger Yag manuellement, il faut donc lui envoyer la requête suivante:
http://localhost/yagapi/pawn/get/id?id=%233

Yag renvoie alors le résultat de la requête sous forme URL-encodée:



Qu'on peut décoder avec n'importe quel site d'encodage/décodage pour retrouver un objet JSON lisible par un humain:



Bien sûr, en pratique, on utilisera toujours les fonctions javascript encode() et decode() suivantes:

```
function encode(TargetStr) {
   return encodeURIComponent(TargetStr).replace(/'/g, "%27").replace(/"/g, "%22");
}

function decode(TargetStr) {
   return decodeURIComponent(TargetStr.replace(/\+/g, " "));
}
```

API REST

Aide-mémoire

Ces sigles ont été inventés pour Yag, à l'exception du GUID qui est un sigle officiel.
Le GUID est un identifiant unique global.
C'est une longue chaîne de caractères qui ressemble à ça:
3F2504E0-4F89-11D3-9A0C-0305E82C3301
Yag utilise en interne quelques guid pour identifier les feuilles de perso, les pions, etc.
Le code javascript peut en avoir besoin dans ses requêtes.
Il existe donc des requêtes qui renvoient un guid, qu'on peut alors utiliser dans d'autres requêtes.

Sigle	**Signification**	**Traduction**
ps	Pawn Sheet	Feuille de Pion

cs	Character Sheet	Feuille de Personnage
guid	Global Unique IDentifier	Identifiant Global Unique
pguid	Pawn Guid	Guid de Pion
csguid	Cs Guid	Guid de la cs
id	IDentifier	Identifiant
sp	Selected Pawns	Pions Sélectionnés
dr	Dice Result	Résultat de Dés

Les pions

/yagapi/pawn/get/guid

- C'est une requête GET
- Elle renvoie la ps du pion cible, son identifiant id, et l'identifiant csguid de sa cs.

Appel

- Elle n'attend qu'un seul paramètre: pguid
  - var myFormData = "pguid=" + encode(pGuid);
  - xhr.open('GET', '/yagapi/pawn/get/guid?' + myFormData);
  - xhr.send();
- pGuid est le guid du pion cible

Retour

- Elle renvoie une chaîne JSON contenant la ps, l'id, et le csguid du pion cible.
  - {"ps" : <ps du pion>,"id" : <id du pion>, "csguid" : <guid de la cs du pion>}

/yagapi/pawn/get/id

- C'est une requête GET
- Elle renvoie la ps du pion cible, son pguid, et l'identifiant csguid de sa cs.

Appel

- Elle n'attend qu'un seul paramètre: id
  - var myFormData = "id=" + encode(pid);
  - xhr.open('GET', '/yagapi/pawn/get/id?' + myFormData);
  - xhr.send();
- id est l'identifiant du pion cible

Retour

- Elle renvoie une chaîne JSON contenant la ps, le pguid, et le csguid du pion cible.
  - {"ps" : <ps du pion>,"pguid" : <guid du pion>, "csguid" : <guid de la cs du pion>}

/yagapi/pawn/get/selected/firstvisible

- C'est une requête GET

- Elle renvoie la ps, l'identifiant pguid du premier pion visible sélectionné, l'identifiant csguid de sa cs, et son id.

Appel

- Elle n'attend aucun paramètre
    - xhr.open('GET', '/yagapi/pawn/get/selected/firstvisible');
    - xhr.send();

Retour

- Elle renvoie une chaîne JSON contenant la ps et la cs du premier pion sélectionné.
    - {"ps" : <ps du pion>,"pguid" : <guid du pion>, "csguid" : <guid de la cs du pion>, "id" : <id du pion>}

/yagapi/pawn/get/selected/firstfromcurrentcs

- C'est une requête GET
- Elle renvoie la ps, l'identifiant pguid du premier pion sélectionné de la cs courante, et son id.

Appel

- Elle n'attend aucun paramètre
    - xhr.open('GET', '/yagapi/pawn/get/selected/firstfromcurrentcs');
    - xhr.send();

Retour

- Elle renvoie une chaîne JSON contenant la ps et l'id du premier pion sélectionné de la cs courante.
    - {"ps" : <ps du pion>,"pguid" : <guid du pion>, "id" : <id du pion>}

/yagapi/pawn/set/ids

- C'est une requête POST
- Elle met à jour la ps des pions de la liste ids.

Appel

- Elle attend deux paramètres: la ps cible et la liste ids des id des pions cibles
    - var PSJson = {};
    - PSJson["health"] = document.getElementById("CSCurrentHealth").value;
    - [...]
    - var IDSJson = ["#2", "#4", "JaneID"];
    -
    - xhr.open('POST', '/yagapi/pawn/set/ids');
    - var myFormData = "ps=" + encode(JSON.stringify(PSJson)) + "&" + "ids=" + encode(JSON.stringify(IDSJson));
    - xhr.send(myFormData);

Retour

- Elle ne renvoie rien

/yagapi/pawn/set/current

- C'est une requête POST
- Elle met à jour la ps et l'id du pion courant.

Appel

- Elle attend 3 paramètres: la ps cible, le pguid du pion, et son id cible.
    - xhr.open('POST', '/yagapi/pawn/set/current');
    - var myFormData = "ps=" + encode(JSON.stringify(PSJson)) + "&" + "id=" + encode(pid) + "&" + "pguid=" + encode(pGuid);
    - xhr.send(myFormData);
- PSJson est la ps cible au format JSon
- pGuid est le guid du pion. Il est nécessaire pour que Yag puisse vérifier que le pion courant sélectionné est bien le pion cible.
- pid est le nouvel id du pion cible

Retour

- Elle ne renvoie rien

/yagapi/pawn/attack

- C'est une requête POST
- Elle lance l'animation d'attaque des pions de la liste ids.

Appel

- Elle attend un paramètre: la liste ids des id des pions cibles
    - var IDSJson = ["#2", "#4", "JaneID"];
    - xhr.open('POST', '/yagapi/pawn/attack);
    - var myFormData = "ids=" + encode(JSON.stringify(IDSJson));
    - xhr.send(myFormData);

Retour

- Elle ne renvoie rien

/yagapi/pawn/get/selected/list

- C'est une requête GET
- Elle renvoie la liste des pions sélectionnés

Appel

- Elle n'attend aucun paramètre
    - xhr.open('GET', '/yagapi/pawn/get/selected/all');
    - xhr.send();

Retour

- Elle renvoie une chaîne JSON contenant la liste sp des pions sélectionnés.
    - {"sp":["#1","#2","#3"]}

/yagapi/pawn/select/nextincharactersheet

- C'est une requête GET
- Elle sélectionne le pion suivant de la feuille de personnage courante.

Appel

- Elle n'attend aucun paramètre
    - xhr.open('GET', '/yagapi/pawn/select/nextincharactersheet');
    - xhr.send();

Retour

- Elle ne renvoie rien

/yagapi/pawn/select/nextvisible

- C'est une requête GET
- Elle sélectionne le pion visible suivant.

Appel

- Elle n'attend aucun paramètre
  - xhr.open('GET', '/yagapi/pawn/select/nextvisible');
  - xhr.send();

Retour

- Elle ne renvoie rien

/yagapi/pawn/select/id

- C'est une requête GET
- Elle sélectionne le pion identifié par son id.

Appel

- Elle n'attend qu'un seul paramètre: id
  - var myFormData = "id=" + encode(pid);
  - xhr.open('GET', '/yagapi/pawn/select/id?' + myFormData);
  - xhr.send();
- id est l'identifiant du pion cible

Retour

- Elle ne renvoie rien

/yagapi/pawn/preview/id

- C'est une requête GET
- Elle prévisualise le pion identifié par son id.

Appel

- Elle n'attend qu'un seul paramètre: id
  - var myFormData = "id=" + encode(pid);
  - xhr.open('GET', '/yagapi/pawn/preview/id?' + myFormData);
  - xhr.send();
- id est l'identifiant du pion cible

Retour

- Elle ne renvoie rien

/yagapi/pawn/preview/firstselected

- C'est une requête GET
- Elle prévisualise le premier pion sélectionné.

Appel

- Elle n’attend aucun paramètre

Retour

- Elle ne renvoie rien

/yagapi/pawn/goto/id

- C’est une requête GET
- Elle envoie le joueur sur le pion identifié par son id.

Appel

- Elle n’attend qu’un seul paramètre: id
    - var myFormData = "id=" + encode(pid);
    - xhr.open('GET', '/yagapi/pawn/select/id?' + myFormData);
    - xhr.send();
- id est l’identifiant du pion cible

Retour

- Elle ne renvoie rien

/yagapi/pawn/goto/firstselected

- C’est une requête GET
- Elle envoie le joueur sur le premier pion sélectionné.

Appel

- Elle n’attend aucun paramètre

Retour

- Elle ne renvoie rien

Les feuilles de personnage

/yagapi/charactersheet/select/guid

- C’est une requête GET
- Elle sélectionne la cs identifiée par le csguid donné en paramètre.

Appel

- Elle n’attend qu’un seul paramètre: le guid csguid de la cs cible
    - var myFormData = "csguid =" + csguid;
    - xhr.open('GET', '/yagapi/charactersheet/select/guid?' + myFormData);
    - xhr.send(myFormData);
- csguid est l’identifiant de la feuille de perso cible

Retour

- Elle ne renvoie rien

/yagapi/charactersheet/get/list

- C'est une requête GET
- Elle renvoie la liste des feuilles de perso visibles pour le joueur

Appel

- Elle n'attend aucun paramètre
    - xhr.open('GET', '/yagapi/charactersheet/get/list');
    - xhr.send(myFormData);

Retour

- Elle renvoie une chaîne JSON contenant la liste CharacterSheets des couples {"Guid", "Name"} de feuilles de perso visibles:
    - {"CharacterSheets":[{"Guid":"4D0E6A7F49A961770FCC749C3DB72A5D","Name":"Jane"},{"Guid":"C566AA19430557D97917D283FCDEFC42","Name":"Joe"}]}
- Dans chaque objet JSON retourné:
    - Guid est l'identifiant (interne) de la feuille de perso (nécessaire pour créer en javascript un bouton pouvant sélectionner cette feuille de perso)
    - Name est le nom de la feuille de perso

/yagapi/charactersheet/get/current

- C'est une requête GET
- Elle renvoie la cs courante.

Appel

- Elle n'attend aucun paramètre
    - xhr.open('GET', '/yagapi/charactersheet/get/current);
    - xhr.send(myFormData);

Retour

- Elle renvoie une chaîne JSON contenant la cs courante, son csguid, le compteur d'id des lignes de la cs cslineidcounter, la liste ids des pions attachés à la cs, et la liste sp des pions sélectionnés.
    - {"cs" : <cs courante>, "csguid": <guid de la cs>, "cslineidcounter": <compteur d'id des lignes>, "ids" : <id des pions de la cs>, "sp" : <id des pions sélectionnés>}

/yagapi/charactersheet/set/current

- C'est une requête POST
- Elle met à jour la cs passée en paramètre

Appel

- Elle attend 3 paramètres cs, csguid cible, et cslinecounter.
    - xhr.open('POST', '/yagapi/charactersheet/set/current');
    - var myFormData = "cs=" + encode(JSON.stringify(CSJson)) + "&" + "csguid=" + encode(CsGuid) + "&" + "cslinecounter=" + encode(LineIdCounter);
    - xhr.send(myFormData);
- CSJson est la cs cible au format JSON
- CsGuid est le csguid cible. Il est nécessaire de l'envoyer pour que Yag puisse vérifier que la cs cible est bien celle actuellement sélectionnée.
- LineIdCounter (integer) est la valeur actuelle du compteur d'id

Retour

- Elle ne renvoie rien

/yagapi/charactersheet/duplicate/guid

- C'est une requête GET
- Elle duplique la cs identifiée par le csguid donné en paramètre.

Appel

- Elle n'attend qu'un seul paramètre: CSGuid
    - var myFormData = "CSGuid=" + csguid;
    - xhr.open('GET', '/yagapi/charactersheet/duplicate/guid?' + myFormData);
    - xhr.send(myFormData);
- CSGuid est l'identifiant de la feuille de perso cible

Retour

- Elle ne renvoie rien

/yagapi/charactersheet/create

- C'est une requête GET
- Elle crée une nouvelle cs.

Appel

- Elle n'attend aucun paramètre

Retour

- Elle ne renvoie rien

Les dés

/yagapi/dice/clear

- C'est une requête GET
- Elle arrête le jet de dés en cours et supprime les dés lancés (procédure d'arrêt d'urgence des jets de dés)

Appel

- Elle n'attend aucun paramètre
    - xhr.open(GET, '/yagapi/charactersheet/set');
    - xhr.send();

Retour

- Elle ne renvoie rien

/yagapi/dice/roll/panel

- C'est une requête POST
- Elle lance les dés pour les formules dans le panneau des dés

Appel

- Elle attend 2 paramètres TryFormula et EffectFormula:
    - var myFormData1 = "TryFormula=" + encode(document.getElementById("TryFormula").value);
    - var myFormData2 = "EffectFormula=" + encode(document.getElementById("EffectFormula").value);
    -
    - xhr.open('POST', '/yagapi/dice/roll/panel');
    - var myFormData = myFormData1 + "&" + myFormData2;
    - xhr.send(myFormData);
- TryFormula et EffectFormula sont respectivement les formules pour l'essai et le résultat (typiquement toucher/dégats)

Retour

- Elle renvoie un objet roll contenant les 2 résultats Try et Effect:
    - roll = {"Effect" : <dr de l'effet>, "Try" : <dr de l'essai>}
- dr est la structure LUA d'un résultat de dés

/yagapi/dice/roll/init/all

- C'est une requête GET
- Elle lance l'initiative pour tous les pions participant au tour

Appel

- Elle n'attend aucun paramètre
    - xhr.open(GET, '/yagapi/dice/roll/init/all');
    - xhr.send();

Retour

- Elle ne renvoie rien

/yagapi/dice/roll/init/current

- C'est une requête GET
- Elle lance l'initiative pour la feuille de perso courante.

Appel

- Elle n'attend aucun paramètre
    - xhr.open(GET, '/yagapi/dice/roll/init/current');
    - xhr.send();

Retour

- Elle renvoie un objet roll contenant les 2 résultats Try et Effect:
    - roll = {"Effect" : <dr de l'effet>, "Try" : <dr de l'essai>}
- dr est la structure LUA d'un résultat de dés

/yagapi/dice/roll/line/current

- C'est une requête POST
- Elle lance les dés pour la ligne LineId de la feuille de perso courante.

Appel

- Elle attend 1 paramètre LineId:
    - xhr.open('POST', '/yagapi/dice/roll/line/current');
    - var myFormData = "LineId=" + encode(TargetLineId);
    - xhr.send(myFormData);
- LineId est l'identifiant de la ligne dans la feuille de perso

Retour

- Elle renvoie un objet roll contenant les 2 résultats Try et Effect:
    - roll = {"Effect" : <dr de l'effet>, "Try" : <dr de l'essai>}
- dr est la structure LUA d'un résultat de dés

/yagapi/dice/load

- C'est une requête POST
- Elle charge un jeu de dés.

Appel

- Elle attend 1 paramètre ds:
    - xhr.open('POST', '/yagapi/dice/load');
    - var myFormData = "ds=" + encode(JSON.stringify(DSJson));
    - xhr.send(myFormData);
- DSJson est le nom du jeu de dés à charger

Retour

- Elle ne renvoie rien

Le journal

/yagapi/journal/get/all

- C'est une requête GET
- Elle renvoie le contenu complet du journal.

Appel

- Elle n'attend aucun paramètre
    - xhr.open('GET', '/yagapi/journal/get/all');
    - xhr.send();

Retour

- Elle renvoie un tableau entries contenant le contenu du journal
    - entries = : ["entrée 1", "entrée 2", ...]

/yagapi/journal/get/last

- C'est une requête GET
- Elle renvoie les last dernières lignes du journal.

Appel

- Elle attend un paramètre: le nombre last de lignes à renvoyer
    - var myFormData = "last=42";
    - xhr.open('GET', '/yagapi/journal/get/last?' + myFormData);

    - xhr.send();

Retour

- Elle renvoie un tableau entries contenant les last dernières lignes:
    - entries = : [“entrée -42”, “entrée -41”, ...]

/yagapi/journal/get/at

- C’est une requête GET
- Elle renvoie le contenu du journal depuis la ligne at.

Appel

- Elle attend un paramètre: le numéro at de la première ligne à renvoyer
    - var myFormData = "at=42";
    - xhr.open('GET', '/yagapi/journal/get/at?' + myFormData);
    - xhr.send();

Retour

- Elle renvoie un tableau entries contenant toutes les lignes suivant la ligne at:
    - entries = : [“entrée 43”, “entrée 44”, ...]

/yagapi/journal/send/public

- C’est une requête POST
- Elle affiche un message public dans le journal

Appel

- Elle attend 1 paramètre Message:
    - xhr.open('POST', '/yagapi/journal/send/public');
    - var myFormData = "Message=" + document.getElementById("JournalMessage").value;
    - xhr.send(myFormData);

Retour

- Elle ne renvoie rien

/yagapi/journal/send/local

- C’est une requête POST
- Elle affiche un message privé dans le journal

Appel

- Elle attend 1 paramètre Message:
    - xhr.open('POST', '/yagapi/journal/send/local);
    - var myFormData = "Message=" + document.getElementById("JournalMessage").value;
    - xhr.send(myFormData);

Retour

- Elle ne renvoie rien

Le tour

/yagapi/initiative/get/list

- C’est une requête GET
- Elle renvoie la liste des pions participant au tour avec leur initiative.

Appel

- Elle n’attend aucun paramètre
    - xhr.open('GET', '/yagapi/initiative/get/list');
    - xhr.send();

Retour

- Elle renvoie un tableau lines contenant les infos de tour pour chaque pion y participant:
    - lines = : [{“id”: “id1”, “acting”: bActing1, “init”: init1, “csguid”, “csguid1” }, ...]
    - Le tableau contient 1 objet JSON par pion, avec:
        - id = identifiant du pion (string)
        - acting = est-ce au tour du pion de jouer ? (booleen)
        - init = initiative du pion (integer)
        - csguid = guid de la cs du pion (string)

/yagapi/initiative/down

- C’est une requête GET
- Elle passe le tour au pion d’initiative immédiatement inférieure à l’actuel.

Appel

- Elle n’attend aucun paramètre
    - xhr.open('GET', '/yagapi/initiative/down');
    - xhr.send();

Retour

- Elle ne renvoie rien

/yagapi/initiative/up

- C’est une requête GET
- Elle passe le tour au pion d’initiative immédiatement supérieure à l’actuel.

Appel

- Elle n’attend aucun paramètre
    - xhr.open('GET', '/yagapi/initiative/up');
    - xhr.send();

Retour

- Elle ne renvoie rien

/yagapi/initiative/reset

- C’est une requête GET
- Elle réinitialise l’initiative pour ce tour.

Appel

- Elle n’attend aucun paramètre
    - xhr.open('GET', '/yagapi/initiative/reset');
    - xhr.send();

Retour

- Elle ne renvoie rien