Sirius API v1.0.5

## Objet du document

Ce document contient toutes les informations afin d’être en mesure de se connecter au serveur du jeu Sirius.

## Table des matières

[Objet du document 1](#_Toc472079185)

[Table des matières 1](#_Toc472079186)

[Le jeu Sirius 2](#_Toc472079187)

[Créer son personnage 3](#_Toc472079188)

[Améliorer son personnage 3](#_Toc472079189)

[Les types de personnage 3](#_Toc472079190)

[Les types de partie (boss) 3](#_Toc472079191)

[Se connecter à l’API 4](#_Toc472079192)

[Les services disponibles (l’API) 5](#_Toc472079193)

[Connexion au serveur 5](#_Toc472079194)

[Déconnexion du serveur 5](#_Toc472079195)

[Avoir des informations sur votre personnage 6](#_Toc472079196)

[L’affichage des parties disponibles 7](#_Toc472079197)

[Joindre une partie 8](#_Toc472079198)

[L’état de la partie en cours 8](#_Toc472079199)

[Pour faire une action (utiliser un pouvoir) 10](#_Toc472079200)

[Notes importantes 11](#_Toc472079201)

[Éviter d’être banni du serveur et les délais d’appel 11](#_Toc472079202)

[Le quasi temps-réel 11](#_Toc472079203)

[Distinguer un message texte d’un objet dans une variable 11](#_Toc472079204)

## Le jeu Sirius

Sirius est un serveur d’un jeu de type RPG basé sur la technologie PHP, HTML, JavaScript, MySQL et AJAX. Celui-ci peut être joué seul ou à plusieurs (4 joueurs dans la même partie maximum).

L’API, qui est disponible aux étudiants, permet de concevoir un jeu avec une interface graphique au choix. Voici deux exemples de jeux qui, en vérité, sont le même. Seul le *LookAndFeel* du jeu est différent.

|  |  |  |
| --- | --- | --- |
| Page de connexion au serveur | Page de choix de la partie | Page de jeu |
|  | C:\Users\Frederic\AppData\Local\Microsoft\Windows\INetCache\Content.Word\journey-1.png | C:\Users\Frederic\AppData\Local\Microsoft\Windows\INetCache\Content.Word\journey-2.png |
|  |  |  |
|  |  |  |

Le commencement

Avant même de se connecter à l’API, l’étudiant doit avoir terminé de créer son compte sur le serveur Sirius.

### Créer son personnage

Pour ce faire, il doit aller au lien suivant, spécifier son matricule (sans mot de passe), et terminer l’inscription et la création de son personnage.

<https://apps-de-cours.com/web-sirius/server/>

Un personnage qui vient d’être créé reçoit 5 points attributs et 1 point de pouvoir.

Actuellement, chacune des 4 classes peut avoir jusqu’à 3 pouvoirs différents.

### Améliorer son personnage

Après chaque partie gagnée, le personnage reçoit de l’expérience, ce qui lui permet de monter de niveau.

À chaque nouveau niveau, il 5 points attributs et 1 point de pouvoir. Ces points peuvent être utilisés via la page Web « Accueil » du serveur Sirius (lorsque l’on est connecté).

### Les types de personnage

Il existe 4 classes :

Melee  Qui se concentre sur la force

Range Qui se concentre sur l’agilité

Heal Plus faible mais peut se guérir et guérir les autres

Magic Attaque très fort, mais peu de défense

### Les types de partie (boss)

Il existe 4 différents types de partie :

Melee  Beaucoup de HP

Range Attaque plus fort

Heal Peut se guérir

Magic Peut attaquer tous les joueurs en même temps

## Se connecter à l’API

Pour créer un jeu, il est nécessaire de communiquer avec le serveur de Sirius. Vous devrez donc faire appel à son API.

Voici un exemple d’appel qui permet de faire l’opération « signin ». Il est **FORTEMENT** conseillé d’utiliser cette fonction tel quel et de la placer dans une classe « Utils » ou dans « CommonAction ».

/\*\*

\* data = array('key1' => 'value1', 'key2' => 'value2');

\*/

public function callAPI($service, array $data) {

$apiURL = "https://apps-de-cours.com/web-sirius/server/api/" . $service . ".php";

$options = array(

'http' => array(

'header' => "Content-type: application/x-www-form-urlencoded\r\n",

'method' => 'POST',

'content' => http\_build\_query($data)

)

);

$context = stream\_context\_create($options);

$result = file\_get\_contents($apiURL, false, $context);

if (strpos($result, "<br") !== false) {

var\_dump($result);

exit;

}

return json\_decode($result);

}

Exemple d’appel (LoginAction, UserDAO, …)

$data = [];

$data["username"] = "guest";

$data["pwd"] = "guest";

$key = parent::callAPI("signin", $data);

## Les services disponibles (l’API)

### Connexion au serveur

Pour faire une authentification auprès du serveur, il faut être préalablement avoir complété son inscription (voir plus haut). Lors d’une connexion, le serveur vous retournera une clé. Celle-ci doit être conservée en session car tous les autres appels au serveur en dépendent. Cette clé permet au serveur de « vous reconnaitre » (savoir qui vous êtes).

|  |  |  |
| --- | --- | --- |
| Nom du service | signin |  |
| Paramètres | username  pwd | Le nom de votre personnage  Votre mot de passe |
| Retour (succès) | Votre clé de session (40 caractères) | Ex :  « afsc9sflasmknc5lkntasd9yhcbasdfnasd9fcn » |
| Retour (erreur) | "EMPTY\_USERNAME"  "USER\_NOT\_FOUND"  "CHARACTER\_NOT\_CREATED"  "USER\_IS\_BANNED"  "TOO\_MANY\_CONNECTIONS\_BAN"  "INVALID\_USERNAME\_PASSWORD" | - Inscription non complétée  - Trop de connexions rapides |

Important : puisque le mot de passe est envoyé dans une connexion « http » (non sécurisé), n’utilisez pas un mot de passe que vous utilisez déjà ailleurs.

Pour savoir comment implémenter ce service, veuillez regarder l’exemple plus haut.

### Déconnexion du serveur

Lorsque vous êtes connectés, vous pouvez vous déconnecter en passant au service votre clef de session.

|  |  |  |
| --- | --- | --- |
| Nom du service | signout |  |
| Paramètres | key | Votre clé  *doit être envoyée dans un tableau, même s’il n’y a que la clé en paramètre. Exemple :*  *$data = array("key" => $\_SESSION["key"]);* |
| Retour (succès) | "USER\_SIGNED\_OUT" |  |
| Retour (erreur) | "EMPTY\_KEY"  "USER\_NOT\_FOUND" |  |

### Avoir des informations sur votre personnage

Service pratique pour savoir si le personnage a monté de niveau, pour connaitre son HP ou simplement pour savoir l’état de la dernière partie (gagnée ou perdue).

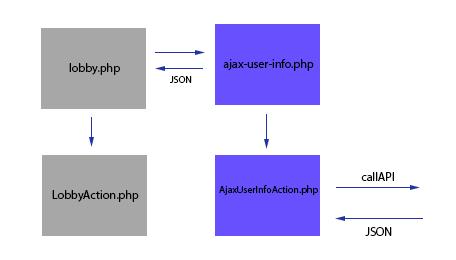
Ce service peut être utilisé à la fin d’une partie (pour savoir si elle a été gagnée ou perdue), et lors de l’affichage des parties. Par exemple :



|  |  |  |
| --- | --- | --- |
| Nom du service | user-info |  |
| Paramètres | key | Votre clé |
| Retour (succès) | Un document JSON avec les informations du personnage |  |
| Retour (erreur) | "EMPTY\_KEY"  "USER\_NOT\_FOUND"  "USER\_IS\_BANNED"  "TOO\_MANY\_CALLS\_BAN" | c  - Trop d’appels à ce service rapidement |

**Astuce :**

Vous devriez utiliser AJAX pour accéder à ce service. Voici un exemple : la page hobby.php se charge sans rien faire de particulier. Après 2 secondes d’attente, une fonction JavaScript utilise AJAX pour appeler la page « ajax-user-info.php ». Dans son action, cette dernière appelle l’API qui retourne les statistiques du personnage. Cette information est ensuite retournée au JavaScript et celui-ci ajoute les informations dans la page (DOM). Cette logique devrait également être utilisée la plupart des services (sauf login/logout)



### L’affichage des parties disponibles

Vous avez accès aux parties qui ont moins de 4 niveaux de différences par rapport au niveau de votre personnage.

Voici un exemple d’affichage des parties disponibles.



|  |  |  |
| --- | --- | --- |
| Nom du service | list |  |
| Paramètres | key | Votre clé |
| Retour (succès) | Un document JSON avec les parties disponibles |  |
| Retour (erreur) | "EMPTY\_KEY"  "USER\_NOT\_FOUND"  "USER\_IS\_BANNED"  "TOO\_MANY\_CALLS\_BAN" | - Êtes-vous connecté?  - Trop d’appels à ce service rapidement |

### Joindre une partie

Il est possible de joindre les parties qui ne sont pas remplies. Certaines parties acceptent 1 joueur maximum, d’autres 2, jusqu’à un maximum de 4. Plus une partie acceptent de personnes, plus elle est difficile mais plus elle donne de l’expérience.

Lorsque votre personnage joint une partie, il est nécessaire de rediriger le joueur vers la page de jeu, car à partir de ce moment, *game is on!*

|  |  |  |
| --- | --- | --- |
| Nom du service | enter |  |
| Paramètres | key  id | Votre clé  ID de la partie |
| Retour (succès) | "GAME\_ENTERED" |  |
| Retour (erreur) | "EMPTY\_KEY"  "EMPTY\_ID"  "USER\_IS\_BANNED"  "USER\_NOT\_FOUND"  "GAME\_NOT\_FOUND"  "EXCEEDED\_LEVEL\_RANGE"  "GAME\_IS\_FULL" | - Êtes-vous connecté? |

### L’état de la partie en cours

Lorsque le personnage est dans la partie, il est possible de demander régulièrement son état afin de savoir si votre HP a diminué, quels sont les autres joueurs présents, etc.

Il y doit y avoir un délai de 2 secondes minimum entre chaque appel. Autrement, l’appel est refusé et vous pourriez même être banni.

|  |  |  |
| --- | --- | --- |
| Nom du service | state |  |
| Paramètres | Key | Votre clé |
| Retour (succès) | Un document JSON contenant l’état de la partie | etat.game.[…] Info sur la partie/boss  etat.player.[…] Info sur le perso et les pouvoirs disponibles  etat.otherplayers.[…] Liste d’info sur les autres joueurs |
| Retour (erreur) | "EMPTY\_KEY"  "USER\_NOT\_FOUND"  "GAME\_NOT\_FOUND\_WIN"  "GAME\_NOT\_FOUND\_LOST"  "GAME\_NOT\_FOUND\_NONE"  "TOO\_MANY\_CALLS\_BAN" | - Êtes-vous connecté?  - Partie terminée (vous avez gagné)  - Partie terminée (vous avez perdu)  - Partie terminée |

**Exemple de réponse**

Voici un exemple complet de réponse :

{

"game":{

"name":"Void 1",

"type":"Melee",

"level":"0",

"hp":990,

"attacked" : false,

"last\_target" : "party",

"max\_hp":1000

},

"player":{

"name":"Guest2",

"level":1,

"type":"Magic",

"victories":0,

"skills":[

{

"name":"Normal",

"heal\_target":null,

"level":1,

"cost":0,

"dmg":5,

"heal":0

}

],

"hp":10,

"mp":10,

"max\_hp":10,

"max\_mp":10

},

"other\_players":[

{

"name":"Guest",

"level":"1",

"welcome\_text":null,

"type":"Melee",

"victories":"0",

"max\_hp":25,

"max\_mp":5,

"hp":25,

"mp":5,

"attacked":"--"

}

]

}

Notes :

* Pour savoir si un autre joueur a utilisé un pouvoir, regarder la propriété « attacked ».
  + -- = le joueur n’a pas fait son action/pouvoir
  + Normal, Special1, Special2 = Nom du pouvoir utilisé
* « dmg » : signifie point de dégât (*damage*)
* La propriété « skills » et « other\_players » sont des tableaux, il faut donc faire une itération (*loop*) pour avoir accès aux données.

### Pour faire une action (utiliser un pouvoir)

Il y doit y avoir un délai de 2 secondes minimum entre chaque action. Autrement, elle est refusée ou vous pourriez même être banni.

|  |  |  |
| --- | --- | --- |
| Nom du service | action |  |
| Paramètres | key  skill-name | Votre clé  Nom du pouvoir utilisé |
| Retour (succès) | "OK" |  |
| Retour (erreur) | "EMPTY\_KEY"  "USER\_NOT\_FOUND"  "GAME\_NOT\_FOUND"  "TOO\_MANY\_CALLS\_BAN"  "EMPTY\_SKILL\_NAME"  "SKILL\_NOT\_FOUND"  "PLAYER\_IS\_DEAD"  "NOT\_ENOUGH\_MP" | - Êtes-vous connecté? |

## Notes importantes

Voici quelques notes importantes concernant le serveur Sirius.

### Éviter d’être banni du serveur et les délais d’appel

Pour le service « user-info » : un délai de 2 secondes minimum entre chaque appel

Pour le service « list » : un délai de 2 secondes minimum entre chaque appel

Pour le service « state » : un délai de 2 secondes minimum entre chaque appel

Pour le service « action » : un délai de 2 secondes minimum entre chaque appel

Il est donc nécessaire d’attendre le retour de l’appel du service avant d’en lancer un autre!

Par exemple :

function list() {

$.ajax({

url : "ajax-list.php",

type : "POST"

})

.done(function (msg) {

var reponse = JSON.parse(msg);

// traitement ici…

setTimeout(list, 2000); // Attendre 2 secondes avant de relancer l’appel

})

}

setTimeout(list, 2000); // Appel initial (attendre 2 secondes)

### Le quasi temps-réel

Étant donné qu’AJAX est utilisé au lieu des Web Sockets pour le jeu, le jeu ne fonctionne pas tout-à-fait en temps réel. C’est donc normal que lorsque l’un des joueurs fait une action, un autre joueur ne la voit pas instantanément.

### Distinguer un message texte d’un objet dans une variable

Voici un exemple de code qui permet de voir si la variable est un objet ou une simple chaîne de caractères. Ça peut être utilisé pour savoir si la partie est terminé ou pas

if (typeof maVariable !== "object") {

if (maVariable == "GAME\_NOT\_FOUND") {

// Fin de la partie. Est-ce que j’ai gagné? Je dois appeler user-info

}

}

else {

// maVariable est un objet. On pourrait faire, par exemple, maVariable.game.hp ou

// maVariable.player.mp

}