Documentation des API

Chapus Louka

Table des matières

[1. Introduction 3](#_Toc136597011)

[2. Javascript 3](#_Toc136597012)

[2.1 Gestion\_utilisateur 3](#_Toc136597013)

[2.2 Gestion\_galaxie 3](#_Toc136597014)

[2.3 Planete\_nav 3](#_Toc136597015)

[2.4 Affichage\_infrastructure 3](#_Toc136597016)

[2.5 Affichage\_recherche 4](#_Toc136597017)

[2.6 Bouton\_sidebar 4](#_Toc136597018)

[2.7 Ressource\_manager 4](#_Toc136597019)

[2.8 Construction 4](#_Toc136597020)

[2.9 Recherche\_manager 4](#_Toc136597021)

[3. Script PHP 5](#_Toc136597022)

[3.1 Selection\_univers 5](#_Toc136597023)

[3.2 Creer\_utilisateur 5](#_Toc136597024)

[3.3 Afficher\_planete 5](#_Toc136597025)

[3.4 Possession\_planete 5](#_Toc136597026)

[3.5 Attribuer\_planete 5](#_Toc136597027)

[3.6 Afficher\_planete\_nav 5](#_Toc136597028)

[3.7 Afficher\_infrastructure 5](#_Toc136597029)

[3.8 Batiment\_en\_construction 5](#_Toc136597030)

[3.9 Afficher\_recherche 6](#_Toc136597031)

[3.10 Recherche\_en\_cours 6](#_Toc136597032)

[3.11 Ressource\_info 6](#_Toc136597033)

[3.12 Nombre\_batiment 6](#_Toc136597034)

[3.13 Infrastructure\_niveau\_date 6](#_Toc136597035)

[3.14 Possede\_laboratoire 6](#_Toc136597036)

[3.15 Recherche\_niveau\_date 6](#_Toc136597037)

[3.16 Connexion 6](#_Toc136597038)

[3.17 Creation\_univers 7](#_Toc136597039)

[3.18 Ressource\_connexion 7](#_Toc136597040)

# Introduction

Le but de ce document est de documenter les actions faites par les différents scripts sur les pages. Pour chaque script il y aura une description détaillée de son utilité dans le bon fonctionnement du jeu.

# Javascript

Pour chaque javascript on décrit le fonctionnement sans forcément rentrer dans les détails de l’implémentation, on explicite aussi quel autre script sont appelés dans un script.

## Gestion\_utilisateur

Ce script va ajouter les options de sélection dans la liste déroulante de la page index en appelant le script [Selection\_univers](#_3.1_Selection_univers). Puis il ajoute des EventListener sur le bouton de connexion et sur le bouton de création de compte. Lorsqu’on appui sur le bouton de connexion on appelle le script [connexion](#_3.16_Connexion) et on met à jour les classes des différents messages d’erreurs si besoin. Pour le bouton de création de compte cette fois-ci on appelle le script [Créer\_utilisateur](#_3.2_Créer_utilisateur) et on met aussi à jour les messages d’erreurs si besoin.

## Gestion\_galaxie

Ce script est appelé dans la page Galaxie. Il commence par ajouter des EventListener sur les barres de sélection de galaxie et de système solaire, si les deux barres sont non vides en même temps alors on fait appel au script [Afficher\_planete](#_3.3_Afficher_planete) avec la valeur de la galaxie et celle du système solaire sélectionner en argument.

## Planete\_nav

Ce script est appelé dans toutes les pages (sauf la page index). Il permet d’attribuer une planète à un joueur et ensuite d’afficher ses « coordonnées » en haut à gauche de chaque page, pour que le joueur connaisse l’emplacement de sa planète. Tout d’abord on regarde si le joueur possède une planète avec le script [Possession\_planete](#_3.4_Possession_planete), s’il n’en n’a pas alors on lui en attribue une grâce au script [Attribuer\_planete](#_3.5_Attribuer_planete), on attribut aussi à ce moment-là les valeurs par défaut pour un nouvel utilisateur. Ensuite si l’utilisateur possède déjà une planète alors on va simplement afficher ses « coordonnée » à l’aide du script [Afficher\_planete\_nav](#_3.7_Afficher_planete_nav).

## Affichage\_infrastructure

Ce script est appelé dans la page Infrastructure. Il permet de faire l’affichage de toutes les informations sur les infrastructures en appelant le script php [afficher\_infrastructure](#_3.7_Afficher_inffrastructure) puis après un petit délai il appelle le script [batiment\_en\_construction](#_3.8_Batiment_en_construction) avec la méthode GET pour savoir s’il y a déjà un bâtiment en construction ou pas et le cas échéant mettre les bonnes classes sur tous les boutons. Mais s’il y a déjà un bâtiment qui est en construction alors on regarde la date de fin de construction et on la compare avec la date actuelle toutes les secondes. Si la date de construction est dépassée alors un réinitialise la date de construction grâce au script [batiment\_en\_construction](#_3.8_Batiment_en_construction) mais avec la méthode POST cette fois-ci.

## Affichage\_recherche

Ce script est appelé dans la page Recherche. Il permet de faire l’affichage des recherches avec les bonnes informations comme le script précédent, on appel le script php afficher\_recherche pour que les recherches soit afficher avec les niveaux stockés dans la base de données. Puis on appel le script recherche\_en\_cours avec la méthode GET pour savoir s’il y une recherche qui est déjà en cours, autrement dit s’il y a une recherche qui possède une date de fin. Puis on regarde toutes les secondes si on a dépassé la date et si c’est le cas on appelle le script recherche\_en\_cours mais avec la méthode POST pour réinitialiser la date de la recherche.

## Bouton\_sidebar

Ce script est appelé dans toutes les pages (sauf la page index). Il ajoute des eventListener sur chaque bouton de la sidebar puis lorsqu’on clique qu’un bouton, on envoie les ressource (avec la fonction envoyer\_ressource du javascript [ressource\_manager](#_Ressource_manager)) à la base de données pour pouvoir les récupérer par la suite et on redirige l’utilisateur sur la bonne page.

## Ressource\_manager

Ce script est aussi appelé dans toutes les pages (sauf la page index). Il permet de générer les ressources d’un utilisateur toutes les secondes, même si la production afficher sur le site est par minute. Donc à chaque chargement de ce script on récupère les ressources et le niveau des différents bâtiments qui peuvent générer des ressources stockées sur la base de données grâce au script [ressource\_info](#_3.11_Ressource_info) avec la méthode GET, puis on incrémente les ressources toutes les secondes, sauf pour l’énergie qui est une ressource fixe. Ensuite on a une autre fonction qui permet d’envoyé les ressources locales sur la base de données, pour cela on fait appel au script [ressource\_info](#_3.11_Ressource_info) mais avec la méthode POST et la quantité des ressources en argument. Enfin on envoi les ressources toutes les minutes au cas où l’utilisateur ferme la page sans sauvegarder.

## Construction

Ce script est appelé dans la page Infrastructure. Il ajoute des eventListener sur chaque bouton de la page, lorsqu’un bouton est cliqué on test dans l’ordre le nom de la classe du bouton elle doit être différentes d’en\_contruction, puis on regarde s’il y a assez de place sur la planète (le nombre de bâtiment est récupéré grâce au script [nombre\_batiment](#_3.12_Nombre_batiment)) et enfin on récupère et on teste si on a assez de ressource pour construire le bâtiment. Si c’est le cas alors on enlève les ressources, on change la classe, on envoie les ressources sur la base de données et on augmente le niveau sur la base de données avec le script [infrastructure\_niveau\_date](#_3.13_Infrastructure_niveau_date) et la méthode POST. On fait la même chose pour tous les boutons de la page.

## Recherche\_manager

Ce script est appelé dans la page Recherche. Il ajoute des eventListener sur chaque bouton de la page, lorsqu’un bouton est cliqué on test dans l’ordre s’il le joueur possède un laboratoire de recherche (on le sait avec le script [possede\_laboratoire](#_3.14_Possede_laboratoire)), le nom de la classe du bouton elle doit être différentes d’en\_contruction et le niveau des recherches précédente si applicable, enfin on regarde si le joueur à assez de ressource pour lancer la recherche. Si c’est le cas alors on augmente le niveau, on met à jours la date de fin dans la base de données avec le script [recherche\_niveau\_date](#_3.15_Recherche_niveau_date), puis on change la classe des boutons et on envoie les ressources sur la base.

# Script PHP

Tous les scripts php ou presque font appelle au script pdo qui permet de se connecter à la base de données avec l’utilisateur esirem et mot de passe esirem il crée donc le PDO et la variable db.

## 3.1 Selection\_univers

On fait une requête sql pour récupérer les noms de tous les univers stockés dans la table univers de la base de données, puis on ajoute toutes les options dans une variable texte que l’on renvoie à la fin du script.

## 3.2 Creer\_utilisateur

On commence par regarder la méthode utilisé (ici POST), puis on regarde si un utilisateur avec le même mail existe déjà, sinon on l’ajoute à la base de données en hachant le mot de passe pour pas qu’il soit visible en claire.

## 3.3 Afficher\_planete

On commence par regarder la méthode utilisé (ici GET), puis on récupère les différents id nécessaire pour faire l’affichage. Ensuite, on fait une boucle for pour chaque planète dans la base de données avec la bonne galaxie et le bon système solaire, s’il y a une planète alors on affiche ses informations sinon on affiche une case vide. On revoie le tableau des planètes rempli avec les bonnes informations.

## 3.4 Possession\_planete

On commence par regarder la méthode utilisé (ici GET), puis on récupère les id nécessaires pour faire la requête ou on compte le nombre de planète possède un joueur, si le joueur ne possède aucune planète on renvoie false, sinon on renvoie true.

## 3.5 Attribuer\_planete

On commence par regarder la méthode utilisé (ici GET), puis on récupère les id nécessaires. On récupère l’id de la première planète et de la dernière planète d’un univers et ensuite on tire aléatoirement un nombre entre les deux id tout en vérifiant que la planète n’est pas déjà attribuée à un autre joueur. Ensuite on récupère les « coordonnées » de la planète que l’on stocke dans la variable de session et on initialise les différentes valeurs pour un nouveau joueur comme les ressources ou le bâtiment niveau 0. On finit par renvoyer les « coordonnées » de la planète.

## 3.6 Afficher\_planete\_nav

On récupère les id nécessaires, puis on tronque les noms en ne gardant que le numéro des différentes données et on les arrange pour que l’affichage soit fait correctement.

## 3.7 Afficher\_infrastructure

On récupère les id nécessaires, puis on fait l’affichage des informations des bâtiments avec les bonnes valeurs, on calcul les couts pour chaque bâtiment en fonction de leur niveau et du niveau des recherches. On met aussi les bon id pour pouvoir récupérer ces informations par la suite.

## 3.8 Batiment\_en\_construction

On commence par récupérer les id nécessaires, ensuite on regarde la méthode utiliser si c’est GET, on regarde s’il y a un bâtiment qui possède une date de fin de construction si c’est le cas on renvoi la date et null sinon. Enfin pour la méthode POST, il s’agit là de réinitialiser la date de construction et de la mettre à null c’est ce qu’on fait avec une requête sql.

## 3.9 Afficher\_recherche

On commence par récupérer les id nécessaires, puis on affiche les informations sur les recherches sachant que leur cout en ressource n’augmente pas, il y a seulement le cout en temps qui augmente à chaque niveau. Encore une fois on met les id nécessaire pour pouvoir récupérer les informations par la suite avec un javascript.

## 3.10 Recherche\_en\_cours

On récupère les id nécessaires, puis on fait la même chose que le script [batiment\_en\_construction](#_3.8_Batiment_en_construction), c’est à dire que si on a la méthode GET on renvoie la date de fin et null s’il n’y a pas de date de fin. Puis avec la méthode POST on met à null la ligne qui contient une date qui n’est pas null.

## 3.11 Ressource\_info

On récupère les id nécessaires, puis on regarde la méthode utiliser. Pour la méthode GET, on récupère les ressources sur la table ressources de la base de données, puis on récupère les différents niveaux de bâtiments générant des ressources et on sépare chaque donnée par une barre verticale pour pouvoir les retrouver dans le javascript. Pour la méthode POST, on récupère les ressources passer en argument et on met simplement à jours les ressources avec les bonnes valeurs, on en profite aussi pour mettre à jours la date de déconnexion de l’utilisateur, elle servira lors de la connexion pour générer les ressources de l’utilisateur lorsqu’il était déconnecté.

## 3.12 Nombre\_batiment

On regarde la méthode utilisée puis on récupère les id, ensuite on calcul le nombre de bâtiment présent sur la planète, c’est simplement la somme des niveaux de ces derniers, les bâtiments de défense ne sont pas pris en compte. Puis on récupère le nombre max de bâtiment constructible grâce à la position et donc la taille de la planète. Enfin si on est en dessous de la limite on revoie en\_dessous et limite sinon.

## 3.13 Infrastructure\_niveau\_date

On récupère les id nécessaires, puis on récupère les informations fournies on va ensuite à la bonne ligne dans la table infrastructure, on prend l’id et on met à jours le niveau et la date, sachant que pour la date on a déjà ajouté le temps de construction dans le javascript.

## 3.14 Possede\_laboratoire

On récupère les id nécessaires, et on va simplement regarder le niveau du laboratoire dans la table infrastructure (il a l’id numéro 1). Si le niveau est supérieur ou égal à 1 alors on renvoie possède et s’il est égale à 0 on renvoie possède\_pas.

## 3.15 Recherche\_niveau\_date

On récupère les id nécessaires, puis on récupère les informations fournies. Dans les informations fournies on a l’id de la technologie, on va donc simplement mettre à jours la table recherche avec le bon niveau et la bonne date de fin.

## 3.16 Connexion

On récupère les informations fournies dans le POST puis on regarde si on a le nom de l’utilisateur qui existe, si c’est le cas on teste que les mots de passe soit identique grâce à la fonction password\_verify car les mots de passe sont chiffrés. Enfin on regarde l’univers sélectionné, si on sélectionne nouveau univers alors on va exécuter le script [creation\_univers](#_3.16_Creation_univers) et on initialiser toutes les variables nécessaires pour la suite et enfin on calcul les ressources générer pendant que le joueur n’était pas la grâce au script [ressource\_connexion](#_3.18_Ressource_connexion).

## 3.17 Creation\_univers

On récupère le nombre d’univers déjà créé et un ajoute le nom du nouveau univers en notant son id. On va ensuite générer les galaxies et les systèmes solaires grâce à des boules for imbriqué. Puis pour les planètes il faut tirer au sort le nombre de planète dans un système solaire et ensuit il faut tirer au sort l’emplacement de la planète dans le système solaire, pour faire cela on utilise la fonction rand(). On ajoute au fur et à mesure les données dans la base de données.

## 3.18 Ressource\_connexion

On récupère les id nécessaires, puis on regarde si l’utilisateur possède une planète sinon on ne fait rien, mais s’il a une planète alors on va récupérer les ressources dans la base de données avec la date de déconnexion. On va ensuite calculer la différence en seconde entre la date actuelle et la date de déconnexion. Puis on calcul la production grâce au niveau des bâtiments stocké dans la base de données et on met à jours les nouvelles ressources ainsi que la date actuelle dans la table ressource.