# Description du code source de Project AG

*Auteur :* Jean-François Grailet

Mars 2020

# Table des matières

# Avant-propos

Project AG est un site web qui a été conçu et créé par pur loisir. Bien qu'il soit actuellement axé sur les jeux vidéo, PAG (pour abréger) s'est construit à la base autour d'un forum et de l'espace membre qui s'y rattache. En ce sens, son code n'est pas forcément spécifique aux sites traitant des jeux vidéo (et autres) et peut être ré-utilisé à d'autres fins.

Comme le code ne contenait rien d'exotique et que le site n'a pas pour vocation de devenir une source de revenus, j'ai décidé de le rendre *open source* afin qu'il puisse aussi bien servir à d'autres développeurs qu'être amélioré et étendu au fil du temps par des contributeurs externes. Ce choix est d'autant plus justifié que le base de données associée au site ne stocke et ne traite aucune donnée sensible (à moins de considérer les adresses e-mail comme des données sensibles), les mots de passe étant *hachés*[1]).

Comme PAG a été conçu sur plusieurs années par à-coups, son code est commenté en long et en large afin de toujours garder une trace de l'intérêt de chaque composant ainsi que de la manière de les utiliser. La lecture du code devrait donc être relativement aisée pour un oeil extérieur, mais les commentaires existants ne seront pas forcément suffisants pour comprendre spontanément l'architecture du site et le pourquoi du comment de certains choix de conception.

C'est précisément pour donner un aperçu d'ensemble du code que ce document a été créé. Les pages qui suivent décrivent donc l'architecture globale du site en tentant de rester le plus général possible, les détails techniques étant (abondamment) commentés dans le code source.

1. Cf. `https://fr.wikipedia.org/wiki/Fonction_de_hachage`

# 1 Introduction

Ce chapitre présente dans un premier temps le type d'architecture employé (cf. 1.1) pour organiser le code source et décrit ensuite les différents chapitres de ce document (cf. 1.2). Dans un second temps, ce chapitre récapitule aussi brièvement les langages utilisés (cf. 1.3) et ainsi que les sous-dossiers qui structurent le code source (cf. 1.4).

## 1.1 Architecture générale

La conception de PAG applique l'architecture **MVC** [1], c.-à-d. *Modèle - Vue - Contrôleur*, illustrée à la figure 1.1. Cette architecture vise à séparer clairement le code qui va représenter les données (le modèle), le code qui va fournir une interface à l'utilisateur (la vue) et le code qui va gérer la logique des opérations réalisées par l'utilisateur (le contrôleur).

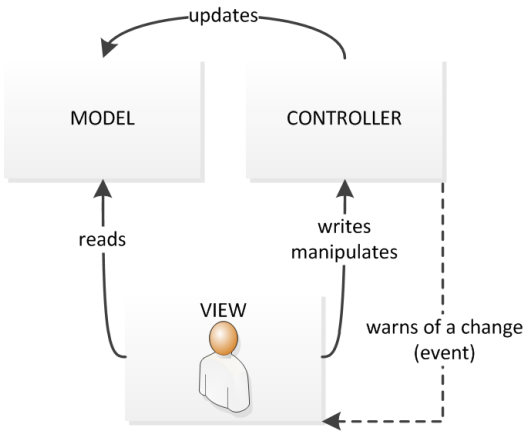


FIGURE 1.1 – Vue conceptuelle du *design pattern* MVC (source : Wikipédia)

Dans le cas de PAG, en pratique, le modèle rassemble un ensemble de classes PHP qui vont servir d'abstraction pour le contenu de la base de données, tandis que la vue est constituée d'une part d'un ensemble de sous-dossiers qui vont isoler les fichiers écrits dans certains langages (ici, le CSS et le JavaScript) et d'autre part d'un ensemble de *templates* et de fonctions d'interprétation intermédiaire qui vont isoler le code HTML et son agencement selon l'utilisateur du reste. Les scripts principaux, écrits en PHP et situés majoritairement à la racine du site, s'apparentent à la partie contrôle et se veulent le plus haut niveau possible (c.-à-d., en reléguant les aspects plus techniques à d'autres parties du code). Notez qu'un détail des sous-dossiers est donné à la section 1.4.

1. Cf. `https://fr.wikipedia.org/wiki/MVC`

En pratique, cela veut dire que les scripts principaux (écrits en PHP)

— ne contiennent aucune requête directe vers la base de données, et mentionnent tout au plus les débuts et les fins de transactions avec celle-ci (c.-à-d. une suite logique d'interactions qu'on doit pouvoir annuler si l'une d'entre elles échoue et compromet l'intégrité des données),

— ne comportent pas ou peu de CSS (langage utilisé pour coder le style visuel) ou de JavaScript (langage utilisé pour coder les interactions), se contentant de lister les fichiers écrits dans ces langages qui seront utilisés pour créer le rendu de la page finale et gérer son interactivité,

— ne comportent pas non plus de HTML (langage utilisé pour structurer une page Web) ou très peu, celui-ci étant écrit pour la majeure partie dans des *templates* isolés dans la partie *vue* du code.

Il ne reste alors que la logique "*haut-niveau*" qui va régir les actions de l'utilisateur. Pouvoir isoler cette logique du reste du code est très important pour sa lisibilité : comme les lecteurs du code pourront s'en apercevoir, certaines pages manipulent plusieurs fonctionnalités (assez souvent optionnelles) d'un coup. S'il est tout à fait possible de reproduire tout un script en mélangeant tout à la fois les différents langages utilisés (cf. 1.3), le résultat serait beaucoup plus verbeux et confus à la lecture.

## 1.2 Contenu de ce document

Le but de ce document est de présenter les grandes lignes du code source en allant progressivement du modèle (le M de MVC) vers le contrôleur (le C), ou plus exactement vers les scripts PHP "en pratique", c.-à-d. ceux qui décrivent le comportement de chaque page du site tout en minimisant les détails techniques.

Le chapitre 2 commence par présenter la librairie d'en-tête `Header.lib.php` (située dans le sous-dossier `libraries/`) qui fournit plusieurs classes [2] statiques servant de base commune à l'ensemble des scripts PHP. Connaître celles-ci et leur utilité est indispensable pour bien comprendre le reste du code. Ces classes incluent notamment la gestion de la connexion à la base de données (et les interactions avec celle-ci) ainsi que la gestion de la connexion de l'utilisateur sous un pseudonyme.

Ensuite, le chapitre 3 présente le *Modèle*, c.-à-d. l'ensemble des classes PHP utilisées pour manipuler sous forme d'objets les données du site (par exemple les utilisateurs, les sujets, les messages qu'ils contiennent, etc.) telles que stockées dans la base de données. La conception de ces classes vise en particulier à isoler les requêtes SQL (le langage utilisé pour dialoguer avec la base de données) liées à la manipulation des données (insertion, lecture ou modification de lignes dans la base de données) du reste du code.

Le chapitre 4 décrit ensuite la *Vue*, c.-à-d. comment les données sont traitées afin de produire une page web affichable. Cette "traduction" des données se fait principalement à l'aide d'un moteur de *templates* fait maison, mais pour certains éléments du site dont l'affichage peut dépendre des préférences et des interactions de l'utilisateur, des *représentations intermédiaires* (dans le code source, *IR* pour *Intermediate Representation*) traduisent les données brutes en un format plus adapté au moteur de templates.

2. En écrivant ce document, je suppose que le lecteur est déjà familier avec la programmation orientée-objet. Des tutoriels pour son utilisation en PHP peuvent être facilement trouvés sur le Web.

Enfin, le chapitre 5 présente les grandes lignes des scripts principaux (c.-à-d. la partie *Contrôle* du code), c.-à-d. comment les différents composants vus dans les 3 chapitres antérieurs sont manipulés pour créer des scripts "haut niveau" qui implémentent les comportements voulus tout en minimisant le mieux possible les détails techniques.

Notez que des sections annexes existent sous forme de chapitre 6 et fournissent quelques justifications quant à certains choix qui ne tiennent pas de l'implémentation.

## 1.3 Langages utilisés

Le code de PAG fait intervenir 5 langages différents couramment utilisés dans le développement web. Pour les lecteurs moins familiers avec le développement web, le tableau 1.1 ci-contre récapitule ces langages et l'utilité de chacun à titre d'information.

| **Langage** | **Acronyme** | **Utilité** |
|---|---|---|
| **HTML** | *HyperText Markup Language* | Structure des pages |
| **CSS** | *Cascade Style Sheet* | Style des pages |
| **PHP** | *PHP : Hypertext Preprocessor* | Traitement des requêtes |
| **SQL** | *Structure Query Language* | Gestion de la base de données |
| **JavaScript** | *Java Script* | Interactivité des pages |

TABLE 1.1 – Langages utilisés.

Il est utile de préciser aussi que la partie JavaScript utilise le *framework jQuery* [3], principalement à des fins de compatibilité avec tous les navigateurs Web répandus (bien qu'il facilite aussi l'écriture du code).

## 1.4 Arborescence des dossiers

Le tableau 1.2 reprend les principaux dossiers et leur contenu.

3. Cf. `https://jquery.com/`

| Dossier | Contenu |
|---|---|
| `./` (racine) | Contient les scripts principaux, écrits en PHP, qui correspondent à la partie ***Contrôle*** du code. Une poignée d'images par défaut sont également présentes. |
| `ajax/` | Contient les scripts PHP qui traitent les requêtes asynchrones envoyées à l'aide de JavaScript, utilisées pour gérer l'interaction entre la page web et l'utilisateur sans rechargement de celle-ci. Le dossier est nommé d'après le nom donné à ce type d'architecture : **Ajax** (pour *asynchronous JavaScript and XML*[4]). |
| `config/` | Contient un unique fichier, `Config.inc.php`, qui renseigne les informations de connexion à la base de données ainsi que quelques paramètres supplémentaires d'ordre pratique. En principe, c'est le seul fichier à configurer pour tester le code sur un serveur. |
| `javascript/` | Contient, comme son nom l'indique, tous les fichiers JavaScript du site. Cela inclut la librairie *jQuery*, mais aussi un fichier `default.js` chargé sur toutes les pages et comportant les fonctionnalités JavaScript les plus courantes et essentielles. |
| `libraries/` | Contient des librairies de fonctions écrites en PHP, notamment celles qui sont utilisées pour gérer les *uploads*. La librairie la plus essentielle est `Header.lib.php` : elle sert de base commune à tous les scripts. Le chapitre 2 revient en détails sur ce qu'elle fournit. |
| `model/` | Contient des classes PHP qui modélisent chacune des entités stockées dans la base de données. Leur but est de rassembler les requêtes SQL associées à une même entité (et leur traitement) dans le script PHP qui y correspond. Ce sous-dossier correspond donc à la partie ***Modèle*** du code. |
| `res_icons/` | Contient toutes les icônes utilisées sur le site. Toutes celles-ci sont libres de droit (et proviennent de sources tel *IconsDB*[5]. |
| `style/` | A l'instar de `javascript/`, ce sous-dossier contient tous les fichiers écrits dans un même langage, ici le CSS. C'est dedans que se trouve l'ensemble des fichiers de style du site. En particulier, le fichier `default.css` contient du style utilisé sur toutes les pages du site (sinon la plupart). |
| `upload/` | Ce sous-dossier ne contient pas de ressource ou de code, mais sert à stocker les *uploads* des utilisateurs. Un sous-dossier `uploads/tmp/` stocke les fichiers temporaires de chaque utilisateur (un sous-dossier par utilisateur). |
| `view/` | Ce dernier sous-dossier correspond à la partie ***Vue*** du code. Sa racine contient les fichiers `Header.inc.php` et `Footer.inc.php` qui encadrent toutes les pages du site. |

TABLE 1.2 – Description de l'arborescence des dossiers principaux.

4. `https://fr.wikipedia.org/wiki/Ajax_(informatique)`
5. `https://www.iconsdb.com/`

# 2 La librairie d'en-tête (`Header.lib.php`)

# 3 Le *Modèle* et la base de données

# 4 La gestion de la *Vue*

# 5 Les scripts principaux (ou *Contrôleur*)

# 6 Annexes

## 6.1 Au sujet de la langue

Le code a été délibérément écrit et documenté en langue anglaise. Il n'est point question d'un rejet de la langue française : en vérité, je suis parti du principe que la plupart des programmeurs ont souvent une bonne maîtrise de la langue anglaise, et que dans l'éventualité où le code intéresserait des programmeurs non-francophones, cela éviterait de devoir le commenter à deux reprises (une tâche qui s'avèrerait bien plus pénible que de traduire ce document-ci).

## 6.2 Origine du nom "Project AG"

Le choix du terme "*Project*" s'explique simplement car il s'agit à la base d'un simple projet personnel qui s'est enrichi au fil du temps jusqu'à justifier une mise en ligne. Le même terme fait également référence à son aspect *open source* et la possibilité que PAG soit enrichi au fil du temps.

Les lettres AG constituent un acronyme pour "*Another G...*". Pourquoi donc ? Tout simplement car la majorité des sites web et forums consacrés aux jeux vidéo ont un nom qui commence en G. Ni plus ni moins ! Il s'agit donc de dire, un peu ironiquement, que PAG n'est jamais qu'un *autre site* sur le sujet.

Enfin, *Project AG* en tant que de nom de domaine (c.-à-d. le `projectag.org`) est aussi un jeu de mot. En effet, ce nom de domaine sonne comme "*project tag*" et fait allusion à la particularité du forum de classer les sujets par mots-clefs (ou *tags*) plutôt que par catégories. Il s'agit par ailleurs d'une des premières fonctionnalités qui a été implémentées lors de la création du site.