Projet Développement Web

# I/ Préparation des données

## A. Collecte des données

Les données ont été collectées sur le site : <https://gamalytic.com/>

Ce sont des données du site de vente de jeux vidéo Steam. La collecte des données est possible via un téléchargement des tableaux de données en format « .csv » (cf. figure 1).

Ce site de données en OpenData nous propose de télécharger les fichiers de données, mais uniquement 50 lignes à la fois (sur plus de 80\_000).

Rajouter figure 1.

Afin de gagner du temps, nous avons développé un code de web scrapping sur Python et plus précisément avec le package BeautifulSoup qui permet cela.

Nous avons ainsi téléchargé massivement les données du site (en OpenData, donc rien d'illégal !).

Il a ensuite fallu télécharger les données du Cloud en local afin de les exploiter.

Les données brutes ne sont pas vraiment utilisables telles quelles au vu de nos objectifs de développement.

## B. Nettoyage des données et création de la Base de données

Mettre une capture d’écran de l’arborescence des fichiers.

Avant toutes manipulations des données (autre qu'exploratoires), il nous a fallu créer nos tables SQL. Cela a été réalisé avec le fichier : creation\_tables\_projet\_dvp\_web.sql

L'utilisation de Pentaho Data Integration (PDI) est essentielle dans cette partie du travail.

En effet, l'outil d'ETL va nous permettre de nettoyer les données et de les insérer dans un Système de Gestion de Bases de Données (SGBD). Le travail réalisé avec PDI a pu être dupliqué dans 2 fichiers différents. Nous avons créé une base de données pour tester et vérifier que notre code fonctionnait avant d'insérer les données dans la table créée avec Doctrine dans Symfony. Nous nous concentreront ici sur nos fichiers tests.

Remarques :

Avec notre base de données de tests, les tables ont effectivement été créées avec le fichier : creation\_tables\_projet\_dvp\_web.sql

Cependant, pour nos bases de données réelles, les tables ont été créées avec l'outil Doctrine intégré au framework Symfony. La création des tables s'est faite à l'aide d'entités et de migrations (expliqué plus tard dans le projet).

1. Extraction et Nettoyage des données.

Les données sont stockées dans divers fichiers CSV. PDI va nous permettre de les réunir et de les joindre quand nécessaire. De plus, cet outil offre de nombreuses fonctionnalités permettant le nettoyage des données notamment en matière de manipulation de chaines de caractères et en termes de changement de type et de format.

2. Export en CSV et insertion dans les tables SQL.

Après avoir réuni les nombreux fichiers, il nous a fallu exporter nos fichiers finaux (qui sont l'équivalent en CSV de nos tables SQL). Cet export nous a ensuite permis la réutilisation de ces fichiers pour insérer les données dans les tables toujours avec PDI. Il nous a aussi permis de faire une analyse exploratoire sur les données avec le langage R.

La réalisation du 1. et l'export des fichiers à l'issue de celui-ci se fait dans le fichier : Extraction\_Nettoyage\_ExportCSV.ktr

Toute la partie d'insertion dans les tables se fait dans les fichiers suivants :

- Insertion\_Workbench\_games.ktr

- Insertion\_Workbench\_genres.ktr

- Insertion\_Workbench\_link\_games\_genres.ktr

Il est possible d'utiliser une "Tâche" (.kjb) de PDI, qui est un fichier de planification d'exécution d'autres programmes de PDI, les "Transformations" (.ktr). La "Tâche" utilisée ici nous sert à ordonner l'exécution des différents fichiers (nettoyage + insertion dans les tables) afin d'ordonner et de centraliser notre travail.

Ce fichier est : Tache\_Insertion\_tables\_Workbench.kjb

Remarque :

Afin de simplifier l'utilisation du programme par les divers membres du groupe, nous avons créé des variables d'environnement modifiables. Le but de ces variables est de gérer la connexion à la base de données locale de chacun afin de pouvoir travailler sur les mêmes données de manière indépendante et de pouvoir progresser dans l'avancement du projet sans être dépendant des autres.

Ces variables sont stockées dans le fichier : ressources/config.txt

# II/ Premier pas en développement

## Apprentissage

Nous avons choisis de nous orienter vers les technologies suivantes pour le développement de notre application :

* Le langage PHP
* Le framework Symfony pour gérer l’architecture de l’application.
* Le framework Bootstrap pour l’aspect visuel de l’application.

La maîtrise de PHP était donc essentielle pour pouvoir se servir des autres outils (notamment Symfony). Pour apprendre ce langage depuis zéro, nous nous sommes orienté vers les cours OpenClassroom qui nous ont permis de nous guider dans notre apprentissage du langage en retrouvant des concepts clé partagés par d’autre langages de programmation (Boucle, Condition, Programmation Orientée Objet de Python entre autre). Nous avons ainsi pu apprendre rapidement les bases du langage et découvrir son utilité dans un contexte de développement web.

Pour apprendre à maitriser Symfony, les cours OpenClassroom furent tout aussi utiles. De plus, ils nous ont aussi permis de nous initier à l’utilisation du framework Bootstrap qui était utilisé dans le code du cours. Des vidéos YouTube complémentaires nous ont permis d’aborder et de comprendre certains concepts plus en détails (ex : l’architecture MVC)

Les liens vers la documentation utilisée est disponible dans la partie Bibliographie.

## Premier code et méthode de travail

Après avoir compris et assimilé les bases de chacune de ces technologies de développement, nous avons essayé de produire notre premier code de manière autonome.

Le but était de développer une première application basée sur l’architecture MVC (Model View Controller). Après avoir structuré l’application, nous nous sommes essayés au développement front pour essayer de faire correspondre l’esthétique de notre première page avec l’une de nos maquettes créées avec Figma. L’objectif était surtout de définir un premier fichier CSS graphique qui nous serait ensuite utile pour le développement graphique de l’application en plus de s’approprier le framework Bootstrap.

Pour faciliter le travail en groupe, nous avons mis en place un Repository git avec l’application GitHub desktop notamment. Cela nous a permis de centraliser notre code, d’éviter les chevauchements en développement et de faciliter le partage de fichier. Un autre avantage de ce Repository est l’historique de développement. En effet, il est possible d’avoir un aperçu des modifications apportées au code pendant tout le développement. Notre Repository contient non seulement le code de l’application mais aussi toutes les parties associées au scrapping, à l’intégration des données ainsi que notre analyse exploratoire et les différents rapports.

## Premiers essais de connexion à la BD

Toutes les manipulations et opérations en lien avec les BD dans Symfony ont été réalisées avec l’outil Doctrine.

Il est important de noter que nous n’avons pas trouvé de documentation pour connecter notre application à une base de données existante. En effet, il semblerait que l’utilisation de tables SQL existante ne soit pas possible. La raison étant que pour pouvoir exploiter la table dans l’application, il faut créer une « Entity » dans Symfony. Cette « Entity » créée ensuite une table dans notre base de données par le biais d’une « Migration ». La création d’une « Entity » sur une table SQL existante écrase cette table et ses données.

Nous avons donc recréé un schéma MariaDB pour pouvoir l’exploiter avec Symfony. Puis nous avons créé les tables dans cette nouvelle base avec des « Entity ». L’insertion des données dans la table a pu être réalisée comme précédemment avec PDI. Après l’insertion des données dans notre base de données, nous pouvons faire des requêtes depuis les Repository Symfony pour récupérer les données pour les utiliser dans notre application. Que ce soit pour l’affichage de données brutes ou leurs utilisations à des fins plus complexes (ex : visualisations), les requêtes construites depuis les Repository sont essentielles. Pour la mise en forme des données et leurs utilisations, cela se passe dans nos Controller Symfony.

## Commandes Symfony utiles

### Installation Symfony

Installer symfony CLI (voir openclassrooms symfony)

Installation globale de composer : <https://getcomposer.org/download/>

Si PHP pas installer -> le réinstaller (en cas d’erreur d’installation de composer.). A faire meme si XAMPP installé

Récupérer les fichiers du git. Taper en ligne de commande : « composer require doctrine/doctrine-bundle »

### Commande de lancement et d’arrêt de l’app Steam

symfony serve -d -> commande pour lancer le serveur

symfony server:stop -> pour stopper le serveur

### Création des Controllers

Pour créer un Controller

symfony console make :controller

### Doctrine – connexion et gestion de la BD

Pour créer une nouvelle entité (équivalent de la table SQL mais dans php) :

php bin/console make :entity

Pour créer le fichier de maj de la table (après avoir cr’éer la migration) :

php bin/console make :migration

Pour enclencher la maj de la BD a partir du fichier précédent :

php bin/console doctrine :migrations :migrate

# III/ Analyse exploratoire et premiers graphiques

Après avoir insérer nos données dans le SGBD MySQL, nous avons utilisé le langage R afin de réaliser une analyse exploratoire sur nos données. Bien que des analyses exploratoires aient déjà été réalisées, il s’agissait ici d’utiliser une méthode plus rigoureuse afin de se faire une idée précise des graphiques que nous souhaiterions réaliser (bien que nous en ayons déjà une petite idée).

Cette utilisation de R nous a aussi permis de définir la forme de nos données voir de certaine tables que nous avons ensuite créés avec des requêtes SQL (les « View » de MySQL) afin de pouvoir créer les graphiques associés.

Précisions sur les fichiers et l’arborescence.

# IV/ Développement

## Nos difficultés (21/10)

### Création des graphiques

La mise en forme des données => requêtes complexe et appel à de la logique POO pas forcement évidente.

Demande beaucoup de travail dans les repository et les controller => possible de bien s’y retrouver en documentant correctement le code.

Difficulté surmontée ? OUI

### Initialisation des modals

Problèmes lors de l’initialisation des modal sur la page recommandation :

Je pensais d’abord que c’était dû à un chevauchement d’ID mais en sortant la modal de la boucle (elle devient unique) on garde le problème. Pourquoi ????

Maj 29/10 : Probelem d’ID, reussite de la creation des modal avec le requêtage des données en AJAX.

Difficulté surmontée ? OUI

### Requêtes AJAX – JQuery

Je ne comprends juste pas comment ça marche => documentation pas claire.

On en a beaucoup besoin : système de filtre, creation de modal (recommandation), radio-btn analysis page (graphe period), etc…

Sans parler d’AJAX, j’ai du mal à comprendre comment envoyer des informations au controller depuis la page twig (interaction utilisateur).

Difficulté surmontée ? NON

Màj 28/10 : première requête AJAX réussie -> affichage réactif d’un graphe en fonction de la valeur d’un radio btn

Explication du fonctionnement des requêtes Ajax :

Le but est de rendre la page interactive. En pratique, on intègre un script JavaScript qui définit une fonction – avec ou sans paramètres. Cette fonction est ensuite appelée dès que l’utilisateur fait une action précise – ex : cliquer sur un bouton précis. La syntaxe d’une requête Ajax est particulière mais il s’agit surtout de préciser que c’est une requête Ajax en utilisant JQuery et envoyant des données dans un format précisé (le plus souvent JSON). Ces données sont ensuite récupérées dans le Controller de la route associée à la requête Ajax. Dans la méthode du Controller, on effectue les opérations nécessaires avant de renvoyer la réponse dans la page twig ou plutôt dans le script JS de la requête Ajax qui se charge ensuite d’utiliser la réponse pour générer l’affichage voulu (affichage brut, graphique, etc…).

### Mise en place d’un cache pour les images (et requêtes ?)

Je comprends l’idée (éviter de devoir télécharger les ressources à chaque initialisation de la page en les téléchargeant en local).

Mais encore une fois, la doc est obscure et j’ai du mal à comprendre comment l’implémenter concrètement dans notre projet.

Difficulté surmontée ? NON

Message 21/10 22h00 :

Bonsoir,

Je me permets de vous déranger à propos des divers problèmes rencontrés dans le développement au cours de ces derniers jours.

Il y a principalement 3 problèmes :

1- Difficultés à mettre ne place plusieurs modal sur une même page.

Sur l'une de nos pages, nous voulons offrir à l'utilisateur la possibilité d'ouvrir une modal associée à un jeux particulier.

Sachant que nous avons déjà une modal présente sur la page (dans la navbar), nous avons toujours, au minimum, 2 modals sur cette page.

Sauf que dès que nous souhaitons activer la 2nd modal, nous obtenons l'erreur suivante dans la console :

"modal.js:160 Uncaught TypeError: Cannot read properties of undefined (reading 'backdrop')"

Après de nombreuses recherches sur le sujet, j'avoue ne pas comprendre le problème ni les solutions proposée en ligne (qui semblent peu correspondre à mon problème).

Si je devais donner un avis, je dirais que les modals ne peuvent pas se créer sur la même 'backdrop' ou au contraire ne peuvent pas en initialiser une nouvelle car il en existe déjà une.

Est-ce que vous avez une piste ou un élément de réponse à nous apporter à ce propos ?

2- Mise en place d'un cache pour les images :

La docummentation n'est pas très claire à ce sujet, l'idée de télécharger des images en local dès leur premier chargement à partir de la ressource web ne semble pas très répandue dans la doc (ou je n'ai pas cherché au bon endroit ?).

Même si le bundle LiipImagineBundle semble adapt, j'avoue être un peu perdu...

Est-ce que vous avez des pistes plus précises ou une documentation claire sur le sujet ?

3- Le gros morceau : les requêtes AJAX et l'utilisation de JQuery.

Comment dire...

Il semblerait que je sois le seul à faire ce que je veux faire, j'explique.

Sur ce sujet la documentation est très volumineuse.

Par contre elle n'est pas toujours très compréhensible (car souvent très technique) ou bien elle n'est pas adaptées (ex : mauvaise version de symfony).

Le vrai problème se situe dans le fait que les requêtes AJAX semblent être la suite logique dans l'interaction (encadrée) de l'utilisateur avec le back.

Mais cette partie-là est encore floue chez nous donc le niveau supérieur nous est encore inaccessible.

J'implore votre aide sur cette partie-là car nous ne voyons pas comment avancer seuls sur cet aspect...

Exemple d'opération que nous souhaitons mettre en place :

Changer dynamiquement l'affichage d'un Chart JS dans un canvas en fonction d'un radio button.

Les 2 graphes à affiché sont réalisés. Le but est de récupérer la valeur du radio button et de la passer en paramètre des fonctions de création de graphe avant de les afficher.

Les fonctions de création marchent correctement quel que soit le paramètre en entrée.

Par contre, la récupération de la valeur de la checkbox cochée est vertigineuse (réussi mais surement pas de la bonne manière).

La vraie difficulté se trouve dans la mise à jour du graphique (zone de canva) sans une relance systématique du chargement de la page -> mais surtout avec les bons paramètres !

# Bibliographie

## Sources

Liens vers les données et sites de collecte :

* <https://www.kaggle.com/datasets/joebeachcapital/top-1000-steam-games>
* <https://gamalytic.com/game-list>
* <https://gamalytic.com/genres>

## Documentation

### Apprentissage et prise en main des technologies

* <https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql>
* <https://openclassrooms.com/fr/courses/1665806-programmez-en-oriente-objet-en-php>
* <https://openclassrooms.com/fr/courses/8264046-construisez-un-site-web-a-laide-du-framework-symfony-7>
* Installation de PHP sur Windows (autre que Xaamp) :

<https://www.youtube.com/watch?v=OVTnj3hcHuc&list=PLjwdMgw5TTLVDv-ceONHM_C19dPW1MAMD&index=3>

* Mieux comprendre l’architecture MVC (Model View Controller) :

<https://www.youtube.com/watch?v=gs-61l4Z32M>

* Mieux comprendre la notion de routeur :

<https://grafikart.fr/tutoriels/router-php-1149>

* Premiers pas avec Doctrine dans Symfony :

<https://symfony.com/doc/6.4/the-fast-track/fr/8-doctrine.html>

### Développement

Symfony UX et Chart js

* <https://symfony.com/bundles/ux-chartjs/current/index.html#using-plugins>
* <https://ux.symfony.com/chartjs>
* <https://www.youtube.com/playlist?list=PLBG4BJt8bo9rrPibhKLbpe-z1Dkk2VJjO>
* Bubble chart avec chart js :
  + <https://www.youtube.com/watch?v=SJxD4gYdX2A>
  + <https://www.youtube.com/watch?v=ySU7wbIEGjk>
  + <https://www.youtube.com/watch?v=7soD4PiWXuY>
  + <https://datavizcatalogue.com/methods/bubble_chart.html>
  + <https://www.chartjs.org/docs/latest/charts/bubble.html>

Gestion du cache :

* <https://www.dnd.fr/tout-savoir-sur-lutilisation-du-cache-dans-symfony/>
* <https://nouvelle-techno.fr/articles/29-la-gestion-du-cache-avec-symfony>

Requêtes depuis le Repository :

* <https://symfony.com/doc/current/doctrine.html#querying-with-sql>

Ajax :

* Simple :
  + <https://www.youtube.com/watch?v=JT-Rq_16HOk>
  + <https://symfony.com/legacy/doc/jobeet/1_4/fr/18?orm=Propel>
* Filtrage :
  + <https://www.youtube.com/watch?v=25IqZbLYoK0>
  + <https://www.youtube.com/watch?v=oG6VWCDK-Ns>