

# Lord of geek

---



Projet Pédagogique : Une approche du MVC

Pré-requis :

- Syntaxe basique de PHP
- POO
- PDO
- Sessions

# Partie 1 : Contexte

Le jeune Loïc est un grand collectionneur de jeux vidéos. Il en a plus de 2000 exemplaires de jeux vidéo. Chaque exemplaire à un état (abimé, bon, neuf).

Il se peut qu'un exemplaire soit en double, voir en triple. Si par exemple, il a eu la chance de trouver la version "collector" il n'a pas pu résister. Il se peut aussi qu'un même jeu, soit sortie sur deux consoles différentes. On différencie donc le **jeu** par exemple Skyrim de **l'exemplaire** :

- un exemplaire sur PS4 en mauvais état
- un exemplaire sur Switch en bon état
- un exemplaire sur PS4 version collector, en bon état

Il souhaite un logiciel pour améliorer la gestion de sa collection.

Il souhaite retrouver rapidement un titre. Si par exemple, on l'appelle pour acheter un exemplaire The Legend of Zelda sur NES. Il souhaite savoir s'il l'a déjà et dans quel état. Si l'exemplaire qu'il a est trop abimé, il n'est pas contre l'acheter de nouveau.

Il aimerait pouvoir les classer, par année de sortie, par genre et par console.

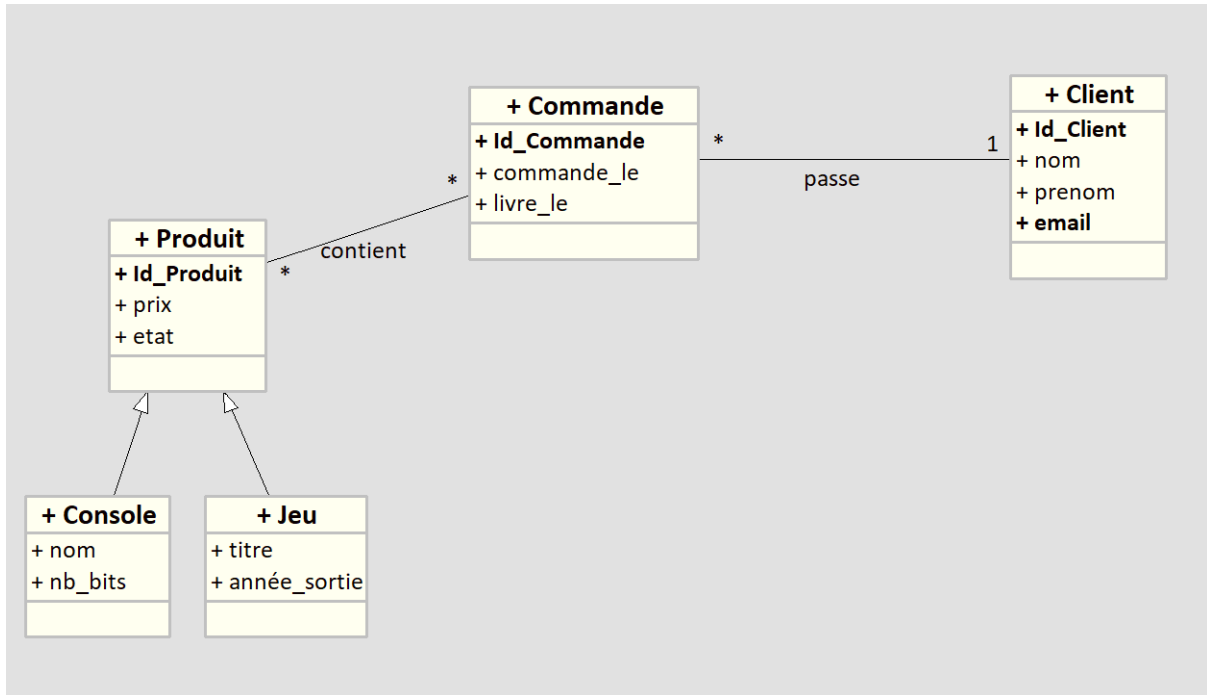
Certains jeux font partie d'une "**serie**" par exemple, assassin creed et assassin creed 2 font partie de la série Assassin Creed et ils sont ordonnés (1, 2, 3...).

Il conserve le prix du jeu qu'il a acheté ainsi que l'année de l'acquisition. Parfois, il ne se souvient plus de quand il l'a acheté ou à quel prix, mais il doit pouvoir enregistrer son jeu tout de même.

## Rattrapage pour la CP 5

Travail à faire ✎

- 1) Proposer des diagrammes MLD qui correspondrait à ses besoins.
- 2) Dérivé le diagramme UML suivant :



Produire le diagramme MLD avec Wordbench. Fournir une image du diagramme au format image.

## Pour tout le monde

Il en a tellement qu'il s'est mis en tête de vendre quelques exemplaires (pas tous !).

Il a commencé à travailler sur un prototype de site commercial. Mais il vous demande de l'aide pour ajouter les fonctionnalités qu'il lui manque.

### Travail à faire

Dans un premier temps, vous allez **étudier ce qu'il a fait**, et lui faire **une proposition d'amélioration par écrit**. Ensuite, vous allez **effectuer lesdites améliorations** qui auront été approuvés.

Voici son travail commenté :

# Prototype de L.O.G

Par Loic - Lord of Geek

---

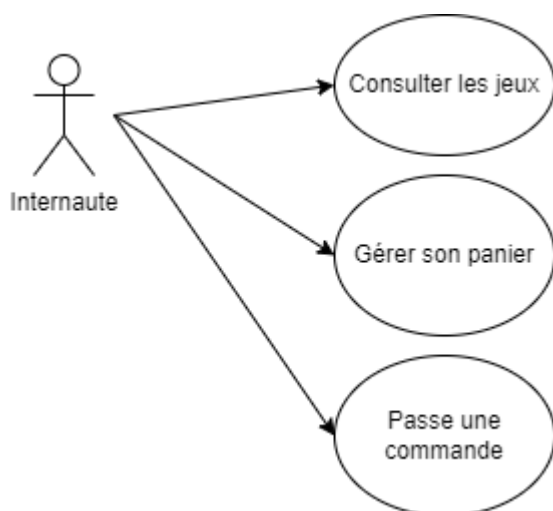
Voici mon repo git : [https://github.com/agnesNeedemand/lord\\_of\\_geek](https://github.com/agnesNeedemand/lord_of_geek)

Par ce document, je vais vous expliquer un peu ce que j'ai voulu faire et ce que j'ai fait.

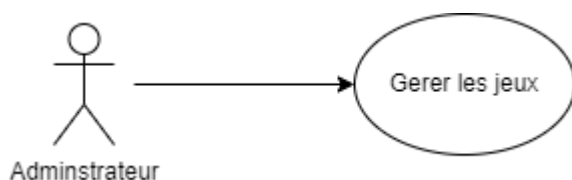
## Cas d'utilisation

Voici les cas d'utilisation que j'ai commencé à cibler.

Trois du *front office* :



Un du *backoffice* :



*Note du formateur : La définition des cas d'utilisation est une étape très importante, c'est à partir de ce découpage que s'organisera l'application ; il ne faut absolument pas négliger cette étape. Les cas du Front office sont déjà développés. Le dernier "use case" est donné en exercice, avec de nombreux éléments de correction*

Les scénarios :

Voici ce que j'ai pu développer et qui devrait fonctionner normalement, si je n'ai pas fait d'erreur.

Use case : Consulter jeux	
Action de l'internaute	Réponse du système
l'internaute demande à consulter les jeux	le système retourne la liste des catégories
l'internaute Sélectionne une catégorie	le système retourne la liste des jeux de cette catégorie.
l'internaute demande d'ajouter un jeu dans le panier	le système ajoute le jeu dans le panier et informe avec un message
Cas particulier	
Le jeu est déjà dans le panier.	Le système en informe avec un message d'erreur

Use case : Gérer son panier	
Action de l'internaute	Réponse du système
L'internaute demande à voir son panier	Le système retourne la liste des articles du panier
L'internaute demande la suppression d'un article.	Le système retire l'article du panier (après confirmation) et retourne la liste des articles du panier
Cas particulier	
Le panier est vide.	Le système en informe avec un message

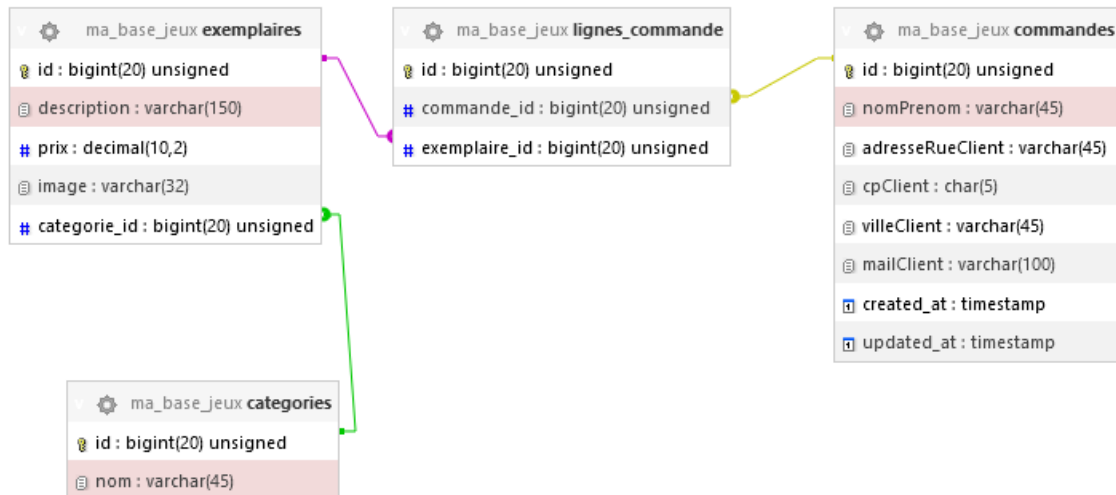
Use case : Passer une commande	
Action de l'internaute	Réponse du système
L'internaute demande à passer commande	Le système retourne un formulaire pour saisies
L'internaute remplit le formulaire et l'envoie	Le système enregistre la commande
Cas particulier	

Le système constate une erreur de saisie	Le système constate une erreur de saisie
Le panier est vide.	Le système informe avec un message

*Note du formateur : le back office sera le sujet du prochain TP, il sera fait en Laravel. Pour l'instant les données seront à entrer directement depuis la BDD à la main.*

# Le diagramme MLD de la base de données

La base de données est sous MySQL. J'ai choisi la modélisation suivante :



En [annexe](#) dans le cours, pour générer la base de données. Elle n'est pas finie. J'aimerais, par exemple, pouvoir ajouter l'état de chaque jeu, je pense que c'est important et que l'image ne suffit pas. J'attends vos propositions.

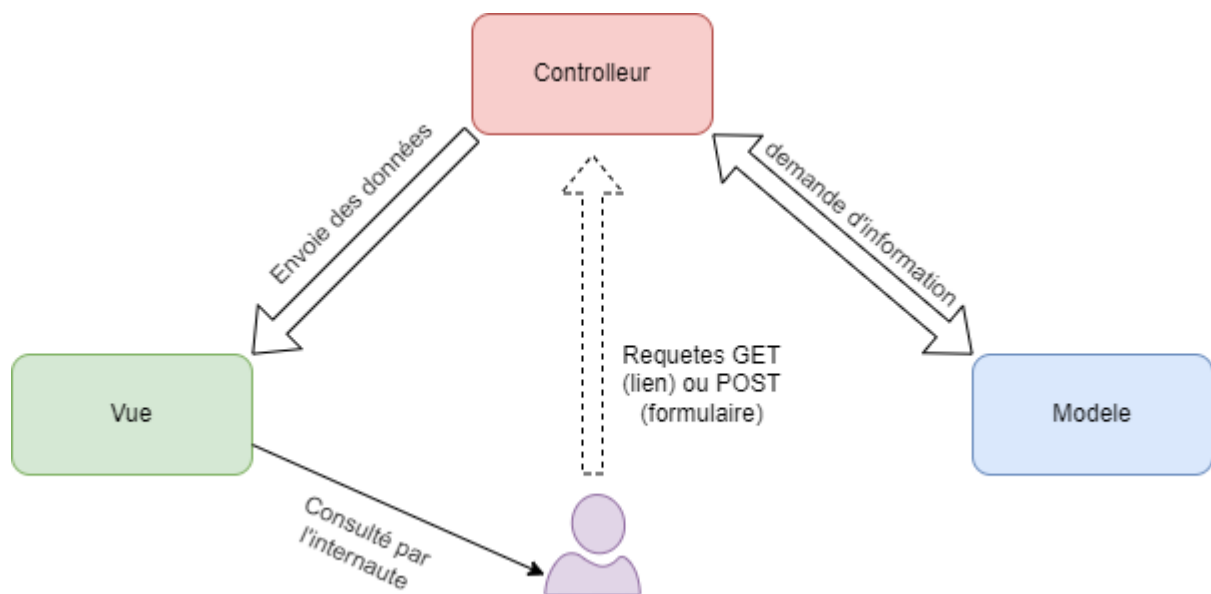


# Le modèle MVC

Pour l'organisation du code, j'ai choisi de suivre le modèle MVC.

- Modèle
- Vue
- Contrôleur

*Note du formateur : Il s'agit d'un **design pattern** en français, un patron de conception; Suivre cette organisation fait partie des bonnes pratiques et vous permet de ne pas vous perdre dans votre code.*



Je me suis efforcé à bien séparer les trois. Et je souhaite que vous en fassiez autant pour continuer le projet.

## 1- Le modèle

Je m'impose une règle d'or : **Seul le modèle a accès la base de données !!!!**

Ce que j'appelle modèle est la couche qui accède à la base de données. Cela correspond à l'ensemble des fichiers qui sont dans le dossier **modèle**; Tous les fichiers ont un M majuscule.

*Note du formateur : Par convention, les classe doivent commencer par une majuscule, et le fichier doit avoir le même nom que la classe;*

J'aurais pu faire de même pour les contrôleurs, et travailler avec des classes pour les contrôleurs, mais j'ai choisi de faire simple (je suis un débutant). Je ne sais pas ce que vous en pensez...

J'utilise la classe PDO, dédiée à l'accès aux données. Cette classe est présente par défaut dans PHP, mais j'avais besoin d'une connexion à MA base de données. J'ai alors créé la classe "**AccesDonnées**" qui fait ce travail.

J'ai choisi le "**static**" pour qu'un seul objet PDO, soit utiliser à chaque fois pour toutes les requêtes. j'aurais pu faire autrement, bien sûr. Mais je voulais bien séparer chaque chose. Ainsi, j'ai aussi opté pour regrouper les requêtes qui concerner la même table ensemble. À chaque table, j'ai créé une classe modèle qui fait des requêtes sur cette table"

Je vous laisse observer mon travail. J'ai lu des articles sur l'injection SQL, et je pense que ma façon de faire les requêtes est à revoir...

## 2- La vue

La vue (V) représente ce qui est exposé à l'utilisateur, en général, il s'agit de HTML statique ou généré en PHP. Il y a deux sortes de vue :

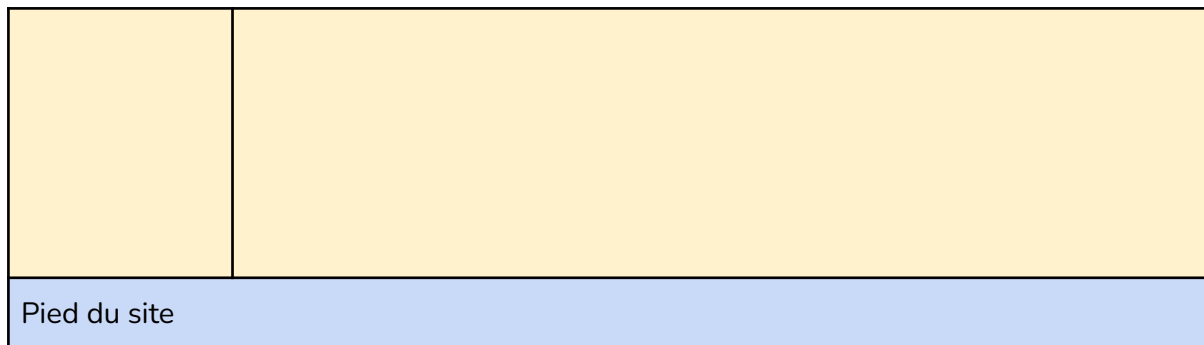
- les pages d'informations navigables grâce aux liens
- les formulaires de saisies d'informations. Ces formulaires peuvent être présentés à plusieurs reprises pour confirmation ou signalement d'erreurs

Vous pouvez les consulter en [annexe](#).

Une vue propose une partie commune à chaque page et des zones liées à la demande de l'utilisateur.

Voici la structure que je souhaitais pour mon site:

Entête - menu de navigation	
Sous-menu de navigation  (Seulement sur certaines pages)	Contenu ( qui change en fonction de la page )



- La partie bleue est Statique.
- La partie jaune est Dynamique.

Aussi, vous trouverez le **template.php** qui contient la partie statique et les différentes vues qui seront inclus dans la partie dynamique par l'intermédiaire de ce que j'ai appelé un "contrôleur de contenu".

Toutes mes vues sont dans le dossier "\_vue". Les vues contiennent des données. Ces données sont fournies par le contrôleur.

**Attention : En php, faire appelle à une variable qui n'est pas définie créer une erreur.**

## 3- Le contrôleur

### 3-1 Contrôler la vue

Ce sont les contrôleurs qui vont être à l'écoute des requêtes de l'utilisateur et fournir ainsi la vue externe correspondante. Pour cela, il faudra, à tout moment, connaître l'état de l'application.

J'ai choisi la variable `$uc` pour "use case" qui m'apporte cette information.

Contrôleur de vues ( dans template.php ) :

```
switch($action){  
    case 'ceci' :  
        include("_vue/v_ceci.php");  
        include("_vue/v_encorececi.php");  
}
```

```
        break;
    case 'cela' :
        include("vues/v_cela.php");
        break;
}
```

## 3-2 Contrôler les données

Pour chaque use case ( cas d'utilisation ) j'ai créé un contrôleur que vous pouvez observer dans le code. Ils sont dans la partie "**contrôleur**". C'est ici que j'effectue les appels au modèle, pour récupérer les données. Ces données sont distribuées à la vue.

Un contrôleur principal qui oriente vers les différents contrôleurs dans le fichier **index.php**. Encore un switch !

Le fichier index.php est appelé le contrôleur général. C'est ici qu'on décide de tout, et tout passe par lui...

D'ailleurs, toutes les pages commencent par : "*index.php?uc=*"

## 3-3 Contrôler la logique

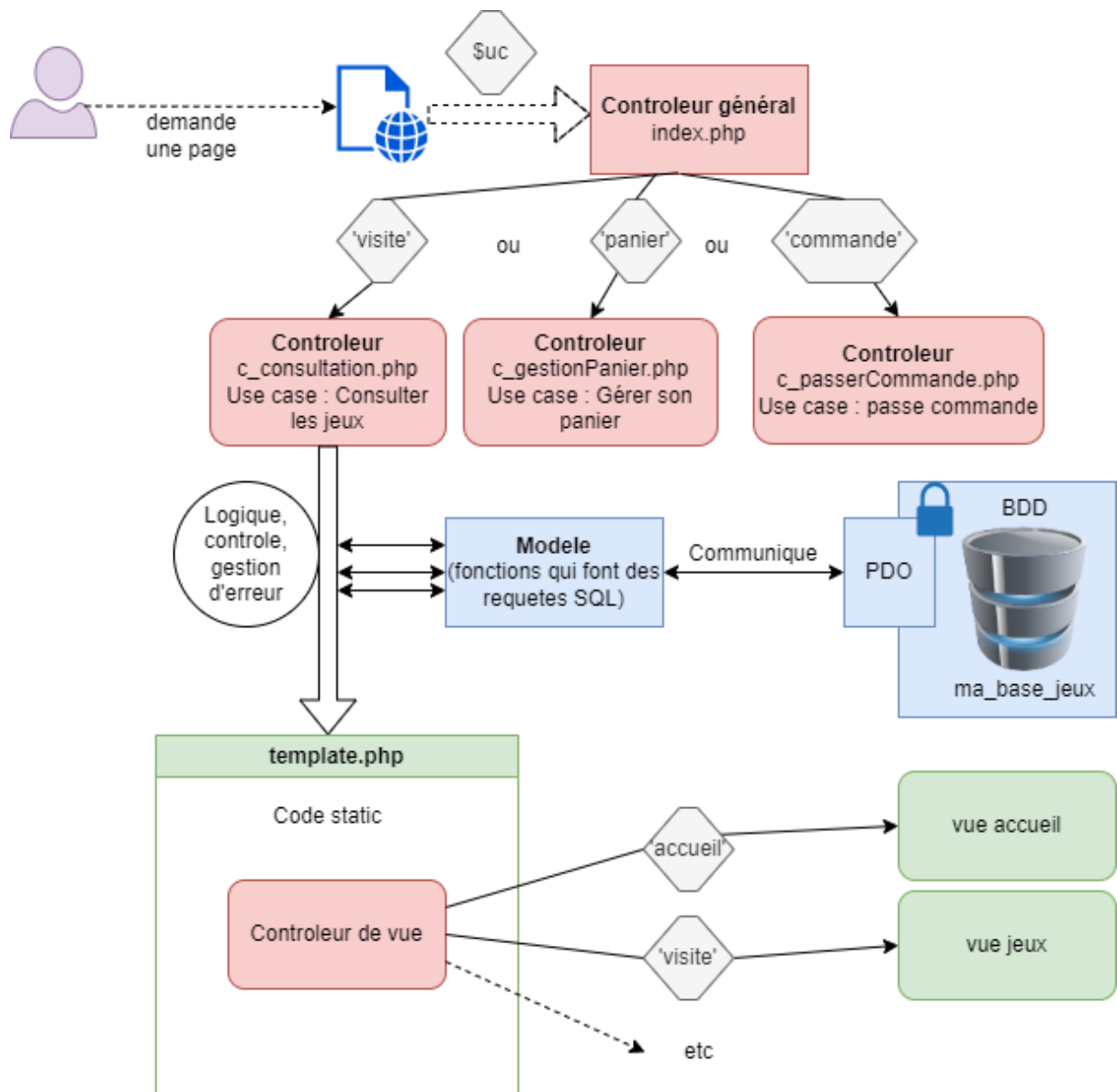
Puis, j'ai eu besoin d'affiner cette information, j'ai alors utilisé aussi la variable **\$action** pour l'action que veut faire l'utilisateur.

Voici un exemple, pour le cas, de la gestion de panier (**c\_gestionPanier.php**), les actions suivantes sont possibles :

- voir le panier
- supprimer un jeu du panier

C'est ici que j'ai placé la **logique** du site.

Finalement, vous pouvez consulter mon fonctionnement :



# Travail à faire

---

- 1) Dans un premier temps, analysez le prototype en vous aidant des commentaires ci-dessus. Faites fonctionner le site sur votre serveur local.
- 2) Finir la partie gestion qui doit permettre d'ajouter, de modifier ou de supprimer des jeux sur le site.
- 3) Proposer des améliorations fonctionnelles possibles qui seront présenter sous forme d'un tableau. Au moins 10 améliorations demandées.

*Attention : Ici, on ne s'intéresse qu'à la partie fonctionnelle. Les améliorations visuelles ne sont pas interdites, mais ne doivent pas apparaître dans le tableau.*

Exemple de tableau :

N°	Amélioration fonctionnelle	Approuvé ?	Réussie ?
1	Ajouter un état sur les jeux	oui	
2			
3			
4			
5			
6			
7			
8			
9			
10			

# Annexe

---

## Annexe : Les différentes vues

### Vue de l'accueil



**Lord Of Geek**

**le prince des jeux sur internet**

Accueil	Voir le catalogue de jeux	Voir son panier	Mon compte
---------	---------------------------	-----------------	------------

Je n'ai pas encore eu le temps de réfléchir à la page d'accueil... pour l'instant c'est simple.

### Vues des catégories




Aventure
Combat
Logique

Accueil	Voir le catalogue de jeux	Voir son panier	Mon compte
---------	---------------------------	-----------------	------------

Ici, je voudrais afficher tous les jeux quand aucune catégorie n'est sélectionnée. Hélas, je n'ai pas eu le temps :(

## Vue de la catégorie "Aventure"




AccueilVoir le catalogue de jeuxVoir son panierMon compte

Aventure


Combat

Logique



The Legend of zelda sur NES

Prix : 30.00 Euros



Skyrim sur 360

Prix : 10.00 Euros

## Image par défaut quand le jeu n'en a pas.



## Vue du panier non vide



AccueilVoir le catalogue de jeuxVoir son panierMon compte



**Votre panier**



The Legend of zelda sur NES(30.00 Euros)



Skyrim sur 360(10.00 Euros)



Tekken(5.00 Euros)





### Formulaire de saisies d'information

Commande

Nom Prénom\*

rue\*

code postal\*

ville\*

mail\*

Valider

Annuler

“Peut-être qu’il serait intéressant que l’utilisateur puisse retrouver ces commandes... mais je ne sais pas comment faire”.

### Les messages d'erreurs :

■ Ce jeu est déjà dans le panier !!

### Les messages informatifs :

Ce jeu a été ajouté

