|  |
| --- |
| MacroHard  Alex-Sam-Mathis  P-Web2  Grégory Charmier |

Table des matières

[1.0 Introduction 4](#_Toc103972379)

[1.1 Objectifs 4](#_Toc103972380)

[1.2 Outils 4](#_Toc103972381)

[1.3 Prérequis 4](#_Toc103972382)

[2.0 Sujet 4](#_Toc103972383)

[2.1 Mise en place d’un site avec Laravel 4](#_Toc103972384)

[2.2 Utilisation du framework Tailwind 4](#_Toc103972385)

[3.0 Planification 5](#_Toc103972386)

[3.1 Organisation sur GitHub 5](#_Toc103972387)

[3.2 Organisation au sein du groupe 5](#_Toc103972388)

[4.0 Réalisation 6](#_Toc103972389)

[4.1 Mise en place des éléments 6](#_Toc103972390)

[4.1.1 Heroku 6](#_Toc103972391)

[4.1.2 Laravel 6](#_Toc103972392)

[4.1.3 Tailwind 6](#_Toc103972393)

[4.2 Git et Github 6](#_Toc103972394)

[4.3 Maquettes du site 7](#_Toc103972395)

[4.4 Base de données 8](#_Toc103972396)

[4.4.1 MCD 8](#_Toc103972397)

[4.4.2 MLD 9](#_Toc103972398)

[4.5 Laravel 9](#_Toc103972399)

[4.5.1 Modèles 9](#_Toc103972400)

[4.5.2 Factories 11](#_Toc103972401)

[4.5.3 Migrations 12](#_Toc103972402)

[4.5.4 Seeders 13](#_Toc103972403)

[4.5.5 Routes 14](#_Toc103972404)

[1.1.1 Jeton CSRF 14](#_Toc103972405)

[1.1.2 Controllers 15](#_Toc103972406)

[1.1.3 Views 15](#_Toc103972407)

[1.2 Tailwind 16](#_Toc103972408)

[1.3 (Heroku) 16](#_Toc103972409)

[2.0 Conclusion 16](#_Toc103972410)

[1.1 Éléments atteints 16](#_Toc103972411)

[1.2 Éléments non atteints 16](#_Toc103972412)

[1.3 Eléments bloquants 16](#_Toc103972413)

[1.4 Comment nous améliorer 17](#_Toc103972414)

[1.5 À refaire ? Avis personnel 17](#_Toc103972415)

[3.0 Bibliographie / Webographie 17](#_Toc103972416)

# Introduction

## Objectifs

* Développer ses capacités analytiques.
* Mettre en œuvre les connaissances apprises dans les modules 133 et 151.
* Mettre en place un site avec l’utilisation du MVC grâce à Laravel.
* Utilisation du Framework Tailwind.

## Outils

* Pc des laboratoires standard
* Vs Code
* Serveur local, Uwamp et DB Main
* Accès à Internet
* Microsoft Office
* Github

## Prérequis

Modules 100, 101, 431, 104, 105, 133, 151, 214, 120

# Sujet

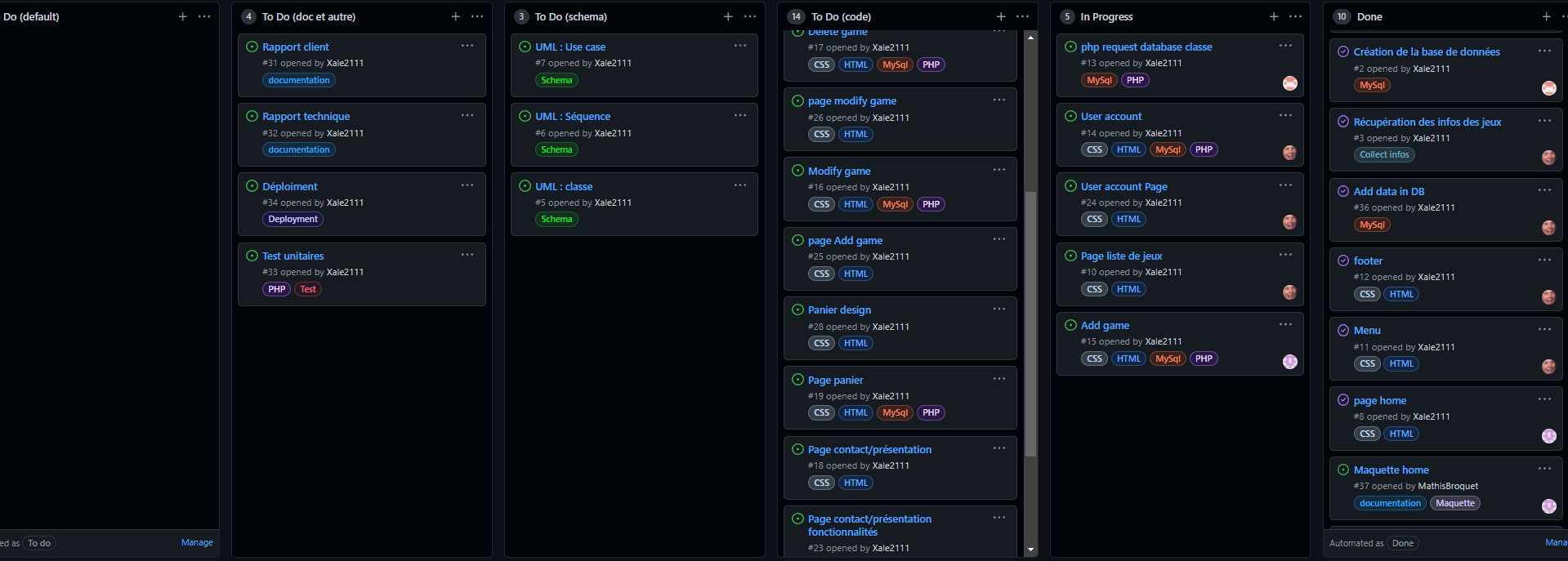
## Mise en place d’un site avec Laravel

Créer une plateforme d’achat de jeux sur laquelle on peut se connecter afin d’ajouter au panier ou s’il on est administrateur, ajouter, modifier ou supprimer un jeu. La mise en place du site web se fera grâce à Laravel. Il faudra donc apprendre et comprendre comment fonctionne Laravel et utiliser les différents outils mis à disposition par Laravel tel que les layouts, les migrations, les vues avec l’extension blade.php ainsi que tout ce qui a un rapport avec le MVC comme les contrôleurs et modèles.

## Utilisation du framework Tailwind

Le CSS sera fait grâce l’utilisation de Tailwind. Il faudra également modifier le fichier tailwind.config.js afin d’ajouter des images et des couleurs. L’utilisation de Tailwind permet entre autres de participer à l’éco-conception web. En effet, grâce à Tailwind, il n’y a plus besoin de créer de feuille de style CSS car tout y est compactés dans les balises HTML. Cela réduit donc le nombre de fichier présent sur le serveur. Grâce à Tailwind, il y a la possibilité de créer des « layout » de base, pour le header/footer du site par exemple. Il y a donc que 1 seul fichier qui contient le header et footer du site. Il suffit juste d’appliquer ce fichier sur les autres fichiers pour avoir le header et footer déjà fait. Cela minimise aussi le code HTML, ce qui rend les fichiers moins lourds et gourmands.

# Planification



## Organisation sur GitHub

L’organisation sur le github est géré en 3 catégories : ToDo, In Progress, Done. La catégorie ToDo est elle-même séparée en 4 colonnes : ToDo(default, là où arrive les issues par défaut), ToDo(Doc et autre, pour la documentation, les tests et le déploiement), ToDo (schema, pour les schémas. Soit uml, maquette et MCD/MLD), ToDo(Code, pour tous les éléments liés au code).

## Organisation au sein du groupe

Pour l’organisation au sein du groupe, cela s’est fait plutôt naturellement. Samuel c’est occupé de tout ce qui avait un rapport avec la BD donc MCD/MLD, Migrations Laravel et créer la BD ainsi que le modèle Laravel. Il a également ajouté des factories. Mathis, quant à lui, c’est occupé des maquettes avant de passer à l’ajout de jeux sur le site. Alexandre a commencé par le front-end du site (header, menu, footer), soit HTML et le CSS avec tailwind. Il a également ajouté le layout pour les vues dans laravel avant d’ajouté la gestion d’utilisateurs.

# Réalisation

## Mise en place des éléments

## Heroku

Mathis a configuré Heroku pour le faire fonctionner dans une branche annexe (juste pour tester au début du projet). Mais on n’a pas eu le temps de changer cela dans le vrai projet et de faire le CI car on manquait de temps.

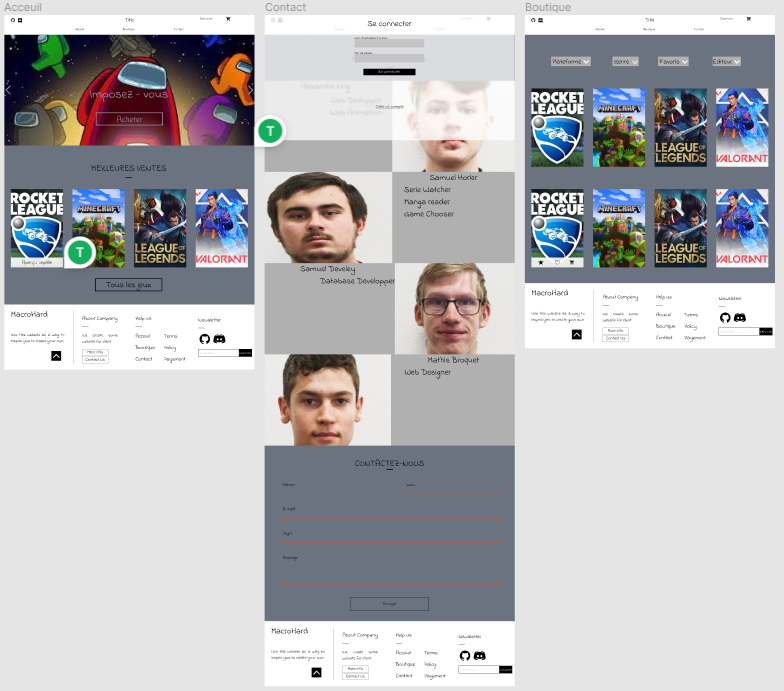
## Laravel

Pour pouvoir installer Laravel nous avons utilisé composer. En même temps de créer le projet on l’a lié à un projet GitHub.

## Tailwind

Tailwind peut s’installer de 3 manières différentes : CLI, Using PostCSS, Play CDN. Nous avons choisi de l’installer avec CLI car nous trouvons cela plus simple et moins complexe à implémenter.

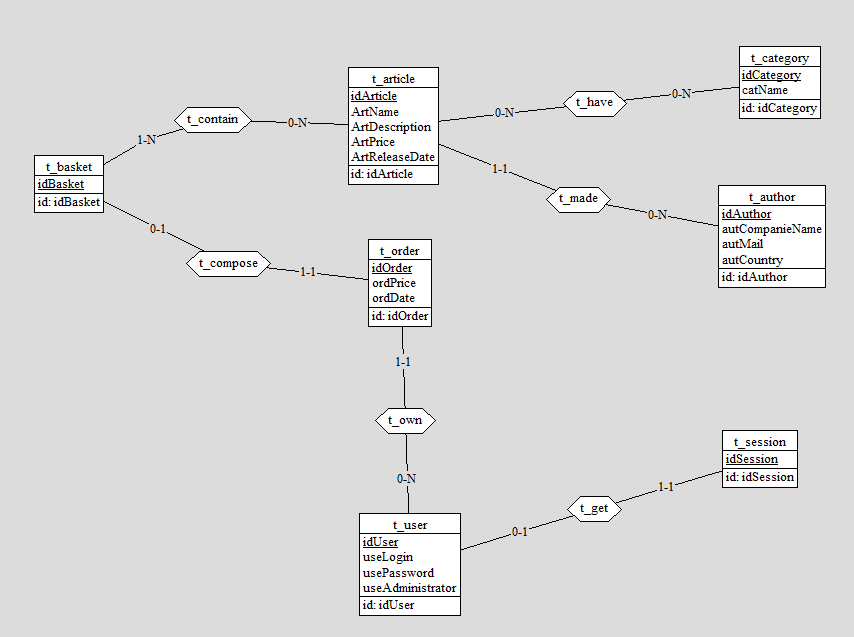
## Maquettes du site



Pour les maquettes du site, faite avec Figma, on a fait que 3 pages car cela prenait trop de temps et que chaque fois que Monsieur Hörler venait il ne fessait rien ou on devait repasser derrière lui.

## Base de données

## MCD



Nous pouvons voir que nous avons conçu un modèle MCD relativement complexe avec plusieurs tables avec différents types de relations. Nous avons donc fait un model assez précis. Nous pouvons aussi observer que nous aurons affaire à des tables intermédiaires sur les relations 1-N/0-N et 0-N/0-N entre les tables : t\_basket/t\_article et t\_article/t\_category. Cela permet à un article d’appartenir plusieurs catégories. Pour le panier, cela lui permet d’avoir plusieurs fois le même article. Pour ce modèle, nous avons donc prévus plusieurs systèmes :

* Jeu :

Pouvoir avoir un maximum d’information sur un jeu (sa/ses catégorie(s), son auteur) et toutes les informations relatives au jeu lui-même (prix, nom, description, date de sortie etc..)

* Commandes :

Nous avons fait en sort que l’utilisateur puisse avoir un système de panier et de commande à disposition. Il pourra enregistrer plusieurs jeux dans un panier et créer une commande. L’utilisateur pourra avoir un historique des commandes et voir le détail de celle-ci.

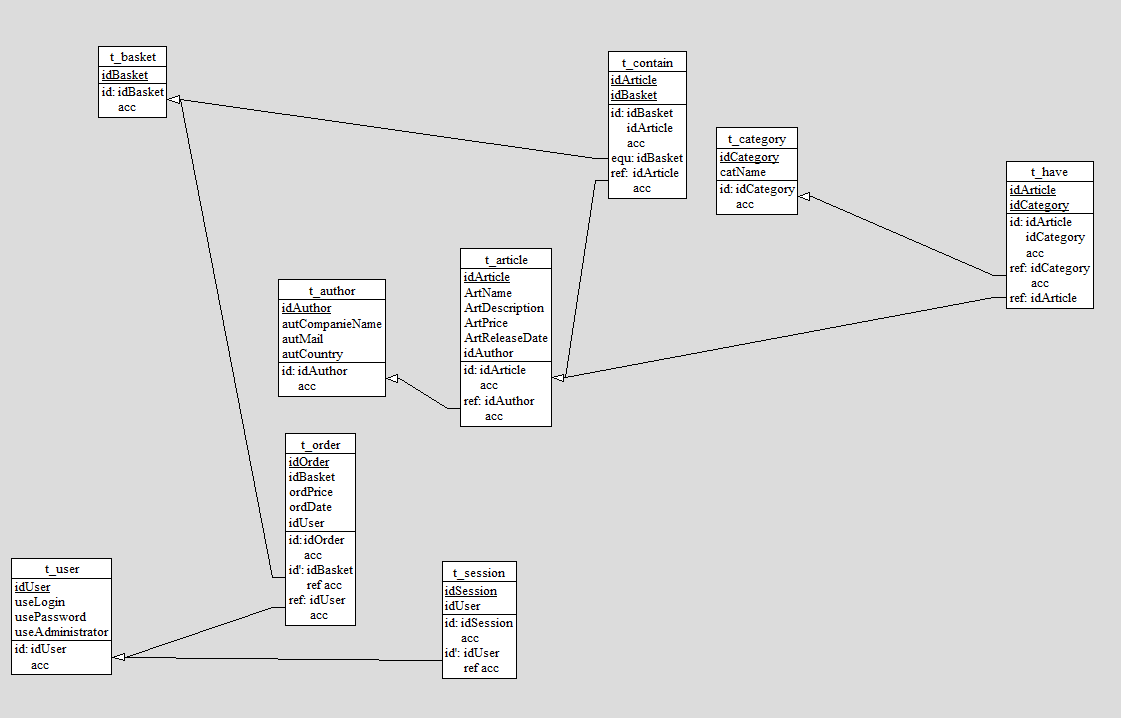
* Login :

L’utilisateur pourra se créer un compte et se loguer. L’utilisateur ne pourra pas passer de commandes tant qu’il n’est pas connecté à un compte.

* Session :

L’utilisateur aura aussi une session afin de stocker un cookie de session pour garder en interne pendant un certain temps, certaines informations en rapport avec lui (login, password). L’utilisateur n’aura donc pas besoin de se loguer à chaque fois qu’il visite le site.

## MLD

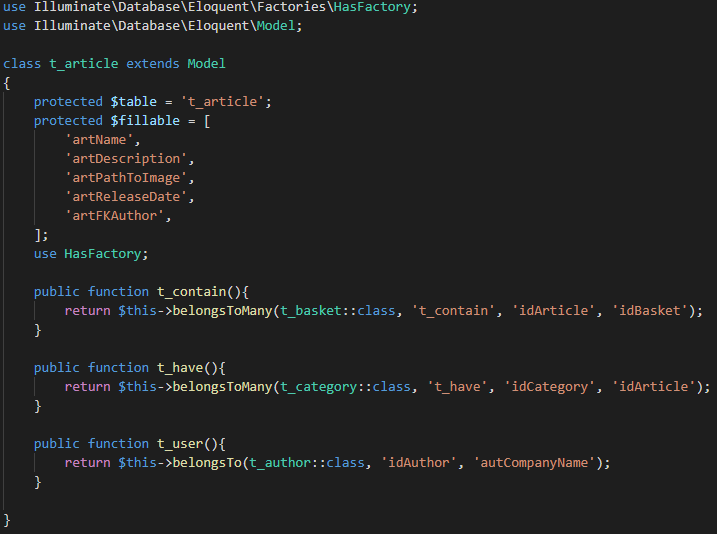


## Laravel

## Modèles

Les modèles servent à décrire en interne, comment les relations sont définies entre les tables de la DB, comment et quelles informations vont pouvoir transiter entre chacune des tables. Ils peuvent utiliser des factories mais je vous en reparle plus tard. L’utilisation des modèles permet l’utilisation de l’ORM Eloquent, un objet assez puissant de Laravel, permettant de sécuriser facilement les requêtes envoyées à la DB. Il permet par la même occasion de simplifier les requêtes

Les modèles se présentent sous cette forme :



C’est une classe qui possède différents éléments, variables et fonctions utiles. On peut facilement les générer via la commande suivante « php artisan make:model nameOfModel –m ».

* L’élément « protected $table = ‘t\_article’ » permet de ‘renommer’ la table avec le nom correcte, si nous n’utilisons pas les normes Laravel.
* L’éléments «  protected $fillable = [] » permet d’indiquer à Laravel, quels vont être les champs de la table avec lesquelles il va pouvoir intéragir (donner/recevoir des données).
* Les relations entre les tables :

Il existe plusieurs type de méthode pour définir les relations entre les tables.

Pour les relations 1-1 nous utliserons la méthode « hasOne()» ainsi que son inverse « belongsTo() ».

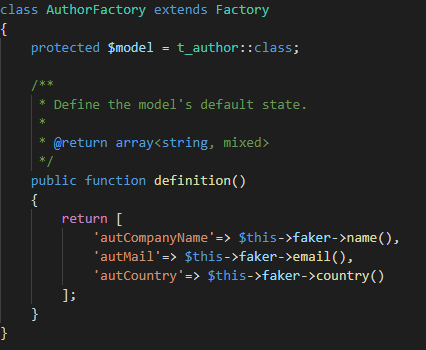
Pour les relations 1-N nous utiliserons la méthode « hasMany() » ainsi que son inverse « belongsTo() ».

Pour les relations N-N nous utiliserons la méthode « belongsToMany() » sur ce type de relation, il va falloir jouer avec une table intermédiaire.

## Factories

Les factories servent à générer facilement des quantités de données aléatoires afin de remplir la DB avec des données de test. Elles vont passer par l’intermédiaire des modèles.

Les factories se présentent sous cette forme :



C’est une classe qui possède différents éléments, variables et fonctions utiles. On peut facilement les générer via la commande suivante « php artisan make:factory Namefactory --model=NameModel». Il faut lier la factories au modèle. Nous pouvons créer des données manuellement via les commandes suivantes :

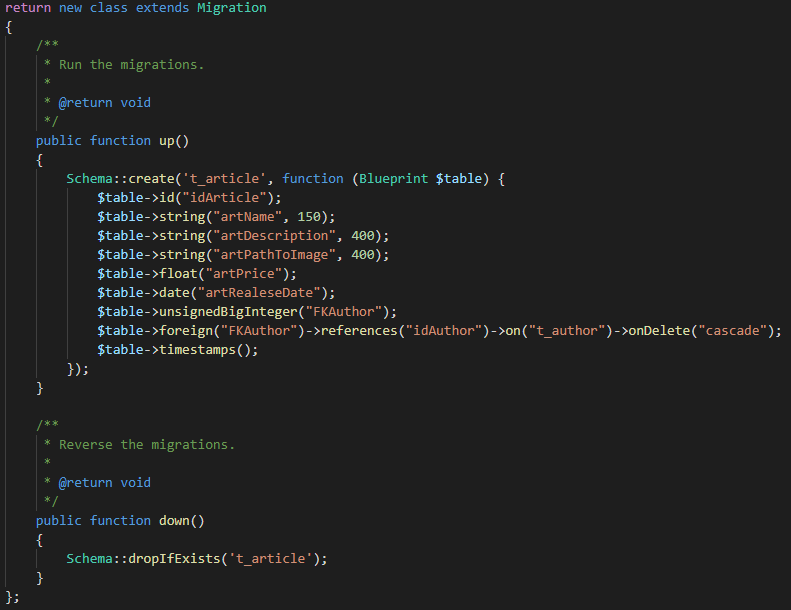
1. php artisan tinker
2. NameModel::factory()->count(10)->create();

La première commande va ouvrir une sorte de « bac-à-sable » où l’on pourra exécuter des commandes permettant la génération de données aléatoire manuelle en appelant le model que l’on veut (2eme commande).

* L’élément « protected $model =t\_author::class» permet de lier le model à la factorie.
* La fonction « definition() » permet la génération de donnée sur demande, à l’aide de l’objet « faker ». Il est possible de générer : des noms, mails, textes, nom de pays, numérot de télépones etc…

## Migrations

Les migrations servent à l’interaction directe avec la DB sur phpMyAdmin par exemple. C’est grâce aux migrations que nous définissons le nom de la table, le nom de ses attributs ainsi que leurs types. Nous pouvons aussi définir les relations entre chaque table en précisant les clés étrangères.



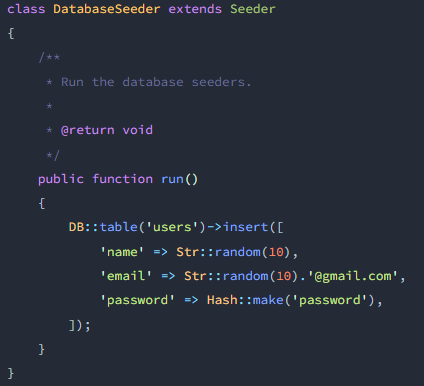
C’est une classe qui possède différents éléments, variables et fonctions utiles. On peut facilement les générer via la commande suivante « php artisan make:migration NamefOfMigration ». Nous pouvons exécuter les migrations via la commande suivante «  php artisan migrate » ou « php artisan migrate:refresh ».

* Dans la fonction « up() » , nous définissons : le nom de la table, ses attributs et leurs types ainsi que l’espace occupé. Nous renseignons aussi les clés étrangères de la table.
* Dans la fonction « down() », nous pouvons drop la table existante. Cela est utilisé lors d’un « php artisan migrate:reset » ou d’un « php artisan migrate :rollback --step=2» par exemple. Cela veut donc dire que Laravel garde en mémoire les différentes migrations que nous avons fait durant le projet.

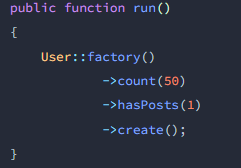
## Seeders

Les seeders permettent d’exécuter les factories faites précédemment mais de manière plus directe et automatisée. Ils permettent donc la génération plus simplifiée de données dans la BD.

Les seeders se présentent sous cette forme :



On peut facilement les générer via la commande suivante « php artisan make:seeder NameOfSeeder ». La méthode « run() » est exécutée lors de l’utilisation de la commande « php artisan db :seed » . Ici nous indiquons directement à Laravel sur quelle table et quels champs il va devoir insérer les données.



Ici, nous passons par l’intermédiaire d’une factory que nous avons créé précédemment. Nous créons 50 utilisateurs qui sont donc eux même lié directement à un titre (hasPost(1)).

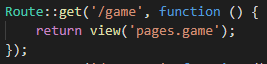


Nous pouvons utiliser la méthode « call() », pour exécuter tous les seeders que nous voulons.

Les seeders s’exécutent lors de la commande « php artisan db :seed » ou « php artisan db :seed –class=nameOfSeeder » pour spécifier un seeder. Il est possible d’utiliser la commande « php artisan migrate :fresh –seed » pour exécuter les migration avec les seeders.

## Routes

Les routes, vont lire les requêtes http qui transitent afin de pouvoir détecter quelle page il faut envoyer à l’utilisateur et quelles actions il faut faire. Les routes sont aussi bien utilisées pour les requêtes de type POST que pour les requêtes de type GET.



Dans la route, on peut aussi définir quand un contrôleur va être appelé et avec quelle méthode. Il est aussi possible de renommer sa route. 

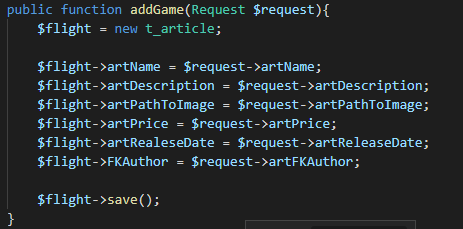
## Jeton CSRF

Le jeton CSRF est présent pour sécuriser l’application au niveau de la falsifications requêtes intersites. Ce qui ne permet pas à l’utilisateur de rentrer des commandes malveillantes. La protection CSRF est utile pour sécuriser les formulaires en particulier.

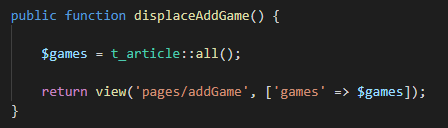
## Controllers

Les contrôleurs, permettent de faire la jonction entre les vues et les modèles, qui eux interagissent avec la DB.

Ici, nous pouvons l’interaction entre le contrôleur et un modèle, afin d’ajouter un jeu dans la DB.



Ici nous voyons clairement l’utilité et le fonctionnement du contrôleur. Il va commencer par récupérer tous les articles en DB puis va les envoyer à la vue spécifiée sous le nom de variable « games ».



## Views

Les vues sont l’interaction directes avec l’utilisateur. C’est-elle qui vont gérer l’affichages des éléments en les reprenant depuis le contrôleur. Comme ci-dessous.



## Tailwind

Pour ce projet, il a fallu réapprendre le CSS. Grâce à la documentation, cela n’a pas été trop complexe. Alors une fois Tailwind appris, le CSS du site à facilement pu être mis en place. L’accès au fichier config.js a également eu lieu Pour l’ajout de couleurs personnalisées et d’images.

## (Heroku)

Pour Heroku Mathis l’a mis en place et il fonctionnait mais sur la mauvaise branche. On n’a pas eu le temps de le changer car on a eu trop peu de temps.

# Conclusion

## Éléments atteints

* Une page d’ajout d’un jeu (accès administrateur)
* Une page d’accueil affichant les jeux les plus vendus et dernières sorties (tout sera en fonction de ce qui se trouve en DB) (la page d’accueil est présente mais pas dynamique)
* Création d’un compte client

## Éléments non atteints

* Une page comprenant la liste des jeux (accès tout public)
* Une page d’information sur nous et un moyen de contact. (accès tout public)
* Un panier avec les jeux ajoutés avant la confirmation de la commande.
* Modification et suppression d’un jeu

## Eléments bloquants

* Pas de connaissance sur le Framework Laravel
* Pas de connaissance de Tailwind
* Monsieur Hörler n’était presque jamais présent durant le projet
* Nous avons découvert le dernier jour que la base de données n’a pas été implémentée correctement au niveau des schéma. En effet, il y a une table t\_basket (représentant le panier d’achat), qui n’a pas trop de sens dans ce contexte. En effet, un panier est temporaire. Il aurait donc fallu stocker les informations du panier dans un cookie par exemple et ainsi garder que la table t\_order en DB, pour l’historiques des commandes.
* Problèmes avec git. En effet, par manque de compréhension de git, nous nous sommes mis à travailler à 3 sur la même branche donc nous avons eu plusieurs fois des problèmes/conflit avec les fichiers. Ce qui nous a pris pas mal de temp à régler parfois.
* Mauvaise gestion du temps mis à disposition et trop d’ambition. Donc manque de temps pour finir le projet.

## Comment nous améliorer

* Prendre moins de temps pour faire les maquettes
* Découper le projet en plus de petites tâche (sur GitHub)
* Apprendre toutes les étapes pour afficher une page (connexion db, récupération/gestion de l’infos, affichage, …)
* Prendre plus de temps pour créer la base de données et réfléchir correctement à sa modélisation car c’était un élément bloquant en fin de projet

## À refaire ? Avis personnel

Mathis : Je pense que si je devais refaire le projet je passerais moins de temps sur la modélisation du site et je vérifierais plus la DB. Ensuite j’essayerai de tout implémenter une fois pour tout comprendre et que cela aille plus vite pour les autres vues.

Samuel : Si je devais refaire ce projet-là, je partirai avec une base de données beaucoup plus simple à gérer afin de comprendre plus facilement tout le côté back-end de Laravel (models, migrations, factories, seeder). Ensuite je pense qu’implémenter des petits ajouts complètement (du back-end 🡪 front-end) serai plus utile pour mieux comprendre le fonctionnement de Laravel en général.

# Bibliographie / Webographie

[Documentation laravel](https://laravel.com/)

[Documentation tailwind](https://tailwindcss.com/)

[NorthCoders](https://www.youtube.com/c/Northcoders)

[Laraguide](https://www.youtube.com/watch?v=BQo-3ohwL5Y&list=PLMWEEzYqZ0em1vnBx8F5GZd94Ejph4TjO)

[Stackoverflow](https://stackoverflow.com/)

[Figma](https://www.figma.com/)