**Mes notes de formation laravel 8 suivant le lien :** <https://m.youtube.com/watch?v=EaSgftRyvAM&list=PLeeuvNW2FHVj4vHJRj9UDeDsXshHlnHJk>

* Composer est le gestionnaire des librairies php. Il permet de gérer les dépendances lors de l’installation des différentes librairies.
* Une route peut renvoyer une chaine de caractère qui sera « parsée » en réponse http
* Une route « closure » retourne une fonction sans nom dont la valeur de retour peut être une vue, une chaine de caractère, une réponse json (sous la forme clé=> valeur), …
* Les vues peuvent et devraient être retournées depuis un contrôleur pour plus de lisibilité du code. Pour retourner une vue depuis le contrôleur, il faut créer le contrôleur (soit manuellement, en prenant soin de mettre le namespace (namespace App\Htpp\Controllers ;) et de faire hériter ce contrôleur de la class Controller, soit en ligne de commande via php artisan make : controller).
* Pour retourner des données à travers une vue, on utilise la fonction ‘compact’ sous forme de chaine de caractère (compact(‘nom\_variable’)). Dans les vues, les variables passées depuis le contrôleur sont utilisées entre double accolades {{ }}. On peut également utiliser la fonction with qui prend en paramètre une clé (la clé étant le nom de la variable sous forme de chaine de caractère, donc entre quotes et sa valeur étant la variable en elle-même. Ex : ->with(‘title’, $title). Lorsqu’il y a plusieurs variables à retourner, on peut utiliser compact ou alors retourner un tableau de clé-valeur

Ex : return view(‘article’, [

‘title’=>$title,

‘title2’=>$title2

]) ;

Généralement, les données passées aux peuvent être des tableaux. Pour utiliser les éléments du tableau dans la vue, on indexe la variable à notre convenance.

* Les « blade » sont des générateurs de templates (gère les vues html en les raccordant avec les contrôleurs)
* Pour passer des paramètres aux routes, on les met entre accolades et comme argument lors de l’appel dans la vue concernée
* Le nom donné aux routes(->name(‘nomroute’)) est avantageux dans la mesure où on peut changer l’arborescence de la route sans avoir à toucher aux fichiers concernés par cette route
* Gestion des relations entre les tables

*Relation 1 à plusieurs (one to many)*

Ex : un post possède un ou plusieurs commentaires. La table commentaire aura comme clé étrangère une référence à la table post qui va indexer l’id du post pour faciliter la recherche.

Dans la définition de la migration associée à la création du modèle commentaire, on écrire les lignes suivantes :

$table->unsignedBigInt(‘post\_id’) pour définir la colonne de la clé étrangère

$table->foreign(‘post\_id’)->reference(‘id’)->on(‘posts’) pour associer à la colonne nouvellement créée la fonction de clé étrangère.

Dans la classe « 1 », on crée une fonction qui retourne « hasMany(ClassName :: class) » ; dans la classe « plusieurs », on crée une fonction qui retourne $this->belongsTo(ClassName ::class)

Petite astuce lorsqu’on boucle sur un tableau : pour gérer le cas du tableau vide, on peut utiliser la structure :

@forelse($variables as variable)

//my code

@empty

//my code if the array is empty

@endforelse

*Relation 1 à 1 (one to one)*

Ex: un post possède une et une seule image, une n’appartient qu’à un seul post

En respectant la cohérence de la relation, le modèle post possède une fonction qui retourne $this->hasOne(Image ::class) tandis que le modèle image a une fonction post() qui retourne $this->belongsTo(Post ::class)

*Relation plusieurs à plusieurs (many to many)*

Pour de telles relations, nul besoin de clés étrangère, mais une table intermédiaire doit être créée appelée table pivot qui matérialise la relation entre les modèles en stockant au moins les différents id.

Convention de nommage d’une table pivot : table1\_table2 (dans l’ordre alphabétique et au singulier.

Les deux modèles relatifs à la table pivot auront des fonctions qui retourneront $this->belongsToMany(ClassName ::class)

*Relation 1 à plusieurs (polymorphique)*

Ce type de relation permet de réutiliser une table donnée en lui permettant d’avoir plusieurs parents sans alourdir le système.

Ex : si on a un modèle vidéo qui lui aussi aura des commentaires, au lieu de lui créer un modèle video\_comment, on peut réutiliser la table comment qu’on utilisait pour les posts, en ajoutant un champ type (post ou video), avec leur id correspondant