LARAVEL (8-2021) – FRAMEWORK PHP 4

SOMMAIRE

ETAPE 4

0 - Présentation

Principes du cours

Les parties surlignées de bleu correspondent à des installation ou exercices à faire.
Les parties surlignées en jaune sont des concepts importants.

Pour chaque chapitre, on commence par une présentation complète, puis on fait le TP.

Bilan des commandes des étapes précédentes

Commandes

- Charger les dépendances

```
npm update                    # dossier nodes_modules
composer update               # dossier vendor
```

- Démarrer l'application

```
npm run dev.                  # démarrer vite
```

```
php artisan artisan serve     # démarrer le serveur web
```

- Vider les caches

```
php artisan optimize:clear
php artisan config:cache
```

- Lister les routes

```
>php artisan route:list
```

- Création d'un contrôleur pour la route a-propos

```
> php artisan make:controller AproposController
```

- Syntaxe Blade

```
     Le PHP est entre {{    }}
     Le PHP est précédé de @
```

Fichier manipulés

- Création d'une route « apropos »

  ⇒ routes / web.php

- Création d'un contrôleur pour la route

  ⇒ App / Http / Controllers / AproposController.php

- Création d'une vue pour le contrôleur : fichier blade

  ⇒ ressources / views / apropos / index.blade.php

Sauvegarde de l'étape précédente

Copie

- On vide les caches :

```
php artisan optimize:clear
php artisan config:cache
```

- On supprime au minimum les dossiers :

  ⇒ vendor,

  ⇒ node_module

- On peut aussi supprimer les fichiers :

  ⇒ package-lock.json

  ⇒ composer.lock

- On peut aussi supprimer les dossiers :

  ⇒ Storage/logs,

Livraison de la copie

- Après suppression, on obtient un dossier de 404 ko.

  ⇒ on zippe et on livre

Test de la copie

- On fait une copie du dossier «matodolist-etape-3-save » dans « matodolist-etape-3-ok » :
- Pour tester la copie :

```
cd matodolist-etape-3-ok
Composer update
npm install && npm run dev
```

```
php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
```

- On teste la route : http://127.0.0.1:8000/apropos

  ⇒ Ca marche !

Documentation

- Les liens sont sur la v12. On peut passer sur la version la plus récente sur la page de la v12.

Controller

- https://laravel.com/docs/12.x/controllers

Route

- https://laravel.com/docs/12.x/routing

Vue

- https://laravel.com/docs/12.x/views

Migration

- https://laravel.com/docs/12.x/migrations

ORM Eloquent

- https://laravel.com/docs/12.x/eloquent

Erreurs possibles sur la migration

- On va faire une première migration : php artisan migrate

  ⇒ Ca peut causer des erreurs

- On met à jour le fichier App/Providers/AppServicesProvider.php :

  ⇒ On ajoute la ligne suivante juste après le 1er use :

```
use Illuminate\Support\Facades\Schema; // juste après le 1er use
```

  ⇒ On ajoute la ligne suivante dans la fonction boot

```
Schema::defaultStringLength(191);      //  dans la fonction boot
```

  ⇒ On passe les commandes suivantes :

```
php artisan optimize:clear     # par prudence, on vide les caches
php artisan config:cache       # par prudence, on vide les caches
php artisan migrate

Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (25.02ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (18.06ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (21.58ms)
```

Objectifs de l'étape 4

Objectifs pédagogiques

- Se familiariser avec l'ORM Eloquent

  ⇒ DDL et Migration **(§1 : 11 pages-slides**) : pour créer la table des todos

  ⇒ DML et Données factices **(§2 : 6 pages-slides)** : pour créer des todos factices

- Se familiariser avec la notion de Contrôleur de ressources **($3 : 11 pages-slides**, dont 8 sur la création des vues).

  ⇒ Se familiariser avec les vues et routes associées à un contrôleur de ressources.

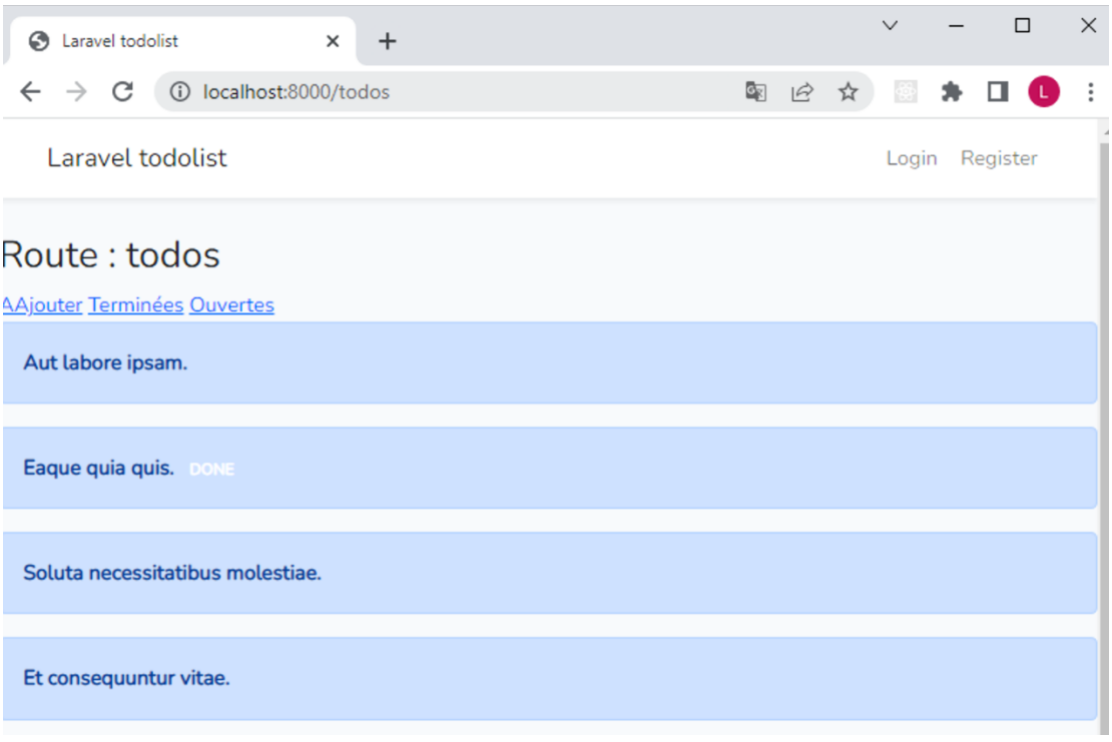
Objectifs fonctionnels

- Créer la route "todos" avec :

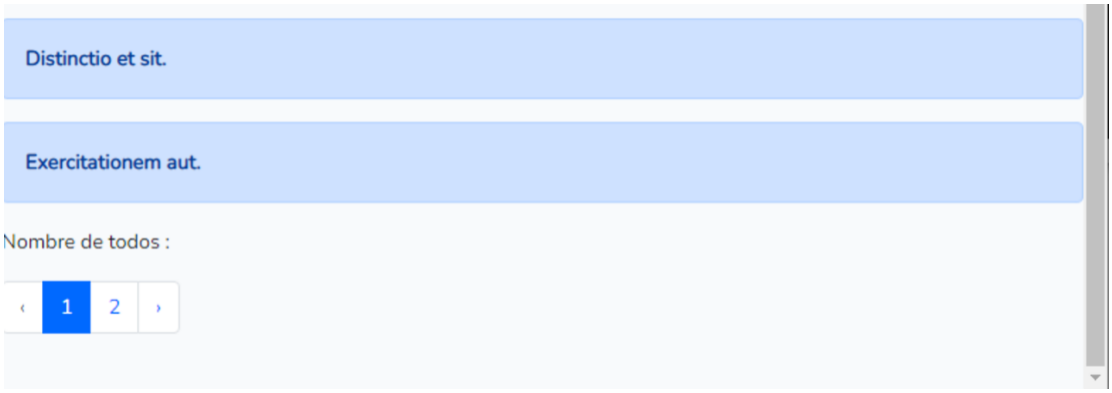
  ⇒ des todos factices

  ⇒ 3 boutons (les boutons terminées et ouvertes fonctionneront)

  ⇒ un système de pagination.



…



Utilisation de VS Code

VS Code

- On peut utiliser VS Code pour travailler notre projet Laravel.
- Dans VS Code, on installera des paquets spécifiques pour Laravel.

PHP

- Installer PHP Intelephense

Laravel

- Installer Laravel Artisan
- On peut ouvrir un terminal pour démarrer par exemple > php artisan serve
- On peut faire bouton droit / palette de commande pour accéder à des commandes artisan.

```
php artisan make:migration create_todos_table --create todos
```

1 – Ajouter des tables dans la BD avec « artisan make :migration » (v7 – pp. 9-20)

0 : Présentation

Principes

- On peut faire du DDL avec l'outil artisan male:migration
  ⇒ Créer / Modifier / Supprimer des tables
  ⇒ Créer / Modifier / Supprimer des attributs
- On peut ajouter ces modifications étape par étape : chaque étape s'appelle une **migration**.
- On peut gérer ces migrations :
  ⇒ Ajouter
  ⇒ Supprimer
  ⇒ Annuler
  ⇒ Lister
- La gestion des migrations correspond à la commande :

```
php artisan make:migration
```

- Aide pour cette commande

```
php artisan make:migration --help
```

- Le DDL se fera en écrivant des fichiers PHP correspondant au DDL-SQL.

Documenation Laravel

- Migration : https://laravel.com/docs/12.x/migrations
- Création de table : https://laravel.com/docs/12.x/migrations - creating-tables
- Création de colonnes : https://laravel.com/docs/12.x/migrations - creating-columns
- Création de clés étrangères : https://laravel.com/docs/12.x/migrations - foreign-key-constraints
- Etc.

Différences de version

- A noter que la syntaxe a pu un peu évoluer en fonction des versions de Laravel.

1 : Création d'une migration

Objectif

- On veut créer une migration (une étape de DDL) pour créer la table des todos.

Création d'une migration pour créer une table

- Commande php artisan make:migration

```
php artisan make:migration create_todos_table --create todos
```

⇒ Le nom de la migration est conventionnel : « **create_todos_table** » : on précise ici qu'il s'agit de **créer la table todos**. Il faut donner un nom explicite.

⇒ L'option « --create todos » permet de donner le nom « todos » à la table, sinon Laravel va s'appeler « todos_table »

Fichier créé : database / migration / date_heure_nom_migration.php

- Fichier créé : date_heure_nom_migration.php

⇒ Par exemple : 2021_03_12_153313_create-todos-table

- Dans ce fichier : la class CreateTodosTable avec 2 méthodes :

⇒ up() : pour créer les table : create()

⇒ down() : pour supprimer la table : dropIfExist()

Création de la table : méthode up()

- La méthode up() fait appel à la méthode **Schema::create()**

  ⇒ id() permet de créer un id. On peut aussi utiliser bigIncrements()

  ⇒ timestamps() permet de créer 2 attributs : created_at et updated_at.

Suppression de la table : méthode down()

- La méthode down() fait appel à la méthode **Schema::dropIfExists()**

  ⇒ La méthode dropIfExists() va permettre de supprimer une table si elle existe.

Variante syntaxique : à éviter !

- On peut donner n'importe quel nom à la migration et rien d'autre :

```
php artisan make:migration m1
```

  ⇒ Dans ce cas, les fonctions up() et down() sont créées avec rien dedans.

2 : Consulter les migrations

- Pour consulter les migrations, on passe la commande :

```
php artisan migrate:status
```

⇒ Attention, le serveur de BD doit tourner !

```
+------+-----------------------------------------------------+-------+
| Ran? | Migration                                           | Batch |
+------+-----------------------------------------------------+-------+
| Yes  | 2014_10_12_000000_create_users_table                | 1     |
| Yes  | 2014_10_12_100000_create_password_resets_table      | 1     |
| Yes  | 2019_08_19_000000_create_failed_jobs_table          | 1     |
+------+-----------------------------------------------------+-------+
```

⇒ Colonne Run : est-ce que la migration a été exécutée

⇒ Migration : nom de la migration

⇒ Batch : numéro de la « transaction » de migration : les migrations qui ont été exécutées ensemble ont un même numéro. Les réalisations de migration sont numérotées de 1 à N.

- Après la création de la migration create_todos_table :

```
+------+-----------------------------------------------------+-------+
| Ran? | Migration                                           | Batch |
+------+-----------------------------------------------------+-------+
| Yes  | 2014_10_12_000000_create_users_table                | 1     |
| Yes  | 2014_10_12_100000_create_password_resets_table      | 1     |
| Yes  | 2019_08_19_000000_create_failed_jobs_table          | 1     |
| No   | 2021_03_12_165402_create-todos-table                |       |
+------+-----------------------------------------------------+-------+
```

⇒ La nouvelle migration n'est pas exécutée. Elle n'a pas de numéro de transaction.

3 : Supprimer une migration non exécutée

- Pour supprimer une migration non exécutée, il suffit de supprimer le fichier.

4 : Création d'une table avec une migration

Exécuter une migration

- A ce stade, notre BD contient les tables suivantes :

```
Mysql>show tables

+------------------------+
| Tables_in_matodolist   |
+------------------------+
| failed_jobs            |
| migrations             |
| password_resets        |
| users                  |
+------------------------+
```

- Pour exécuter les migrations qui n'ont pas encore été exécutées (Run à No), on passe la commande suivante :

```
php artisan migrate
```

Consulter les migrations

```
php artisan migrate:status

+------+-----------------------------------------------+-------+
| Ran? | Migration                                     | Batch |
+------+-----------------------------------------------+-------+
| Yes  | 2014_10_12_000000_create_users_table          | 1     |
| Yes  | 2014_10_12_100000_create_password_resets_table | 1     |
| Yes  | 2019_08_19_000000_create_failed_jobs_table    | 1     |
| Yes  | 2021_03_12_165402_create-todos-table          | 2     |
+------+-----------------------------------------------+-------+
```

⇒ Avec cette commande, les migrations nouvellement exécutées ont leur Run qui passe à Yes et récupère un numéro de Batch (2 en l'occurrence).

Conséquences dans la BD

- A ce stade, notre BD contient les tables suivantes :

```
Mysql>show tables

+-------------------------+
| Tables_in_matodolist    |
+-------------------------+
| failed_jobs             |
| migrations              |
| password_resets         |
| todos                   |
| users                   |
+-------------------------+
```

- La table todos a été créée par la méthode **Schema::create()**

- La table todos contient les attributs suivants :

```
Mysql>desc todos ;

+------------+---------------------+------+-----+---------+----------------+
| Field      | Type                | Null | Key | Default | Extra          |
+------------+---------------------+------+-----+---------+----------------+
| id         | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| created_at | timestamp           | YES  |     | NULL    |                |
| updated_at | timestamp           | YES  |     | NULL    |                |
+------------+---------------------+------+-----+---------+----------------+
```

5 : Suppression d'une table avec une migration : annulation de la migration

Annuler une migration

➢ ***Migration:***

- Pour annuler la dernière migration, on passe la commande suivante :

```
php artisan migrate:rollback
```

➢ ***Conséquences dans Laravel***

```
php artisan migrate:status

+------+-----------------------------------------------------+-------+
| Ran? | Migration                                           | Batch |
+------+-----------------------------------------------------+-------+
| Yes  | 2014_10_12_000000_create_users_table                | 1     |
| Yes  | 2014_10_12_100000_create_password_resets_table      | 1     |
| Yes  | 2019_08_19_000000_create_failed_jobs_table          | 1     |
| No   | 2021_03_12_165402_create-todos-table                |       |
+------+-----------------------------------------------------+-------+
```

➢ ***Conséquences dans Laravel***

⇒ A ce stade, notre BD contient les tables suivantes :

```
Mysql>show tables

+------------------------+
| Tables_in_matodolist   |
+------------------------+
| failed_jobs            |
| migrations             |
| password_resets        |
| users                  |
+------------------------+
```

⇒ La table todos a été supprimée par la méthode **Schema::dropIfExists()**

6 : Ajout d'attributs dans une table issue d'une migration

technique 1 : Ajouter des attributs dans le Schema ::create('todos'..) avant la migration

- Pour le moment, on a créé automatique un id et des timestamps.
- On va ajouter d'autres attributs dans la table.
- L'objectif est d'obtenir cette table :

todos	
id	int
created_at	timestamp
updated_at	timestamp
name	string(100)
text	texte
done	booléen

- Création de colonnes : https://laravel.com/docs/12.x/migrations#creating-columns
- Dans la méthode Schema ::create() de la méthode up() de la migration, on rajoute :

```
Schema::create('todos', function (Blueprint $table) {
    ...
    $table->string('name', 100);
    $table->text('description');
    $table->Boolean('done');
```

- On crée la table :

```
php artisan migrate
```

- On vérifie le status :

```
php artisan migrate:status
```

⇒ Le Run est à yes,

⇒ Le Batch est à 2

- On vérifie sur le serveur de BD :

```
mysql>show tables
mysql>desc todos
```

technique 2 : Créer une migration d'update pour la table : --table todos

- Comment ajouter des colonnes à une table qui existe déjà : --table todos
- On va créer une migration d'update : https://laravel.com/docs/12.x/migrations#updating-tables
- Au lieu d'avoir une fonction Schema ::create() on aura une fonction Schema ::table()
- On commence par créer la migration :

```
php artisan make:migration alter_todos_table --table todos
```

- Dans la méthode Schema ::table() de la méthode up() de la migration, on écrit :

```
Schema::table('todos', function (Blueprint $table) {
    $table->string('name', 100);
    $table->text('description');
    $table->Boolean('done');
});
```

- Pour la méthode Schema ::table() de la méthode down() de la migration, on écrit :

```
Schema::table('todos', function (Blueprint $table) {
    $table->dropColumn('name');
    $table->dropColumn('description');
    $table->dropColumn('done');
});
```

⇒ https://laravel.com/docs/12.x/migrations#dropping-columns

7 : Bilan des commandes de migration

Artisan et le DDL

- Artisan et Laravel nous permettent de créer et modifier notre BD avec du code PHP.
- C'est très pratique car ça nous permet d'avancer le projet et de faire des mises à jour de la BD en fonction des besoins du projet.
- C'est adapté à la méthode AGILE.

Toutes les commandes migrate

```
>php artisan
>php artisan migrate
>php artisan migrate:status
>php artisan migrate:rollback

>php artisan make:migration -help
>php artisan make:migration
>php artisan make:migration create-todos-table --create todos
```

DDL table

- CREATE TABLE : create()
- ALTER TABLE : table()
- DROP TABLE : drop(), dropIfExists()

Type des colonnes

- https://laravel.com/docs/12.x/migrations#available-column-types

Index, unique, clé étrangère

- https://laravel.com/docs/12.x/migrations#indexes

Toutes les commandes

- https://laravel.com/docs/12.x/migrations

TP 1

Installer la table todos comme présentée dans le chapitre

- On utilisera 2 migrations comme dans le document (la création et l'alter).

Synthèse

Essai :

On crée la migration de création de la table

```
php artisan make:migration create_todos_table --create todos
```

On en crée une 2ème

```
php artisan make:migration m1
```

On regarde les migrations (aussi dans database/migrations)

```
php artisan migrate:status
```

On lance la migration pour créer les tables

```
php artisan migrate
```

```
php artisan migrate:status
```

On annule les migration

```
php artisan migrate:rollback
```

```
php artisan migrate:status
```

On supprimer le fichier database/migrations/m1

Version finale:

On crée la migration de création de la table

```
php artisan make:migration create_todos_table --create todos
```

On lance la migration pour créer la table

```
php artisan migrate
```

On crée une migration de modification de la table

```
php artisan make:migration alter_todos_table --table todos
```

On met à jour les fonctions up et down pour gérer les attributs qu'on veut ajouter :
Ajout de. :

```
    $table->string('name', 100);
    $table->text('description');
    $table->Boolean('done');
```

On lance la migration pour modifier la table

```
php artisan migrate
```

On vérifie dans la BD la presence des attributs

2 – Création d'un modèle et de données factices avec Eloquent (v8 – pp. 21-27)

0 : Présentation de l'ORM Eloquent et des modèles

ORM et Eloquent

- ORM : Object-Relational Mapping
- Un ORM permet de faciliter les relations entre les tables de la base de données (relational) et les classes de notre application (object).
- En général, les frameworks utilisent un ORM.
- Laravel utilise l'ORM Eloquent : https://laravel.com/docs/12.x/eloquent

Eloquent et les modèles

- MVC : Modèle – Vue – Contrôleur.
- Le Modèle du MVC, c'est le Modèle des données. Il correspond aux tables de la BD.
- Eloquent va nous servir pour la mise en place et l'exploitation de la BD.
- Eloquent va nous permettre de créer des données factices.

Exemple : le modèle App\Models\User

- ***Le dossier App\Models***
  - Les modèles sont rangés dans App\Models.
  - On trouve déjà le modèle User présent par défaut dès l'installation de base (composer create-project laravel/laravel monProjet).
  - C'est une **classe User** qui correspond à la table users.

- ***Convention de nommages des tables Laravel***
  - ⇒ table : todos // minuscule au début, pluriel
  - ⇒ modèle : Todo // Majuscule au début, singulier
  - On peut choisir ce qu'on veut, mais ce sera pratique d'appliquer cette convention.

- ***Gestion de la sécurité : attribute $fillable***
  - Dans la classe App/Models/User, on trouve la définition d'un attribut $filable.
  - Cet attribut permet de définir certains les attributs de la table : ceux qui correspondent à des champs de saisie pour lesquels le PHP gérera une protection des données particulière.

```
    protected $fillable = [
        'name',
        'email',
        'password',
    ];
```

⇒ https://laravel.com/docs/12.x/eloquent#mass-assignment

1 : Créer le modèle Todo

A ce stade, la table todos a déjà été créée :

```
+---------------+---------------------+------+-----+---------+----------------+
| Field         | Type                | Null | Key | Default | Extra          |
+---------------+---------------------+------+-----+---------+----------------+
| id            | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| name          | varchar(100)        | NO   |     | NULL    |                |
| description   | text                | NO   |     | NULL    |                |
| done          | tinyint(1)          | YES  |     | 0       |                |
| creator_id    | bigint(20)          | NO   |     | 0       |                |
| affectedTo_id | bigint(20)          | NO   |     | 0       |                |
| affectedBy_id | bigint(20)          | NO   |     | 0       |                |
| created_at    | timestamp           | YES  |     | NULL    |                |
| updated_at    | timestamp           | YES  |     | NULL    |                |
+---------------+---------------------+------+-----+---------+----------------+
```

Créer la classe du modèle : app/Models/Todo

- La commande php artisan make:modele permet de créer un nouveau modèle :

```
php artisan make:model Todo
```

⇒ On donne le nom du modèle : Todo

⇒ avec --migration ou -m ça va créer la migration en même temps. Inutile dans ce cas : c'est déjà fait.

- Le modèle App/Models/Todo.php a été créé, à côté de User.
- On pourra créer des dossiers dans Models pour ranger nos modèles.

⇒ Dans ce cas, il faudra changer le namespace : App\Models\TelDomaine

Mettre à jour le modèle Todo

- Pour le moment, la classe du modèle Todo, App\Models\Todo.php, c'est un template vide.
- A noter les couleurs de la coloration syntaxique.
- On applique le template du modèle User pour créer le $filable.
- name, description et done sont trois champs sont les champs de saisie de masse.
- On ajoute donc :

```
    protected $fillable = [
        'name',
        'description',
        'done',
    ];
```

⇒ On a besoin de rien d'autres pour le moment.

2 : Créer des données factices

Principes

- On va pouvoir générer des fausses données qui vont nous permettre de tester notre application facilement.
- Pour ça, on utilse la commande :

```
php artisan db:seed
```

- Cette commande utilise la classe « /databases/seed/DatabaseSeeder ».

  ⇒ Dans cette classe, on trouve une méthode run et un exemple mis en commentaire :

  ⇒ \App\Models\User::factory(10)->create();

  ⇒ Ca permet de créer des users selon le code qui est dans UserFactory.php

- Dans la classe database/factories/UserFactory.php :

  ⇒ On a un exemple de création de données factices.

  ⇒ On va s'en servir pour créer les données factices des Todos.

Test

- On peut tester la création de user factices en dé-commentant le factory des User dans la classe databases/seed/DatabaseSeeder :

```
public function run()
{
    \App\Models\User::factory(5)->create();
}
```

- Ensuite on passe la commande artisan :

```
php artisan db:seed
```

- On vérifie la présence des données dans la BD

```
mysql>select * from users \G
```

- On peut supprimer les données si on veut :

```
mysql>delete from users where id > 1;
```

Création d'une factory pour le modèle Todo

- On va créer une factory (usine) pour le modèle Todo.

  ⇒ Pour ça on va dupliquer la factory du modèle User.

- On a déjà : databases\factories\UserFactory pour le modèle User.

  ⇒ La table users était la suivante :

```
+--------------------+---------------------+------+-----+---------+----------------+
| Field              | Type                | Null | Key | Default | Extra          |
+--------------------+---------------------+------+-----+---------+----------------+
| id                 | bigint(20) unsigned | NO   | PR  | NULL    |auto_increment
| name               | varchar(191)        | NO   |     | NULL    |                |
| email              | varchar(191)        | NO   | UNI | NULL    |                |
| email_verified_at  | timestamp           | YES  |     | NULL    |                |
| password           | varchar(191)        | NO   |     | NULL    |                |
| remember_token     | varchar(100)        | YES  |     | NULL    |                |
| created_at         | timestamp           | YES  |     | NULL    |                |
| updated_at         | timestamp           | YES  |     | NULL    |                |
+--------------------+---------------------+------+-----+---------+----------------+
```

- La classe Factory définit comment vont être créées les données factices dans la BD.
- La méthode definition() permet cela : elle utilise un attribut « faker ».

  ⇒ Les 5 champs non timestamp sont traités

  ⇒ L'attribut « email_verified_at » est traité à part et mis à null

Creation de la TodoFactory

- On dupplique la classe UserFactory en TodoFactory
- On met à jour : user devient todo
- On met à jour la méthode definition()

```
'name' => $this->faker->sentence(3),          // + ou moins 3 mots
'description' => $this->faker->sentence(10),// + ou - 10 mots
'done' => $this->faker->numberBetween(0,1), // 0 ou 1
```

⇒ sentence(max) : génère des mots pour l'attributs. On fixe le nombre max.

⇒ numberBetween permet de choisir un entier entre 2 valeurs

- Pas besoin de plus : la « factory » est faite

Utilisation de la TodoFactory

- On revient dans database\seeders\DatabaseSeeders
- Dans le run, on duplique la factory des users et on l'adapte pour 25 Todo

```
\App\Models\Todo::factory(25)->create();
```

- On peut lancer la commande :

```
php artisan db:seed
```

⇒ Les données ont été générées.

- On peut vérifier dans la BD :

```
mysql>select * from todos \G
```

- On peut supprimer les données dans la BD :

```
mysql>delete from todos \G
```

TP 2

Créez des données factices pour la table todos

Synthèse

Création du modèle Todo

```
php artisan make:model Todo
```

Dans app/Models/Todo.php :

```
    protected $fillable = [
        'name',
        'description',
        'done',
    ];
```

Créer TodoFactory.php

Mettre à jour le code :

Changer le nom de la classe : TodoFactory

Mettre à jour le code de la fonction « definition » :

```
'name' => $this->faker->sentence(3),         // + ou moins 3 mots
'description' => $this->faker->sentence(10),// + ou - 10 mots
'done' => $this->faker->numberBetween(0,1), // 0 ou 1
```

Dans le DatabaseSeeder.php

```
\App\Models\Todo::factory(25)->create();
```

Exécution du code :

```
   php artisan db:seed
```

Vérification en BD

3 – Création d'un contrôleur de ressources (v9-11 – pp. 28-38)

0 : Principes d'un contrôleur de ressources

Présentation

- Un contrôleur de ressources est un outil pour accéder à nos données (les tables de la BD) indépendamment de l'application.
- Un contrôleur de ressources permet de faire du CRUD sur une table (CRUD : Create = Insert, Read = Select, Update, Delete)
- On sait créer nos données et nos modèles. On sait les alimenter en données factices.
- On veut pouvoir manipuler nos données indépendamment de l'application.
- Pour ça, on met en place un contrôleur de ressources (les ressources sont les données).
- https://laravel.com/docs/12.x/controllers#resource-controllers
- Ensuite il faudra des routes pour utiliser le contrôleur.
- Ca reprend l'étape 3-0.

Précision sur le contrôleur de resource

- Pour chaque table, on a une classe.
- Pour chaque classe correspondant à une table, on aura un contrôleur particulier : on l'appelle contrôleur de ressources.

  ⇒ Le contrôleur du MVC, c'est le « main », le programme principal. Ici un « main » pour simplement contrôler nos données.
- Ce contrôleur aura une fonction index(), comme tous les contrôleurs, qui pourra retourner une vue.

  ⇒ La fonction index() c'est la vue du contrôleur dans le modèle MVC.
- Mais il offre aussi des méthodes qui permettent de manipuler les données de la table : faire du **CRUD** (Create = Insert, Read = Select, Update, Delete) :

  ⇒ create()

  ⇒ show()

  ⇒ edit()

  ⇒ update()

  ⇒ destroy()
- Le pattern d'un contrôleur de ressources est le même pour toutes les classes du modèle :

  ⇒ php artisan va fournir un outil pour fabriquer ce pattern.

  ⇒ https://laravel.com/docs/12.x/controllers#actions-handled-by-resource-controller

1 : Mise en place d'un contrôleur de ressources

php artisan make:controller --help

- On sait déjà faire un contrôleur avec php artisan make :controller
- L'aide nous montre l'option --ressource (ou -r)
- On va donc passer la commande :

```
php artisan make:controller -r TodoController
```

- Ca génère le contrôleur App/Http/Controller/TodoController avec ses méthodes index(), create(), show(), edit(), update(), destroy().
- Ce contrôleur servira à notre application finale.
- Pour le moment, on veut un contrôleur de ressources avec ses routes.

2 : Création des routes

Nouvelle syntaxe pour les routes

- Dans le fichier routes\web.php, on ajoute une route « todos » pour le contrôleur de ressources.
- Nouvelle syntaxe : resource('route', Controller ::class)

```
use App\Http\Controllers\TodoController;
Route::resource('todos', TodoController::class);
```

⇒ https://laravel.com/docs/12.x/controllers#resource-controllers

Test de la route : http://127.0.0.1:8000/todos

- La route existe mais la page est blanche (pas d'erreur 404).
- Chemin :

⇒ Route::ressources('todos', 'TodoController')

⇒ App\Http\Controllers\TodoController.index() :

-> ne fait rien
-> On peut y mettre un echo 'hello world' ;

- Tout est prêt pour exploiter nos données.

3 : Comment afficher les données de la table todos ? La vue.

Version debug

Dans l'index du app/Http/TodoController, à la place de l'echo

```
use App\Models\Todo;   // à ne pas oublier : App majuscule !

public function index()
{
    // echo 'hello world';
    $datas=Todo::all();
    dd($datas);
}
```

⇒ $datas est un tableau de données

⇒ Todo c'est la classe app\Models\Todo

⇒ :: permet d'accéder à une méthode de classe

⇒ all() c'est un select de tous les tuples (pas de where).

⇒ dd($datas) permet un affichage pratique type "console"

=> on comprend bien qu'on est dans une logique de debug et d'outils de debug

Version debug avec une vue

- On transmet les données à un fichier blade
- Au lieu du dd($datas) on passe une vue :

```
    return view('todos.index', compact('datas'))
```

⇒ Compact('datas') : le nom de la variable, sans $

- A ce stade :

⇒ ressources\views\todos\index.blade.php : n'existe pas encore

⇒ On va donc créer : ressources\views\todos\index.blade.php

- On reprend la syntaxe de la vue apropos :

```
<x-app-layout>

</x-app-layout>
```

- Syntaxe blade :

⇒ https://laravel.com/docs/12.x/blade

⇒ https://laravel.com/docs/12.x/blade#the-loop-variable

- Code avec une boucle foreach de blade

```
<x-app-layout>
    <h3>hello world</h3>
    @foreach($datas as $data)
        @php
            // on met du php dans le html-blade
            dd($data);
        @endphp
    @endforeach
</x-app-layout>
```

⇒ On perd Breeze

⇒ dd($data) : le dd() fait quitter l'affichage : ça affiche le premier et ça s'arrête (comme un return).

Variantes de code debug avec vue

```
<x-app-layout>
    <h1>Hello world</h1>
    @php
        echo '<pre>'; print_r($datas); echo '</pre>';
    @endphp
    @foreach($datas as $data)
        @php
            dd($data);
        @endphp
    @endforeach
</x-app-layout>
```

On peut remplacer le blade par du php :

```
<x-app-layout>
    <h1>Hello world</h1>
    <?php
        echo '<pre>'; print_r($datas[0]); echo '</pre>';
    ?>
    <?php foreach($datas as $data){
        dd($data);
    } ?>
</x-app-layout>
```

Version HTML avec une vue:

- On met du HTML-Blade dans le foreach :

```
<x-app-layout>
    <h1>Hello world</h1>
    @foreach($datas as $data)
      <h4>
        Nom : {{$data->name}} --
        Done : {{$data->done}}
      </h4>
    @endforeach
<x-app-layout>
```

- A ce stade on n'a perdu le CSS.
- // {{ }} on met du php dans le html : c'est <?php echo ?>
- On peut remplacer le {{ }} par du php

```
@foreach($datas as $data)
  <h4>
    Nom : <?php echo $data->name ?> --
    Done : <?php echo $data->done ?>
  </h4>
@endforeach
```

4 : SQL

Ajout d'un filtre : where done == 1

- A ce stade :

  ⇒ Route::ressources('todos', 'TodoController')

  ⇒ App\Http\Controllers\TodoController.index()

  =>ressources\views\todos\index.blade.php

- L'objectif est de récupérer un tableau de datas avec le filtre : done == 1
- Ca se passe dans le TodoController

  ⇒ Il faut faire un équivalent de « where done=1
- On passe de :

```
$datas = Todo::all();
```

- A :

```
$datas = Todo::where('done', 1)->get();
```

- Il faut faire un get() pour obtenir le résultat.

SQL Laravel :

- https://laravel.com/docs/12.x/queries#basic-where-clauses

5 : Affichage avec Layout et TailWind (ou Bootstrap)

Situation

- On sait parcourir les données de la BD
- On sait y mettre des données
- On a créé un design plus abouti
- Notre vue hérite du layout

  ⇒ http://127.0.0.1:8000/todos

Version TailWind

```
@foreach($datas as $data)
    <div class="bg-blue-100 border-l-4 border-blue-500 text-blue-
700 p-4" role="alert">
        <strong>
            {{ $data->name }}
            @if($data->done)
                <span class="bg-green-500 text-white text-xs
font-semibold px-2 py-1 rounded">
                    done
                </span>
            @endif
        </strong>
    </div>
@endforeach
```

⇒ C'est déjà un résultat présentable. En quelques lignes de code !

⇒ bg-blue-100 : background, couleur bleu, transparence 100

⇒ border-l-4 : une bordure de largeur 4

⇒ border-blue-500 : couleur de la bordure: bleu, transparence 500

⇒ text-blue-700 : texte bleu, transparence 700

⇒ p-4 : écartement des paragraphes de 4

⇒ Le role alert sert pour l'environnement (on peut s'en passer dans un premier temps).

Version sans blade avec seulement du PHP :

```
<?php foreach($datas as $data) { ?>
    <div class="alert alert-primary" role="alert">
        <strong>
            <?php echo $data->name ?>
            <?php if ($data->done ) { ?>
                <span class="badge badge-success">
                    done
                </span>
            <?php } ?>
        </strong>
    </div>
<?php } ?>
```

Version Bootstrap

- On utilise les composants « alert » et « badge » de bootstrap :

```
@foreach($datas as $data)
    <div class="alert alert-primary" role="alert">
        <strong>
            {{ $data->name }}
            @if($data->done )
                <span class="badge badge-success">
                    done
                </span>
            @endif
        </strong>
    </div>
@endforeach
```

Installation et utilisation d'un plugin bootstrap

- On installe le dernier plugin bootstrap. Il faudra peut-être redémarrer VS Code pour qu'il fonctionne.
- On veut un composant alerte :

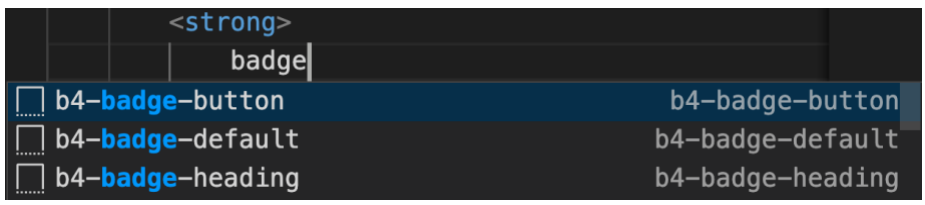
  ⇒ Dans le fichier de code : on tape alerte :

- On choisit default. On obtient :

```
    <div class="alert alert-primary" role="alert">
        <strong>primary</strong>
    </div>
```

- On veut un composant badge :

  ⇒ Dans le fichier de code : on supprime le primary du et on tape badge :



- On choisit default puis success. On obtient :

```
    <div class="alert alert-primary" role="alert">
        <strong>
            <span class="badge badge-success"></span>
        </strong>
    </div>
```

- Ensuite on complète avec ce qu'on veut afficher.

  on affiche toutes les todos : les terminés ont le bouton, les autres rien

Exemples de tuto bootstrap :

- Sur les composant alert : https://www.pierre-giraud.com/bootstrap-apprendre-cours/alerte-modal-toast/
- Sur les composant badge : https://www.pierre-giraud.com/bootstrap-apprendre-cours/badge-jumbotron/

6 : Amélioration de la présentation : système de pagination v11

Problèmatique de la pagination

- Actuellement, on a toutes les todos : ce n'est pas agréable visuellement et ça peut ralentir le système puisqu'il faut charger toutes les données.
- On va donc afficher les données « page à page », c'est-à-dire par parquet (de 10, 15, au choix).
- En PHP, gérer la pagination est une tâche assez lourde, mais aussi répétitive.
- Laravel va nous fournir des outils pour gérer ça de façon très simple.

Technique de pagination

➢ ***Mise à jour du contrôleur de ressources***

```
$datas=Todo::all(10);
```

- Remplacé par :

```
$datas=Todo::paginate(10);
```

⇒ Ça marche !

⇒ Mais on n'a pas la barre de pagination.

- https://laravel.com/docs/12.x/pagination

Quand on fait :

```
$datas = Todo::where('done', 1)->paginate(10);
```

On obtient la pagination sur les done à 1.

Barre de pagination

- Pour afficher la barre de pagination, on modifie le template : views/todos/index.php
- On ajoute, juste après le @endforeach :

```
{{ $datas->links()}}
```

⇒ De là, on accède aux pages : on voit l'url.

⇒ http://127.0.0.1:8000/todos?page=2

- https://laravel.com/docs/12.x/pagination#displaying-pagination-results

➢ ***Bootstrap***

- On a un problème si on utilise Bootstrap :

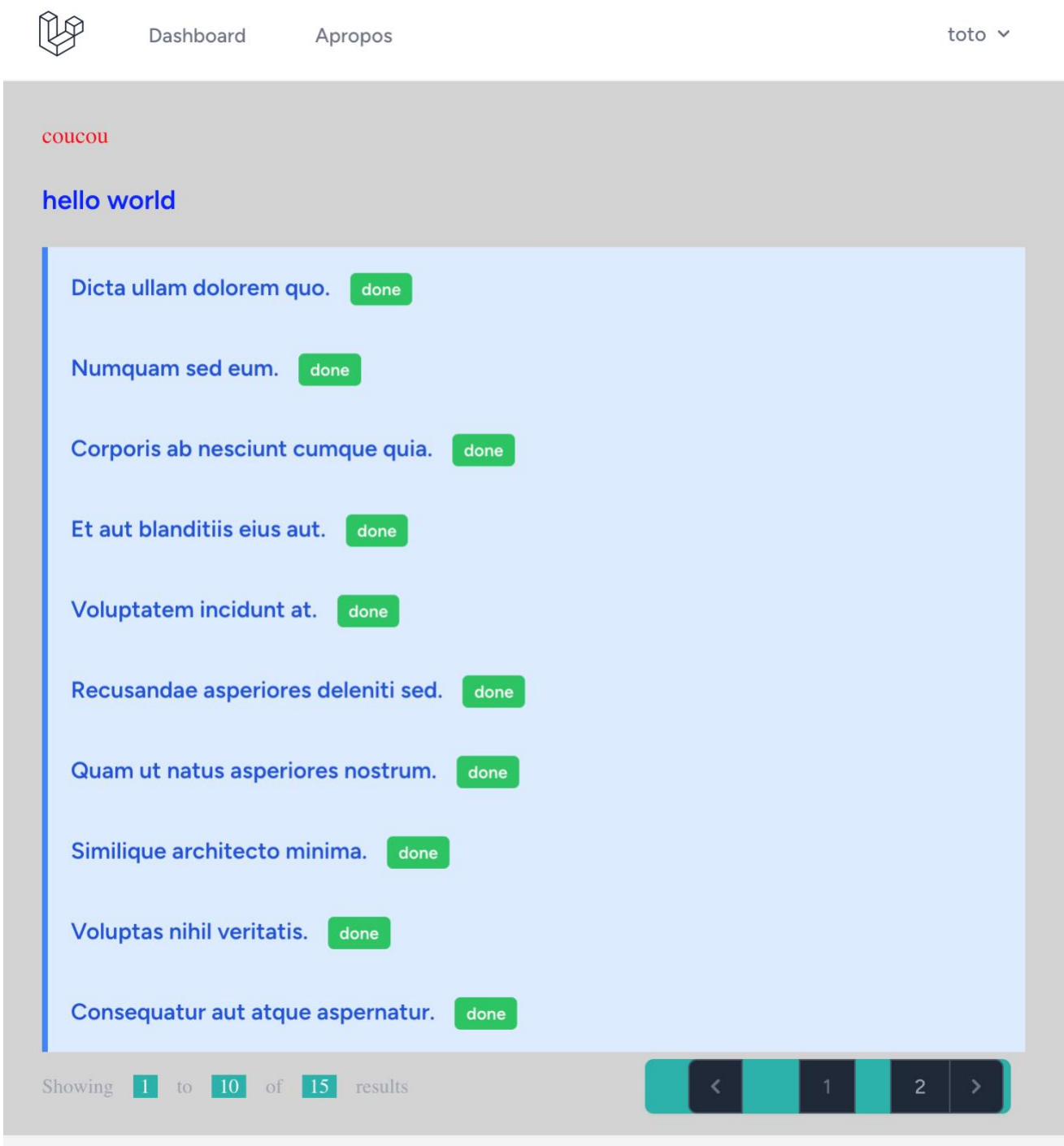
⇒ https://laravel.com/docs/12.x/pagination - using-bootstrap

- Il faut mettre à jour le fichier App\Providers\AppServiceProvider :

```
use Illuminate\Pagination\Paginator;
public function boot()
{
    Paginator::useBootstrap();
}
```

TP 3

Faites tout le TP jusqu'à arriver à ce résultat :



S'il vous reste du temps, améliorez le menu : supprimer le menu dashboard. Changer le logo. Faites-en sortes qu'il amène sur l'Apropos. Supprimer le menu Apropos. Ajoutez un menu todos qui amène sur la route todos.
Ajoutez ce qui a été fait à l'étape 3.0 : un autre contrôleur de ressources sur une autre table de todos.

Synthèse

1) Création du contrôleur de ressources pour le modèle Todo

```
php artisan make:controller -r TodoController
```

2) Création de la route :

```
use App\Http\Controllers\TodoController;
Route::resource('todos', TodoController::class);
```

Test de la route : http://127.0.0.1:8000/todos

La route existe mais la page est blanche.

3) Affichage des données en mode debug :

```
use App\Models\Todo;
class TodoController extends Controller
{
    public function index()
    {
        //echo 'hello world';
        $datas=Todo::all();
        dd($datas);
    }
```

Au lieu du dd($datas) on passe une vue :

```
        $datas=Todo::all();
        return view('todos.index', compact('datas'));
```

Version HTML-Blade :

```
<x-app-layout>
    <h1>Hello world</h1>
    @foreach($datas as $data)
      <h4>
        Nom : {{$data->name}} --
        Done : {{$data->done}}
      </h4>
    @endforeach
<x-app-layout>
```

4) SQL : On filtre les done = 1

```
$datas = Todo::where('done', 1)->get();
```

⇒ http://127.0.0.1:8000/todos

5) Affichage TailWind

```
@foreach($datas as $data)
    <div class="bg-blue-100 border-l-4 border-blue-500 text-blue-700 p-4"
role="alert">
        <strong>
            {{ $data->name }}
            @if($data->done)
                <span class="bg-green-500 text-white text-xs font-
semibold px-2 py-1 rounded">
                    done
                </span>
            @endif
        </strong>
    </div>
@endforeach
```

6) Pagination :

```
$datas=Todo::paginate(10);
```

Ou

```
$datas = Todo::where('done', 1)->paginate(10);
```

Et

Dans views/todos/index.php, juste après le @endforeach :

```
{{ $datas->links()}}
```