Composer.json => il fait la liste automatiquement des dépendances **nécessaires en PHP**

Vendor=> les dépendances sont installées dans le vendor

Package.js => il fait la liste automatiquement des dépendances **nécessaires en JAVASCRIPT**

**Commande NPM INSTAL**

**Node\_modules =>** les dépendances sont installées dans node\_modules

1. **Routes**

Route ::get (‘/details/{id}’, PostsContoller@index);

Route ::get (‘/details/{specific\_id}’, PostsContoller@index);

Astuces :

Route dynamique l’id est entouré e de bracket { }

**Depuis Laravel 8**

* Route::get('/checking', '\App\Http\Controllers\HomeController@index');
* Nommer une route
* Renvoyer directement vers une vue : Route::view('/test','test', ['name' => 'Taylor']);

1. **Controller**

**Commandes terminal :**

php artisan make :controller PostsController

php artisan make:contoller –resource PostController

1. **Blade**

Afficher l’id de l’url

 <p>i'am the number <h1>{{request()->route('specific\_id')}}</h1></p>

Le layout

Créer un fichier layouts/app.blade.php

!html

<div class=’container’>

@yield(‘body’)

@yield(‘footer’)

L’importer dans la vue

@extends(‘layouts.app’)

@section(‘body’)

@endsection

1. **Migrations**

.env => complétetr DB\_DATABASE = name

php artisan migrate => Dossier databe/migrations => distingue les tables

Pour éviter d’avoir à coder ces méthodes **down** la version 5.5 de Laravel a prévu la commande **fresh** qui supprime automatiquement les tables concernées :

migrate:fresh Drop all tables and re-run all migrations => drop table

Pour annuler et relancer en une seule opération on utilise la commande **refresh** :

migrate:refresh Reset and re-run all migrations =>rollback table

Pour annuler la dernière migration dans mysql :

migrate:rollback Rollback the last database migration => la methode down est

migrate:install Create the migration repository

migrate:reset Rollback all database migrations

migrate:status Show the status of each migration

Convention pour créer une table « \_ »  et « pluriel »

$ php artisan make:migration create\_posts\_table --create="posts" " (on flag pour nommer la table)

**$table->string**('name')->**default**('user');

**$table->string**('name')->unique();

**$table-> string** ('name',2)-> unsigned (); option pour limiter

**$table->integer**('is\_admin')->unsigned(); ne peut pas être un chiffre négatif

Pour modifier la table sans rollback toutes les données déjà enregistrées.

Php artisan make:migration add\_admin\_column\_to \_posts\_table --table="posts" (on flag la table concernée)

!!! modifier la méthode down pour utiliser le rollback methode dropolumn()

$table->dropColumn("is\_admin");

Enfin on relance un php artisan migrate

1. **SQL QUERIES**

**Notes : ?, ? permet d’éviter injection SQL, le paramètre est remplace par , [1, ‘dayle’] ;**

insert

DB::insert('insert into users (id, name) values (?, ?)', [1, 'Dayle']);

select

$users = DB::select('select \* from users where active = ?', [1]);

ensuite

foreach ($users as $user) {

echo $user->name;

}

update

$affected = DB::update('update users set votes = 100 where name = ?', ['John']);

delete

$deleted = DB::delete('delete from users');

1. **ELOQUENT – ORM ->MODEL**

Votre application Web Laravel pourrait communiquer directement avec les tables de la base de données. Cependant, il sera plus intéressant d'utiliser des modèles, c'est-à-dire une représentation objet de chacune des tables.

Convention pour créer un model « PascalCase»  et « singulier »

!!! la table correspondante sera écrite en minuscule et pluriel

Ex : modèle :Post, en bdd la table: posts

**php artisan make:model Post -m**

astuce on ajoute le flag -m pour générer automatiquement la migration

**CRUD**

**Read**

$posts = **Post::**get();

Return $posts;

**create**

$post = new **Post;** => on appel le model (qui est l’objet de la table)

------------------------------------------------

$post->title =’titre 1’

$post->save();

--------------------------ou par Affectation de masse--------------------

Il faut modifier la protection dans la modele. Pour éviter injection sql

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

$flight = Flight::create(['name' => 'Flight 10']

If you already have a model instance, you may use the fill method to populate it with an array of attributes:

$flight->fill(['name' => 'Flight 22']);

**update**

Classique

$flight = App\Flight::find(1);

$flight->name = 'New Flight Name';

$flight->save();

--------------------------ou par Affectation de masse--------------------

Flight::where('active', 1)

->where('destination', 'San Diego')

->update(['delayed' => 1]);

**Delete**

$deletedRows = Flight::where('active', 0)->delete();

destroy est la méthode correcte pour supprimer directement une entité (via un objet ou un modèle).

Post ::destrouy([4,5]) ;

Sinon

Post ::where(id,4)->delete();

1. **RELATION ELOQUENT**

**One to One**

Model User:

Public function Post()

{

return $this->hasOne(App\Post)

}

**------------------------------------Inverse---------------------------------**

La clé étrangère est dans la table post

**Model Post**

Public function User ()

{

return $this->belongsTo(App\User)

}

Route::get('/user/{id}/post', function($id){

    return User::find($id)->post;

});

Le controller appelle la méthode de du modèle cad pos. Méthode qui de trouve dans le modèle USER

**One to Many**

Model User:

Public function Posts() !!! La méthode est a écrire au pluriel car HasMany

{

return $this->hasMany(App\User)

}

-------inverse----

Model Post

Public function user()

{

Return $this->belongTo(App\User)

}

**Many to Many**

Ex: entre 2 tables users et roles

* Table pivot selon ordre alphabétique : **role\_user** au singulier !!

php artisan make:migraton create\_roles\_users\_table –create=’role\_user’

* **users**
* id - integer
* name - string
* **roles**
* id - integer
* name - string
* **role\_user**
* user\_id - integer
* role\_id - integer

Modèle User

Public function roles()

{

Return $this->belongsToMany(App\Role)

}

------INVERSE-----

Modèle Role

Public function users()

{

Return $this->belongsToMany(App\User)

}

**Récupérer données de la table pivot**

**NOTES: Différence entre appel de la relation ->posts ou posts()**

* $store->distributors() will return the actual relationship object (\Illuminate\Database\Eloquent\Relations\BelongsToMany).
  + When do you use it?

You usually would call the relationship function if you want to further specify the query before you run it. For example add a where statement:

$distributors = $store->distributors()->where('priority', '>', 4)->get();

* $store->distributors will be a collection containing the results of the relationship query (\Illuminate\Database\Eloquent\Collection)
* $store->distributors()->get() will be a collection containing the results of the relationship query (\Illuminate\Database\Eloquent\Collection).
* $store->distributors->get() should return an error since you're calling get() on a Collection object and the first parameter is not optional. If not an error, it should at least return null.

1. **CRUD**

**first, edit your route**

Route::get('invitation/show/{group}', 'InvitationController@show')->name('show.invitation');

with **route()** method, pass the name of the route

<a href="{**{ route('show.invitation', $group) }}"** class="btn btn-primary">Invite User to Group</a>

with **url()** method, pass URL of the route

<a href="**{{ url('/invitation/show/', $group) }}"** class="btn btn-primary">Invite User to Grou

 <ul>

        <li><a href="{{route('form.show', $post->id)}}">{{$post->title}}</a></li>

 </ul>

<a href="{{url('/form/'.($post1->id).'/edit')}}"><button>Editer </button> </a>

**Collective**

**url method**

    {!! Form::open(['url' => '/form/' . $editPost->id, 'method' => 'put']) !!}

**Route method**

{!! Form::open(['route'=>['form.destroy',$editPost->id],'method'=>'delete']) !!}

**Action method => controller**

echo Form::open(['action' => ['Controller@method', $user->id]])

1. **Validation**

**2 méthodes:**

* Directement dans le controller

$request->validate([

'title' => 'required|unique:posts|max:255',

'body' => 'required',

'publish\_at' => 'nullable|date',

]);

* Générer une class request

php artisan make:request CreatePostRequest

    public function authorize()

    {

        return true; => passer à true

    }

    /\*\*

     \* Get the validation rules that apply to the request.

     \*

     \* @return array

     \*/

    public function rules()

    {

        return [

            'title' => 'required|max:8'

        ];

    }

1. **DIVERS**

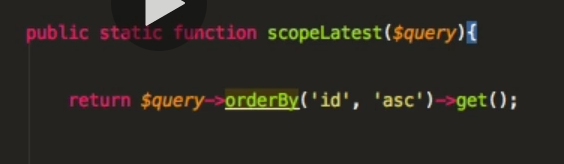
**Date**

        $date = new DateTime();

        return $date->format('d-m-Y');

**query**

methode static dans le modele



Appelle de la static dans le controller

Une image contenant texte

Description générée automatiquement

1. **Ee**
2. **A**
3. **A**
4. **A**
5. **A**
6. **A**