Zouibi Imadeddine : 202

Exercices:

Ex 1:

composer create-project laravel/laravel example-app

Ex 2 :

avec la methode compact() :

$tab= ['nom' => 'imad', ‘email’=>’imad@gmail.com’];

return view('vue', compact('tab'));

dans le fichier vue.blade.php:

Nom: {{ $data['nom'] }}

Age: {{ $data['email’'] }}

Ex 3:

Le moteur de template utilisé par Laravel s'appelle Blade. Blade est un moteur de template simple et puissant intégré à Laravel.

EX 4:

vous pouvez utiliser la classe ‘Validator’ en impoeter la classe tout dabord :

use Illuminate\Support\Facades\Validator;

et en utilise la methode make() , voici un exemple :

```
$validator = Validator::make($request->all(), [
        'nom' => 'required|string|max:255',
        'email' => 'required|email|unique:utilisateurs,email',
            ]);
```

EX 5:

php artisan make:model NomDuModele

EX 6:

Pour ajouter une colonne à une table existante via une migration, vous pouvez utiliser la commande php artisan make:migration pour créer une migration, puis utiliser les méthodes addColumn() ou table() dans la migration pour ajouter la colonne. exemple :

php artisan make:migration ajouter_email_a_users –table=users

EX 7:

La méthode utilisée pour faire une redirection vers une autre route est redirect().

EX 8:

Pour récupérer toutes les données d'une table avec Eloquent, vous pouvez utiliser la méthode all() ,exemple:

$donnees = Modele::all();

EX 9:

Pour créer un middleware, vous pouvez utiliser la commande artisan :

php artisan make:middleware nomdeMiddl

EX 10:

Pour définir une route qui accepte n'importe quelle méthode HTTP, vous pouvez utiliser la méthode any()

Route::any('/url', 'Controller@methode');

EX 11:

Pour accéder à l'instance de la requête actuelle dans un contrôleur, vous pouvez injecter la classe Illuminate\Http\Request :

```
use Illuminate\Http\Request;


public function methode(Request $request) {
   $nom = $request->input('nom');
 }
```

EX 12:

Pour limiter le nombre de requêtes qu'un utilisateur peut faire, vous pouvez utiliser la fonction throttle()

```
Route::middleware('throttle:nombre')->group(function () {
    // Routes ici
});
```

EX 13:

Pour envoyer un email en utilisant Laravel, vous pouvez utiliser la classe Mail :

```
use Illuminate\Support\Facades\Mail;
Mail::to('destinataire@example.com')->send(new NomDuMailable());
```

EX 14 :

```
php artisan serve
```

EX 15 :

Pour accéder aux paramètres d'URL dans une route, vous pouvez les capturer en utilisant des paramètres dans la définition de la route :

```
Route::get('/url/{parametre}', 'Controller@methode');
```

Ex 16:

Pour ajouter une contrainte de clé étrangère dans une migration, vous pouvez utiliser la méthode foreign()

```
$table->foreign('cle_etrangere')->references('cle_primaire')->on('table_reference');
```

EX 17:

Pour créer une tâche planifiée (scheduled task) dans Laravel, vous pouvez utiliser la commande artisan :

```
php artisan make:command NomDeLaCommande
```

EX 18 :

Pour valider une requête entrante et rediriger automatiquement en cas d'erreur, vous pouvez utiliser la méthode validate():

```
$request->validate([
    'champ' => 'regles_de_validation',
]);
```

EX 19 :

```
public function enfants() {
    return $this->hasMany('App\Enfant');
}
```

EX 20 :

Pour charger les relations d'un modèle de manière conditionnelle dans Laravel Eloquent, vous pouvez utiliser la méthode with() :

```
use App\Models\Utilisateur;

// Charger les utilisateurs avec leur rôle uniquement si un certain critère est rempli
$utilisateurs = Utilisateur::when($condition, function ($query) {
    return $query->with('role');
})->get();
```

EX 21:

```
public function getNomAttribute($value) {

    return strtoupper($value);

}
```

EX 22:

Le Tinker est une interface en ligne de commande incluse avec Laravel qui permet d'interagir avec l'application Larave
en utilisant l'interpréteur PHP interactif.

EX 23:

```
php artisan vendor:publish --provider="NomDuProvider"
```

EX 24 :

```
php artisan key:generate
```

EX 25 :

```
php artisan make:event NomDeLEvenement && php artisan make:listener NomDuListener
--event=NomDeLEvenement
```

EX 26 :

Pour implémenter une stratégie de cache personnalisée, vous pouvez créer une classe de cache personnalisée et la configurer dans config/cache.php.

EX 27 :

Le fichier de configuration qui détermine les services externes que l'application utilise est config/services.php

EX 28 :

```
@component('component')
    @slot('slotName')
        Contenu du slot
    @endslot
@endcomponent
```

EX 29:

Un trait dans Laravel est une sorte de classe partielle qui peut être réutilisée dans plusieurs modèles ou contrôleurs. Vous pouvez l'utiliser en l'important dans vos classes à l'aide de la déclaration use.

EX 30:

Pour définir un accesseur dans un modèle Eloquent, vous pouvez définir une méthode avec la convention d'appel getNomAttribute dans le modèle.

EX 31:

```
use Illuminate\Support\Facades\DB;

DB::beginTransaction();

try {

    DB::table('table1')->update(['column1' => 'valeur']);

    DB::table('table2')->delete();

    DB::commit();

}

catch(\Exception $e){

DB::rollBack();

}
```

EX 32:

Dans Laravel, vous pouvez utiliser la méthode encrypt() pour crypter des données avant de les sauvegarder dans la base de données

EX 34:

Pour créer une réponse JSON dans un contrôleur Laravel, vous pouvez utiliser la méthode json() de l'objet response():

return response()->json($donnees);

Ex 37 :

Ouvrez le fichier config/queue.php de votre projet Laravel.

Dans ce fichier, recherchez la clé 'default'. Cette clé indique le pilote de file d'attente par défaut à utiliser. Vous pouvez définir cette clé sur le pilote que vous souhaitez utiliser. Les options courantes sont 'sync', 'database', 'redis', 'beanstalkd', 'sqs', etc. Par exemple, pour utiliser le pilote de file d'attente Redis, vous pouvez définir 'default' => 'redis'.

EX 38:

la méthode broadcast