Résumé Laravel

Laravel Design Pattern : *MVC*

MVC

Installation

step 1: download composer

step 2: verify composer

```
composer --version
```

step 3: install laravel

```
composer global require laravel/installer

laravel new example-app
```

Problem : i got this error

- Failed to download laravel/laravel from dist: The zip extension and unzip/7z commands are both missing, skipping. The php.ini used by your command-line PHP is: C:\php-8.2.5\php.iniNow trying to download from source
- I went to php.ini and uncommented the extention zip and ffi

step 4: run the server

```
cd example-app
php artisan serve
```

step 5: open the browser and go to localhost:8000

Bootstrapping and Lifecycle

Step 1: When a request is made to a Laravel application, the first file that is loaded is typically index.php.

Step 2: The next step involves retrieving an instance of the Laravel application from bootstrap/app.php. This instance contains various service providers that are responsible for handling different aspects of the application.

Step 3: The request is then sent to the HTTP kernel, which is responsible for handling incoming HTTP requests. The HTTP kernel runs a series of bootstrappers that are defined in the Illuminate\Foundation\Http\Kernel class.

Step 4: Middleware is then applied to the request. Middleware acts like a series of filters that can modify or block incoming requests based on certain criteria.

Step 5: One of the most important bootstrapping actions involves loading service providers. Each service provider is registered and then booted one by one.

Step 6: Once all of the service providers have been registered, the request is handed off to the router for dispatching. Routing is also implemented as a service provider.

Step 7: After the controller returns a response, the response travels back outward through the middleware and the HTTP kernel. Finally, the response is sent to the user's web browser using the send() method in index.php.

Database

In Laravel we don't work with SQL Queries, there's an abstraction layer that transforms, the database into classes and queries into methods.

Configuration

Migration

Model

In Laravel, a model represents a single table in a database, and it is responsible for interacting with that table. The model provides an abstraction layer between the database and the application code, allowing developers to perform database operations using an object-oriented syntax.

Models

Factories

Traits

Factories Helpers

Seeds

Mutators

Accessors

Relationships

Controller

Routing

Api Routes

In REST Architecture we have 4 main operations called CRUD:

- GET (Read) :
  - '/users' //these are route uri
  - '/users/{id}'
- POST (Create) : '/users'
- PATCH/PUT (Update) :
  - '/users'
  - '/users/{id}'
- DELETE (delete) : '/users/{id}'

Here's the Route design for RESTful API :

```
//'api.php'
Route::request_type('uri',function(arguments_to_pass){
       return new your_reponse;
});

//example of get request

Route::get(/users/{users},function($user){
       return new \Illuminate\Http\JsonResponse([
       'data' => $user
]);
});
```

Generally api routes return JSON files

laravel uses substitute binding middleware to automatically load model instance to the controller

there's 2 ways to link routes with there controller:

```
Route::apiResource('/users',[\App\Http\Controllers\UserController::class]);
```

or

```
use App\Http\Controllers\UserController;
use Illuminate\Support\Facades\Route;

// Define a GET route that calls the "index" method in the UserController
Route::get('/users', [UserController::class, 'index']);

// Define a POST route that calls the "store" method in the UserController
Route::post('/users', [UserController::class, 'store']);
```

```
// Define a GET route that calls the "show" method in the UserController with an
ID parameter
Route::get('/users/{id}', [UserController::class, 'show']);

// Define a PUT route that calls the "update" method in the UserController with an
ID parameter
Route::put('/users/{id}', [UserController::class, 'update']);

// Define a DELETE route that calls the "destroy" method in the UserController
with an ID parameter
Route::delete('/users/{id}', [UserController::class, 'destroy']);
```

to check our routes :

```
php artisan route:list
```

to avoid a messy code we can create a folder where we will put route file for each model

and copy the code and paste it in 'api.php' then link it using :

```
require __DIR__ . '/api/users.php'
```

in case, you wanted to add a middleware before your route you can add it like following:

```
Route::middleware('auth')->get('/users', [UserController::class, 'index']);
```

to apply it to a group use :

```
Route::middleware('auth')->prefix(user)->group(function(){
// Define a GET route that calls the "index" method in the UserController
Route::get('/users', [UserController::class, 'index'])->name('index');

// Define a POST route that calls the "store" method in the UserController
Route::post('/users', [UserController::class, 'store'])->name('store');

// Define a GET route that calls the "show" method in the UserController with an
ID parameter
Route::get('/users/{id}', [UserController::class, 'show'])->name('show');;

// Define a PUT route that calls the "update" method in the UserController with an
ID parameter
Route::put('/users/{id}', [UserController::class, 'update'])->name('update');;

// Define a DELETE route that calls the "destroy" method in the UserController
with an ID parameter
Route::delete('/users/{id}', [UserController::class, 'destroy'])-
>name('destroy');;
})
```

> make sure you **name** your routes to make there access stable , and use **prefix** to add a layer of organisation

> If you wanna add multiple middlewares you should pass to middleware as arguments an assotiative array

> in case we have 2 API Versions, to maintain the first version we add a folder containing that version and we add the prefix of the version to the route

to remove a middleware from a route we use:

```
Route::get('/users', [UserController::class, 'index'])->name('index')
->WithoutMiddleware('auth');
```

to impose a type to use in a route (or to trigger a controller) we can use:

```
Route::get('/users', [UserController::class, 'index'])->name('index')
->Where('user','[0-9]+');
```

there's also:

- WhereAlpha
- WhereAlphaNumeric
- WhereNumber

in the model there's hidden and fillable attributes:

```
protected $fillable=([
'title',
'body',
])

protected $hidden=([
'password',
]);

protected $append=([
'title_upper_case' //here you can add accessors
])
```

Eloquent ORM

an eloquent ORM is the way to write database queries in Laravel

To select we use:

```
$posts = post::query()->where('id','<',3)->get();

return new JSONResponse([
'data' => $posts
]);
```

to insert we use create() and we pass our array

```
$created = post::query()->create([
'title' => $request->title,
'body' =>$request->body,
]);

return new JSONResponse([
'data' => $created
]);
```

to update we use update():

```
$updated = $post->update([
'title' => $request->title ?? $post->title,
'body' => $request->body ?? $post->body,
]);

if(!$updated)
{
return new JsonResponse([
'errors' =>[
'Failed to update model'
]
],400);
}

return new JsonResponse([
'data' => $post
]);
```

to delete we use forcedelete() method :

```
$updated = $post->forcedelete()

if(!$updated)
```

```
{
return new JsonResponse([
'errors' =>[
'Failed to delete'
]
],400);
}

return new JsonResponse([
'data' => 'success'
]);
```

We can also manage validation using validation():

```
$request->validate([
'title' => 'required',

]);
```

Web routes

Generally web routes return HTML files