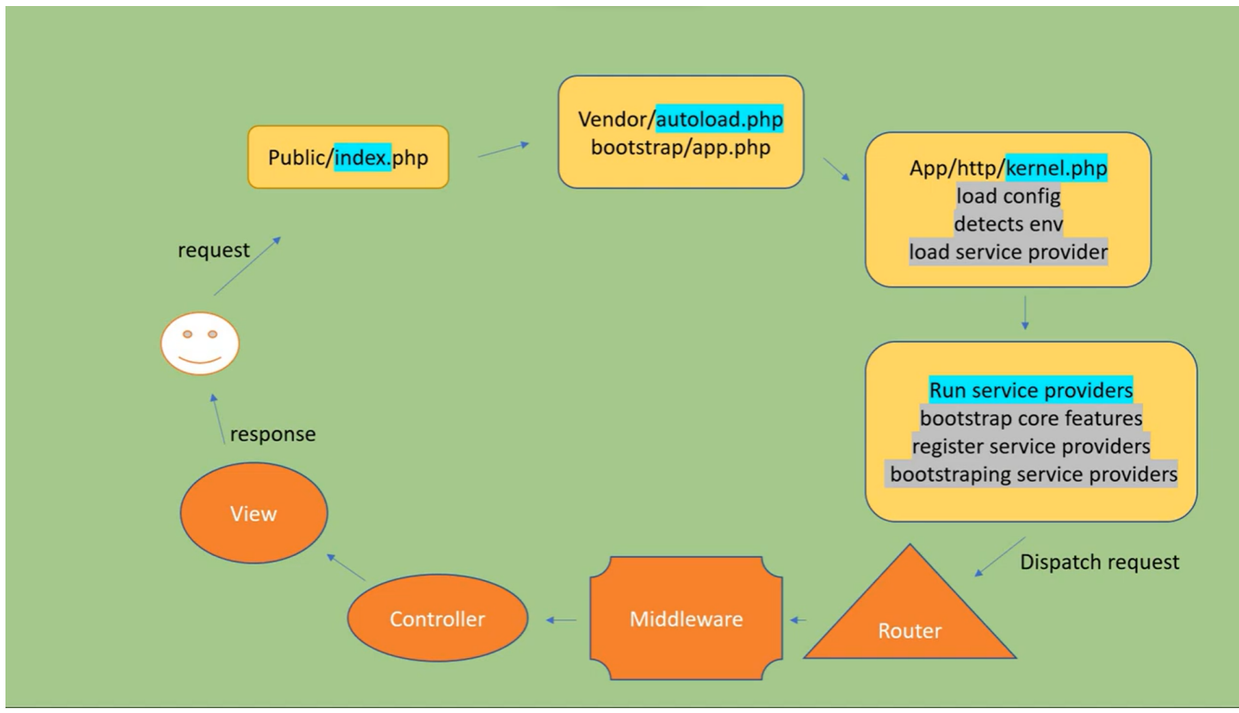
Laravel Request Life cycle



دى الدورة اللى بياخدها كل request علشان يتنفذ

1- اول حاجة انا بروح اعمل ال request بتاعى عادى

2- بعد كده بيبقى فيه عندى ملف اسمه .htaccess علشان يعمل rewrite لكل ال requests علشان يروح ينفذ ملف ال single entry point

3- بعد كده بعمل autoloadلكل الملفات وكل ال packages اللى عندى عن طريق ملف ال autoload.php اللى موجود في folder ال vendor

ملف ال bootstrap هو الملف الذى يحتوى على ال classes الخاصة بالمشروع ويسمى service container

4- بعد ما خلصنا ملف ال autoload هنروح نستدعى الملف اللى اسمه kernel.php والملف ده هو اللى فيه الكود الخاص بتجهيز ال environment variables لل project بتاعنا

5- Service provider: يتحمل مقدمو الخدمة مسؤولية تمهيد المكونات المختلفة للتطبيق. يقومون بتسجيل الخدمات وربط الواجهات بالتطبيقات وتنفيذ مهام الإعداد الأخرى. عادةً ما يتم تنفيذ موفري الخدمة في بداية دورة حياة الطلب.

6- Routing: عادةً ما يتم تكوين خادم الويب لتمرير الطلب إلى ملف public/index.php. في ملف Laravel، Index.php يتولى نظام التوجيه Laravel المهمة. يقوم Laravel بمطابقة عنوان URL للطلب الوارد مع المسار المحدد في تكوين مسارات التطبيق.

7- Middleware: بمجرد مطابقة المسار، يتم تنفيذ أي برنامج وسيط مرتبط بهذا المسار. يمكن للبرامج الوسيطة أداء مهام مثل المصادقة والتسجيل وطلب التعديل.

8- controller: بعد المرور عبر البرامج الوسيطة، يتم تسليم الطلب إلى طريقة وحدة التحكم. وحدة التحكم مسؤولة عن معالجة الطلب وإرجاع الاستجابة. داخل وحدة التحكم، يمكنك الوصول إلى البيانات والتفاعل مع النماذج وتنفيذ منطق آخر خاص بالتطبيق.

9- Middleware (مرة أخرى): بعد انتهاء وحدة التحكم من عملها، قد يتم تنفيذ أي برامج وسيطة إضافية محددة في تعريف المسار.

10- response: بمجرد مرور الطلب عبر مكدس البرامج الوسيطة، يتم إنشاء استجابة. يمكن أن تكون هذه الاستجابة عبارة عن عرض HTML أو بيانات JSON أو أي تنسيق آخر يتطلبه التطبيق.

Facade design pattern

ودى معناها نمط الواجهة

طب هو ليه اتعمل ؟؟؟؟ او هو بيحل مشكلة ايه بالظبط ؟؟؟؟؟

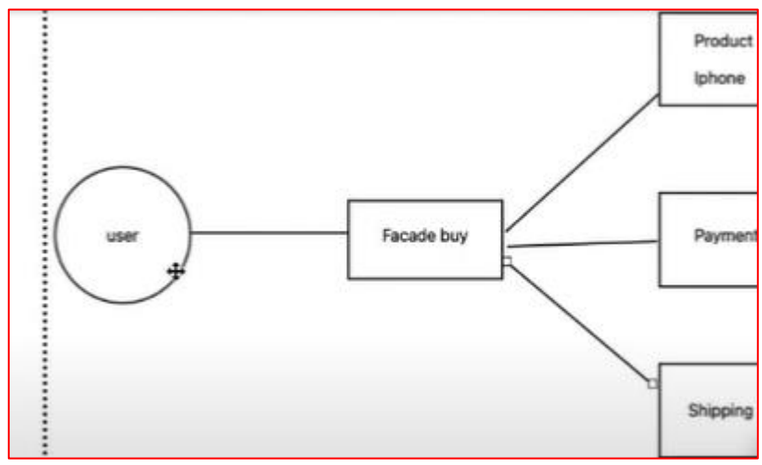
بص يا صديقي انت دلوقتي عندك شوية مكتبات معقدة عايز تشغلها مع بعض عشان تقدر في الآخر تطلع اللوجك بتاعك فانت دلوقتي محتاج انت تشغل المكتبات دي وتهيئها للعمل بالشكل المطلوب نتيجة لذلك، سيصبح business layer مرتبطا بإحكام بتفاصيل التنفيذ لفئات الجهات الخارجية ، مما يجعل من الصعب فهمها والحفاظ عليها

طب علشان نتغلب على المشكلة دى نعمل ايه ؟؟؟؟؟؟؟؟

واجهة بسيطة لنظام فرعي معقد يحتوي على الكثير من الأجزاء المتحركة. قد توفر الواجهة وظائف محدودة مقارنة بالعمل مع النظام الفرعي مباشرة. ومع ذلك، فهو يتضمن فقط تلك الميزات التي يهتم بها العملاء حقا

يعد الحصول على واجهة أمرًا مفيدًا عندما تحتاج إلى دمج تطبيقك مع مكتبة متطورة تحتوي على عشرات الميزات ، ولكنك تحتاج فقط إلى جزء ضئيل من وظائفها

على سبيل المثال، قد يستخدم التطبيق الذي يقوم بتحميل مقاطع فيديو مضحكة قصيرة مع القطط على وسائل التواصل الاجتماعي مكتبة تحويل فيديو احترافية. ومع ذلك ، كل ما تحتاجه حقا هو (اسم الملف ، التنسيق). بعد إنشاء مثل هذا الفصل وربطه بمكتبة تحويل الفيديو ، سيكون لديك واجهتك الأولى



Service Provider

Service providers are the central place of all Laravel application bootstrapping.

تخيل ان Laravel عبارة عن عربية والعربية دى ليها tank يبقى Laravel عبارة عن العربية و ال tank عبارة عن ال service provider

يبقى انا علشان اشغل العربية لازم احط بنزين جوا ال tank

يبقى علشان اشغل اى service جوا Laravel لازم احطها في المكان بتاع ال bootstrap

Laravel's core services are bootstrapped via service providers.

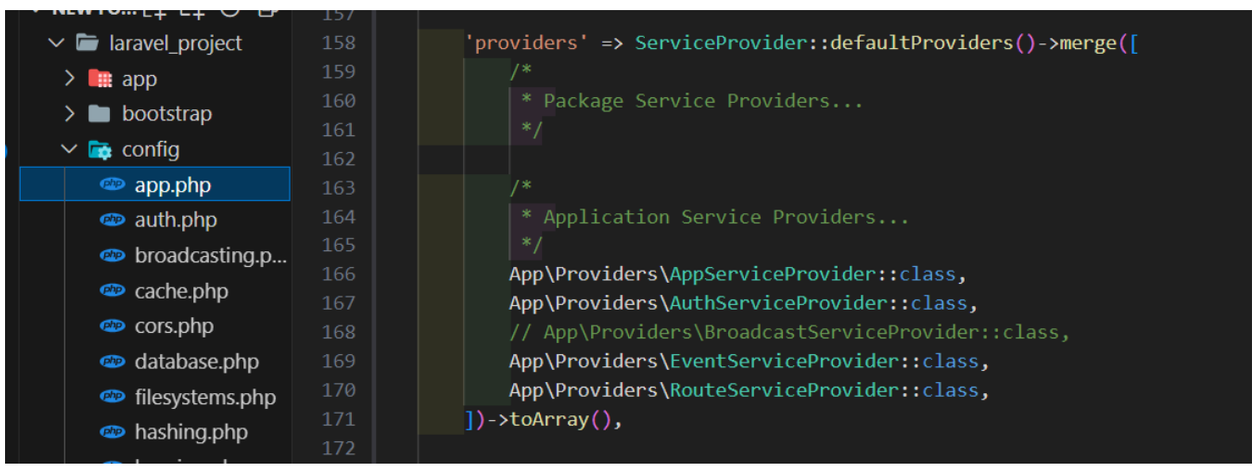
event listeners, middleware, and even routes.

ودى معناها انه Laravel نفسها بيبقى جاى معاها شوية service زى ال session زى ال cookies وهكذا

وعلشان ال service دى تشتغل لازم احطها جوا ال service provider

config/opp.php file included with Laravel, you will see a providers array

ملف ال service provider ده موجود في config ال folder في ملف اسمه app.php



deferred providers, meaning they will not be loaded on every request, but only when the services they provide are actually needed.

فيه نوعين من ال service provider :-

- **service provider انه لما بروح انشا ملف هو مبيروح ويحمل كل المكتبات اللى فيه لا ده هو بيحمل المكتبات على حسب ال request اللى مبعوت في الملف**
- **service provider انه بيروح يحمل كل الملفات**

Service Container

حاوي خدمات Laravel (أي Laravel service container) هو أداةٌ قوية لإدارة اعتِماديَّات (dependencies) الصِّنف والقيام بإضافة اعتِماديَّات (dependency injection)." إضافة الاعتِماديَّات" هو مصطلح تقني يعني في مُجمله "إضافة" اعتماديات الصنف باستخدام التابع الباني (constructor) أو في بعض الحالات توابع ضبط القيم (setter).

لنلقِ نظرة على هذا المثال البسيط:

```php
namespace App\Http\Controllers;

use App\User;
use App\Repositories\UserRepository;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * مستودع المُستخدم تطبيق
     *
     * @var UserRepository
     */
    protected $users;

    /**
     * أنشئ نسخة وحدة تحكم جديدة.
     *
     * @param  UserRepository  $users
     * @return void
     */
    public function __construct(UserRepository $users)
    {
        $this->users = $users;
    }

    /**
     * عرض الملف الشخصي للمُستخدِم المُعطى
     *
     * @param  int  $id
     * @return Response
     */
    public function show($id)
    {
        $user = $this->users->find($id);
        return view('user.profile', ['user' => $user]);
    }
}
```

في هذا المثال يحتاج UserController إلى جلب المُستخدمين من مصدر بيانات. لهذا السبب سنضيف (inject) خدمة (service) تقدر على جلب المُستخدمين. يَستخدمUserRepository ، في هذا السياق، Eloquentلجلب معطيات المُستخدم من قاعدة البيانات. ولكن لمّا كان المستودع (repository) مُضافًا(injected) ، فنستطيع تبديله إلى تعريف استخدام آخر (implementation) بسهولة. نستطيع أيضا أن "نُقلِّد" أو نصنع تعريف استخدام مزيّف من UserRepository عند اختبار تطبيقنا.

فهم حاوي خدمات Laravel فهمًا عميقًا هو مسألةٌ جوهريةٌ في بناء تطبيقات ضخمة قوية، وفي المساهمة لأساس إطار Laravel ذاته.

Validations

يوفّر Laravel عدّة طرق مختلفة للتحقّق من صحّة البيانات الواردة للتطبيقك. يستخدم صنف وحدة التحكّم الأساسي في Laravel الخاصيّة ValidatesRequests التي توفّر طريقة ملائمة للتحقق من صحة الطلب HTTP الوارد بمجموعة متنوّعة من قواعد التحقق الفعّالة.

لمعرفة المزيد عن ميزات التحقق الفعالة فيLaravel ، دعنا نلقي نظرة على مثال كامل من التحقق من صحّة استمارة وعرض رسائل الخطأ على المستخدم.

تعريف المسارات

لنفترض أولًا أنّ لدينا المسارات التالية المحدّدة في ملفّنا:routes/web.php

```
Route::get('post/create', 'PostController@create');

Route::post('post', 'PostController@store');
```

سيعرض المسار GET استمارة للمستخدم لإنشاء منشور مدوّنة جديد بينما سيخزّن المسار POST منشور المدونة الجديد في قاعدة البيانات.

إنشاء وحدة التحكّم

فارغًا حاليًا `store` فلنلقي في الخطوة التالية نظرة على وحدة تحكم بسيطة تتعامل مع هذه المسارات. سنترك التابع:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class PostController extends Controller
{
    /**
     *  إظهار الاستمارة لإنشاء منشور مدوّنة جديد
     *
     * @return Response
     */
    public function create()
    {
        return view('post.create');
    }

    /**
     * إنشاء منشور مدوّنة جديد
     *
     * @param  Request  $request
     * @return Response
     */
```

```
    public function store(Request $request)
    {
        // تحقّق وتخزّن المنشور الجديد ...
    }
}
```

كتابة منطق التحقّق نحن الآن مُستعدّون لملء التابع store بالمنطق للتحقق من مشاركة المدوّنة الجديدة. لذلك، سنستخدم التابع validate المقدّم من الكائن Illuminate\Http\Request. إذا نجحت قواعد التحقّق، ستواصل التعليمات البرمجيّة التنفيذ بشكل طبيعي؛ ولكن في حالة فشل التحقق من الصحّة، سيُطرح استثناء وتُرسل استجابة الخطأ المناسبة إلى المستخدم تلقائيًا. ستُنشئ في حالة طلب HTTP تقليدي استجابة إعادة توجيه، في حين تُرسل استجابة JSON لطلبات AJAX. لفهم التابع validate بشكل أفضل، فلنعد مرّة أخرى إلى التابع store:

```
/**
 *  تخزين منشور المدوّنة الجديد
 *
 * @param  Request  $request
 * @return Response
 */
public function store(Request $request)
{
    $validatedData = $request->validate([
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);

   // منشور المدونة صالح ...
}
```

كما ترى نُمرّر قواعد التحقق المرغوبة للتابع validate. ومرّة أخرى، إذا فشل التحقّق من الصحّة، ستُنشئ الاستجابة المناسبة تلقائيًا. إن نجحت عمليّة التحقّق، ستستمر وحدة تحكّمنا في العمل بشكل طبيعي.

Request

To obtain an instance of the current HTTP request via dependency injection, you should type-hint the Illuminate\Http\Request class on your route closure or controller method. The incoming request instance will automatically be injected by the Laravel service container:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;

class UserController extends Controller
{
    /**
     * Store a new user.
     */
    public function store(Request $request): RedirectResponse
    {
        $name = $request->input('name');

        // Store the user...

        return redirect('/users');
    }
}
```

As mentioned, you may also type-hint the Illuminate\Http\Request class on a route closure. The service container will automatically inject the incoming request into the closure when it is executed:

```php
use Illuminate\Http\Request;

Route::get('/', function (Request $request) {
    // ...
});
```

REFERENCES

- https://www.youtube.com/watch?v=9ULRmpmzshE
- https://laravel.com/docs/10.x/lifecycle#main-content
- https://laravel.com/docs/10.x/facades#main-content
- https://youtu.be/p0pAt8NSyj8?si=rCuF7-l6wlHDZF5-
- https://www.youtube.com/watch?v=gZ0R7DuLZIs
- https://laravel.com/docs/10.x/providers#main-content
- https://wiki.hsoub.com/Laravel/container
- https://laravel.com/docs/10.x/container#main-content
- https://laravel.com/docs/10.x/requests
- https://wiki.hsoub.com/Laravel/validation