

Esta é uma cópia de uma conversa entre ChatGPT e Anônimo.



O que você sugere para instalar em um projeto robusto de laravel 12? O projeto já está configurado com php 8.3, apache e mariadb, quero multitenancy, websockets, emails, queues e tudo mais.

Abaixo vai um “kit de fundação” opinativo para um projeto **Laravel 12 + PHP 8.3 + Apache + MariaDB** com **multitenancy**, **websockets**, **e-mail**, **filas** e mais—focado em robustez, observabilidade e DX.

Visão geral de arquitetura

- **Banco principal (central):** app (usuários, billing, catálogo de tenants, chaves, etc.)
- **Bancos por tenant:** `tenant_{id}` (isolamento forte) ou **schema por tenant** (se usar PostgreSQL).
- **Cache/Filas/broadcast:** Redis (único backend para tudo).
- **WebSockets:** Laravel WebSockets (BeyondCode) em cluster com Redis Pub/Sub.
- **Multitenancy:** `stanc1/tenancy` (maduro, flexível, compatível com queues, events, routes).
- **E-mails:** Mailpit em dev; Postmark/SES em prod (+ Spatie MailEclipse opcional p/ templates).
- **Filas:** Horizon (monitoramento e orquestração dos workers).
- **Jobs/Schedule:** `php artisan schedule:run` via cron (ou Windows Task Scheduler) + queues workers (Supervisor no Linux; NSSM no Windows).
- **Auth/API:** Sanctum (tokens first-party) ou Passport (OAuth2) se você precisa de 3rd parties.
- **Performance:** OPcache + Redis + Octane (Swoole/RoadRunner) se desejar throughput alto (pode manter Apache como *edge* e Octane como backend via reverse proxy, se quiser).
- **Observabilidade:** Telescope, Sentry/Flare, Laravel Debugbar (apenas dev), Larastan, Pest.
- **Segurança/Qualidade:** Spatie Permission, Spatie Activitylog, Pest, PHPStan/Larastan, Laravel Pint, Rector.

Pacotes recomendados (composer)

Instale conforme a necessidade. Em produção, lembre de **fixar versões**.

- Core/Infra
- - Redis, Horizon, Scout (busca), Telescope (obs), Octane (perf), Cache limpa
- Multitenancy, WebSockets, Auth/API, Segurança, Utilidades.

```
```bash
```

## Cache / queues / monitoração

```
composer require predis/predis
composer require laravel/horizon
```

## **WebSockets (broadcast sem Pusher)**

```
composer require beyondcode/laravel-websockets
```

## **Multitenancy**

```
composer require stancl/tenancy
```

## **Auth/API**

```
composer require laravel/sanctum
```

**ou, se precisar OAuth2 (terceiros):**

**composer require laravel/passport**

**E-mail (dev) - use Mailpit externamente (binário), sem composer**

**Opcional para templates de e-mail:**

```
composer require --dev qoraiche/laravel-mail-editor
```

## **Observabilidade / DX**

```
composer require --dev laravel/telescope barryvdh/laravel-debugbar
composer require sentry/sentry-laravel
composer require --dev pestphp/pest pestphp/pest-plugin-laravel nunomaduro/larastan laravel/pint
rector/rector
```

## **Busca full-text (opcional)**

```
composer require laravel/scout
```

**+ driver (meilisearch/meilisearch-php ou algolia/algoliasearch-client-php)**

## **Segurança / auditoria (opcional)**

```
composer require spatie/laravel-permission spatie/laravel-activitylog spatie/laravel-csp
...
```

## **.env base (principais chaves)**

```
``env
APP_ENV=local
APP_DEBUG=true
APP_URL=https://app.localhost ↗
```

## **DB central**

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=app
DB_USERNAME=root
DB_PASSWORD=secret
```

## **Redis único p/ cache, queue e websockets pub/sub**

```
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
```

## **Filas + Horizon**

```
QUEUE_CONNECTION=redis
```

## **Broadcasting**

```
BROADCAST_DRIVER=redis
CACHE_STORE=redis
SESSION_DRIVER=redis
```

## **WebSockets (BeyondCode)**

```
PUSHER_APP_ID=local
PUSHER_APP_KEY=local
PUSHER_APP_SECRET=local
PUSHER_HOST=127.0.0.1
PUSHER_PORT=6001
PUSHER_SCHEME=http
PUSHER_APP_CLUSTER=local
```

## **Mail (DEV - Mailpit)**

```
MAIL_MAILER=smtp
MAIL_HOST=127.0.0.1
```

```
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="no-reply@hero.local" ↗
MAIL_FROM_NAME="Hero"
```

## Telescope / Sentry

```
TELESCOPE_ENABLED=true
SENTRY_LARAVEL_DSN=
...
```

**Mailpit:** execute o binário do Mailpit localmente (`mailpit -smtp :1025 -ui :8025`), UI em <http://localhost:8025> ↗.

## Multitenancy com stancl/tenancy

**Modelo de tenants** no banco central (ex.: tabela `tenants`) e **tenancy por subdomínio:** `{tenant}.app.localhost`. Você pode escolher **single DB + prefix** ou **DB por tenant** (recomendado para isolamento).

Passos típicos:

1. Publicar e migrar:
 

```
```bash
php artisan vendor:publish --tag=tenancy
php artisan migrate
```
```
2. Definir **identificação do tenant** (subdomínio) e **middlewares** da tenancy (isolam cache, filesystem, DB).
3. Criar **migrations de tenant** (pasta `database/migrations/tenant`), além das do app central.
4. Ao criar um tenant, rodar `tenants:artisan migrate` no DB dele.

Exemplo de **rota tenant-aware**:

```
```php
// routes/tenant.php
use Illuminate\Support\Facades\Route;

Route::middleware(['tenancy'])->group(function()
{
Route::get('/dashboard', function(){ return 'Tenant dashboard'; });
});
```
```

Exemplo de **rota central (sem tenancy)**:

```

`php
// routes/web.php
use Illuminate\Support\Facades\Route;

Route::get('/', function() { return 'Landing do app central'; });
`

```

**Jobs/Queues multitenant:** o pacote propaga o contexto de tenant para jobs, eventos e notificações (evita vazar dados entre tenants).

## WebSockets (BeyondCode)

1. Publish/config + migration de stats (opcional):

```

`bash
php artisan websockets:install
php artisan vendor:publish --
provider="BeyondCode\LaravelWebSockets\WebSocketsServiceProvider"
`

```

2. Rodar servidor WS:

```

`bash
php artisan websockets:serve
`

```

3. No **broadcasting.php**, use `pusher` apontando para seu servidor local (as variáveis `PUSHER\_\*` do .env).
4. No front, use Echo com driver pusher apontando para `ws://localhost:6001`.
5. Em produção, **coloque atrás do Apache/Nginx** com proxy para a porta 6001 e sticky sessions desnecessárias (Redis pub/sub cuida do fan-out).

## Filas + Horizon

1. Configurar `QUEUE\_CONNECTION=redis`.

2. Instalar e publicar Horizon:

```

`bash
php artisan horizon:install
php artisan vendor:publish --provider="Laravel\Horizon\HorizonServiceProvider"
`

```

3. Subir workers:

```

`bash
php artisan horizon
`

```

4. **Produção (Linux):** gerencie com **Supervisor** (processos resilientes). **Windows:** use **NSSM** ou o Agendador de Tarefas.

## E-mails

- **DEV:** Mailpit (captura tudo, templates via MailEclipse opcional).
- **PROD:** Postmark, AWS SES ou Mailgun (DKIM/SPF, DMARC, bounce handling).
- **Templates:** Blade/MJML (compilado), teste com `php artisan tinker` enviando mailable para Mailpit.

## Autenticação e API

- **Sanctum** (SPA/1st party, simples e seguro).
- **Passport** se precisar **OAuth2** para terceiros.
- **Permissions:** `spatie/laravel-permission` (roles & permissions por tenant — salve no DB do tenant).

## Observabilidade e Qualidade

- **Telescope** (somente para ambientes não-prod ou com gate).
- **Sentry/Flare** para erros em prod.
- **Logs** rotacionados (Monolog + daily).
- **Larastan + Pest + Pint + Rector** no CI:
 

```
```bash
vendor/bin/pest
vendor/bin/phpstan analyse
vendor/bin/pint
vendor/bin/rector process
```
```

## Performance

- **OPcache** ligado e afinado.
- **Octane** (Swoole/RoadRunner) se você quiser throughput bem acima do Apache/PHP-FPM.
- **Redis** como driver padrão de cache/session/queue.
- **Scout** + Meilisearch para buscas rápidas.

## Tarefas de sistema (CRON / Scheduler)

- **Linux:** cron: `\* \* \* \* \* php /path/artisan schedule:run >> /dev/null 2>&1`
- **Windows:** Agendador de Tarefas chamando `php artisan schedule:run` a cada minuto.

## Checklist de implantação

- ☐ Definir estratégia de **multitenancy** (DB por tenant vs único DB).
- ☐ Configurar **Redis** (cache, queues, broadcast) e ajustar `.env``.
- ☐ Subir **WebSockets** e testar ``Echo`` (canais privados/presença).
- ☐ Configurar **Mailpit** (dev) e provedor de e-mail (prod).
- ☐ Ativar **Horizon** com processo residente (Supervisor/NSSM).
- ☐ Habilitar **Scheduler** (cron/Task Scheduler).
- ☐ Adicionar **Telescope** (dev) e **Sentry** (prod).
- ☐ Implantar **Sanctum** (ou Passport) e **Spatie Permission** por tenant.
- ☐ Criar **migrations**: centrais e por tenant; automação de ``tenants:artisan migrate``.
- ☐ Configurar **backups** (DB central + DBs de tenants) e retenção.
- ☐ CI/CD: testes (Pest), análise estática (Larastan), Pint, builds.

## Dicas específicas (Apache + MariaDB + Windows)

- No Windows, evite rodar workers/sockets em janelas soltas. Use **NSSM** para manter ``php artisan horizon`` e ``php artisan websockets:serve`` como serviços.
- Considere **mover para Linux** em produção (Docker ou VM) para usar Supervisor e facilitar scaling.
- Se ficar no Apache, ative **HTTP/2**, **gzip/brotli** via módulo, e **OPcache JIT** (com parcimônia).
- Para *tenancy por subdomínio* no Windows, garanta os registros de host (arquivo ``hosts``) e VirtualHosts apontando para o mesmo app (tenancy resolve o tenant).

Se quiser, te entrego um **esqueleto inicial** (providers, configs ``tenancy.php``, ``websockets.php``, ``broadcasting.php``, rotas ``tenant.php``/central, e um **Command** para criar tenant + migrar + provisionar usuário admin) pronto pra colar no teu projeto.



quero autenticação via tokens, acho que o sanctum é mais básico (não sei)

Ótima pergunta. Se você quer **autenticação via tokens** e não precisa de **OAuth2 completo** (terceiros, consent screens, refresh tokens padrão OAuth, etc.), então:

- **Sanctum** é o caminho mais simples e robusto para **tokens "first-party"** (SPA, mobile, CLI da própria organização).
- **Passport** só vale se você realmente precisa de **OAuth2** (apps de terceiros, authorization code, client credentials...).
- Alternativa **JWT "puro"** (ex.: `tymon/jwt-auth`) é útil para **totalmente stateless** e interoperabilidade, mas você perde conveniências nativas (revogação por DB, abilities/escopos fáceis, integração out-of-the-box).

Para o seu caso (Laravel 12, multi-tenant, API + websockets, filas, etc.), **recomendo Sanctum** com **Personal Access Tokens** + **abilities** + um **vínculo de tenant no token** (para isolar o acesso).

## Como implantar com Sanctum (tokens de API)

### 1) Instalar e migrar

```

bash
composer require laravel/sanctum
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
php artisan migrate

```

No `config/auth.php`, deixe o guard API como `sanctum` (normalmente já fica ok sem mexer).

Opcional: expiração (minutos) em `config/sanctum.php`:

```

php
// config/sanctum.php
'expiration' => 60*24, // 1 dia, por exemplo

```

### 2) Emitir tokens (com abilities e tenant)

Supondo multi-tenant por subdomínio usando `stancel/tenancy`, guarde o `tenant_id` no token (via `abilities` ou `metadata` simples).

```

php
// app/Http/Controllers/Auth/IssueTokenController.php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class IssueTokenController extends Controller
{
 public function __invoke(Request $request)
 {
 $request->validate(['email'=>'required|email','password'=>'required','tenant'=>'required']);
 $user = \App\Models\User::where('email',$request->email)->first();
 if(!$user || !\Hash::check($request->password,$user->password)){ abort(401,'Credenciais inválidas'); }

```

```

php

 $tenantId = (string) $request->tenant; // valide se o user pertence a este tenant
 $abilities = ['tenant:'.$tenantId,'orders:read','orders:write'];

 $token = $user->createToken('api-'.$tenantId, $abilities);
 return response()->json(['token'=>$token->plainTextToken]);
 }

```

```

}

```



Rotas:

```
```php
// routes/api.php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Auth\IssueTokenController;

Route::post('/auth/token', IssueTokenController::class);

Route::middleware(['auth:sanctum','ability:orders:read'])->get('/orders', function(){ /* ... */ });
Route::middleware(['auth:sanctum','ability:orders:write'])->post('/orders', function(){ /* ... */ });
```
```

### 3) Enforçar o tenant do token (middleware)

Assim você garante que o token só acesse o **tenant correto** (evita “troca de subdomínio” indevida).

```
```php
// app/Http/Middleware/EnsureTokenMatchesTenant.php
namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class EnsureTokenMatchesTenant
{
    public function handle(Request $request, Closure $next)
    {
        $token = $request->user()?->currentAccessToken();
        if(!$token){ abort(401); }
    }
}
```

```
ruby

$abilities = $token->abilities ?? [];
$tenantAbility = collect($abilities)->first(function($a){ return
str_starts_with($a,'tenant:'); });
if(!$tenantAbility){ abort(403,'Token sem tenant'); }

$tokenTenant = explode(':', $tenantAbility)[1] ?? null;

// Exemplo: se você resolve o tenant pelo subdomínio
$requestTenant = tenant()?->getTenantKey(); // com stancl/tenancy
if(!$requestTenant || $requestTenant !== $tokenTenant){ abort(403,'Tenant inválido para este
token'); }

return $next($request);
}
```

```
}
```
```

Registre e use nas rotas do tenant:

```
```php
// app/Http/Kernel.php
protected $routeMiddleware = [
    // ...
    'token.tenant' => \App\Http\Middleware\EnsureTokenMatchesTenant::class,
]
```

```
];
...

```php
// routes/tenant.php (separado das rotas centrais)
use Illuminate\Support\Facades\Route;

Route::middleware(['auth:sanctum','token.tenant'])->group(function()
{
Route::get('/dashboard', fn()->['ok'=>true]);
});
...

```

## 4) Revogação, rotação e escopos

- **Revogar:** `request()->user()->currentAccessToken()->delete();` ou apague via `personal_access_tokens`.
- **Listar tokens:** `$user->tokens`.
- **Rotacionar** (boa prática em login/refresh manual): revogue o antigo e emita um novo.
- **Escopos/abilities:** valide com `ability:foo,bar` no middleware da rota.

## 5) Protegendo Broadcast/WebSockets

Use Sanctum nas rotas de broadcast para canais privados/presença:

```
```php
// routes/channels.php
use Illuminate\Support\Facades\Broadcast;

Broadcast::routes(['middleware'=>['auth:sanctum']]);

Broadcast::channel('orders.{id}', function($user,$id){ return (int)$user->id === (int)$id; });
...

```

No Echo (front), envie o token no header **Authorization: Bearer <token>**.

Quando escolher Passport em vez de Sanctum

- Você precisa de **OAuth2** formal (authorization code, client credentials, refresh tokens padrão com escopo).
- Integração com **terceiros** que esperam **OAuth2**.

Instalação é mais pesada (chaves, `passport:install`, fluxo OAuth). Para seu cenário **first-party** multi-tenant, **Sanctum cobre muito bem**.

Boas práticas para produção

- Defina **SANCTUM_STATEFUL_DOMAINS** ao usar SPA (se usar cookies de sessão/CSRF). Para puro **token Bearer**, não é necessário.
- Ative **expiração** dos tokens em `config/sanctum.php` e/ou implemente revogação ativa.
- Logue emissão/revogação de tokens (auditoria).
- Rate limit nos endpoints de emissão (`throttle:login`).
- Para multi-tenant: valide **tenant** ↔ **usuário** antes de emitir o token e aplique o middleware de **consistência de tenant** (acima).

Se quiser, eu já te deixo um **pacote de arquivos** (controller, middleware, rotas de exemplo e testes Pest) pronto pra colar. Quer que eu gere isso agora?



quero o passport, o sanctum é obrigatório?

Não, o **Sanctum** não é obrigatório 👍.

O **Laravel Passport** é um pacote oficial, baseado em **OAuth2**, para quando você quer **autenticação via tokens padronizada** (access tokens, refresh tokens, client credentials, authorization code flow, etc.). Ele substitui totalmente a necessidade do Sanctum se o seu caso de uso é **API-first** com **tokens Bearer**.

Diferença rápida

- **Sanctum** → tokens simples + sessão SPA. Bom para apps próprios (SPA, mobile, CLI). Mais leve.
- **Passport** → implementação completa de **OAuth2**. Bom se você precisa:
 - Autenticar **apps de terceiros** (ex.: parceiros).
 - Usar **authorization code flow** ou **client credentials**.
 - Emitir **refresh tokens automáticos**.

Instalação do Passport

1. Instale:

```
bash

composer require laravel/passport
```

2. Migre e gere chaves de criptografia:

```
bash

php artisan migrate
php artisan passport:install
```

3. No `App\Models\User` adicione o trait:

```
php

use Laravel\Passport\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, Notifiable;
}
```

4. No `config/auth.php`, mude o driver do guard `api` para `passport`:

```
php

'guards' => [
    'api' => [
        'driver' => 'passport',
        'provider' => 'users',
    ],
],
```

5. Registre o Passport no `AuthServiceProvider`:

```
php

use Laravel\Passport\Passport;

public function boot()
{
    $this->registerPolicies();

    Passport::routes();
}
```

Exemplo de emissão de token

- **Password Grant Tokens:**

```
bash

curl -X POST http://app.localhost/oauth/token \
  -d 'grant_type=password' \
  -d 'client_id=CLIENT_ID' \
  -d 'client_secret=CLIENT_SECRET' \
  -d 'username=user@example.com' \
  -d 'password=secret' \
  -d 'scope='
```

Resposta:

```
json

{
  "token_type": "Bearer",
  "expires_in": 31536000,
  "access_token": "...",
```

```
"refresh_token": "..."  
}
```

Escopos (scopes)

No `AuthServiceProvider`:

```
php  
  
Passport::tokensCan([  
    'orders:read' => 'Ler pedidos',  
    'orders:write' => 'Criar e editar pedidos',  
]);
```

Na rota:

```
php  
  
Route::get('/orders', fn()=>...) ->middleware('scopes:orders:read');
```

Multi-tenant com Passport

Você pode:

- Criar **clients por tenant** (cada cliente OAuth atrelado ao tenant).
- Ou incluir o `tenant_id` no token (claims extras).

Isso garante que o token de um tenant não tenha acesso a outro.

👉 Resumindo:

- Se você quer **OAuth2 completo com tokens Bearer + refresh tokens** → use **Passport**.
- O **Sanctum não é obrigatório**, só é recomendado quando você não precisa da complexidade do Passport.

Quer que eu monte um **fluxo pronto de autenticação multi-tenant com Passport** (incluindo refresh tokens e restrição por tenant)?



Veja:

```
$ composer require laravel/passport  
./composer.json has been updated  
Running composer update laravel/passport  
Loading composer repositories with package information  
Updating dependencies  
Your requirements could not be resolved to an installable set of packages.
```

Problem 1

- Root composer.json requires laravel/passport * -> satisfiable by laravel/passport[v0.1.0, ..., v0.2.4, v1.0.0, ..., v1.0.18, v2.0.0, ..., v2.0.11, v3.0.0, v3.0.1, v3.0.2, v4.0.0, v4.0.1, v4.0.2, v4.0.3, v5.0.0, v5.0.1, v5.0.2, v5.0.3, v6.0.0, ..., v6.0.7, v7.0.0, ..., v7.5.1, v8.0.0, ..., v8.5.0, v9.0.0, ..., v9.4.0, v10.0.0, ..., v10.4.2, v11.0.0, ..., v11.10.6, v12.0.0, ..., v12.4.2, v13.0.0, ..., v13.2.0].
- laravel/passport[v0.1.0, ..., v0.2.4, v1.0.0, ..., v1.0.18, v2.0.0, ..., v2.0.11, v3.0.0, ..., v3.0.2, v4.0.0, ..., v4.0.3, v5.0.0, ..., v5.0.3, v6.0.0, ..., v6.0.7, v7.0.0, ..., v7.5.1] require guzzlehttp/guzzle ~6.0 -> found guzzlehttp/guzzle[6.0.0, ..., 6.5.8] but the package is fixed to 7.10.0 (lock file version) by a partial update and that version does not match. Make sure you list it as an argument for the update command.
- laravel/passport[v10.0.0, ..., v10.0.1] require php ^7.3 -> your php version (8.3.14) does not satisfy that requirement.
- laravel/passport[v10.1.0, ..., v10.2.2] require illuminate/auth ^8.2 -> found illuminate/auth[v8.2.0, ..., v8.83.27] but these were not loaded, likely because it conflicts with another require.
- laravel/passport[v10.3.0, ..., v10.3.2] require illuminate/auth ^8.2|^9.0 -> found illuminate/auth[v8.2.0, ..., v8.83.27, v9.0.0, ..., v9.52.16] but these were not loaded, likely because it conflicts with another require.
- laravel/passport[v10.3.3, ..., v10.4.2] require illuminate/auth ^8.37|^9.0 -> found illuminate/auth[v8.37.0, ..., v8.83.27, v9.0.0, ..., v9.52.16] but these were not loaded, likely because it conflicts with another require.
- laravel/passport[v11.5.0, ..., v11.10.6] require illuminate/auth ^9.0|^10.0 -> found illuminate/auth[v9.0.0, ..., v9.52.16, v10.0.0, ..., v10.48.28] but these were not loaded, likely because it conflicts with another require.
- laravel/passport[v11.0.0, ..., v11.4.0] require illuminate/auth ^9.0 -> found illuminate/auth[v9.0.0, ..., v9.52.16] but these were not loaded, likely because it conflicts with another require.
- laravel/passport v12.0.0 requires illuminate/auth ^9.0|^10.0|^11.0 -> found illuminate/auth[v9.0.0, ..., v9.52.16, v10.0.0, ..., v10.48.28, v11.0.0, ..., v11.45.2] but these were not loaded, likely because it conflicts with another require.
- laravel/passport[v12.0.1, ..., v12.4.0] require illuminate/auth ^9.21|^10.0|^11.0 -> found illuminate/auth[v9.21.0, ..., v9.52.16, v10.0.0, ..., v10.48.28, v11.0.0, ..., v11.45.2] but these were not loaded, likely because it conflicts with another require.
- laravel/passport[v12.4.1, ..., v12.4.2] require lcobucci/jwt ^4.3|^5.0 -> satisfiable by lcobucci/jwt[4.3.0, 5.0.0, ..., 5.5.0].
- laravel/passport[v13.0.0, ..., v13.2.0] require league/oauth2-server ^9.2 -> satisfiable by league/oauth2-server[9.2.0].
- laravel/passport[v8.0.0, ..., v8.5.0, v9.0.0, ..., v9.3.2] require php ^7.2 -> your php version (8.3.14) does not satisfy that requirement.
- laravel/passport v9.4.0 requires illuminate/auth ^6.18.31|^7.22.4 -> found illuminate/auth[v6.18.31, ..., v6.20.44, v7.22.4, ..., v7.30.6] but these were not loaded, likely because it conflicts with another require.
- lcobucci/jwt[4.3.0, 5.1.0, ..., 5.5.0] require ext-sodium * -> it is missing from your system. Install or enable PHP's sodium extension.

Alternatively you can require one of these packages that provide the extension (or parts of it):

Keep in mind that the suggestions are automated and may not be valid or safe to use

- paragonie/sodium_compat_ext_sodium
- jigsaw/expresscheckout-php-sdk Express Checkout SDK for PHP allows easy integration

with Juspay's EC payments as a service platform

- vufind/vufind A flexible discovery layer.

- lcobucci/jwt 5.0.0 requires php ~8.1.0 || ~8.2.0 -> your php version (8.3.14) does not satisfy that requirement.

- league/oauth2-server 9.2.0 requires lcobucci/jwt ^5.0 -> satisfiable by lcobucci/jwt[5.0.0, ..., 5.5.0].

To enable extensions, verify that they are enabled in your .ini files:

- F:\wamp64\bin\php\php8.3.14\php.ini

You can also run `php --ini` in a terminal to see which files are used by PHP in CLI mode.

Alternatively, you can run Composer with `--ignore-platform-req=ext-sodium` to temporarily ignore these required extensions.

Use the option `--with-all-dependencies (-W)` to allow upgrades, downgrades and removals for packages currently locked to specific versions.

You can also try re-running `composer require` with an explicit version constraint, e.g. `"composer require laravel/passport:*"` to figure out if any version is installable, or `"composer require laravel/passport:^2.1"` if you know which you need.

Installation failed, reverting `./composer.json` and `./composer.lock` to their original content.

Você esbarrou em **3 coisas**:

1. **Versão do Passport incompatível** com seu stack atual (Laravel 12 / PHP 8.3 / Guzzle 7).
2. Falta da extensão **ext-sodium** (obrigatória para `lcobucci/jwt 4.3+/5.x`, usada pelo Passport moderno). php.net/GitHub
3. O Composer está preso pelo **lock** (e tenta instalar Passports antigos que exigem Guzzle 6).

Como resolver (passo a passo)

A) Habilite ext-sodium no PHP 8.3 do Wamp

- Wamp ícone > **PHP > PHP Extensions > sodium** (marque) e reinicie serviços. Ou edite `php.ini` e garanta `extension=sodium`. Confirme com `php -m | findstr sodium`. [Stack Overflowphp.net](https://stackoverflow.com/questions/76822222/php-8-3-sodium-extension)

B) Instale o Passport compatível com Laravel 12

Use a linha abaixo para forçar versões modernas e permitir atualizar dependências travadas no lock:

```
bash

composer require laravel/passport:^13.2 --with-all-dependencies
```

- **Por quê ^13.2?** É a geração atual compatível com **Laravel 12** e usa `league/oauth2-server 9.2 + lcobucci/jwt 5.x` (compatível com PHP 8.3) e **Guzzle 7** — evitando o erro de "Guzzle ~6.0". [laravel.compackageagist.org/GitHub](https://laravel.com/packageagist.org/GitHub)

Se ainda travar, rode:

```
bash

composer update laravel/passport league/oauth2-server lcobucci/jwt guzzlehttp/guzzle -W
```

C) Passos de setup (depois de instalar)

```
bash

php artisan migrate
php artisan passport:install
```

- No `App\Models\User`: use `Laravel\Passport\HasApiTokens`;
- Em `config/auth.php`, no guard `api`: `"driver" => "passport"`.
- Em `App\Providers\AuthServiceProvider@boot`: `Passport::routes()`;
Documentação oficial (Laravel 12): laravel.com

O que causou o erro original?

- Composer tentou versões antigas do Passport (v0-v7) que **exigem Guzzle 6**, mas seu projeto já tem **Guzzle 7.10** no lock; daí o conflito.
- Versões novas do Passport pedem `lcobucci/jwt 4.3/5.x`, que **precisa de ext-sodium** — sua instalação não tinha a extensão ativa. php.net/GitHub

Se quiser, te deixo já um **snippet completo** (AuthServiceProvider, User, rotas de token e refresh, scopes e middleware multi-tenant) pronto pra colar.



como eu delete tudo no docker de uma só vez? Escreva uma única linha de comando

Se você quer **apagar tudo de uma vez no Docker** (containers, volumes, imagens, redes), pode usar:

```
bash

docker system prune -a --volumes -f
```

Essa linha:

- `system prune` → limpa tudo
- `-a` → remove **todas** as imagens não usadas (inclusive dangling e não utilizadas)
- `--volumes` → remove volumes também
- `-f` → força (sem pedir confirmação)

⚠ Atenção: isso vai limpar absolutamente tudo que não estiver em uso no momento — containers, imagens, volumes e redes vão embora.

Quer que eu monte também um equivalente para **Windows PowerShell** que inclua **remove todos os containers ativos** antes do prune?