

Manual de Programador

Instalación del entorno

Guía Técnica

Manual de programador — Preparación del entorno en Windows

Objetivo: dejar máquina Windows lista para ejecutar proyectos Laravel que usan **Livewire** y **TailwindCSS (Vite)**.

Orden de lectura recomendado: lee el checklist, luego sigue la sección “Instalación” paso a paso y finalmente la sección de “Comprobaciones y resolución de problemas”.

CHECKLIST RÁPIDO (antes de empezar)

- Windows 10/11 (preferible) con permisos de administrador.
- Espacio en disco suficiente (2–5 GB libre mínimo para dependencias).
- Conexión a Internet para descargas.
- Terminal: PowerShell (recomendado) o Git Bash/CMD.
- Decide: usarás **XAMPP/Laragon** (instaladores todo-en-uno) o **WSL2** (Linux dentro de Windows). Este manual usa XAMPP/Laragon como ejemplo, pero incluyo notas de WSL cuando convenga.

1) ELECCIÓN DEL ENTORNO PHP (En base a Windows)

Opciones válidas:

- **XAMPP** (fácil, trae Apache + MySQL + PHP).
- **Laragon** (más ligero y orientado a desarrolladores).
- **WSL2** (recomendado si quieres un entorno Linux real; más avanzado).

En este caso se usará XAMPP por la inclusión de MySQL y PHP.

2) INSTALAR COMPONENTES (paso a paso)

2.1 Instalar XAMPP (con PHP 8.2+)

1. Descarga XAMPP desde apachefriends.org y elige la versión con **PHP 8.2** o superior.
2. Ejecuta el instalador → instala Apache y MySQL (MariaDB).
3. Inicia **XAMPP Control Panel** y arranca **Apache** y **MySQL**.
4. Añade PHP al PATH (opcional pero útil):
    - Abre *Configuración del sistema → Variables de entorno → PATH → Editar*
    - Añade la ruta: C:\xampp\php
5. Verifica en PowerShell/CMD:
6. *php -v*

Si prefieres Laragon: la instalación es similar y Laragon configura PATH automáticamente.

Laragon también ofrece “Start All” y un menú rápido para bases de datos.

2.2 Instalar Composer (gestor de dependencias PHP)

1. Descarga **Composer-Setup.exe** desde getcomposer.org.
2. Ejecuta instalador → cuando te pida la ruta de php.exe, selecciona C:\xampp\php\php.exe (o la ruta de Laragon).
3. Marca “Add composer to PATH” para usar composer desde cualquier terminal.
4. Probar:
5. composer -V

Tip: Si obtienes error de memoria (out of memory) al usar Composer, usa:

php -d memory_limit=-1 C:\path\to\composer.phar install

ó

composer install --ignore-platform-reqs

(este último ignora requisitos de plataforma, usar solo para desarrollo cuando entiendas riesgos).

2.3 Instalar Node.js y npm

1. Descarga Node.js LTS (recomendado) desde nodejs.org (instalador para Windows).
2. Verifica:

node -v

npm -v

2.4 Instalar Git

1. Descarga Git for Windows (git-scm.com) y sigue el instalador.
2. Verifica:

git --version

2.5 (Opcional) Instalar pgAdmin / MySQL Workbench

- Si vas a usar PostgreSQL instala PostgreSQL y pgAdmin.
- Si usas MySQL (XAMPP trae phpMyAdmin), puedes usar phpMyAdmin o MySQL Workbench.

3) CONFIGURACIONES IMPORTANTES DE PHP (php.ini)

Edita php.ini (por ejemplo C:\xampp\php\php.ini) y verifica/ajusta:

- Activar extensiones necesarias (quita el ; delante si están comentadas):

extension=openssl

extension=pdo_mysql

extension=pdo_pgsql ; si usas postgres

extension=mbstring

extension=tokenizer

extension=xml

extension=zip

extension=curl

- Aumentar límites si manejarás uploads grandes:

upload_max_filesize = 50M

post_max_size = 50M

memory_limit = 512M

max_execution_time = 300

- Reinicia Apache desde XAMPP Control Panel tras cambiar php.ini.

4) CREAR EL PROYECTO LARAVEL (si crearas uno desde cero)

(Solo si necesitas crear proyecto nuevo; en tu caso ya tienes repos o código, pero dejo esto por completitud)

composer create-project laravel/laravel nombre_proyecto

cd nombre_proyecto

php artisan serve

php artisan serve levantará el servidor en http://127.0.0.1:8000.

5) PREPARAR UN PROYECTO EXISTENTE (COMANDOS y ARCHIVOS CLAVE)

Nota: omití la parte de clonar. Empiezo desde que ya tienes el código en una carpeta local.

En la carpeta raíz del proyecto (donde está composer.json y package.json):

5.1 Copiar el entorno (.env)

En PowerShell:

copy .env.example .env

o en Git Bash/Linux:

cp .env.example .env

5.2 Instalar dependencias PHP (Composer)

composer install

- Si falla por memoria, ver la nota en 2.2.

- Si te pide platform: ext-* (extensiones de PHP faltantes), instala esas extensiones vía php.ini o usa XAMPP que ya las trae.

5.3 Generar la clave de aplicación

php artisan key:generate

5.4 Crear enlace de almacenamiento (storage link)

Laravel usa public/storage para archivos subidos. En Windows hay dos métodos:

Método A (comando artisan — recomendado):

php artisan storage:link

Si da error por permisos, abre PowerShell como **Administrador** e inténtalo de nuevo.

Método B (si falla): crear un enlace simbólico manual

Abre PowerShell como administrador:

mklink /J "%CD%\public\storage" "%CD%\storage\app\public"

(Este crea una **junction** que suele funcionar mejor en Windows.)

5.5 Configurar .env (base de datos y URLs)

Ejemplo MySQL (XAMPP):

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=nombre_bd

DB_USERNAME=root

DB_PASSWORD= # normalmente vacío en XAMPP por defecto

Ejemplo PostgreSQL:

DB_CONNECTION=pgsql

DB_HOST=127.0.0.1

DB_PORT=5432

DB_DATABASE=nombre_bd

DB_USERNAME=postgres

DB_PASSWORD=tu_password

Otros valores útiles en .env (Vite)

- Laravel + Vite típicamente no necesita variables extra. Asegúrate APP_URL esté correcto:

APP_URL=http://127.0.0.1:8000

Vite detecta automáticamente cuando ejecutas *npm run dev.*

5.6 Ejecutar migraciones y seeders

Crear la base de datos desde phpMyAdmin o pgAdmin antes de migrar.

En terminal:

php artisan migrate

Si existen seeds:

php artisan db:seed

O todo junto:

php artisan migrate --seed

Si quieres recrear todo (desarrollo):

php artisan migrate:fresh --seed

6) FRONT-END: instalar y preparar Tailwind + Vite + Livewire

6.1 Instalar dependencias Node (desde la raíz del proyecto)

npm install

Esto instalará Tailwind, Vite, PostCSS, etc., según package.json.

6.2 Verificar archivos clave

Asegúrate que existen y revisa su contenido mínimamente:

- package.json (tiene scripts "dev": "vite", "build": "vite build" u similares)
- vite.config.js (configuración de Vite; Laravel usa plugin laravel-vite-plugin)
- tailwind.config.js (contenidos / rutas a los Blade y JS)
- postcss.config.js (si aplica)
- resources/css/app.css (debe contener las directivas Tailwind):
- @tailwind base;
- @tailwind components;

- @tailwind utilities;

6.3 Inicializar / revisar Tailwind

Si el proyecto no incluye configuración (o quieres recrearla):

npx tailwindcss init -p

Edita tailwind.config.js:

```
module.exports = {
  content: [
    './resources/**/*.blade.php',
    './resources/**/*.js',
    './resources/**/*.vue',
  ],
  theme: { extend: {} },
  plugins: [],
}
```

Es esencial que las rutas en content incluyan las vistas Blade para que Tailwind "purge" correctamente en prod y compile las clases en dev.

6.4 Compilar assets en desarrollo (Vite dev + HMR)

Abre **dos terminales** (o usa pestañas):

- Terminal A (Laravel backend):

- php artisan serve
- Terminal B (Frontend dev server — Vite):
- npm run dev

npm run dev abrirá el dev server de Vite (HMR). Manténlo abierto mientras desarrollas.

Accede a la app en el navegador en http://127.0.0.1:8000. Las vistas Blade deberían incluir la directiva @vite(['resources/css/app.css','resources/js/app.js']) o usar @vite de Laravel para cargar assets.

Si usas Livewire, no olvides:

- En el <head> principal de tu layout:
- @vite(['resources/css/app.css', 'resources/js/app.js'])
- @livewireStyles
- Antes de </body>:
- @livewireScripts

6.5 Compilar para producción

Cuando vayas a producción:

npm run build

Esto genera assets optimizados en public/build.

7) INSTALAR LIVEWIRE (si no viene ya instalado por composer)

En proyecto:

composer require livewire/livewire

Publicar assets o configs no es obligatorio en la mayoría de casos, pero si el paquete requiere:

php artisan vendor:publish --tag=livewire:config

En las vistas Blade (layouts) añade:

@livewireStyles

...

@livewireScripts

Comandos útiles de Livewire (generadores):

php artisan make:livewire NombreComponente

Para MinervaLab en particular hay comandos personalizados:

php artisan make:admin-livewire Nombre

php artisan make:site-livewire Nombre

(estos generan scaffold para CRUD, subida de archivos, etc. — si están incluidos en el proyecto).

8) EJECUTAR TODO CONCURRENTE Y COMPROBAR FUNCIONAMIENTO

Pasos:

1. En terminal 1: backend
2. *php artisan serve*
3. En terminal 2: frontend (vite)
4. *npm run dev*
5. Abre navegador: http://127.0.0.1:8000
6. Prueba Livewire: abre una página con componentes Livewire y verifica interactividad (sin recarga de página).
7. Prueba subida de archivos: sube una imagen; verifica que aparezca en storage/app/public y que public/storage apunte correctamente.

9) PROBLEMAS COMUNES Y SOLUCIONES (troubleshooting detallado)

Error: Class "PDO" not found o pdo_mysql missing

- Edita php.ini y habilita extension=pdo_mysql. Reinicia Apache.

Error: Composer memory exhausted

- Ejecuta:

php -d memory_limit=-1 composer install

Error: php artisan storage:link falla por permisos

- Ejecuta PowerShell/CMD como **Administrador** y vuelve a ejecutar php artisan storage:link.
- Si sigue fallando, crea manualmente una junction:

mklink /J "C:\ruta\a\proyecto\public\storage" "C:\ruta\a\proyecto\storage\app\public"

Tailwind no aplica estilos (clases no aparecen)

- Revisar tailwind.config.js → content debe incluir rutas a los .blade.php.
- Asegúrate de ejecutar npm run dev durante desarrollo.
- Borrar cache del navegador y recargar.

Vite HMR no funciona (hot reload)

- Asegúrate que npm run dev está corriendo sin errores.
- Revisa consola del navegador por errores CORS o por puerto bloqueado.
- Si la app carga assets desde rutas equivocadas, revisa @vite en Blade y APP_URL en .env.

Livewire no responde o no detecta eventos

- Verifica @livewireStyles y @livewireScripts están incluidos en el layout.
- Si hay JS personalizado, usa Livewire.on('evento', ...) dentro de document.addEventListener('livewire:load', () => { ... }).

Archivos subidos no guardan (error 419 o CSRF)

- Asegura que los formularios Livewire tengan wire:submit.prevent o que los formularios Blade incluyan @csrf.
- Revisa session.cookie_secure o configuraciones de cookies si usas HTTPS local.

Error: You must set the APP_KEY o app no sirve

- Ejecuta php artisan key:generate y reinicia.

Problema con puertos (Vite usa 5173 por defecto)

- Si puerto 5173 está en uso, detén proceso que lo usa o cambia el puerto en vite.config.js o en package.json script:

"dev": "vite --port 5174"

Problemas de antivirus / firewall

- Windows Defender o antivirus pueden bloquear puertos HMR. Permite Node/Vite y Apache en el firewall o añade excepciones.

10) COMANDOS Y ATALLOS ÚTILES

PowerShell / CMD:

Variables básicas

php -v

composer -V

node -v

npm -v

git --version

Laravel (desde la raíz del proyecto)

copy .env.example .env # Windows

composer install

php artisan key:generate

php artisan storage:link

php artisan migrate

php artisan migrate --seed

php artisan serve

Node / Frontend

npm install

npm run dev

npm run build

Livewire

composer require livewire/livewire

php artisan make:livewire NombreComponente

En Windows (si storage:link falla)

mklink /J "%CD%\public\storage" "%CD%\storage\app\public"

11) SUGERENCIAS AVANZADAS Y OPTIMIZACIONES

- **Usar WSL2 con Ubuntu**: si planeas replicar producción Linux (muy recomendable para proyectos serios), instala WSL2 y mueve instalación de PHP/Composer/Node al entorno Linux. Evita problemas de symlinks y permisos.
- **Usar Docker**: montar contenedores para PHP, DB y Node. Más trabajo inicial pero mayor consistencia.
- **Configurar Git ignore**: no subas .env, node_modules, vendor, public/build si no corresponde.

- **Automatizar con scripts**: crea un setup-dev.ps1 que ejecute los comandos básicos (composer install, npm install, php artisan key:generate, migrate, storage:link) para nuevos desarrolladores.

12) CHECKLIST FINAL (antes de empezar a programar)

- php -v muestra PHP 8.1+
- composer -V OK
- node -v y npm -v OK
- composer install terminó sin errores
- npm install terminó sin errores
- .env configurado con DB correcta
- php artisan key:generate ejecutado
- php artisan storage:link creado (o junction)
- Migraciones aplicadas (php artisan migrate --seed)
- php artisan serve en ejecución
- npm run dev en ejecución (puede ser en otra terminal)
- Acceso a la app en http://127.0.0.1:8000 y Livewire/Tailwind funcionan

13) Si quieres: plantilla de setup-dev.ps1 para Windows (opcional)

Copia este script en la raíz del proyecto como setup-dev.ps1 y ejecútalo en PowerShell (ejecuta PowerShell como Admin si necesitas crear symlink):

```
# setup-dev.ps1 (ejemplo)

Write-Host "Instalando dependencias PHP..."

composer install



Write-Host "Copiando .env..."

Copy-Item -Path .env.example -Destination .env -Force



Write-Host "Generando APP_KEY..."

php artisan key:generate



Write-Host "Creando storage link..."

try {

    php artisan storage:link

} catch {

    Write-Host "storage:link falló, intentando mklink..."

    cmd /c mklink /J "%CD%\public\storage" "%CD%\storage\app\public"
```

```
}

Write-Host "Migrando base de datos..."

php artisan migrate --seed



Write-Host "Instalando dependencias JS..."

npm install

Write-Host "¡Listo! Ejecuta 'php artisan serve' y 'npm run dev' en terminales separadas."
```