Contenido

[Documentación Técnica SyverumX 3](#_Toc195738763)

[1 Introducción 3](#_Toc195738764)

[2 Objetivos 3](#_Toc195738765)

[3 Desarrollo 4](#_Toc195738766)

[3.1 Dependencias 4](#_Toc195738767)

[4 Configuración inicial 4](#_Toc195738768)

[Paso 1: Creación de aplicación 4](#_Toc195738769)

[Paso 2: Configuración del Servidor Local 4](#_Toc195738770)

[Integración de un nuevo puerto 4](#_Toc195738771)

[4.1.1 VirtualHost 5](#_Toc195738772)

[Paso 3: Instalar dependencias de Composer y Node.js 5](#_Toc195738773)

[Paso 4: Arranque de Xampp y Tailwinds 6](#_Toc195738774)

[5 Estructura de directivos 6](#_Toc195738775)

[5.1 Directivo app 6](#_Toc195738776)

[5.1.1 Directivo Http 6](#_Toc195738777)

[5.1.2 Modulo Controllers 6](#_Toc195738778)

[5.1.3 Modulo Middleware 6](#_Toc195738779)

[5.1.4 Directorio Models 7](#_Toc195738780)

[5.1.5 Directorio Providers 7](#_Toc195738781)

[5.2 Directivo Core 7](#_Toc195738782)

[5.2.1 Directivo Boot 7](#_Toc195738783)

[5.2.1.1 Modulo Enviroment 8](#_Toc195738784)

[5.2.1.2 Modulo Services 8](#_Toc195738785)

[5.2.2 Directivo Database 8](#_Toc195738786)

[5.2.3 Modulo Facades 9](#_Toc195738787)

[5.2.4 Modulo Http 9](#_Toc195738788)

[5.2.5 Modulo Panel 9](#_Toc195738789)

[5.2.5.1 Modulo Attributes 10](#_Toc195738790)

[5.2.5.2 Modulo Resouces 10](#_Toc195738791)

[5.2.6 Modulo Routing 10](#_Toc195738792)

[5.2.7 Modulo Support 10](#_Toc195738793)

[5.2.8 Directorio Controllers 11](#_Toc195738794)

[5.2.9 Directorio Views 11](#_Toc195738795)

[5.3 Directivo public 12](#_Toc195738796)

[5.4 Directivo resources 12](#_Toc195738797)

[5.5 Directivo routes 12](#_Toc195738798)

[5.6 Directivo vendor 12](#_Toc195738799)

[6 Archivo .env 12](#_Toc195738800)

# Documentación Técnica SyverumX

# Introducción

**SyverumX** es un framework PHP diseñado para simplificar la gestión de peticiones HTTP, manteniendo el patrón de diseño **MVC**. Inspirado en frameworks reconocidos como **Laravel, Symfony y React**, su principal característica es su arquitectura **modular y flexible**, permitiendo que tanto sus funcionalidades como su estructura sean fragmentadas y adaptadas según las necesidades del desarrollador.

Esta flexibilidad garantiza que el **SyverumX** pueda evolucionar con nuevas características o simplificarse eliminando módulos innecesarios, sin comprometer su documentación ni su facilidad de uso.

# Objetivos

Se espera lograr:

* **Eficiencia en el desarrollo**: Simplificar la construcción de aplicaciones web, reduciendo la complejidad en la gestión de rutas, controladores y modelos.
* **Modularidad y personalización**: Permitir que los desarrolladores agreguen o eliminen funcionalidades según sus necesidades, sin afectar el núcleo del framework.
* **Documentación clara y accesible**: Garantizar que el framework sea fácil de entender y extender, proporcionando guías detalladas para su implementación y personalización.
* **Optimización del rendimiento**: Mejorar la velocidad de respuesta en las solicitudes HTTP, manteniendo una arquitectura ligera y escalable.

# Desarrollo

## Dependencias

Las tecnologías utilizadas para que podamos utilizar a **SyverumX** son:

* Xampp
* Composer
* Node.js
* Blade – Librería externa
* Phpdotenv – Librería externa
* Tailwinds
* IDE (Recomendado que sea Visual Studio Code para mayor comodidad)

# Configuración inicial

Para comenzar a utilizar **SyverumX**, es necesario configurar **XAMPP** para redirigir las solicitudes al archivo public/index.php. Además, se debe asignar un puerto específico al proyecto, lo que permitirá el uso de **túneles de desarrollo** para depurar el código tanto en el **frontend** como en el **backend**.

Esta configuración garantiza un entorno de desarrollo más estructurado, facilitando el manejo de rutas y el debugging de la aplicación en futuras etapas.

## Paso 1: Creación de aplicación

Descarga los archivos base de **SyverumX** con el siguiente comando de composer  
***Coming soon***

## Paso 2: Configuración del Servidor Local

### Integración de un nuevo puerto

1. Accede a la carpeta:  
   ***xampp/apache/conf/***
2. Abre el archivo **httpd.conf** con un editor de texto o código.
3. Busca la línea:  
   ***Listen 80***
4. Justo debajo, agrega una nueva línea con el puerto que deseas utilizar. Por ejemplo:  
   ***Listen 3000***
5. Guardas los cambios y reinicias el servidor local para que se apliquen los cambios

### VirtualHost

Una vez que hayas configurado el puerto en el paso anterior, necesitamos establecer un VirtualHost para que el servidor pueda reconocer el proyecto y redirigir las solicitudes al directorio donde se encuentra el archivo index.php.

Abre el archivo ***httpd-vhosts.conf*** ubicado en:  
***xampp/apache/conf/extra/httpd-vhosts.conf***

Al final de este archivo, agrega la siguiente configuración para definir el VirtualHost:

***<VirtualHost \*:3000>***

***# Establece el punto de partida donde van a diriguirse las peticiones del usuario***

***DocumentRoot "C:/xampp/htdocs/ nombre\_proyecto /public"***

***#Reglas aplicadas a el directorio asignado***

***<Directory "C:/xampp/htdocs/nombre\_proyecto/public">***

***Options Indexes FollowSymLinks***

***AllowOverride All***

***Require all granted***

***</Directory>***

***ServerName localhost***

***</VirtualHost>***

## Paso 3: Instalar dependencias de Composer y Node.js

Una vez descargado el proyecto, es necesario instalar las dependencias tanto del backend como del frontend. Para ello, ejecuta los siguientes comandos en la raíz del proyecto:

***composer install***

***npm install***

## Paso 4: Arranque de Xampp y Tailwinds

Con las dependencias instaladas, puedes iniciar los servicios de Apache y MySQL desde el panel de XAMPP (en caso de que el proyecto utilice una base de datos).

Luego, ejecuta el siguiente comando en la terminal para compilar y observar los archivos CSS usando Tailwind CSS. Esto permitirá que los cambios en los estilos se reflejen automáticamente:

***npx @tailwindcss/cli -i ./resources/css/app.css -o ./public/css/output.css --watch***

# Estructura de directivos

## Directivo app

### Directivo Http

El **directivo Http** es responsable de gestionar todas las interacciones relacionadas con las peticiones y respuestas dentro del framework. Este módulo proporciona una estructura organizada para manejar solicitudes entrantes, procesarlas a través de controladores o middleware, y generar respuestas adecuadas.

### Modulo Controllers

En este apartado se almacenan todos los archivos de **Controllers** de la aplicación. Como su nombre lo indica, los **controladores** son responsables de gestionar la lógica de las solicitudes HTTP, procesando la información y devolviendo una respuesta adecuada, ya sea una vista, retornar un estado, redirigiendo al usuario a otra ruta, etc.

Cada **Controller** actúa como un intermediario entre el modelo de datos y las vistas, permitiendo mantener una estructura organizada y modular dentro del framework.

### Modulo Middleware

El apartado **Middleware** contiene archivos que actúan como filtros para las solicitudes HTTP antes de que lleguen a los controladores o después de que se genere una respuesta. Su principal función es aplicar reglas y procesamientos adicionales, como:

* **Autenticación y autorización**: Restringir el acceso a ciertas rutas según los permisos del usuario.
* **Protección contra ataques**: Implementar medidas de seguridad como la protección contra CSRF o la validación de tokens.
* **Modificación de solicitudes y respuestas**: Permitir la manipulación de datos antes de ser procesados o enviados al cliente.

Cada middleware se puede asignar a rutas específicas o aplicarse globalmente para garantizar un flujo de datos seguro y estructurado dentro de la aplicación.

### Directorio Models

El directorio **Models** contiene las clases que representan las entidades de la aplicación y se encargan de la interacción con la base de datos.

### Directorio Providers

El directorio **Providers** contiene las clases responsables de la configuración y el registro de servicios dentro de la aplicación. Los **Service Providers** permiten inicializar componentes clave, enlazar dependencias y extender las funcionalidades del framework.

## Directivo Core

Contiene el núcleo de archivos y directivos que hacen que SyverumX funcione. Constituyen la base sólida de cada funcionalidad y lógica de la aplicación. Cada carpeta dentro del framework está estructurada de manera modular y su nombre representa la característica específica que implementa en el sistema.

A continuación, se describen las principales carpetas y su función dentro del framework:

### Directivo Boot

Tiene como propósito principal inicializar y configurar los componentes esenciales de la aplicación. Es el núcleo del proceso de arranque y contiene clases y servicios que preparan el entorno, cargan configuraciones, y establecen las bases para que la aplicación funcione correctamente.

**Application:** Este archivo actúa como el punto de entrada principal del sistema de arranque del framework. Su función es coordinar la carga y configuración inicial de la aplicación.

#### Modulo Enviroment

* **EnvValidator:** Se encarga de gestionar y validar las variables de entorno utilizadas por la aplicación. Su funcionalidad principal incluye obtener valores de entorno con soporte para valores predeterminados cuando una variable no está definida, verificar si el modo de depuración (APP\_DEBUG) se encuentra habilitado, y detectar la ausencia de variables de entorno críticas para garantizar una configuración adecuada del entorno de ejecución. Esta clase centraliza el acceso seguro a la configuración basada en el entorno, mejorando la robustez y mantenibilidad del sistema.

#### Modulo Services

* **DatabaseService:** Este archivo se encarga de configurar e inicializar la conexión a la base de datos de la aplicación. Lee las credenciales necesarias desde las variables de entorno (DB\_CONNECTION, DB\_HOST, DB\_PORT, DB\_DATABASE, DB\_USERNAME, DB\_PASSWORD) y utiliza esos parámetros para configurar una instancia de la clase **Connection**. Una vez configurado, establece una conexión activa, asegurando que la base de datos esté lista para operaciones. Este servicio es invocado automáticamente por la clase Application durante el proceso de arranque del sistema.
* **PanelService**: Este archivo registra y habilita funcionalidades relacionadas con el monitoreo y la depuración de la aplicación. Se encarga de integrar componentes clave —como rutas, solicitudes HTTP y conexiones a la base de datos— dentro del sistema de monitoreo, facilitando el diagnóstico de errores y el análisis del comportamiento del entorno en tiempo real. Al igual que el servicio de base de datos, este también es cargado automáticamente por la clase **Application** durante la inicialización de la aplicación.

### Directivo Database

 Centraliza la gestión de la conexión y las operaciones con la base de datos. Contiene:

* **Connection**: Maneja la configuración y creación de la conexión a la base de datos utilizando PDO. Implementa el patrón **Singleton** para garantizar una única instancia activa y soporta múltiples drivers (actualmente solo MySQL).
* **Database:** Proporciona métodos estáticos para ejecutar consultas de forma sencilla, utilizando la conexión gestionada por **Connection**.

El propósito de esta carpeta es abstraer y simplificar las interacciones con la base de datos, asegurando una conexión eficiente y segura.

### Modulo Facades

Los archivos que se encuentran aquí, son clases que son intermediarías entre la complejidad interna de la aplicación y el desarrollador para que sean mucho mas sencillo e intuitivo acceder a la característica. Nos encontramos con las siguiente clases/archivos:

* **Request:** Permite acceder a las propiedades HTTP de las solicitudes que el usuario realice
* **Route:** Por medio de esta clase, se establecen las rutas que van a estar disponibles dentro de la aplicación.

### Modulo Http

Se encarga de gestionar y abstraer los datos globales relacionados con las solicitudes HTTP. Contiene:

* **Globals**: Centraliza y organiza la información global de las solicitudes HTTP, como parámetros de consulta ($\_GET), datos del cuerpo de la solicitud ($\_POST), cookies ($\_COOKIE), archivos ($\_FILES), la URL actual, el método HTTP y el endpoint solicitado. También incluye métodos para obtener la URL previa y otros datos relevantes del entorno HTTP.

### Modulo Panel

El objetivo de esta carpeta es facilitar la depuración y el análisis del estado de la aplicación en tiempo de ejecución, proporcionando una herramienta visual y programática para identificar problemas o verificar configuraciones.

* **Monitoring**: Administra los módulos registrados para monitoreo. Permite registrar clases con atributos específicos (Debuggable) y extraer datos relevantes de sus propiedades estáticas para inspección.

#### Modulo Attributes

* **Debuggable**: Define un atributo personalizado que marca propiedades como "depurables". Esto permite que el sistema de monitoreo identifique qué propiedades deben ser inspeccionadas.

#### Modulo Resouces

* **views/panel.blade.php**: Es la interfaz visual del panel de monitoreo. Muestra información organizada sobre rutas, solicitudes HTTP y la base de datos, permitiendo a los desarrolladores inspeccionar el estado de la aplicación de manera interactiva.

### Modulo Routing

Para el módulo Routing, nos encontramos con archivos que permiten el almacenamiento y resolución de las rutas de la aplicación. Aquí, el desglose de la descripción de cada archivo:

* **RouteManager:** Guarda las rutas de la aplicación.
* **RouteResolver:** Resuelve las rutas de la aplicación devolviendo la vista correspondiente.

### Modulo Support

El módulo **Support** contiene archivos de configuración que proporcionan métodos públicos reutilizables en toda la aplicación. Estos métodos están organizados en carpetas según su ámbito de uso, lo que facilita su acceso y mantenimiento.

La estructura del módulo se basa en la **denominación y disponibilidad** de los métodos, es decir, cada carpeta dentro de Support agrupa funciones según el contexto en el que pueden ser utilizadas.

Por ejemplo, las funciones diseñadas para ser utilizadas en los controladores se encuentran dentro de la carpeta Controllers, ubicada en Support. Allí se almacenan los archivos de configuración que hacen posible su uso dentro de los controladores de la aplicación.   
  
Estos archivos van a poder ser modificados para agregar o eliminar metodos que requieran.

### Directorio Controllers

El directorio **Controllers** dentro del módulo **Support** contiene herramientas específicas para facilitar el trabajo con los controladores de la aplicación. Incluye:

* **Factory**: Este archivo inicializa el motor de plantillas Blade y permite renderizar vistas desde los controladores. También gestiona la inyección de datos dinámicos y, si está habilitado, integra el panel de monitoreo en las vistas.
* **Helpers**: Proporciona funciones auxiliares reutilizables diseñadas para simplificar tareas comunes en los controladores, como el manejo de respuestas o el formateo de datos.

Ambos archivos están diseñados para ser extendidos o modificados según las necesidades de los controladores.

### Directorio Views

El directorio Views agrupa herramientas enfocadas en la gestión y personalización de las vistas de la aplicación. Contiene:

1. **Factory**: Administra la generación de vistas utilizando el motor de plantillas Blade. Permite renderizar vistas con datos dinámicos y asegura la integración con el sistema de monitoreo si está habilitado.
2. **Helpers**: Incluye funciones auxiliares para las vistas, como utilidades para formatear datos o generar elementos HTML dinámicos, mejorando la experiencia de desarrollo en la capa de presentación.

Ambos archivos están diseñados para facilitar la personalización y reutilización de las vistas en toda la aplicación.

## Directivo public

El directorio **public** contiene el archivo **index.php,** que es el punto de entrada para todas las solicitudes que ingresan a la aplicación y configura la carga automática. Este directorio también alberga sus activos, como imágenes, JavaScript y CSS.

## Directivo resources

Se encuentran archivos relacionados con las vistas(views), componentes, partials, archivos JavaScript y CSS.

## Directivo routes

El directivo **routes** contiene archivos como **web.php** y **api.php** para establecer las rutas disponibles de la aplicación

## Directivo vendor

Contiene las dependencias de composer y las librerías que se deseen integrar al proyecto

# Archivo .env

El archivo [**.env**](vscode-file://vscode-app/c:/Users/shago/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) contiene las configuraciones clave del entorno de la aplicación, permitiendo una fácil personalización sin modificar el código fuente. Incluye:

* **APP\_DEBUG**: Activa o desactiva el modo de depuración (true para habilitarlo, false para deshabilitarlo).
* **Configuración de la base de datos**:
  1. **DB\_CONNECTION:** Define el driver de la base de datos (en este caso, mysql).
  2. **DB\_HOST y DB\_PORT:** Especifican el host y el puerto del servidor de la base de datos.
  3. **DB\_DATABASE, DB\_USERNAME, DB\_PASSWORD:** Configuran el nombre de la base de datos, el usuario y la contraseña para la conexión.

Este archivo permite modificar parámetros del entorno de ejecución sin alterar el código, facilitando la portabilidad y el mantenimiento de la aplicación.