**SyverumX: Framework PHP Liviano y Modular Para el Desarrollo Ágil de Aplicaciones Web**

Presentado por:

Daniel Santiago Morales Ariza

Dirigida a:

Javier Hernan Jimenez Beltran

Escuela de Ciencias Básicas, Tecnología e Ingeniería – ECBTI

Tecnología en Desarrollo de Software

Universidad Nacional Abierta y a Distancia

12 de Septiembre del 2025

# Dedicatoria

A quienes me enseñaron a pensar con rigor y construir con humildad: a **Programación ATS** porsu guía constante; y a mi equipo y colegas, por compartir la curiosidad que hizo posible SyverumX.

# Agradecimientos

Quiero expresar mi agradecimiento a **Universidad Nacional Abierta y a Distancia** y al **Programa de Tecnología en Desarrollo de Software** por el entorno académico que hizo posible este trabajo. A mi **director,** Javier Hernan Jimenez Beltranpor su orientación, exigencia y confianza a lo largo del proceso. A los **docentes y jurados,** por sus observaciones que enriquecieron la calidad del documento.

A mis **compañeros y colegas**, por las discusiones técnicas y el apoyo durante el desarrollo de **SyverumX**. A la **comunidad de software libre**, cuyos proyectos y documentación inspiraron decisiones de diseño; en particular, agradezco a quienes colaboraron y brindaron retroalimentación en los repositorios del proyecto.

A mi **familia y amigos**, por su ánimo constante y comprensión ante las horas de estudio y desarrollo.

# Resumen

Esta tesis presenta **SyverumX**, un framework liviano y modular en PHP para la construcción de aplicaciones web bajo el patrón Modelo–Vista–Controlador (MVC). Se aplicó la metodología **Design Thinking** para identificar necesidades reales de desarrollo en proyectos de tamaño medio y, a partir de una revisión comparativa de soluciones existentes, se definieron principios de diseño orientados a minimizar la complejidad accidental y la configuración innecesaria.

La propuesta se materializó en una arquitectura con núcleo reducido, módulos desacoplados y convenciones claras para ruteo, controladores, vistas y acceso a datos. Se documentó el flujo de trabajo y se desarrollaron implementaciones de referencia que validan la viabilidad del enfoque en escenarios web típicos. Los resultados evidencian una reducción del esfuerzo inicial de arranque y una mejora en la mantenibilidad sin sacrificar extensibilidad, posicionando a SyverumX como alternativa pragmática entre micro-frameworks y stacks de mayor envergadura.

Las contribuciones principales incluyen: (1) una base MVC optimizada para contextos medianos, (2) lineamientos para integración modular y evolución del sistema, y (3) la aplicación documentada de Design Thinking al diseño de frameworks.

**Palabras clave:** PHP, MVC, framework liviano, Design Thinking, desarrollo web.

# Abstract

This thesis reports the development of **SyverumX**, a lightweight and modular **PHP** framework for building web applications using the **Model–View–Controller (MVC)** pattern. Guided by **Design Thinking** and a comparative review of existing solutions, we elicited requirements and defined an architecture that couples a clear MVC surface (routing, controllers, **Blade** views) with a **hexagonal core**, **DI/IoC**, and conformance to **PSR** standards (PSR-4, PSR-7, PSR-15). We implemented a router, controller layer, Blade rendering, a service container, a minimal **CLI** for scaffolding, and a project skeleton with **.env** configuration and **Composer**. The prototype was validated through unit/functional tests and example applications, showing reduced time-to-first-feature and clean extensibility for small-to-medium projects. **SyverumX** thus offers a lean alternative to full-stack frameworks without sacrificing organization, interoperability, or testability.

**Keywords:** PHP; MVC; Hexagonal Architecture; Clean Architecture; PSR-4; PSR-7; PSR-15; Dependency Injection; Blade; CLI; Developer Experience.

INDICE

[1 Dedicatoria 2](#_Toc208583276)

[2 Agradecimientos 3](#_Toc208583277)

[3 Resumen 4](#_Toc208583278)

[4 Abstract 5](#_Toc208583279)

[5 Presentacion del Proyecto 10](#_Toc208583280)

[5.1 Problema de Investigacion 10](#_Toc208583281)

[5.2 Pregunta de investigación 11](#_Toc208583282)

[6 Objetivos 12](#_Toc208583283)

[6.1 Objetivo General 12](#_Toc208583284)

[6.2 Objetivos Específicos 12](#_Toc208583285)

[7 Marco Conceptual y Teorico 13](#_Toc208583286)

[7.1 Frameworks web y patrón MVC 13](#_Toc208583287)

[7.2 Arquitectura Hexagonal (Ports & Adapters) 13](#_Toc208583288)

[7.3 Clean Architecture y la “Regla de Dependencias” 14](#_Toc208583289)

[7.4 DDD (Domain-Driven Design) 14](#_Toc208583290)

[7.5 Principios SOLID 15](#_Toc208583291)

[7.6 Inversión de Control (IoC) e Inyección de Dependencias (DI) 15](#_Toc208583292)

[7.7 Estándares de interoperabilidad PSR (PHP-FIG) 16](#_Toc208583293)

[7.8 Motores de plantillas y Blade 16](#_Toc208583294)

[7.9 CLI y automatización 16](#_Toc208583295)

[7.10 Gestión de dependencias, configuración y assets 17](#_Toc208583296)

[7.11 Relación con frameworks existentes (Laravel/Symfony) 17](#_Toc208583297)

[7.12 Derivaciones teóricas aplicadas a un framework liviano en PHP 18](#_Toc208583298)

[8 Diseno Metodologico 21](#_Toc208583299)

[9 Requerimientos del Sistema 22](#_Toc208583300)

[10 Arquitectura del Sistema 23](#_Toc208583301)

[11 Pruebas del Sistema 24](#_Toc208583302)

[12 Despliegue del Sistema 25](#_Toc208583303)

[13 Trabajos futuros 26](#_Toc208583304)

[14 CONCLUSIONES 27](#_Toc208583305)

[15 BIBLIOGRAFIA 28](#_Toc208583306)

[16 Anexos 29](#_Toc208583307)

INDICE DE TABLAS

[**Tabla 1** 4](#_Toc126749708)

INDICE DE FIGURAS

[**Figura 1** 6](#_Toc126749806)

# Presentacion del Proyecto

## Problema de Investigacion

En el desarrollo de aplicaciones web, los frameworks full-stack en PHP —como Laravel— han consolidado prácticas y componentes que facilitan ruteo, controladores, vistas, contenedores de servicios y una CLI madura. Esa completitud se consigue a través de múltiples service providers y subsistemas internos, lo que incrementa la superficie del framework y la complejidad de su configuración, especialmente para proyectos pequeños y para desarrolladores en etapa de formación o equipos con recursos limitados **(Laravel. s. f).**

A la vez, existen microframeworks (p. ej., Slim o Lumen) que reducen la huella y el número de decisiones iniciales; están orientados a núcleos mínimos para construir APIs y servicios con rapidez, delegando al usuario la integración de capas adicionales (templating, middlewares extendidos, etc.). Esta liviandad aporta velocidad, pero puede requerir más trabajo para ofrecer una experiencia MVC completa y pedagógica “out-of-the-box”. **(Surfside Media. 2024)**

En la práctica, una parte significativa de las fricciones que reportan los desarrolladores no proviene solo del lenguaje, sino de stacks complejos y pipelines de herramientas que elevan la curva de aprendizaje y el time-to-first-feature. Diversas encuestas del sector subrayan la carga que introducen tecnologías pesadas o heredadas y la complejidad del stack como factor de frustración, lo cual respalda la necesidad de alternativas ligeras que mantengan buenas prácticas sin sobrecargar el arranque de los proyectos **(Atlassian, DX & Wakefield Research. 2024)**

Bajo ese contexto, el problema que aborda esta investigación es la ausencia de un framework ligero y modular en PHP que combine, de forma nativa: (i) interfaz MVC amigable para enseñanza y prototipado, (ii) desacoplamiento interno al estilo Arquitectura Hexagonal / Clean Architecture para proteger el dominio, (iii) DI/IoC mediante contenedor de servicios, (iv) interoperabilidad PSR (autoloading, HTTP y middleware), y (v) DX mejorada con una CLI simple para scaffolding; todo ello pensado para proyectos pequeños y medianos sin renunciar a prácticas profesionales. La implementación de SyverumX se planteó precisamente para cerrar esa brecha: exponer una superficie MVC clara (rutas-controladores-vistas con Blade) sobre un núcleo desacoplado (puertos/adaptadores), con tooling mínimo pero suficiente para iniciar y evolucionar una aplicación.

## Pregunta de investigación

¿Cómo puede un framework ligero y modular en PHP, con interfaz MVC y núcleo hexagonal, mejorar la experiencia y productividad de desarrolladores que requieren entornos rápidos de configurar, fáciles de extender y alineados con estándares PSR?

**Sub-preguntas (operativas):**

* ¿En qué medida una CLI mínima (scaffolding, route:list, make:\*) reduce el tiempo de arranque frente a configurar manualmente los componentes?
* ¿Cómo impacta el desacoplamiento (Hexagonal/Clean) en testabilidad y evolución del código respecto de un MVC estrechamente acoplado a infraestructura?
* ¿Qué tanto aporta la adhesión a PSR-4/PSR-7/PSR-15 a la sustituibilidad de componentes (router, HTTP, middlewares) y a la integración con el ecosistema PHP?
* ¿Es Blade (como motor de plantillas compilado a PHP) una opción con buen equilibrio entre simplicidad y rendimiento para un framework educativo y ligero?
* ¿Qué limitaciones aparecen al usar un framework liviano en apps medianas (p. ej., necesidades de ORM, colas, i18n), y cómo mitigarlas sin perder ligereza?

# Objetivos

## Objetivo General

Desarrollar un prototipo de framework en PHP para la creación de aplicaciones web dinámicas adaptadas a diversas necesidades y propósitos.

## Objetivos Específicos

1. Realizar el levantamiento de requerimientos funcionales y no funcionales del framework Syverum, tomando como referencia frameworks existentes y necesidades comunes del desarrollo web.
2. Diseñar la arquitectura de la solución que permita la separación entre el núcleo de negocio, las interfaces de entrada - salida y los adaptadores, asegurando la interoperabilidad entre ellos.
3. Construir un framework para generar soluciones que incorporen Modelos, Vistas y Controladores (MVC) para garantizar una arquitectura modular y escalable.
4. Ejecutar pruebas unitarias, funcionales y de seguridad sobre los distintos componentes del framework, con el fin de validar su estabilidad, integridad y protección frente a posibles vulnerabilidades.

# Marco Conceptual y Teorico

## Frameworks web y patrón MVC

Un framework web es un conjunto de componentes reutilizables que estandariza tareas como ruteo, controladores, vistas, CLI y manejo de dependencias **(Laravel, s. f.; Symfony, s. f.; Composer, s. f.).** En la web, estos frameworks suelen organizar la interacción usuario–sistema mediante el patrón Modelo–Vista–Controlador (MVC), que separa presentación, lógica de aplicación y dominio para favorecer la mantenibilidad y la prueba. El patrón MVC se originó a fines de los 70 en el entorno Smalltalk de Xerox PARC; la formulación temprana de Trygve Reenskaug describe explícitamente modelos, vistas y controladores como entidades diferenciadas con responsabilidades claras **(Reenskaug, 1979).**

SyverumX adopta MVC como interfaz para el desarrollador (rutas, controladores y vistas), pero desacopla internamente las dependencias de infraestructura, lo que sienta la base para pruebas y extensibilidad sin acoplar el dominio a detalles técnicos. Este enfoque converge con arquitecturas contemporáneas (Hexagonal/Clean) vistas más abajo

## Arquitectura Hexagonal (Ports & Adapters)

La Arquitectura Hexagonal (o *Ports & Adapters*), propuesta por Alistair Cockburn, plantea una aplicación independiente de la UI o la base de datos mediante puertos (interfaces) y adaptadores alrededor de un núcleo de dominio. El autor ha actualizado y ampliado este material en años recientes **(Cockburn, 2025)**.  
Para **SyverumX**, esto implica:

* El núcleo define servicios/contratos.
* Adaptadores HTTP (ruteo, middleware, controladores) y de CLI se conectan vía puertos.
* Motores de vistas (p. ej., Blade) y almacenamiento son detalles intercambiables.

Este desacoplamiento permite exponer MVC de forma pedagógica manteniendo la independencia del dominio respecto a la web o la consola.

## Clean Architecture y la “Regla de Dependencias”

Clean Architecture **(Robert C. Martin)** sintetiza principios para mantener las reglas de negocio independientes de detalles de frameworks, UI o bases de datos: toda dependencia apunta hacia adentro (al dominio). Este principio se alinea naturalmente con Hexagonal e inspira la organización por capas/concéntrica que SyverumX busca facilitar.

## DDD (Domain-Driven Design)

Domain-Driven Design (DDD) de Eric Evans propone centrar el desarrollo en el modelo de dominio y el lenguaje ubicuo dentro de contextos delimitados. Para un framework educativo y modular, DDD ofrece criterios sobre dónde ubicar lógica, cómo nombrar y cómo proteger el core domain del “ruido” técnico. SyverumX puede beneficiarse al reservar el núcleo para políticas del dominio y desplazar infraestructura/adaptadores a los bordes. **(Evans, E. 2015)**

## Principios SOLID

Los principios SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) guían diseños mantenibles. En el contexto de un framework, resaltan:

* SRP: clases con una sola razón de cambio (p.ej., rutas vs. render vs. inyección).
* ISP y DIP: contratos pequeños y dependencias invertidas hacia abstracciones (puertos). Aplicados a SyverumX, ayudan a mantener el contenedor de servicios, el ruteador y el renderer de vistas con responsabilidades nítidas **(Mentores Tech. 2025, abril 13).**

## Inversión de Control (IoC) e Inyección de Dependencias (DI)

IoC caracteriza a los frameworks: el flujo lo orquesta la infraestructura; DI provee dependencias externamente, reduciendo acoplamiento y facilitando pruebas. Fowler distingue DI vs. Service Locator y aboga por roles/contratos para reducir el acoplamiento. Un contenedor hace práctica la DI registrando bindings y resolviendo dependencias en tiempo de ejecución.

En PHP, Laravel popularizó el **Service Container** (registros/bindings y resolución automática), un patrón que SyverumX adopta de manera ligera: servicios esenciales (ruteo, vistas, config, logger) se resuelven por contrato y no por implementación concreta **(Atlassian, DX & Wakefield Research. 2024).**

## Estándares de interoperabilidad PSR (PHP-FIG)

Para favorecer interoperabilidad, SyverumX sigue PSR:

* PSR-4: autoloading por namespaces ↔ rutas de archivos, evitando require manual y facilitando empaquetado con Composer.
* PSR-7: interfaces para mensajes HTTP (Request/Response, URI, headers, streams).
* PSR-15: interfaces para request handlers y middleware de servidor, sobre PSR-7.

Adherirse a PSR permite que SyverumX consuma/exponga componentes compatibles del ecosistema PHP sin ataduras de framework **(PHP-FIG. s.f).**

## Motores de plantillas y Blade

Blade (Laravel) permite herencia de layouts, componentes y directivas, compilando a PHP puro y con caché, lo que minimiza overhead y facilita vistas reutilizables. Integrar Blade como módulo opcional da a SyverumX una vía conocida y productiva para el renderizado, sin imponerlo al núcleo **(Laravel. s. f.).**

## CLI y automatización

Las interfaces de línea de comandos (CLI) mejoran DX (developer experience): scaffolding, seeds, comandos de mantenimiento y orquestación de builds. Symfony Console es un componente maduro, extensible y probado **(Symfony. s. f.)**; Laravel Artisan lo adopta para su CLI. SyverumX define su propia CLI liviana inspirada en estas prácticas, priorizando comandos de creación de proyectos, generación de estructuras MVC y utilidades de entorno **(Laravel. s. f).**

## Gestión de dependencias, configuración y assets

* **Composer**: gestor oficial de dependencias PHP; define librerías y resuelve autoloading. (Composer. s. f.)
* **.env / phpdotenv**: separación de configuración/credenciales del código fuente **(vlucas/phpdotenv. s. f.).**
* **Node.js + Tailwind CSS** (en documentación/sitios demo): pipeline moderno de assets; Tailwind aporta utilidades CSS para prototipado rápido y consistente.  
  Estos elementos son **periféricos** al núcleo del framework, pero críticos para la **experiencia de arranque** del desarrollador (Tailwind CSS. s. f.; Node.js. s. f.)

## Relación con frameworks existentes (Laravel/Symfony)

El ecosistema PHP se ha beneficiado de componentización; de hecho, Laravel consume múltiples componentes de Symfony (HttpFoundation, Console, EventDispatcher, etc.). SyverumX se posiciona como alternativa liviana y pedagógica, que expone MVC y adopta PSR/DI sin la complejidad de un full-stack, pero manteniendo compatibilidad con estándares y bibliotecas **(Laravel. s. f; Symfony. s. f.).**

## Derivaciones teóricas aplicadas a un framework liviano en PHP

A partir del patrón MVC, la Arquitectura Hexagonal (Ports & Adapters) y Clean Architecture, puede derivarse un conjunto de criterios para el diseño de un framework liviano orientado a proyectos pequeños y medianos. En primer lugar, la Regla de Dependencias exige que el dominio permanezca independiente de detalles de infraestructura (web, consola, persistencia). Esto implica definir puertos (interfaces) para las operaciones del dominio y ubicar la infraestructura en adaptadores externos, de modo que las dependencias apunten hacia adentro y la aplicación sea testeable, sustituible y evolutiva **(Cockburn, 2005; Martin, 2017).**

Desde DDD, se privilegia un modelo explícito del dominio y un lenguaje ubicuo que guíe los nombres y límites de los componentes. En términos prácticos, el framework debe facilitar contextos delimitados (bounded contexts) y promover que la lógica de negocio no quede dispersa en controladores o vistas. El MVC se adopta como interfaz pedagógica para el desarrollador —rutas, controladores y vistas— sin convertir a los controladores en fuente de reglas de negocio; estos orquestan casos de uso y devuelven modelos/vistas **(Evans, 2015).**

Los principios SOLID orientan la granularidad y el acoplamiento de los componentes del framework. El SRP demanda responsabilidades acotadas (p. ej., ruteo ≠ renderizado ≠ inyección), el OCP desaconseja modificar el núcleo para extender capacidades, el ISP sugiere interfaces pequeñas y específicas, y el DIP formaliza que los módulos de alto nivel dependan de abstracciones, no de implementaciones concretas. En este contexto, la Inversión de Control (IoC) y la Inyección de Dependencias (DI), instrumentadas mediante un contenedor de servicios, reducen el acoplamiento, hacen explícitas las dependencias y facilitan la prueba de unidades **(Fowler, 2004).**

Para asegurar interoperabilidad y reemplazabilidad de componentes, es deseable adherirse a los estándares PSR definidos por PHP-FIG: PSR-4 para el autoloading por *namespaces*, PSR-7 para mensajes HTTP (Request/Response) y PSR-15 para handlers y middlewares de servidor. Con ello, el framework puede integrarse con bibliotecas del ecosistema sin adaptaciones ad hoc ni dependencias “duras” de un proveedor específico **(PHP-FIG, s. f.).**

En la capa de presentación, un motor de plantillas compilado a PHP con herencia de layouts, componentes y directivas minimiza la lógica en las vistas y favorece la reutilización sin sacrificar rendimiento. La separación de responsabilidades se refuerza con escape por defecto para mitigar XSS y con un middleware transversal para CSRF y políticas de cabeceras seguras; estas prácticas armonizan seguridad con claridad de la capa de vista **(Laravel, s. f.).**

La experiencia de desarrollo (DX) es otro eje derivado de la teoría: una CLI mínima —para *scaffolding*, utilidades de configuración y tareas rutinarias— reduce el time-to-first-feature, estandariza flujos y facilita la reproducibilidad de pasos (p. ej., generar controladores/vistas, listar rutas). El lineamiento aquí no es “acumular comandos”, sino ofrecer los puntos de extensión críticos para mantener el núcleo pequeño y extensible **(Symfony, s. f.; Laravel, s. f.).**

Finalmente, de la conjunción de estos marcos (Hexagonal/Clean, DDD, SOLID, DI/IoC y PSR) se desprende una propuesta de diseño para un framework liviano en PHP:

1. **Núcleo independiente** del transporte y la persistencia (puertos/adaptadores).
2. **Interfaz MVC** simplificada para el desarrollador (rutas→controladores→vistas) sin comprometer la pureza del dominio.
3. **Contenedor de servicios** para DI, configuración modular y *providers*.
4. **Compatibilidad PSR** para aprovechar e intercambiar componentes del ecosistema.
5. **Presentación** con motor de plantillas compilado y políticas de seguridad por defecto.
6. **CLI focalizada** en tareas de arranque y mantenimiento, optimizando la DX.

Este conjunto de derivaciones **justifica teóricamente** un framework ligero que prioriza claridad, extensibilidad y buenas prácticas profesionales, sin la sobrecarga típica de soluciones *full-stack* en escenarios donde la **rapidez de configuración**, la **curva de aprendizaje** y la **modularidad** son críticas.

# Diseno Metodologico

# Requerimientos del Sistema

# Arquitectura del Sistema

# Pruebas del Sistema

# Despliegue del Sistema

# Trabajos futuros

# CONCLUSIONES

# BIBLIOGRAFIA

Cockburn, A. (2005). Hexagonal Architecture (Ports & Adapters). Recuperado de su sitio. [Alistair Cockburn](https://alistair.cockburn.us/hexagonal-architecture)

Cockburn, A. (2025). Hexagonal Architecture Explained (v1.1b) [PDF]. [Alistair Cockburn](https://alistaircockburn.com/hexarch%20v1.1b%20DIFFS%2020250420-1012%20paper%2Bepub.docx.pdf)

Reenskaug, T. (1979). Models–Views–Controllers [PDF]. Universidad de Oslo. [mvc.givan.se](https://mvc.givan.se/papers/Models-Views-Controllers.pdf)

Martin, R. C. (2017). Clean Architecture. (resumen/introducción técnica). [George Aristy](https://georgearisty.dev/posts/clean-architecture/)

Evans, E. (2015). Domain-Driven Design Reference [PDF]. Domain Language. [Domain Language](https://www.domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf)

Fowler, M. (2004). Inversion of Control Containers and the Dependency Injection Pattern. [martinfowler.com](https://martinfowler.com/articles/injection.html)

PHP-FIG. (s. f.). PSR-4, PSR-7, PSR-15. [PSR-4: Autoloader](https://www.php-fig.org/psr/psr-4/)

Laravel. (s. f.). Blade Templates; Service Container; Artisan. [Laravel 12](https://laravel.com/docs/12.x/blade)

Symfony. (s. f.). Console Component; Projects using Symfony (Laravel). [Symfony](https://symfony.com/doc/current/components/console.html)

Composer. (s. f.). Introducción. [Composer introduction](https://getcomposer.org/doc/00-intro.md)

vlucas/phpdotenv. (s. f.). Repositorio oficial. [Dotenv - Environment variables](https://github.com/vlucas/phpdotenv)

Tailwind CSS. (s. f.). Documentación. [Tailwind CSS](https://tailwindcss.com/docs?utm_source=chatgpt.com)

Node.js. (s. f.). About the documentation. [nodejs.org](https://nodejs.org/api/documentation.html?utm_source=chatgpt.com)

Repos oficiales del proyecto: syverum-framework, syverum-skeleton, syverum-installer. [Syverum-framework](https://github.com/Santiagomoo/syverum-framework), [Syverum-skeleton](https://github.com/Santiagomoo/syverum-skeleton), [Syverum-installer](https://github.com/Santiagomoo/syverum-installer)

Mentores Tech. (2025, abril 13). *Cómo se aplican los principios SOLID en arquitecturas modernas*. [Arquitectura SOLID](https://mentorestech.com/resource-blog-content/como-se-aplican-los-principios-solid-en-arquitecturas-modernas)

Surfside Media. (2024). *PHP Microframeworks - Slim, Lumen, and Others*. [PHP Microframeworks](https://www.surfsidemedia.in/post/php-microframeworks-slim-lumen-and-others)

Atlassian, DX & Wakefield Research. (2024). *State of Developer Experience in 2024: Key Findings and Highlights*. Oobeya.io. [State of the developer experience](https://oobeya.io/blog/state-of-developer-experience-in-2024-key-findings-and-highlights)

# Anexos