Syverum: framework PHP liviano y modular para el desarrollo ágil de aplicaciones web

Daniel Santiago Morales Ariza

Asesor

Javier Hernan Jimenez Beltran

Universidad Nacional Abierta y a Distancia - UNAD

Escuela de Ciencias Básicas, Tecnología e Ingeniería – ECBTI

Tecnología en Desarrollo de Software

2025

Dedicatoria

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

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 Beltran por 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 Syverum**.** 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 Syverum, un framework liviano y modular en PHP para la construcción de aplicaciones web bajo el patrón Modelo–Vista–Controlador (MVC). Se aplica la metodología Design Thinking para identificar necesidades 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 Syverum como alternativa pragmática entre micro-frameworks y stacks de mayor envergadura. Las contribuciones principales incluyen una base MVC optimizada para contextos medianos, lineamientos para integración modular y evolución del sistema, y la aplicación documentada de Design Thinking al diseño de frameworks.

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

Abstract

This thesis presents Syverum, a lightweight and modular PHP framework for building web applications following the Model–View–Controller (MVC) pattern. The Design Thinking methodology is applied to identify development needs in medium-sized projects, and based on a comparative review of existing solutions, design principles were defined to minimize accidental complexity and unnecessary configuration. The proposal materialized in an architecture with a reduced core, decoupled modules, and clear conventions for routing, controllers, views, and data access. The workflow was documented, and reference implementations were developed to validate the feasibility of the approach in typical web scenarios. The results show a reduction in initial setup effort and an improvement in maintainability without sacrificing extensibility, positioning Syverum as a pragmatic alternative between micro-frameworks and larger technology stacks. The main contributions include an MVC foundation optimized for medium-scale contexts, guidelines for modular integration and system evolution, and the documented application of Design Thinking to framework design.

Keywords: PHP, MVC, lightweight framework, Design Thinking, web development.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Lista de Apéndices

Introducción

En el desarrollo de software actual, los frameworks desempeñan un papel fundamental al proporcionar estructuras reutilizables que agilizan la creación de aplicaciones y estandarizan buenas prácticas. Sin embargo, en el ecosistema PHP persiste una brecha entre los frameworks full-stack, que ofrecen amplias capacidades a costa de una mayor complejidad, y los micro-frameworks, que priorizan la ligereza pero exigen al desarrollador integrar manualmente múltiples componentes. En este contexto surge Syverum, una propuesta de framework liviano y modular que busca equilibrar ambos enfoques al ofrecer una base sólida bajo el patrón Modelo–Vista–Controlador (MVC), sin imponer configuraciones extensas ni dependencias innecesarias.

La motivación de este proyecto radica en las necesidades de los desarrolladores que enfrentan limitaciones de tiempo, recursos o experiencia, y que requieren herramientas simples pero robustas para iniciar proyectos web de manera ágil. Para orientar su diseño se aplicó la metodología Design Thinking, lo que permitió identificar las principales dificultades en los procesos de arranque, modularización y mantenimiento de proyectos de tamaño medio. A partir de este análisis se definieron principios arquitectónicos centrados en la claridad, la mantenibilidad y la extensibilidad.

Syverum integra conceptos de Arquitectura Hexagonal, Clean Architecture y Domain-Driven Design (DDD) para lograr un núcleo desacoplado y coherente. Asimismo, adopta estándares de interoperabilidad PSR promovidos por el PHP-FIG y una interfaz de línea de comandos (CLI) que facilita tareas de scaffolding y despliegue. Con estas características, el framework busca reducir el tiempo hasta la primera funcionalidad (Time-to-First-Feature, TTFF) y ofrecer una experiencia de desarrollo (DX) más fluida.

La presente tesis documenta el proceso de diseño, desarrollo y validación del framework, desde el levantamiento de requerimientos hasta la ejecución de pruebas funcionales y no funcionales. Los resultados obtenidos demuestran que es posible construir un entorno de desarrollo PHP liviano que mantenga la modularidad y escalabilidad propias de soluciones más complejas, posicionando a Syverum como una alternativa pragmática para el desarrollo ágil de aplicaciones web.

Justificación

El desarrollo de aplicaciones web en la actualidad requiere herramientas que permitan equilibrar productividad, calidad y mantenibilidad. En el ecosistema PHP, frameworks como Laravel o Symfony ofrecen soluciones completas pero con estructuras complejas que implican una curva de aprendizaje considerable y una configuración inicial extensa. Por otro lado, los micro-frameworks, aunque simplifican la puesta en marcha, suelen carecer de componentes integrados que faciliten una experiencia de desarrollo coherente bajo el patrón Modelo–Vista–Controlador (MVC). Esta situación evidencia la necesidad de contar con alternativas intermedias que conserven las buenas prácticas de la ingeniería de software sin imponer una sobrecarga innecesaria en el desarrollo de proyectos medianos o educativos.

Syverum surge como respuesta a esa brecha, al proponer un framework liviano y modular construido en PHP que incorpora principios de Arquitectura Hexagonal, Clean Architecture y Domain-Driven Design (DDD), junto con estándares PSR definidos por el PHP-FIG. El proyecto busca ofrecer una base estructurada que facilite la separación de responsabilidades, promueva el desacoplamiento y mejore la mantenibilidad del código, sin sacrificar la simplicidad en la configuración ni la facilidad de adopción.

La pertinencia de esta investigación se fundamenta en su aporte tanto académico como práctico. Desde el punto de vista académico, contribuye a la literatura técnica sobre diseño de frameworks mediante la aplicación documentada de la metodología Design Thinking, centrada en la empatía con el usuario desarrollador y la iteración de soluciones orientadas a sus necesidades reales. En el ámbito profesional, la propuesta brinda una herramienta que puede ser utilizada por desarrolladores, docentes y estudiantes en entornos donde se requiera un framework

ágil, comprensible y adaptable, reduciendo el tiempo hasta la primera funcionalidad (Time-to-First-Feature) y mejorando la experiencia de desarrollo (DX).

En suma, Syverum representa una contribución significativa al campo del desarrollo web en PHP, al ofrecer un modelo de framework que integra claridad arquitectónica, estándares de interoperabilidad y una curva de aprendizaje accesible. Con ello, se promueve el uso de buenas prácticas, la experimentación controlada y la consolidación de competencias técnicas en comunidades de desarrollo emergentes.

Presentación del Proyecto

Problema de Investigación

En el desarrollo de aplicaciones web, los frameworks full-stack en PHP —como Laravel— han consolidado prácticas y componentes que facilitan el ruteo, los controladores, las vistas, los contenedores de servicios y una interfaz de línea de comandos (CLI). Estas capacidades se logran mediante múltiples service providers y subsistemas internos, lo que incrementa la superficie del framework (Laravel, s. f.). Esto, a su vez, puede aumentar la complejidad de su configuración, especialmente en proyectos pequeños, para desarrolladores en etapa de formación o equipos con recursos limitados.

Por otro lado, existen microframeworks —como Slim o Lumen— que simplifican la configuración inicial y reducen el número de pasos necesarios para construir APIs y servicios de forma rápida. Estos frameworks delegan al usuario la integración de capas adicionales como templating, middlewares extendidos, entre otros. Si bien esto aporta velocidad, también puede requerir más trabajo para ofrecer una experiencia MVC completa “out-of-the-box” (Surfside Media, 2024).

En la práctica, una parte significativa de las fricciones que reportan los desarrolladores no proviene únicamente del lenguaje, sino de la acumulación de factores que ralentizan su trabajo: stacks complejos y pipelines de herramientas que elevan la curva de aprendizaje y el tiempo que tarda un usuario en experimentar el primer beneficio o valor significativo de un producto (*time-to-first-feature*). Diversas encuestas del sector subrayan la carga que introducen tecnologías pesadas o heredadas, así como la complejidad del stack, como factores de frustración. Esto 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 oportunidad de construir un framework ligero y modular en PHP que combine, de forma nativa, una interfaz Modelo-Vista-Controlador (MVC) con capacidades de prototipado, junto con un desacoplamiento interno inspirado en la Arquitectura Hexagonal o *Clean Architecture*, lo que permite mantener independencia entre los componentes del framework y proteger el dominio. Además, busca integrar mecanismos de Inyección de Dependencias e Inversión de Control (DI/IoC) mediante un contenedor de servicios, garantizar la interoperabilidad con los estándares PSR (*PHP Standard Recommendations*) —incluyendo autoloading, manejo de HTTP y middleware—, y ofrecer una experiencia de desarrollador (DX) mejorada mediante una CLI simple para *scaffolding*.

Todo esto está pensado para proyectos que puedan ser soportados en Syverum, sin renunciar a prácticas profesionales. La implementación de Syverum se plantea como una solución para cerrar esta brecha: exponer una superficie MVC clara (rutas-controladores-vistas) sobre un núcleo desacoplado (puertos/adaptadores), con herramientas, utilidades y sistemas de software mínimos pero suficientes 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?

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

Realizar el levantamiento de requerimientos funcionales y no funcionales del framework Syverum, tomando como referencia frameworks existentes y necesidades comunes del desarrollo web.

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.

Construir un framework para generar soluciones que incorporen Modelos, Vistas y Controladores (MVC) para garantizar una arquitectura modular y escalable.

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 Teórico

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).

Syverum 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).

Arquitectura Hexagonal (Ports & Adapters)

La Arquitectura Hexagonal, también conocida como *Ports & Adapters*, fue propuesta por Alistair Cockburn con el objetivo de desacoplar la aplicación de detalles como la interfaz de usuario (UI) o la base de datos. Esta arquitectura organiza el sistema en torno a un núcleo de dominio rodeado por puertos (interfaces) y adaptadores. En años recientes, Cockburn ha actualizado y ampliado este enfoque (Cockburn, 2025).

En el contexto de Syverum, esta arquitectura implica que el núcleo define los servicios y contratos del dominio, mientras que los adaptadores —como HTTP (ruteo, middleware, controladores) y CLI— se conectan al núcleo a través de puertos. Asimismo, elementos como los

motores de vistas (por ejemplo, Blade) y los sistemas de almacenamiento se consideran detalles intercambiables. Cabe resaltar que Blade es el motor de plantillas nativo del framework Laravel, diseñado para facilitar la creación de vistas en aplicaciones web.

Clean Architecture y la "Regla de Dependencias"

La Clean Architecture sintetiza principios que buscan mantener las reglas de negocio independientes de detalles técnicos como frameworks, interfaces de usuario o bases de datos. Su principio central, la "Regla de Dependencias", establece que todas las dependencias deben apuntar hacia el núcleo del dominio. (Martin R.C. 2017)

Este enfoque se alinea con la Arquitectura Hexagonal y se implementa en Syverum mediante capas concéntricas que refuerzan la separación entre lógica de negocio e infraestructura.

DDD (Domain-Driven Design)

El Domain-Driven Design (DDD), introducido por Eric Evans (2015), propone centrar el desarrollo en el modelo de dominio y en el lenguaje ubicuo dentro de contextos delimitados (*bounded contexts*). Para un framework educativo y modular como Syverum, DDD ofrece criterios claros sobre la ubicación de la lógica de negocio, la denominación de entidades y servicios, y la protección del *core domain* frente al "ruido" técnico. Syverum incorpora estas prácticas al reservar el núcleo para las políticas del dominio, desplazando la infraestructura y los adaptadores hacia los bordes del sistema.

Principios SOLID

Son un conjunto de buenas prácticas orientadas a mejorar la mantenibilidad, escalabilidad y claridad del software. En el contexto de un framework como Syverum, estos principios permiten una arquitectura más robusta y desacoplada:

SRP (Single Responsibility Principle) o *Principio de Responsabilidad Única*: establece que cada clase debe tener una única responsabilidad bien definida. En Syverum, esto se refleja en la separación clara entre componentes como el enrutador, el motor de renderizado de vistas y el sistema de inyección de dependencias.

ISP (Interface Segregation Principle) o *Principio de Segregación de Interfaces*: promueve el uso de interfaces específicas y pequeñas, evitando que los módulos dependan de métodos que no utilizan.

DIP (Dependency Inversion Principle) o *Principio de Inversión de Dependencias*: sugiere que los módulos de alto nivel no deben depender de módulos de bajo nivel, sino de abstracciones. (Mentores Tech. 2025, abril 13).

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

La Inversión de Control (IoC) es una característica fundamental de los frameworks modernos, donde el flujo de ejecución es orquestado por la infraestructura en lugar del código del usuario. En este contexto, la Inyección de Dependencias (DI) permite que las dependencias sean proporcionadas externamente, lo que reduce el acoplamiento entre componentes y facilita la realización de pruebas.

Según Fowler, es importante distinguir entre DI y el patrón Service Locator, recomendando el uso de roles y contratos (interfaces o abstracciones) para disminuir el acoplamiento entre módulos. Un ejemplo práctico de DI es el uso de un contenedor de dependencias, que registra *bindings* (reglas que indican qué implementación usar para cada contrato) y resuelve automáticamente las dependencias en tiempo de ejecución (Fowler, M. 2004)/

El framework Laravel popularizó el uso del Service Container, un patrón que permite registrar servicios y resolverlos automáticamente. Syverum adopta este enfoque de manera ligera, utilizando contratos en lugar de implementaciones concretas para resolver servicios esenciales como el ruteo, las vistas, la configuración y el sistema de logging (Atlassian, DX & Wakefield Research, 2024).

Estándares de Interoperabilidad PSR – PHP Framework Interoperability Group (PHP-FIG)

Syverum adopta los estándares PSR propuestos por PHP-FIG para garantizar interoperabilidad con el ecosistema PHP. PSR-4 define el autoloading basado en namespaces y rutas de archivos, eliminando la necesidad de require manual y facilitando la integración con Composer. Por su parte, PSR-7 establece interfaces para mensajes HTTP, incluyendo Request, Response, URI, encabezados y flujos, mientras que PSR-15 introduce interfaces para manejadores de solicitudes y middleware, construidos sobre PSR-7.

La adhesión a estos estándares permite que Syverum consuma y exponga componentes de manera compatible, evitando dependencias rígidas y asegurando flexibilidad frente a diferentes frameworks (PHP-FIG, s.f.-a).

Motores de Plantillas y Blade

Blade, el motor de plantillas nativo de Laravel, ofrece funcionalidades avanzadas para la creación de vistas reutilizables. Entre sus características principales se destacan la herencia de layouts, que permite definir estructuras base que otras vistas pueden extender para facilitar la organización visual; los componentes, que son bloques reutilizables de interfaz que promueven la modularidad; y las directivas, instrucciones propias de Blade que simplifican la lógica dentro de las vistas. Todo el contenido de Blade se compila a PHP puro y se ejecuta con caché, lo que

reduce la carga adicional de procesamiento y mejora el rendimiento. Integrar Blade como un módulo en Syverum proporciona una alternativa conocida, eficiente y productiva para el renderizado de vistas (Laravel, s. f.).

CLI y Automatización

Las interfaces de línea de comandos (CLI, por sus siglas en inglés: Command Line Interfaces) mejoran la experiencia del desarrollador (DX, Developer Experience) al ofrecer herramientas que agilizan el trabajo cotidiano. Entre estas herramientas destaca el scaffolding, que consiste en la generación automática de código base y estructuras iniciales. Esta funcionalidad está implementada en Syverum, facilitando la creación rápida de proyectos.

Otras funcionalidades comunes en entornos automatizados, como los seeds (datos de prueba o iniciales para poblar la base de datos), los comandos de mantenimiento y la orquestación de builds (automatización de tareas de construcción y despliegue), no se contemplan actualmente en Syverum debido a su alcance limitado.

En este contexto, Symfony Console se destaca como un componente maduro, extensible y ampliamente probado (Symfony, s.f.), sobre el cual Laravel Artisan construye su propia CLI. Inspirado en estas prácticas, Syverum define una interfaz de línea de comandos liviana, orientada a facilitar tareas esenciales como la creación de proyectos y la generación de estructuras bajo el patrón MVC (Model-View-Controller) (Laravel, s.f.).

Gestión de Dependencias, Configuración y Assets

Composer es el gestor oficial de dependencias de PHP, que permite declarar las librerías requeridas por un proyecto y resuelve el autoloading (Composer, s. f.).

El archivo .env y la librería phpdotenv permiten separar la configuración y las credenciales del código fuente, favoreciendo la seguridad y la portabilidad del sistema (vlucas/phpdotenv, s. f.).

Finalmente, Node.js y Tailwind CSS se utilizan en la documentación y sitios demostrativos como parte del pipeline moderno de assets; Tailwind aporta utilidades CSS que facilitan un prototipado rápido y consistente. Estos elementos son periféricos al núcleo del framework, pero contribuyen a mejorar la experiencia inicial 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.). Syverum es una alternativa liviana, que usa MVC y adopta PSR/DI como lo hacen otros framework comerciales sin tornarse complejo (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 conocida 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 (Single Responsibility Principle) demanda responsabilidades acotadas (por ejemplo, ruteo ≠ renderizado ≠ inyección). El OCP (Open/Closed Principle) recomienda no modificar el núcleo para extender capacidades. El ISP (Interface Segregation Principle) sugiere interfaces pequeñas y específicas. El DIP (Dependency Inversion Principle) indica que los módulos de alto nivel deben depender de abstracciones, no de implementaciones concretas. En este contexto, la Inversión de Control (IoC) y la Inyección de Dependencias (DI), implementadas mediante un contenedor de servicios, reducen acoplamiento, hacen explícitas las dependencias y facilitan las pruebas unitarias (Fowler, 2004; Martin, 2003).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. -b).

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.).

De la integración de marcos como Hexagonal/Clean Architecture, DDD, SOLID, DI/IoC y los estándares PSR surge una propuesta para un framework liviano en PHP, cuyo núcleo permanece independiente del transporte y la persistencia mediante puertos y adaptadores. Se plantea una interfaz MVC simplificada para el desarrollador sin comprometer la pureza del dominio, un contenedor de servicios que facilite inyección de dependencias, configuración modular y providers, además de compatibilidad con PSR para aprovechar e intercambiar componentes del ecosistema. La presentación se soporta en un motor de plantillas compilado con políticas de seguridad por defecto, mientras que la CLI se orienta a tareas de arranque para optimizar la experiencia del desarrollador. Este conjunto de decisiones justifica teóricamente un framework ligero que prioriza claridad, extensibilidad y buenas prácticas, evitando la sobrecarga típica de soluciones full-stack en escenarios donde la rapidez de configuración, la curva de aprendizaje y la modularidad son críticas.

Diseño Metodológico

Enfoque y Tipo de Estudio

De acuerdo con Sampieri (2020), la investigación aplicada tiene como finalidad resolver problemas prácticos específicos, a diferencia de la investigación básica, cuyo propósito es generar conocimiento universal y generalizable. En este sentido, el desarrollo de software planteado se clasifica como investigación aplicada, ya que busca construir un framework que sirva como solución general para apoyar la creación de software. Además, puede considerarse como una investigación proyectiva, dado que se orienta a la propuesta y diseño de un producto tecnológico. A su vez, incorpora un enfoque exploratorio–descriptivo, pues en la fase de análisis se emplea la técnica de benchmarking para identificar, describir y comparar referentes existentes, lo que permite establecer los requerimientos necesarios que guían la propuesta.

Método y Estrategia General

Se adopta Design Thinking para guiar la identificación de necesidades y la iteración del diseño. El Design Thinking es una metodología de innovación centrada en el usuario que busca resolver problemas complejos mediante empatía, creatividad y experimentación iterativa (Brown, 2009).

En la siguiente tabla se presenta la relación entre el desarrollo del framework y las etapas propuestas en la metodología Design Thinking, mostrando cómo cada objetivo del proyecto se vincula con una fase específica del proceso.

Tabla 1

Relación Entre Objetivos del Proyecto y Etapas de Design Thinking

Objetivo	Etapas de Design Thinking	Relación
1. Levantamiento de requerimientos funcionales y no funcionales del framework Syverum, tomando como referencia frameworks existentes y necesidades comunes del desarrollo web.	Empatizar – Definir	Se identifican necesidades de los usuarios desarrolladores y se precisan los problemas a resolver mediante el análisis de referentes (benchmarking) y la definición de requerimientos.
2. Diseñar la arquitectura de la solución que permita la separación entre núcleo de negocio, interfaces de entrada-salida y adaptadores, asegurando la interoperabilidad.	Idear	Se generan el diseño arquitectónico para garantizar modularidad e interoperabilidad.
3. Construir un framework que incorpore Modelos, Vistas y Controladores (MVC) para garantizar una arquitectura modular y escalable	Prototipar	Se desarrolla un prototipo funcional del framework aplicando la arquitectura definida.

4. Ejecutar pruebas unitarias, funcionales y de seguridad sobre los distintos componentes del framework, para validar estabilidad, integridad y protección.	Evaluar/Testear	Se somete el framework a pruebas que permiten verificar si cumple los requerimientos funcionales y no funcionales definidos.

Nota. Adaptación de las fases del Design Thinking al desarrollo del framework Syverum.

Variables e Instrumentos

Variable Independiente

Corresponde a la implementación del framework Syverum, entendida como el uso de sus componentes principales: sistema de enrutamiento, contenedor de dependencias, motor de vistas Blade, CLI y estructura base del proyecto. Esta variable representa la herramienta tecnológica cuya adopción permite evaluar su impacto en el proceso de desarrollo.

Variables Dependientes

Se evalúan los efectos del uso de Syverum en el trabajo del desarrollador. La productividad se mide según el tiempo para iniciar un proyecto, la rapidez para crear funcionalidades y las tareas completadas. La facilidad de adopción se analiza mediante la curva de aprendizaje, la comprensión de la arquitectura y la claridad de la documentación. La mantenibilidad se valora revisando la organización del proyecto, el grado de desacoplamiento y la facilidad para realizar cambios. Finalmente, la experiencia de desarrollo (DX) se estima según la comodidad, eficiencia y satisfacción percibida al trabajar con el framework.

Instrumentos para el Desarrollo de Benchmarking

Se emplea la técnica de benchmarking, una metodología que permite comparar frameworks existentes para extraer requerimientos funcionales y no funcionales alineados con buenas prácticas de ingeniería de software. Este proceso se apoya en tres instrumentos principales: (1) el instrumento para la elección de frameworks de referencia, (2) el instrumento para el estudio detallado de los frameworks seleccionados y (3) el instrumento para la consolidación de la propuesta de innovación. Estos elementos permiten fundamentar de manera objetiva las decisiones tomadas durante el diseño de Syverum.

Instrumento para Elección de Aplicaciones de Software de Frameworks.

Tabla 2

Elección de Aplicaciones de Software de Framework

Nombre de la Aplicación	Acceso a Documentación	Valor Cualitativo	Escala Cuantitativa	URL
Laravel	Si	Total	10	https://laravel.com/docs
Symfony	Si	Total	9	https://symfony.com/doc
CakePHP	Si	Total	9	https://book.cakephp.org
Codelgniter	Si	Parcial	6	https://codeigniter.com
Yii	Si	Parcial	7	https://www.yiiframework.com/doc
Phalcon	Si	Parcial	6	https://docs.phalcon.io
Slim	Si	Parcial	7	https://www.slimframework.com/docs

Lumen	Si	Parcial	6	https://lumen.laravel.com/docs
FuelPHP	Si	Nulo	5	https://fuelphp.com/docs

Nota. Se tendrán en cuenta los tres Frameworks con más puntaje para la revisión de características.

Instrumento para Estudio de las Aplicaciones de Software de Framework Elegidos.

Sinopsis de las aplicaciones a analizar

En el estudio se seleccionaron tres frameworks de desarrollo en PHP con los mejores puntajes obtenidos durante la evaluación**:** Laravel, Symfony y CakePHP. Cada uno de ellos presenta características particulares que los hacen relevantes dentro del ecosistema de desarrollo web moderno.

Laravel es un framework de desarrollo en PHP cuyo propósito principal es facilitar la creación rápida de aplicaciones web mediante un ecosistema integral que incluye enrutamiento, ORM, colas, trabajos, plantillas y pruebas automatizadas, priorizando la productividad y la elegancia del código. Su arquitectura se basa en el patrón Modelo-Vista-Controlador (MVC) y está compuesta por herramientas modernas como Eloquent ORM, el motor de plantillas Blade, middleware, colas y eventos. Además, ofrece un ecosistema oficial muy amplio que incluye servicios como Forge, Vapor, Nova, Horizon y Scout, junto con una comunidad altamente activa. Laravel destaca por su curva de aprendizaje accesible, gracias a su documentación extensa y a la claridad de sus convenciones, lo que lo convierte en una opción ideal para equipos y desarrolladores que buscan reducir el tiempo de salida al mercado (*time-to-market*). Está orientado a proyectos de negocio de pequeña, mediana y gran escala que valoran la estandarización, el ecosistema y la rapidez de desarrollo. Entre sus características principales se encuentran Eloquent ORM, el sistema de enrutamiento expresivo, el motor de plantillas Blade, las migraciones y *seeders* para gestión de datos, soporte para colas y eventos en tiempo real,

autenticación lista para usar, integración con PHPUnit o Pest para pruebas y el CLI Artisan para automatizar tareas.

Por su parte, Symfony se define como un framework y conjunto de componentes reutilizables orientados a la construcción de aplicaciones PHP altamente configurables, escalables y mantenibles. Su enfoque modular se basa en más de cincuenta componentes independientes —como *HttpFoundation*, *Routing*, *DependencyInjection* o *Console*—, los cuales son utilizados o sirven de inspiración para otros frameworks, incluido Laravel. Symfony destaca por su soporte a arquitecturas limpias y dominios complejos, con un fuerte tipado y la promoción de buenas prácticas de ingeniería de software. Está dirigido principalmente a equipos que priorizan la robustez, la extensibilidad, el control fino y la adherencia a estándares empresariales, así como a organizaciones que manejan sistemas heredados o que requieren cumplir normativas específicas (*compliance*). Entre sus características sobresalen los componentes desacoplados, un potente contenedor de inyección de dependencias con autoconfiguración, un sistema avanzado de enrutamiento y despacho de eventos, el uso del motor de plantillas Twig, y el soporte de Doctrine ORM/DBAL para la persistencia de datos. Además, integra *bundles* y *Flex* para la gestión de paquetes, un completo conjunto de herramientas de consola, perfiles de depuración mediante *Web Profiler*, y sólidos mecanismos de *testing*, internacionalización, caché y seguridad.

CakePHP es un framework basado en MVC orientado al desarrollo rápido de aplicaciones (RAD) mediante el principio de “convención sobre configuración”. Ofrece un conjunto cohesivo de herramientas, como un ORM propio con Query Builder y behaviors, validación, manejo de formularios y autenticación integrada. Su enfoque está dirigido a equipos que prefieren estructuras definidas y mínima configuración, especialmente en proyectos

medianos que requieren estabilidad y mantenimiento sencillo. Incluye la herramienta Bake para generar modelos, controladores y vistas de forma automática, soporte para migraciones, seeds, internacionalización y caché. Su carácter opinado reduce la toma de decisiones y permite concentrarse en la lógica del negocio desde el inicio.

Instrumento Benchmarking de Software de Frameworks.

Tabla 3

Características Técnicas de los Framework Seleccionados

Características Técnicas				
Aplicacion/caracteristica	Laravel	Symfony	CakePHP	Syverum
Lenguaje y compatibilidad (PHP 8.x, HTML/CSS/JS, DB comunes)	x	x	x	x
Escalabilidad (caché, colas, eventos, horizontal)	x	x	x	Parcial
Escalabilidad (caché, colas, eventos, horizontal)	x	x	x	x
Compatibilidad multiplataforma (Linux/Win/macOS; Apache/Nginx; FPM/CLI)	x	x	x	x
Mantenimiento y facilidad de actualización.	x	x	x	x
ORM / Query Builder + Migraciones/Seeders	x	x	x	-
Autenticación y autorización (login, hashing, policies/gates)	x	x	x	-

Colas de trabajos + Planificador (scheduler/cron wrapper)	x	x	x	-

Nota. “X” = soportado; “Parcial” = soporte limitado/en desarrollo (Syverum 2.x); “-” = no evidenciado.

Instrumento para Consolidar Propuesta de Innovación.

De acuerdo con los resultados del cuadro anterior las características que adopta Syverum en su desarrollo

Tabla 4

Aportes a la Innovación de Syverum

Aportes a la Innovación de Syverum		
Características para Tener en Cuenta en el Desarrollo	Prioridad Orden (1 - 10)	Motivación
ORM/Query Builder ligero + Migraciones/Seeders	1	Acelera CRUD y versionado de esquema; estandariza despliegues y pruebas, equiparando a Eloquent/Doctrine/Cake *ORM.
Autenticación y Autorización base (sesiones, hashing, middlewares auth, policies/gates)	2	Habilita casos reales y seguridad mínima lista para usar; reduce tiempo de arranque en proyectos.
Colas de trabajos + Planificador (scheduler)	3	Permite tareas asíncronas (emails, webhooks, sincronizaciones) y mejora la escalabilidad operativa.

Nota. Ordenado por impacto en productividad y adopción temprana.

Un ORM (Mapeador Objeto-Relacional) permite utilizar objetos del lenguaje de programación en lugar de escribir consultas SQL manualmente, facilitando la interacción con la

base de datos y reduciendo errores comunes en la construcción de consultas. Al abstraer las operaciones CRUD, el desarrollador puede concentrarse en la lógica del negocio mientras el ORM traduce automáticamente las acciones sobre los objetos en operaciones equivalentes en la base de datos. Esto aporta consistencia, mantiene un estilo de desarrollo uniforme y mejora la mantenibilidad del proyecto a largo plazo.

Eloquent es el ORM nativo de Laravel y se destaca por su enfoque intuitivo, su sintaxis expresiva y sus relaciones definidas de manera fluida, lo que lo convierte en un estándar dentro del ecosistema PHP. Doctrine, en contraste, es un ORM más flexible y completamente desacoplado, muy utilizado en Symfony, que permite un control profundo mediante mapeo basado en anotaciones, YAML o XML, orientado a proyectos de mayor complejidad. Por su parte, CakePHP incorpora un ORM propio diseñado bajo el principio de "convención sobre configuración", lo que simplifica el trabajo al permitir que gran parte de la estructura se genere automáticamente siguiendo reglas predefinidas, reduciendo la configuración inicial.

La integración de un ORM ligero en Syverum busca combinar las fortalezas de estos enfoques: la simplicidad y fluidez de Eloquent, la estructura robusta de Doctrine y la eficiencia basada en convenciones de CakePHP. Esto permitirá ofrecer a los desarrolladores una herramienta ágil para gestionar datos, compatible con migraciones, seeders y buenas prácticas modernas, garantizando así un proceso de desarrollo más rápido, ordenado y escalable.

Consideraciones Éticas

Alidoosti, Lago, Razavian y Tang (2022) destacan que los principales valores éticos en este campo incluyen la privacidad, la justicia, la sostenibilidad y la responsabilidad, subrayando que estos deben ser considerados desde el diseño inicial del software y no únicamente como medidas reactivas frente a problemas. De acuerdo con estas recomendaciones Syverum

contempla aspectos éticos como la transparencia y confianza mediante una documentación clara, la ausencia de prácticas engañosas y la comunicación de limitaciones; la seguridad y privacidad.

Alcances y Limitaciones

Alcances

El presente trabajo se centra en el diseño, implementación y evaluación de Syverum, un framework liviano y modular desarrollado en PHP, orientado a proyectos pequeños y medianos. El alcance del proyecto abarca la construcción de una interfaz clara basada en el patrón Modelo–Vista–Controlador (MVC), así como la implementación de un núcleo desacoplado que sigue los principios de la Arquitectura Hexagonal y la Clean Architecture, priorizando la separación entre dominio e infraestructura.

Además, se incluye la incorporación de herramientas esenciales que mejoran la experiencia del desarrollador, como una interfaz de línea de comandos (CLI), el motor de vistas Blade, un contenedor de servicios y una estructura base del proyecto. También se contempla el levantamiento de requerimientos a partir de un proceso de benchmarking, junto con pruebas de software documentadas que validan la funcionalidad del sistema.

En cuanto al licenciamiento, Syverum adopta un modelo open source que garantiza la transparencia, la auditoría del código, su modificación y redistribución. Este enfoque fomenta la colaboración comunitaria, mejora la seguridad mediante la revisión abierta y reduce costos al eliminar barreras de acceso (Grupo Castilla, 2024). Operativamente, la colaboración se gestiona a través de repositorios públicos en GitHub —*syverum/installer*, *syverum/skeleton* y *syverum/framework*—, donde se administran *issues* (para registrar tareas, errores y mejoras), *pull requests* (para la incorporación y revisión de cambios de código) y el versionado semántico, lo

que facilita la trazabilidad de las actualizaciones, la revisión por pares y la integración de aportes externos alineados con la hoja de ruta del proyecto.

Licenciamiento

Syverum se distribuye bajo la Licencia MIT, una de las licencias de software libre más utilizadas y reconocidas en el ámbito del desarrollo de código abierto. Esta licencia se distingue por su simplicidad, permisividad y compatibilidad, permitiendo a cualquier usuario utilizar, copiar, modificar, fusionar, publicar, distribuir, sublicenciar e incluso vender copias del software, siempre que se conserve el aviso de derechos de autor y el texto de la licencia original (Sistemas Operativos, 2025).

El uso de la Licencia MIT, disponible en el enlace https://opensource.org/license/MIT, garantiza transparencia y acceso total al código fuente, fomentando la colaboración comunitaria. Además, ofrece flexibilidad legal al permitir su integración en proyectos comerciales o privados sin restricciones severas. También brinda protección legal a los autores, ya que el software se distribuye “tal cual”, sin garantías, eximiendo de responsabilidad por fallos o daños derivados de su uso. Finalmente, su compatibilidad con otras licencias facilita la reutilización de componentes y la interoperabilidad con diversos proyectos.

Gracias a estas características, Syverum puede evolucionar mediante aportes externos, manteniendo su carácter abierto, colaborativo y adaptable a distintos entornos de desarrollo.

Limitaciones

A pesar de los avances logrados, el estudio presenta algunas limitaciones. En primer lugar, las pruebas de rendimiento fueron acotadas, ya que las mediciones de latencia y carga se realizaron únicamente sobre rutas simples, sin incluir escenarios complejos que involucraran procesamiento intensivo o concurrencia elevada. En segundo lugar, se excluyeron componentes

avanzados del análisis, debido a que en esta fase del desarrollo no se integraron ni evaluaron funcionalidades como el ORM, los sistemas de colas, la internacionalización (i18n) o la autenticación avanzada, las cuales podrían ser relevantes en aplicaciones de mayor escala. Finalmente, el estudio se desarrolló en un contexto controlado, con pruebas realizadas en entornos estandarizados que podrían diferir de las condiciones reales de producción, como servidores compartidos o configuraciones personalizadas.

Estas limitaciones no invalidan los resultados obtenidos, pero sí delimitan el alcance de las conclusiones y abren espacio para futuros trabajos que profundicen en los aspectos no abordados en esta fase.

Requerimientos del Sistema

El estándar ISO/IEC/IEEE 29148:2018 establece prácticas y lineamientos para la ingeniería de requisitos durante todo el ciclo de vida de sistemas y productos de software. Publicada por ISO, IEC e IEEE, esta norma define procesos para la elicitación, análisis, documentación y gestión de requisitos, junto con directrices para elaborar artefactos claros, verificables y trazables (ISO/IEC/IEEE, 2018). Su aplicación permite reducir ambigüedades y mejorar la alineación entre las necesidades de los interesados y las funcionalidades desarrolladas.

Por su parte, la ISO/IEC 25010:2011, perteneciente a la familia SQuaRE, define un modelo de calidad compuesto por ocho características principales: adecuación funcional, fiabilidad, eficiencia de desempeño, usabilidad, seguridad, compatibilidad, mantenibilidad y portabilidad (ISO/IEC, 2011). Asimismo, incorpora un modelo de calidad en uso que evalúa la efectividad, eficiencia, satisfacción, ausencia de riesgos y cobertura del contexto, permitiendo analizar el software desde diversas perspectivas de calidad.

La integración de ambas normas ofrece un marco sólido para proyectos de software: mientras la ISO/IEC/IEEE 29148 guía la construcción y gestión de requisitos, la ISO/IEC 25010 establece parámetros para los requisitos de calidad y la evaluación del producto final. En conjunto, garantizan que el sistema cumpla con lo solicitado por los usuarios y alcance estándares internacionales de calidad, fiabilidad y seguridad, aspectos esenciales en aplicaciones de comercio electrónico.

Con base en este marco metodológico, se presentan los requisitos funcionales y no funcionales del proyecto. Estos elementos constituyen la base para el diseño, desarrollo y validación de la solución propuesta, asegurando su alineación con las necesidades identificadas y con las buenas prácticas de ingeniería de software.

Requerimientos Funcionales

REQ-F01

Tabla 5

Requerimiento Funcional Número 1

#ID Requerimiento	REQ-F01	Tipo de Requerimiento	Funcional
Descripción	El sistema debe permitir la creación de proyectos web mediante una estructura base predefinida (skeleton).		
Justificación	Facilita el inicio rápido de proyectos con una base consistente y estandarizada.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Alta		
Verificación	Revisión de la estructura generada por el skeleton en pruebas locales.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-F02

Tabla 6

Requerimiento Funcional Número 2

#ID Requerimiento	REQ-F02	Tipo de Requerimiento	Funcional
Descripción	Debe ofrecer una interfaz de línea de comandos (CLI) con comandos para scaffolding, generación de controladores, middlewares y modelos.		
Justificación	Mejora la productividad y estandariza el flujo de trabajo del desarrollador.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Alta		
Verificación	Ejecución de comandos CLI y verificación de la correcta creación de archivos.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-F03

Tabla 7

Requerimiento Funcional Número 3

#ID Requerimiento	REQ-F03	Tipo de Requerimiento	Funcional
Descripción	Debe implementar un sistema de enrutamiento que permita definir rutas HTTP y asociarlas a controladores específicos.		
Justificación	Garantiza la correcta gestión de las peticiones HTTP en la aplicación.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Alta		
Verificación	Pruebas de acceso a rutas y respuesta desde controladores asociados.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-F04

Tabla 8

Requerimiento Funcional Número 4

#ID Requerimiento	REQ-F04	Tipo de Requerimiento	Funcional
Descripción	Debe incluir un motor de vistas basado en Blade, con soporte para layouts, componentes y directivas personalizadas.		
Justificación	Permite separar la lógica de presentación y reutilizar estructuras de vistas.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Media		
Verificación	Renderizado de vistas con layouts y componentes personalizados.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-F05

Tabla 9

Requerimiento Funcional Número 5

#ID requerimiento	REQ-F05	Tipo de Requerimiento	Funcional
Descripción	Debe contar con un contenedor de servicios para la inyección de dependencias (DI), permitiendo el registro y resolución de clases por contrato.		
Justificación	Favorece la modularidad, reduce el acoplamiento y facilita las pruebas unitarias.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Alta		
Verificación	Pruebas unitarias de inyección de dependencias y resolución de contratos.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-F06

Tabla 10

Requerimiento Funcional Número 6

#ID requerimiento	REQ-F06	Tipo de Requerimiento	Funcional
Descripción	Debe permitir la configuración del entorno mediante archivos .env, integrando variables de entorno de forma segura.		
Justificación	Garantiza flexibilidad en la configuración sin exponer datos sensibles.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Media		
Verificación	Carga y lectura correcta de variables definidas en archivos .env.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-F07

Tabla 11

Requerimiento Funcional Número 7

#ID requerimiento	REQ-F07	Tipo de Requerimiento	Funcional
Descripción	Debe incluir pruebas unitarias y funcionales para validar el comportamiento de sus componentes principales.		
Justificación	Garantiza la calidad, confiabilidad y mantenibilidad del sistema.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Media		
Verificación	Ejecución de pruebas unitarias y funcionales automatizadas.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

Requerimientos No Funcionales

REQ-NF01

Tabla 12

Requerimiento No Funcional Número 1

#ID requerimiento	REQ-NF01	Tipo de Requerimiento	No funcional
Descripción	Optimización para entornos con recursos limitados		
Justificación	Optimiza el rendimiento y facilita su adopción en entornos con recursos limitados.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Alta		
Verificación	Medición del consumo de memoria y espacio en disco tras la instalación.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-NF02

Tabla 13

Requerimiento No Funcional Número 2

#ID requerimiento	REQ-NF02	Tipo de Requerimiento	No funcional
Descripción	Debe ser modular, permitiendo la sustitución o extensión de componentes sin modificar el núcleo.		
Justificación	Favorece la extensibilidad y facilita la evolución futura del framework.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Alta		
Verificación	Pruebas de sustitución e integración de componentes en el framework.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-NF03

Tabla 14

Requerimiento No Funcional Número 3

#ID requerimiento	REQ-NF03	Tipo de Requerimiento	No funcional
Descripción	Debe ser fácilmente instalable mediante Composer y comandos CLI.		
Justificación	Facilita la adopción del framework y reduce la complejidad de la instalación inicial.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Alta		
Verificación	Ejecución de instalación mediante Composer y validación de comandos CLI.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-NF04

Tabla 15

Requerimiento No Funcional Número 4

#ID requerimiento	REQ-NF04	Tipo de Requerimiento	No funcional
Descripción	Debe garantizar la mantenibilidad del código mediante principios SOLID y desacoplamiento arquitectónico.		
Justificación	Asegura que el sistema pueda evolucionar y mantenerse en el tiempo con bajo costo de cambios.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Alta		
Verificación	Revisión de la arquitectura y cumplimiento de principios SOLID en auditorías de código.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

REQ-NF05

Tabla 16

Requerimiento No Funcional Número 5

#ID requerimiento	REQ-NF05	Tipo de Requerimiento	No funcional
Descripción	Debe ofrecer una experiencia de desarrollo clara y eficiente, reduciendo el tiempo hasta la primera funcionalidad (TTFF).		
Justificación	Incrementa la productividad del desarrollador y mejora la curva de adopción del framework.		
Autor	Daniel Santiago Morales Ariza	Versión	1
Fecha Creación	18/09/2025	Fecha modificación	18/09/2025
Porcentaje Implementación	100%		
Origen	Documento de tesis		
Prioridad	Alta		
Verificación	Medición del tiempo promedio necesario para implementar la primera funcionalidad.		

Nota. Elaboración propia a partir de la definición de requerimientos del sistema.

Arquitectura del Sistema

La arquitectura de Syverum está diseñada bajo un enfoque modular y desacoplado, siguiendo principios de Domain-Driven Design (DDD), Clean Architecture y Arquitectura Hexagonal (Ports & Adapters). Este diseño busca separar de forma estricta las responsabilidades entre el núcleo de negocio (dominio), la lógica de aplicación y la infraestructura tecnológica, garantizando así mantenibilidad, escalabilidad y facilidad de pruebas.

El sistema se construye a partir de un Kernel central que orquesta la ejecución del framework, un contenedor de dependencias (IoC/DI) y un esquema estandarizado para módulos y servicios.

Capas Principales

Propósito: encapsular las reglas del negocio y las entidades centrales.

Contiene:

Entidades (Entities) con sus invariantes.

Objetos de Valor (Value Objects) para representar conceptos inmutables.

Interfaces de Repositorio (Repository Interfaces) como contratos para persistencia, independientes de la infraestructura.

El dominio es completamente independiente de librerías externas, frameworks o bases de datos.

Capa de Aplicación (Application Layer)

Propósito: coordinar los casos de uso del sistema, orquestando la interacción entre el dominio y los adaptadores externos.

Contiene:

Servicios de Aplicación (Use Cases / Application Services) que definen flujos específicos.

Mediadores entre el Kernel y el Dominio, recibiendo comandos y ejecutando lógica a través de los servicios.

Aquí no se implementa lógica de negocio, sino que se asegura la correcta secuencia de interacciones.

Capa de Infraestructura (Infrastructure Layer)

Propósito: proveer implementaciones concretas de interfaces definidas en el dominio o la aplicación.

Contiene:

Repositorios concretos (ej. MySQL, PostgreSQL, Redis).

Servicios externos (ej. APIs como HubSpot, Deepgram).

Controladores HTTP y sistema de enrutamiento.

Motor de vistas (Blade-like) para renderizado de plantillas.

Esta capa puede cambiar sin afectar el dominio ni los casos de uso.

Kernel del Framework

El Kernel es el núcleo central que inicializa y coordina los componentes del sistema. Entre sus principales responsabilidades se encuentran el registro de servicios en el contenedor IoC, la carga de módulos de infraestructura como el enrutamiento, las vistas y la interfaz de línea de comandos (CLI), el manejo del ciclo de vida de una petición HTTP (Request → Middleware → Controller → Response), y la orquestación de los comandos CLI utilizados para el *scaffolding* y las utilidades del desarrollador.

Estilo Arquitectónico Adoptado

La arquitectura responde a una combinación de tres enfoques complementarios: la Arquitectura Hexagonal (Cockburn, 2005), que promueve la separación entre el núcleo y los adaptadores externos; la Clean Architecture (Martin, 2017), que busca la independencia de frameworks y librerías; y el Domain-Driven Design (Evans, 2004), que enfatiza el modelo de dominio y el lenguaje ubicuo. Gracias a esta integración, la lógica de negocio permanece inmutable frente a los cambios tecnológicos, mientras que la infraestructura puede evolucionar libremente.

Diagramas de Arquitectura

Diagrama de Flujo de Ejecución (Request - Application - Response)

El flujo de ejecución en Syverum inicia en el archivo index.php, que actúa como punto de entrada único para todas las solicitudes. En esta fase, se realiza un proceso de *bootstrapping* que incluye la carga de variables de entorno (`.env`) y proveedores de servicios que inicializan dependencias antes de ejecutar la aplicación.

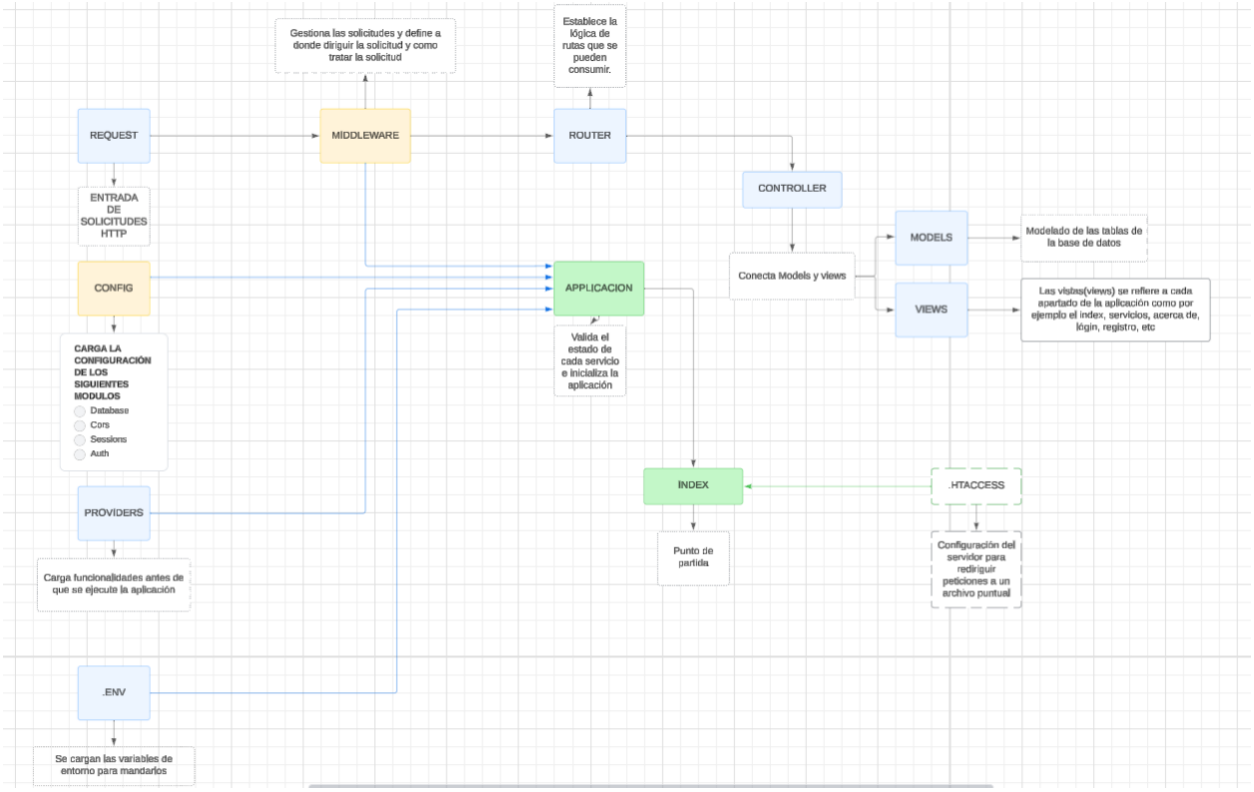
De forma conceptual, se incluye en el diagrama un módulo de `Config`, inspirado en la organización de Laravel. Aunque Syverum aún no incorpora este directorio de configuración, su arquitectura desacoplada hace posible integrar fácilmente archivos de configuración para servicios como base de datos, autenticación o sesiones. Este planteamiento constituye una extensión futura, que mantiene la coherencia con el diseño modular del framework.

Una vez cargada la infraestructura inicial, la solicitud pasa por los Middleware, que validan o transforman datos antes de alcanzar el Router. Este último determina qué controlador atenderá la petición, delegando al Kernel de la aplicación la ejecución del caso de uso

correspondiente. Finalmente, el controlador coordina modelos y vistas para generar una Response, que es enviada al cliente.

Figura 1

Flujo de Ejecución en Syverum de una Solicitud



Nota. El diagrama muestra el flujo de una petición HTTP desde la entrada hasta la respuesta del framework.

Diagrama de Arquitectura General (Domain / Application / Infrastructure)

La arquitectura de Syverum se fundamenta en la separación en capas propuesta por principios como Domain-Driven Design, Arquitectura Hexagonal y Clean Architecture. En la capa Domain se encuentran los elementos fundamentales del negocio: entidades con identidad propia, objetos de valor inmutables, servicios de dominio y contratos abstractos (interfaces de

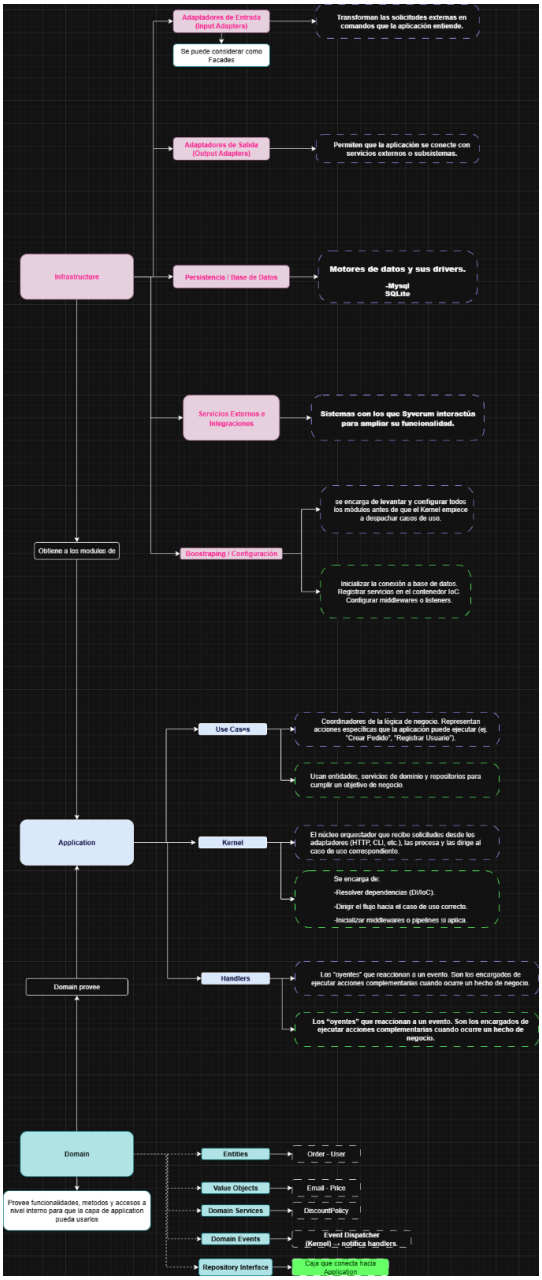
repositorio). Además, en esta capa se definen los eventos de dominio que representan hechos relevantes en la lógica empresarial.

La capa Application actúa como intermediaria entre el dominio y la infraestructura. Contiene los casos de uso, que coordinan las reglas de negocio, y el Kernel, que orquesta el flujo de ejecución. A través de un event dispatcher, los casos de uso publican eventos que son atendidos por handlers, encargados de ejecutar acciones complementarias sin acoplarse directamente al caso de uso.

Finalmente, la capa Infrastructure provee las implementaciones concretas que el dominio y la aplicación requieren. Aquí se encuentran los repositorios que acceden a bases de datos, los adaptadores de entrada (HTTP, CLI) y de salida (servicios externos), así como la configuración de persistencia. Esta capa también gestiona un proceso de bootstrapping, que consiste en inicializar conexiones, registrar servicios en el contenedor IoC y configurar middlewares antes de ejecutar la aplicación.

Figura 2

Diagrama de Arquitectura por Capas de Syverum



Nota. El diagrama representa la arquitectura en capas: Domain (núcleo del negocio), Application (coordinación de casos de uso) e Infrastructure (repositorios, adaptadores y persistencia mediante bootstrapping).

Flujo de Ejecución del Sistema en Syverum

En Syverum, cada petición HTTP sigue un flujo estructurado basado en la separación de responsabilidades. Todo inicia en index.php, donde se ejecuta el proceso de bootstrapping: carga del archivo .env, registro de servicios en el contenedor IoC y activación del Kernel, encargado de coordinar todo el ciclo de la petición.

Luego, la solicitud pasa por los middlewares, que actúan como filtros para validar, transformar o bloquear datos antes de que lleguen al núcleo de la aplicación. Aquí pueden aplicarse procesos como autenticación, validación de cabeceras o protección contra ataques.

El router identifica la ruta correspondiente y determina qué controlador y método atenderán la solicitud, siguiendo la convención ruta → controlador → acción. El controlador, por su parte, no contiene lógica de negocio; su función es invocar un servicio de aplicación, que representa un caso de uso concreto.

Los servicios interactúan con la capa de dominio, donde se encuentran las reglas de negocio, validaciones y entidades. Si se requiere persistencia u operaciones externas, se utilizan repositorios, cuyas implementaciones están en la infraestructura.

Finalizada la ejecución del caso de uso, se genera un objeto Response compatible con PSR-7, que puede devolver JSON, vistas Blade u otros formatos. El Kernel envía esta respuesta al servidor web, cerrando el ciclo.

Este proceso completo se ilustra en el Diagrama de Flujo de Ejecución (Figura 1).

Pruebas del Sistema

Objetivo de las Pruebas

El objetivo de las pruebas de software en Syverum es validar la calidad, confiabilidad y mantenibilidad del framework, asegurando que los requerimientos funcionales (creación de proyectos, CLI, enrutamiento, motor de vistas, IoC, configuración, etc.) se cumplan conforme a lo especificado en el capítulo de requerimientos.

Planificación de Pruebas

Tabla 17

Requerimientos Funcionales para Probar

Ítem	ID Requerimiento	Nombre del Caso de Prueba	Descripción	Prioridad	Responsable
1	REQ-F01	Creación de proyectos	Se verifica que el instalador genere correctamente la estructura base de un proyecto.	Alta	Daniel Morales
2	REQ-F02	Uso de la CLI	Comprueba que los comandos de scaffolding generen controladores, vistas y middlewares de manera correcta.	Alta	Daniel Morales

3	REQ-F03	Sistema de enrutamiento	Se evalúa la definición de rutas HTTP y su correcta asociación a controladores.	Alta	Daniel Morales
4	REQ-F04	Motor de vistas Blade	Se revisa que layouts y componentes se rendericen con éxito.	Media	Daniel Morales
5	REQ-F05	Contenedor de dependencias (IoC/DI)	Se valida la inyección de dependencias y resolución de contratos.	Alta	Daniel Morales
6	REQ-F06	Configuración por .env	Se comprueba que las variables de entorno se carguen y utilicen adecuadamente.	Media	Daniel Morales
7	REQ-F07	Pruebas unitarias y funcionales	Se verifica que se ejecuten y pasen los tests incluidos.	Media	Daniel Morales

Nota. La tabla presenta los requerimientos funcionales que serán verificados durante la fase de pruebas del sistema. Se incluyen el identificador del requerimiento, el caso de prueba correspondiente, su descripción, prioridad y responsable.

Desarrollo de Casos de Prueba

Caso de Prueba 1 -REQ-F01

Tabla 18

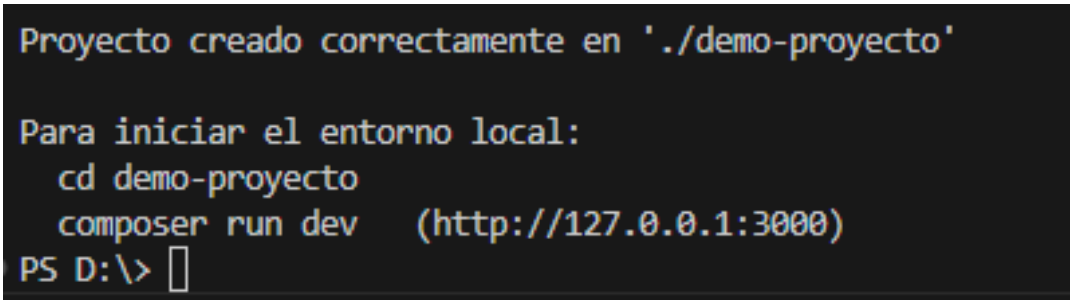
Caso de Prueba REQ-F01: Creación de Proyectos Web

Caso de Prueba 1 -REQ-F01	
Nombre	Creación de proyectos web
Precondiciones	Tener instalado syverum/installer globalmente con Composer.
Pasos de la Prueba	1. Ejecutar syverum new demo-proyecto. 2. Acceder al directorio generado y verificar estructura (routes/, app/, views/, public/).
Resultado Esperado	La estructura se crea completa y lista para ejecutar composer run dev
Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	El framework facilitó el desarrollo del proyecto, lo que permitió la creación de un proyecto denominado demo-proyecto, como se evidencia en la siguiente figura.

Nota. Corresponde al Caso de prueba 1 (REQ-F01) sobre la creación de proyectos web mediante skeleton en Syverum. Elaboración propia a partir de la ejecución práctica realizada el 29/09/2025.

Figura 3

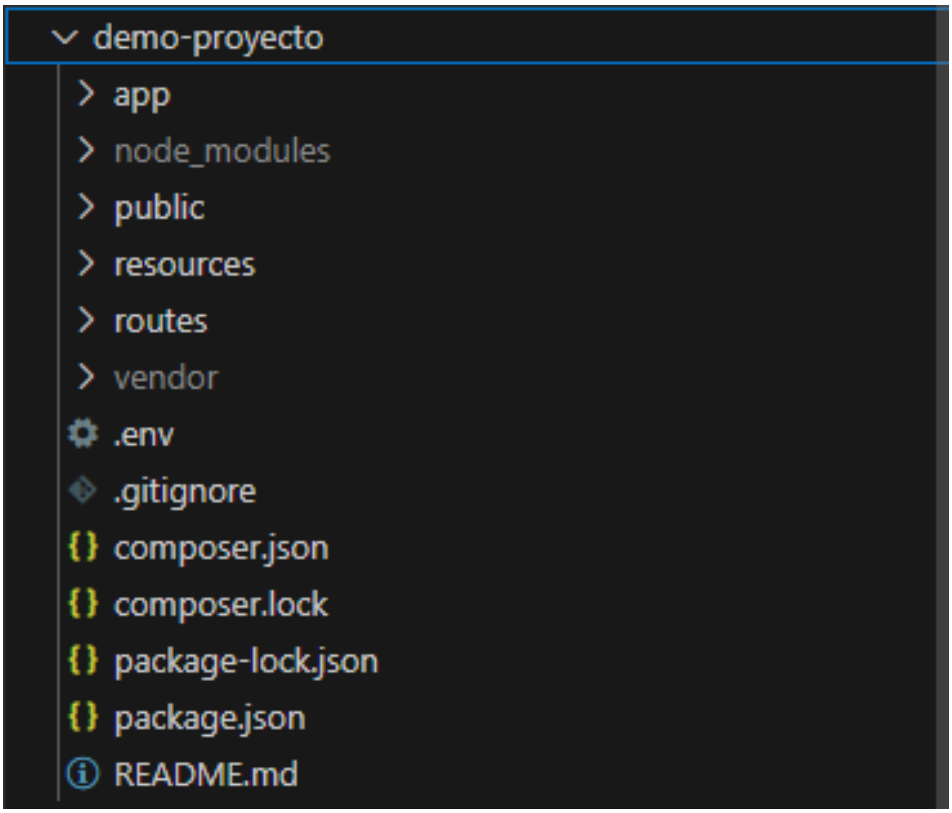
Ejecución de Comando para Crear un Nuevo Proyecto en Syverum



Nota. Paso 1; testeo caso de prueba 1. Captura elaborada por el autor.

Figura 4

Estructura Inicial de Archivos y Directorios Generada por el Skeleton de Syverum



Nota. Paso 2; testeo caso de prueba 1. Captura elaborada por el autor.

Caso de Prueba 2 -REQ-F02

Tabla 19

Caso de Prueba REQ-F02: Uso de la CLI

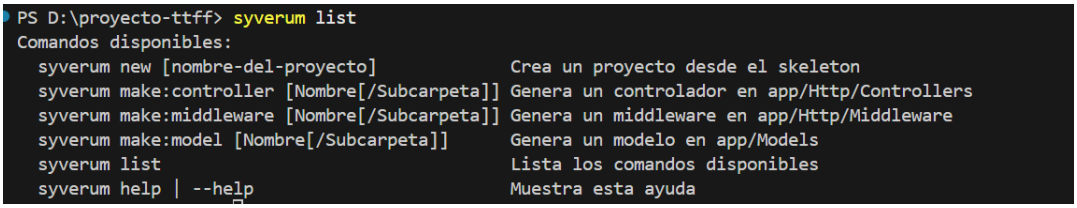
Caso de Prueba 2 -REQ-F02	
Nombre	Uso de la CLI.
Precondiciones	1. PHP 8.x y Composer instalados. 2. Proyecto Syverum existente (skeleton). 3. CLI disponible vía vendor/bin/syverum (o global si aplica).
Pasos de la Prueba	1. Ejecutar php vendor/bin/syverum list y verificar que muestre los comandos disponibles. 2. Ejecutar php vendor/bin/syverum make:controller HomeController y verificar archivo en app/Controllers/HomeController.php con namespace correcto. 3. Ejecutar php vendor/bin/syverum make:model Post y verificar archivo en app/Models/Post.php con propiedades base. 5. Ejecutar php vendor/bin/syverum make:middleware AuthMiddleware y

	verificar archivo en app/Middleware/AuthMiddleware.php.
Resultado Esperado	La estructura se crea completa y lista para ejecutar composer run dev
Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	El comando list muestra la ayuda general y la lista de comandos de scaffolding (controller, model, middleware, etc.). - Cada comando make:* crea el archivo en la ruta correcta con plantilla base y namespace App\.... - No se reportan errores de PHP ni Composer; los archivos son autoloadables (PSR-4).

Nota. Corresponde al Caso de prueba 2 (REQ-F02) sobre el uso de la interfaz de línea de comandos de Syverum para generar artefactos de aplicación (scaffolding).

Figura 5

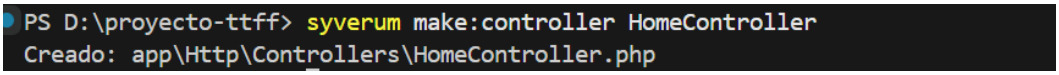
Salida del Comando Syverum List (CLI de Syverum)



Nota. Paso 1; testeo caso de prueba 2. Captura elaborada por el autor.

Figura 6

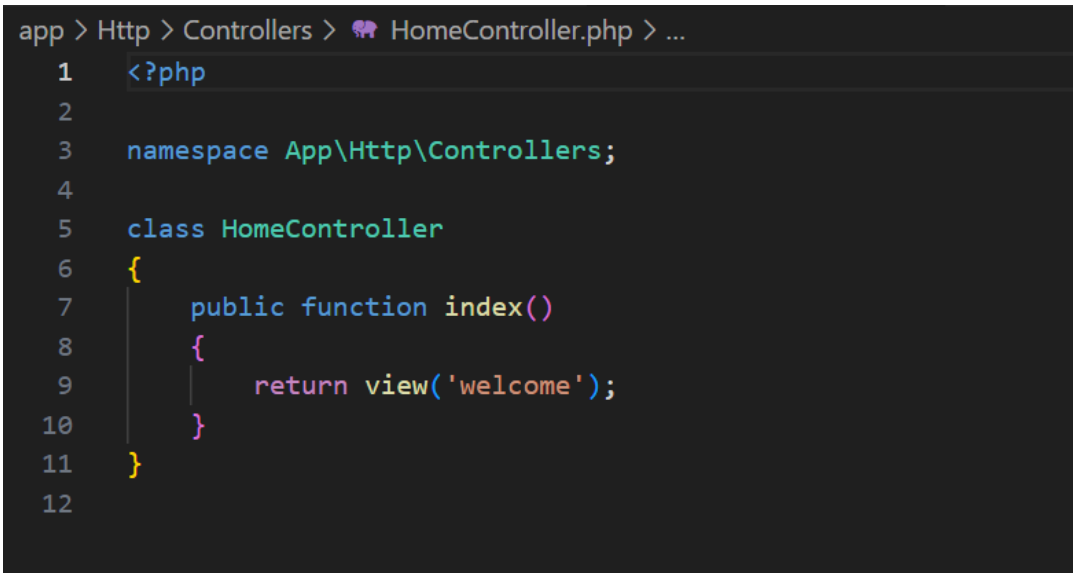
Generación de Controlador con syverum make:controller



Nota. Paso 2; testeo caso de prueba 2. Captura elaborada por el autor.

Figura 7

Confirmación de Archivo Creado en app\Http\Controllers



Nota. Paso 3; testeo caso de prueba 2. Captura elaborada por el autor.

Figura 8

Generación de Modelo con syverum make:model

```
PS D:\proyecto-ttff> syverum make:model Post
Creado: app\Models\Post.php
```

Nota. Paso 4; testeo caso de prueba 2. Captura elaborada por el autor.

Figura 9

Plantilla Base del Modelo Generado en app\Models\Post.php

app > Models > Post.php > ...

```php
<?php

namespace App\Models;

use Core\Support\Database\Model;

class Post extends Model
{
    /**
     * Si es null, Syverum resuelve la tabla por convención:
     * Post -> snake_case y plural (p.ej. Post -> posts)
     */
    protected ?string $table = null;

    /**
     * Clave primaria (por defecto 'id').
     */
    protected string $primaryKey = 'id';

    /**
     * Indica si la PK es autoincremental.
     */
    protected bool $incrementing = true;
```

Nota. Paso 5; testeo caso de prueba 2. Captura elaborada por el autor.

Figura 10

Generación de Middleware con syverum make:middleware

```
PS D:\proyecto-ttff> syverum make:middleware AuthMiddleware
Creado: app\Http\Middleware\AuthMiddleware.php
```

Nota. Paso 6; testeo caso de prueba 2. Captura elaborada por el autor.

Figura 11

Plantilla Base del Middleware Generado (AuthMiddleware.php)

app > Http > Middleware > AuthMiddleware.php > ...

```php
<?php

namespace App\Http\Middleware;

use Core\Support\Http\Request;
use Core\Support\Http\Response;
use Core\Support\Middleware\Contracts\MiddlewareInterface;

final class AuthMiddleware implements MiddlewareInterface
{
    public function __construct(private Request $request) {}

    public function process(callable $next): mixed
    {
        if (!$this->estaAutenticado()) {
            return Response::text('Unauthorized', 401);
        }
        return $next();
    }

    private function estaAutenticado(): bool
    {
        /* tu lógica */
        return false;
    }
}

```

Nota. Paso 7; testeo caso de prueba 2. Captura elaborada por el autor.

Para validar el REQ-F02 se ejecutaron todos los comandos de la CLI de Syverum: syverum list, make:controller HomeController, make:model Post y make:middleware AuthMiddleware. En cada caso se verificó la creación de artefactos en las rutas previstas (app/Http/Controllers, app/Models, app/Http/Middleware), con *namespaces* PSR-4 y *stubs* listos para su ajuste. No se registraron errores y los archivos fueron reconocidos por el *autoload* de Composer. Desde las figuras 5 hasta la 11 evidencian el cumplimiento del requerimiento.

Caso de Prueba 3 -REQ-F03

Tabla 20

Caso de Prueba REQ-F03: Resolución de Rutas HTTP y Despacho a Controladores

Caso de Prueba 3 -REQ-F03	
Nombre	Resolución de rutas HTTP y despacho a controladores
Precondiciones	1. Proyecto Syverum operativo (skeleton instalado). 2. Servidor en desarrollo con composer run dev (o php -S 127.0.0.1:3000 -t public). 3. Archivo de rutas disponible en routes/web.php.
Pasos de la Prueba	1. Ruta GET simple: Route::get('/', fn() => 'OK'); y visitar /. 2. Ruta con parámetro: Route::get('/hello/{name}', fn($name) => "Hello $name"); y visitar /hello/Daniel. 3. Ruta a controlador: Router::get('/home', [HomeController::class, 'index']); (retornar texto o vista). 4) Verificaciones rápidas: hacer POST /items con Router::post('/items', fn() => Response::json(['created'=>true])); y solicitar /no-existe para 404.

Resultado Esperado	El sistema debe resolver correctamente las rutas definidas en routes/web.php, garantizando que cada una responda con el contenido esperado. La ruta / debe retornar el texto “OK”, mientras que /hello/{name} debe responder con un saludo que incluya el parámetro dinámico, por ejemplo, “Hello Daniel”. La ruta /home debe despachar la solicitud al controlador HomeController@index, retornando el texto o la vista definida en dicho método. Por su parte, la ruta /items debe aceptar peticiones POST y responder en formato JSON con el contenido {"created": true}. Finalmente, cualquier ruta que no exista en el archivo de definición debe devolver un error 404 Not Found.
Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	Durante la ejecución de las pruebas en el servidor local (php -S 127.0.0.1:3000 -t public), todas las rutas respondieron de acuerdo con lo esperado: • La ruta raíz (/) devolvió OK. • La ruta con parámetro (/hello/Daniel) devolvió Hello Daniel. • La ruta a controlador (/home) mostró el contenido proveniente de HomeController@index. • La petición POST a /items devolvió el JSON {"created": true}. • La consulta a /no-existe devolvió el estado 404 Not Found.

Nota. Resume los resultados de la validación del sistema de enrutamiento del framework Syverum, comprobando la resolución correcta de rutas y respuestas.

Figura 12

Definición de Ruta GET Simple en routes/web.php

```php
// 1. Ruta GET simple
Route::get('/', fn() => 'OK');
```

Nota. Paso 1; testeo caso de prueba 3. Captura elaborada por el autor.

Figura 13

Respuesta del Navegador al Acceder a la Ruta /

Nota. Paso 2; testeo caso de prueba 3. Captura elaborada por el autor.

Figura 14

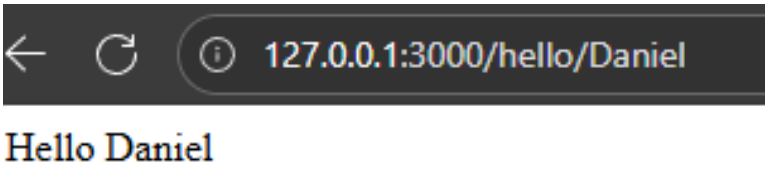
Definición de Ruta con Parámetro Dinámico en routes/web.php

```php
// 2. Ruta con parámetro
Route::get('/hello/{name}', fn($name) => "Hello $name");
```

Nota. Paso 3; testeo caso de prueba 3. Captura elaborada por el autor.

Figura 15

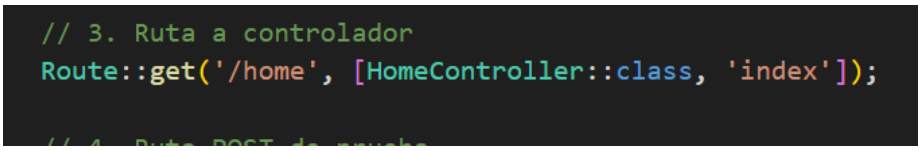
Respuesta del Navegador al Acceder a la Ruta /hello/Daniel



Nota. Paso 4; testeo caso de prueba 3. Captura elaborada por el autor.

Figura 16

Definición de la Ruta /home Asociada al Controlador HomeController@index en routes/web.php



```php
// 3. Ruta a controlador
Route::get('/home', [HomeController::class, 'index']);
```

Nota. Paso 4; testeo caso de prueba 3. Captura elaborada por el autor.

Figura 17

Implementación del Método Index en HomeController para Retornar la Vista

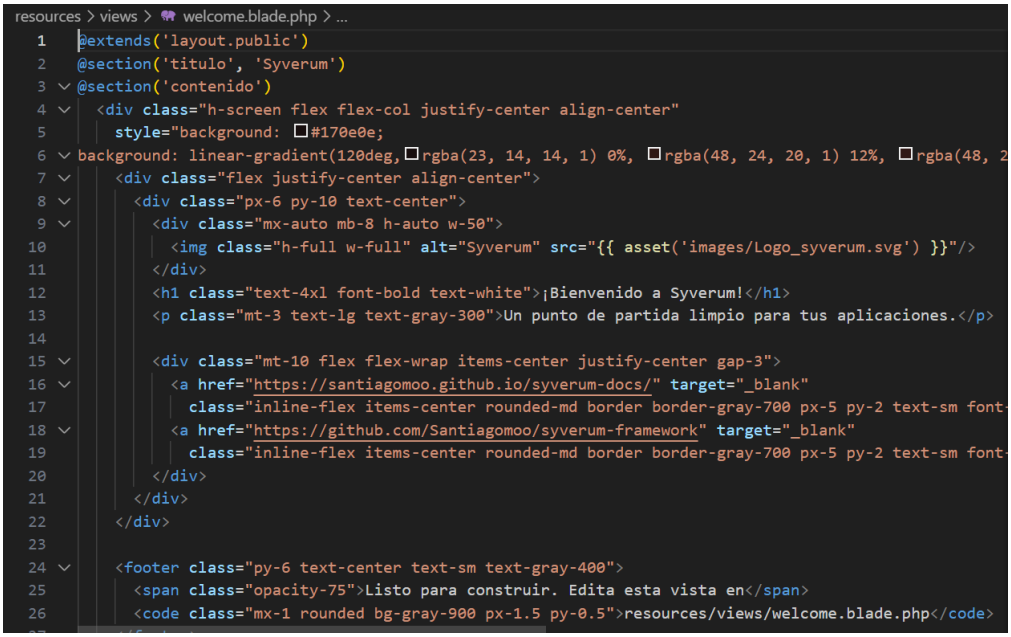
app > Http > Controllers > HomeController.php > ...

```php
<?php

namespace App\Http\Controllers;

class HomeController
{
    public function index()
    {
        return view('welcome');
    }
}
```

Nota. Paso 5; testeo caso de prueba 3. Captura elaborada por el autor.

Figura 18

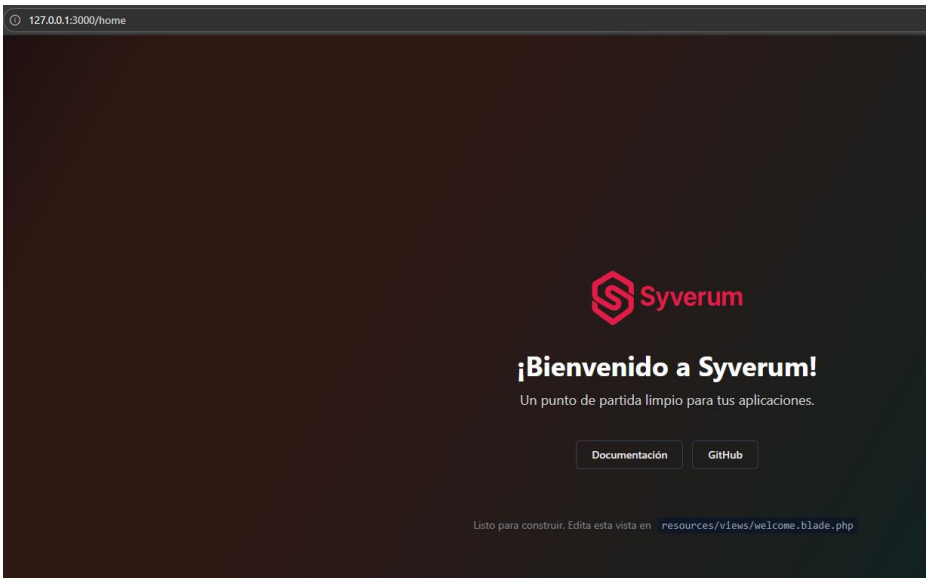
Definición de la Vista welcome.blade.php en el Directorio resources/views



Nota. Paso 6; testeo caso de prueba 3. Captura elaborada por el autor.

Figura 19

Respuesta en el Navegador al Acceder a la Ruta /home



Nota. Paso 7; testeo caso de prueba 3. Captura elaborada por el autor.

Figura 20

Definición de la Ruta POST /items que Retorna una Respuesta en Formato JSON

```
// 4. Ruta POST de prueba
Route::post('/items', fn() => Response::json(['created' => true]));
```

Nota. Paso 8; testeo caso de prueba 3. Captura elaborada por el autor.

Figura 21

Resultado al Intentar Acceder Vía GET a la Ruta /items, Mostrando que Solo Admite Método POST

Nota. Paso 9; testeo caso de prueba 3. Captura elaborada por el autor.

Es importante destacar que, aunque el comportamiento del enrutador cumple con lo esperado al rechazar las rutas no definidas, la presentación del error carece de un formato amigable para el usuario final. Actualmente se expone un mensaje de error crudo (stack trace), lo cual es adecuado en entornos de desarrollo, pero no recomendable en producción. Como trabajo futuro, se plantea la inclusión de un manejador centralizado de errores que devuelva vistas prediseñadas (ej. resources/views/errors/404.blade.php y resources/views/errors/500.blade.php), similar a lo implementado en otros frameworks como Laravel. Esto mejoraría la experiencia del usuario final y mantendría la coherencia estética de la aplicación.

Caso de Prueba 4 -REQ-F04

Tabla 21

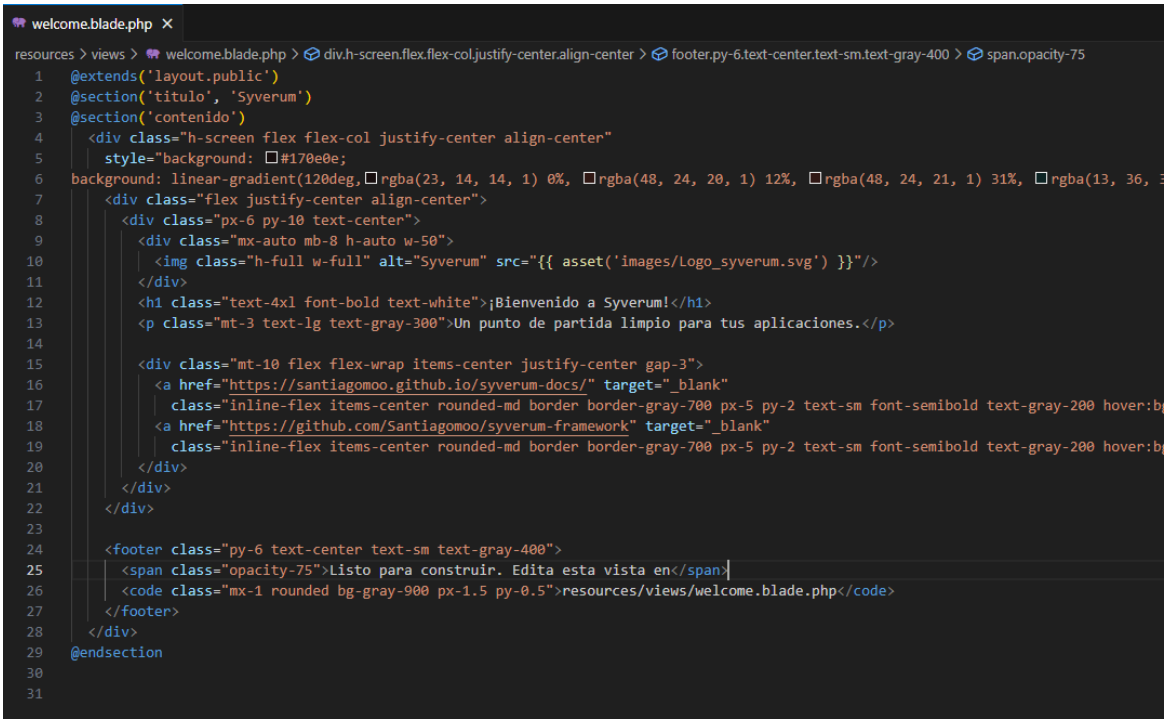
Caso de Prueba REQ-F04: Validación del Motor de Vistas Blade

Caso de Prueba 4 -REQ-F04	
Nombre	Motor de vistas Blade
Precondiciones	Proyecto Syverum creado y servidor en ejecución (composer run dev).
Pasos de la Prueba	1. Crear un archivo welcome.blade.php en resources/views/ 2. Añadir variables y layouts dentro de la vista. 3. Acceder desde el navegador a la ruta raíz
Resultado Esperado	La vista se renderiza correctamente mostrando el contenido dinámico definido.
Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	El framework procesó correctamente la plantilla Blade, mostrando contenido dinámico en la página inicial, según lo configurado en la vista welcome.blade.php.

Nota. Resume la validación del motor de plantillas Blade en Syverum, comprobando la correcta renderización de vistas dinámicas y la separación entre lógica y presentación.

Figura 22

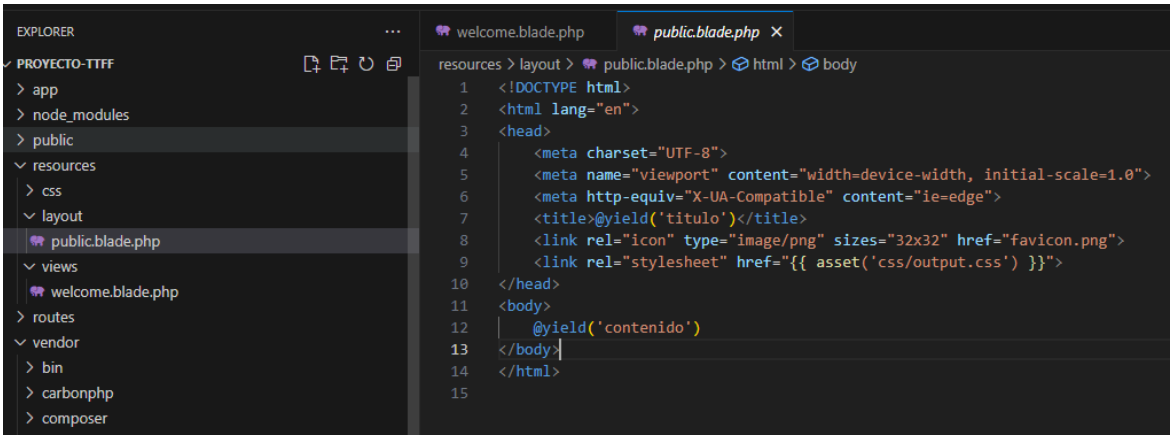
Vista welcome.blade.php con Directivas de Blade en Syverum



Nota. Paso 1; testeo caso de prueba 4. Captura de pantalla propia que muestra el uso de las directivas @extends, @section y {{ }} de Blade en la vista inicial welcome.blade.php, evidenciando que el motor de plantillas está integrado

Figura 23

Layout Base public.blade.php con Secciones Dinámicas



Nota. Paso 2: testeo del caso de prueba 4. Captura elaborada por el autor. El archivo *public.blade.php* funciona como plantilla base del proyecto, definiendo la estructura HTML y usando directivas como @yield('contenido') y @yield('titulo') para inyectar contenido dinámico en las vistas.

Al analizar la estructura generada por defecto en un proyecto creado con Syverum, se evidencia que el framework incluye el motor de vistas Blade de manera nativa. Esto se observa en el archivo welcome.blade.php (Figura 22), donde se emplean directivas propias de Blade tales como @extends y @section para la definición de plantillas y secciones reutilizables. Asimismo, la interpolación de variables mediante la sintaxis {{ }} confirma que el motor procesa dinámicamente los datos enviados desde los controladores.

De forma complementaria, el archivo public.blade.php funciona como layout principal (Figura 23), en el cual se definen secciones dinámicas con @yield('titulo') y @yield('contenido'). Esta separación de responsabilidades es característica del motor Blade y permite centralizar los elementos comunes de la interfaz, tales como encabezados, scripts y estilos, mientras cada vista específica hereda y define únicamente su contenido particular.

Caso de Prueba 5 -REQ-F05

Tabla 22

Caso de Prueba REQ-F05: Inyección de Dependencias (IoC/DI)

Caso de Prueba 5 -REQ-F05	
Nombre	Contenedor de dependencias IoC/DI
Precondiciones	Proyecto inicializado con Syverum.
Pasos de la Prueba	1. Registrar un servicio en el contenedor (App\Services\Logger). 2. Solicitar la resolución automática en un controlador.
Resultado Esperado	El contenedor inyecta correctamente la implementación del contrato.
Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	El contenedor resolvió correctamente la dependencia registrada (App\Services\Logger) e inyectó la implementación en el controlador sin errores, cumpliendo con el comportamiento esperado.

Nota. Presenta los resultados obtenidos al validar el comportamiento del contenedor de dependencias del framework Syverum. Se evaluó la capacidad del sistema para registrar e inyectar automáticamente servicios en los controladores, conforme a los principios de Inversión de Control (IoC) y de Inyección de Dependencias (DI).

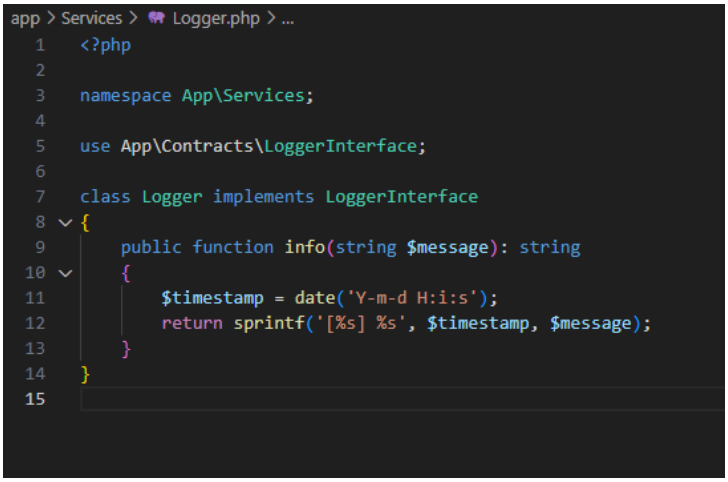
Figura 24

Ejecución del Comando Composer Run Dev e Inicio del Servidor de Desarrollo en Syverum

Nota. Paso 1; testeo caso de prueba 5. Captura de pantalla propia que evidencia la correcta inicialización del servidor local (PHP 8.2.12 + TailwindCSS) en el puerto 3000 (2025).

Figura 25

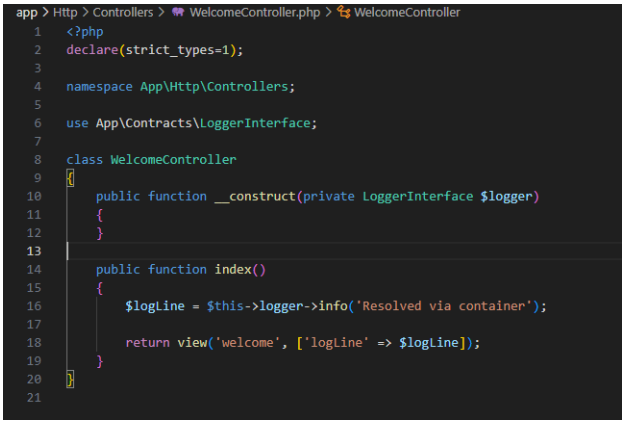
Implementación de un Servicio Logger en la Carpeta Services



Nota. Paso 2; testeo caso de prueba 5. Captura de pantalla propia de la clase Logger implementando la interfaz LoggerInterface en Syverum (2025).

Figura 26

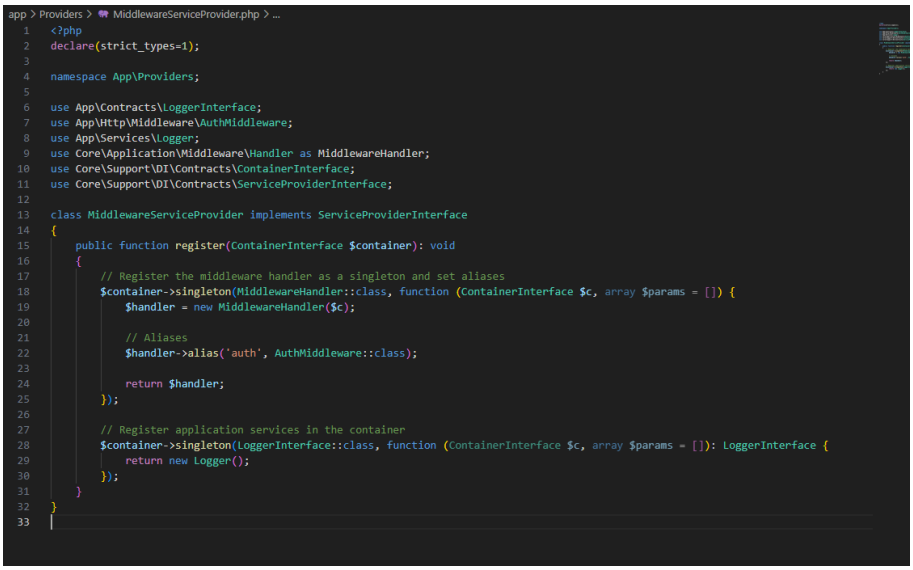
Uso de LoggerInterface en el Controlador WelcomeController



Nota. Paso 3; testeo caso de prueba 5. Captura de pantalla propia del controlador WelcomeController con inyección de dependencias mediante el contenedor de Syverum (2025).

Figura 27

Registro de Servicios en el Contenedor Mediante MiddlewareServiceProvider



Nota. Paso 4; testeo caso de prueba 5. Captura de pantalla propia del registro de LoggerInterface en el contenedor IoC de Syverum (2025).

Figura 28

Resultado de la Inyección de Dependencias con LoggerInterface en Syverum

Nota. Paso 5; testeo caso de prueba 5. Captura de pantalla propia que muestra la correcta resolución e inyección del servicio Logger mediante el contenedor IoC, evidenciada en la vista inicial del framework (2025).

Caso de Prueba 6 -REQ-F06

Tabla 23

Caso de Prueba REQ-F06: Configuración del Entorno Mediante Archivo .env

Caso de Prueba 5 -REQ-F05	
Nombre	Contenedor de dependencias IoC/DI
Precondiciones	1. Proyecto Syverum creado y operativo. 2. Archivo .env disponible en la raíz del proyecto. 3. Librería vlucas/phpdotenv instalada (ya incluida en composer.json).
Pasos de la Prueba	Acceder a una ruta de prueba que devuelva el valor de una variable (ej. APP_NAME) para confirmar su lectura.
Resultado Esperado	El framework debe cargar las variables definidas en el archivo .env y permitir su uso en cualquier parte de la aplicación. Al acceder a /env-test debe mostrarse el valor configurado en APP_NAME.
Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	El framework procesó correctamente el archivo .env, cargando las variables definidas y permitiendo su uso en controladores, rutas y

	vistas. La prueba con la ruta /env-test devolvió el valor configurado en la variable APP_NAME, confirmando la correcta integración del sistema de configuración de entorno.

Nota. Presenta los resultados obtenidos al validar el uso de archivos .env en el framework Syverum. Se comprobó la correcta carga de variables de entorno mediante la librería *phpdotenv*, garantizando flexibilidad en la configuración de la aplicación sin exponer datos sensibles en el código fuente.

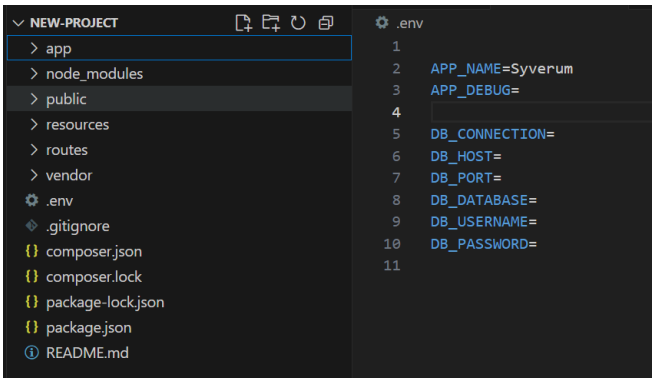
Figura 29

Definición de una Ruta de Prueba que Retorna el Valor de la Variable de Entorno APP_NAME

Nota. Paso 1; testeo caso de prueba 6. Captura elaborada por el autor.

Figura 30

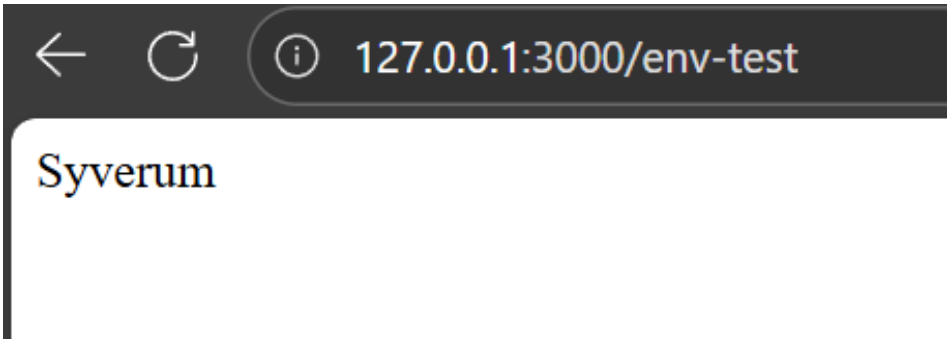
Contenido del Archivo .env Configurado en el Proyecto Syverum



Nota. Paso 2 testeo caso de prueba 6. Captura elaborada por el autor.

Figura 31

Resultado en el Navegador al Acceder a la Ruta /env-test Mostrando el Valor de APP_NAME



Nota. Paso 3; testeo caso de prueba 6. Captura elaborada por el autor.

Caso de Prueba 7 -REQ-F07

Tabla 24

Caso de Prueba REQ-F07: Pruebas Unitarias y Funcionales

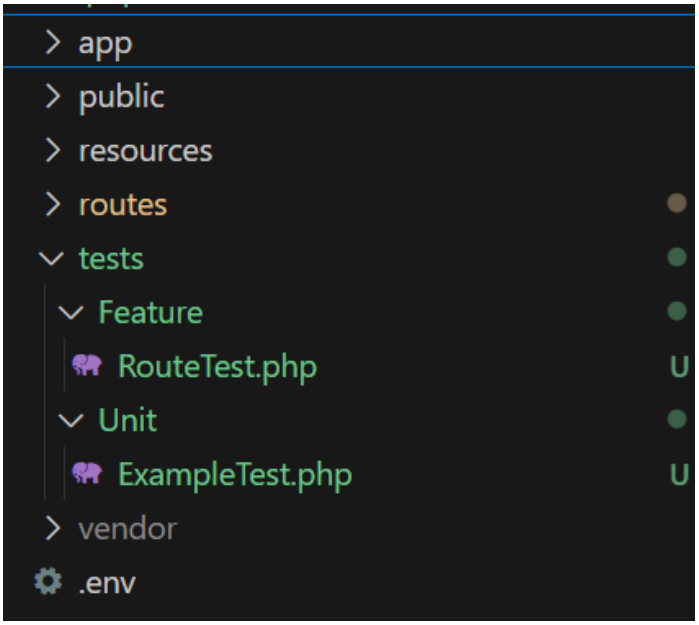
Caso de Prueba 5 -REQ-F05	
Nombre	Pruebas unitarias y funcionales en Syverum
Precondiciones	1. Proyecto Syverum creado y operativo. 2. Dependencia de desarrollo instalada: phpunit/phpunit (o pestphp/pest). 3. Carpeta tests/ configurada en el proyecto.
Pasos de la Prueba	1. Crear una prueba unitaria en tests/Unit/ExampleTest.php para validar una función simple. 2. Crear una prueba funcional en tests/Feature/RouteTest.php para verificar que la ruta / devuelve la respuesta esperada. 3. Ejecutar las pruebas 4. Registrar los resultados obtenidos en consola.
Resultado Esperado	Las pruebas unitarias y funcionales deben ejecutarse correctamente, confirmando el comportamiento esperado de los componentes del framework y la resolución de rutas.
Fecha de Inicio de la Prueba	29/09/2025

Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	Se ejecutaron con éxito las pruebas unitarias y funcionales en el proyecto Syverum utilizando PHPUnit. La prueba unitaria confirmó el funcionamiento de una operación básica y la prueba funcional validó la correcta respuesta de la ruta /, lo que evidencia la posibilidad de integrar un sistema de pruebas automatizadas en el framework.

Nota. Resume la validación de pruebas unitarias y funcionales en Syverum, comprobando la integración de PHPUnit para asegurar la calidad y confiabilidad del sistema.

Figura 32

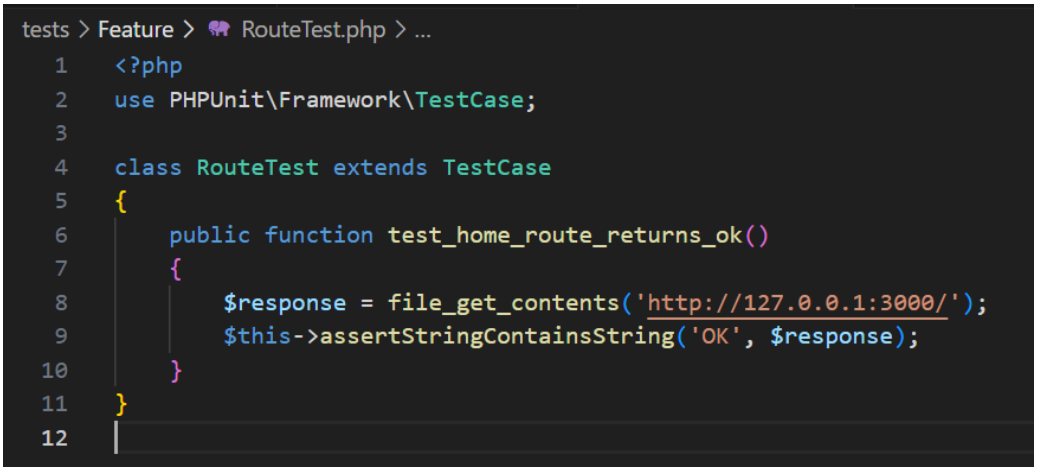
Estructura de Directorios del Proyecto Syverum con Integración de PHPUnit y Carpeta /tests



Nota. Paso 1; testeo caso de prueba 7. Captura elaborada por el autor.

Figura 33

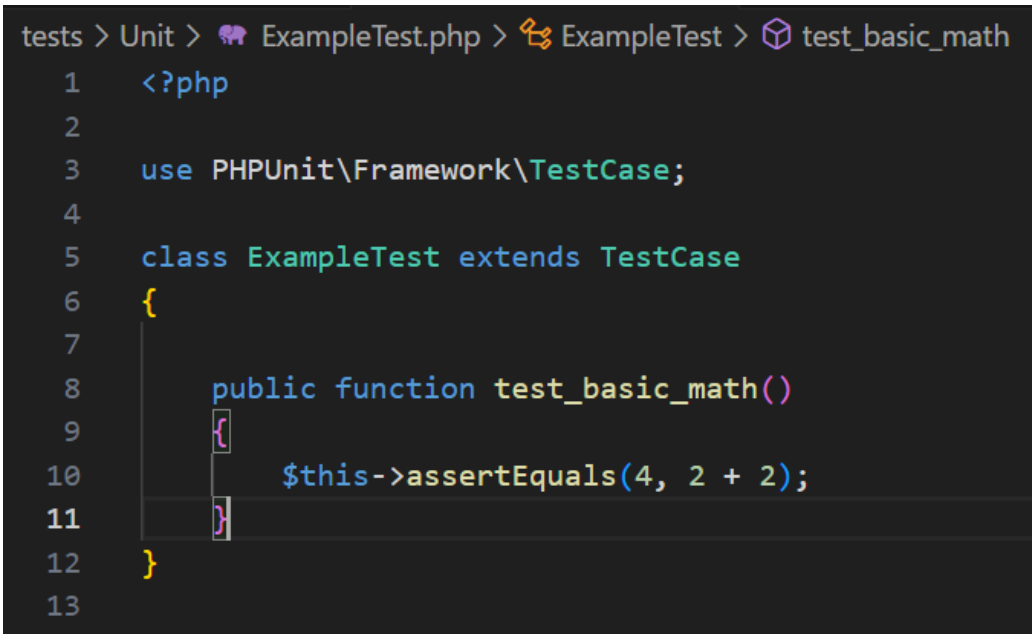
Prueba Funcional en RouteTest.php, Validando la Respuesta de la Ruta raíz /



Nota. Paso 2; testeo caso de prueba 7. Captura elaborada por el autor.

Figura 34

Prueba Unitaria en ExampleTest.php, Verificando una Operación Aritmética Básica.



Nota. Paso 3; testeo caso de prueba 7. Captura elaborada por el autor

Figura 35

Definición de Ruta Simple en routes/web.php para Devolver la Cadena "OK".

```php
Route::get('/', fn() => 'OK');
```

Nota. Paso 4; testeo caso de prueba 7. Captura elaborada por el autor.

Figura 36

Ejecución de PHPUnit Mostrando Resultados Exitosos de las Pruebas Unitarias y Funcionales.

```
PS D:\syverum-skeleton> composer test
PHPUnit 11.5.42 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12
Configuration: D:\syverum-skeleton\phpunit.xml

..                                                                  2 / 2 (100%)

Time: 00:00.060, Memory: 8.00 MB

Example
 ✓ Basic math

Route
 ✓ Home route returns ok

OK (2 tests, 2 assertions)
```

Nota. Paso 5; testeo caso de prueba 7. Captura elaborada por el autor.

Caso de Prueba 8 -REQ-NF01

Tabla 25

Caso de Prueba REQ-NF01: Optimización para Entornos con Recursos Limitados

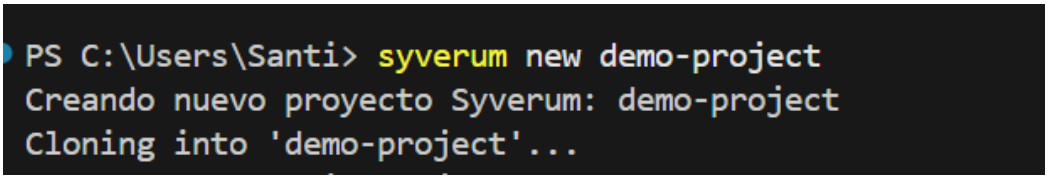
Caso de Prueba 8 -REQ-NF01	
Nombre	Medición de consumo de recursos del framework.
Precondiciones	1. Tener un entorno limpio con PHP 8.x y Composer. 2. No tener instalado previamente Syverum (para medir la instalación desde cero). 3. Herramientas de medición disponibles (comando time, du -sh, top o equivalente en Windows/Linux).
Pasos de la Prueba	1. Instalar un proyecto con syverum new demo-proyecto. 2. Medir el espacio ocupado por el directorio (du -sh demo-proyecto). 3. Ejecutar composer run dev y, en paralelo, monitorear el consumo de memoria del proceso PHP (top, htop, o el Administrador de Tareas en Windows).
Resultado Esperado	- El tamaño del proyecto no supera un umbral razonable (ejemplo: <100 MB). - El consumo de memoria en ejecución se mantiene bajo (ejemplo: <80 MB en pruebas locales). - El framework se considera ligero y viable para entornos con recursos limitados.

Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	Durante la ejecución del proyecto demo-project, los procesos asociados a Syverum registraron un consumo aproximado de 30 MB de memoria para PHP y 60 MB para Node.js. En conjunto, el framework requirió menos de 100 MB en ejecución, lo cual, junto al espacio en disco ocupado de 29.97 MB, confirma que cumple con el requerimiento no funcional REQ-NF01 sobre eficiencia en el uso de recursos.

Nota. Corresponde al caso de prueba del requerimiento no funcional REQ-NF01. Elaboración propia a partir de la ejecución práctica realizada el 29/09/2025.

Figura 37

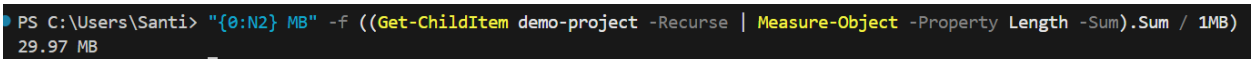
Creación de un Nuevo Proyecto Syverum Mediante el Comando syverum new demo-project



Nota. Paso 1; testeo caso de prueba 8. Captura elaborada por el autor.

Figura 38

Medición del Espacio en Disco Ocupado por el Proyecto Syverum



Nota. Paso 2; testeo caso de prueba 8. Captura elaborada por el autor.

Figura 39

Ejecución del Proyecto Syverum con el Comando composer run dev

Nota. Paso 3; testeo caso de prueba 8. Captura elaborada por el autor.

Figura 40

Consumo de Memoria de los Procesos PHP y Node.js Durante la Ejecución del Proyecto Syverum.

Nota. Paso 4; testeo caso de prueba 8. Captura elaborada por el autor.

Caso de Prueba 9 -REQ-NF02

Tabla 26

Caso de Prueba REQ-NF02: Modularidad y Extensibilidad del Framework

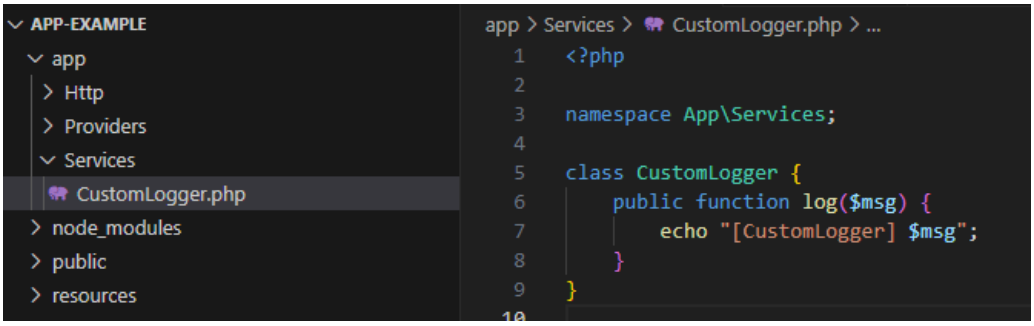
Caso de Prueba 9 -REQ-NF02	
Nombre	Modularidad y extensibilidad del framework.
Precondiciones	Proyecto inicializado con Syverum. Se cuenta con un módulo base cargado en el framework.
Pasos de la Prueba	1. Implementar un nuevo servicio CustomLogger en app/Services. 2. Registrar el CustomLogger en el contenedor de dependencias desde MiddlewareServiceProvider. 3. Usar el CustomLogger en el HomeController para verificar su inyección automática. 4. Eliminar el módulo de middleware por defecto (AuthMiddleware) y sus referencias en el MiddlewareServiceProvider. 5. Ejecutar nuevamente el framework y validar que funciona con el nuevo logger y sin el middleware eliminado.
Resultado Esperado	El framework permite añadir nuevos componentes (ej. CustomLogger) y eliminar módulos existentes (ej. AuthMiddleware) sin errores de ejecución ni necesidad de modificar el núcleo del framework.

Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	Se integró correctamente el servicio CustomLogger, demostrando la extensibilidad del sistema. Posteriormente, se eliminó el módulo AuthMiddleware y sus referencias, confirmando que el framework siguió funcionando sin errores. Esto evidencia la modularidad y la capacidad de evolución del framework.

Nota. Se validó la capacidad del framework Syverum para admitir la sustitución y extensión de componentes sin necesidad de alterar el núcleo, lo que demuestra su carácter modular y favorece la extensibilidad y evolución futura del sistema.

Figura 41

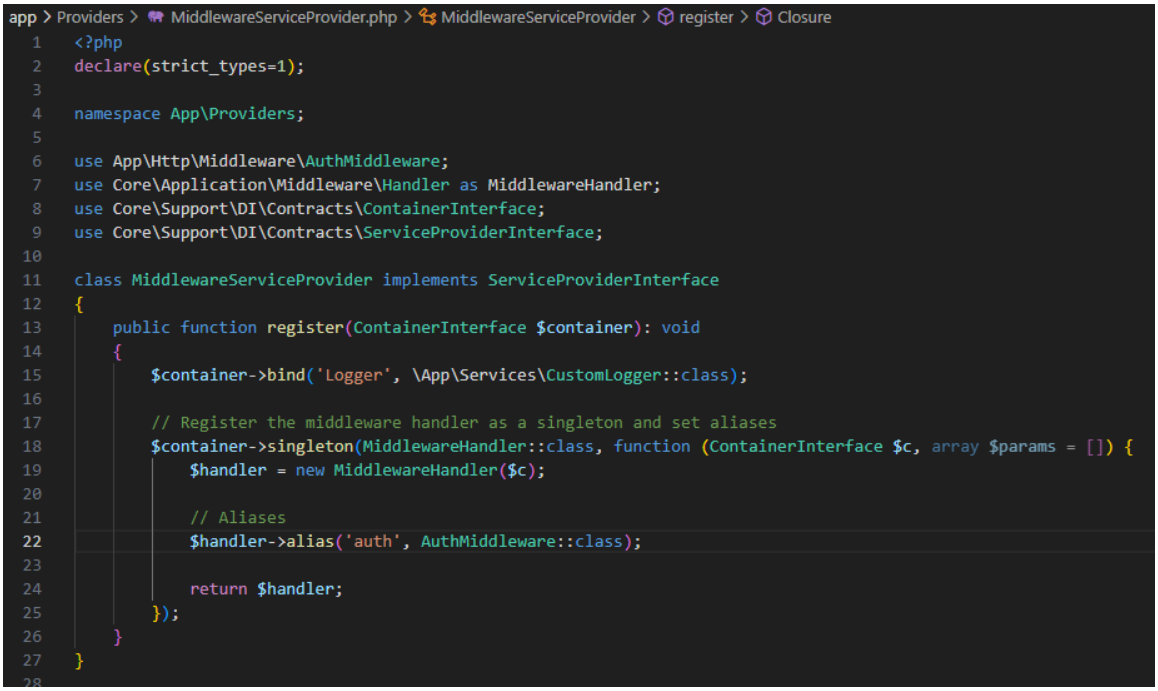
Implementación del Servicio CustomLogger en la Carpeta app/Services



Nota. Paso 1; testeo caso de prueba 9. Captura elaborada por el autor

Figura 42

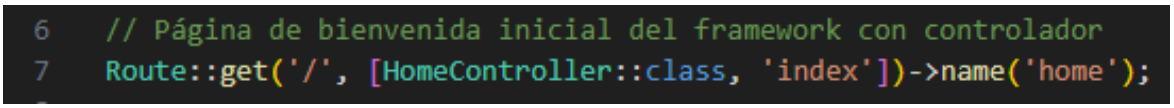
Registro del Servicio CustomLogger en el Contenedor de Dependencias desde MiddlewareServiceProvider.



Nota. Paso 2; testeo caso de prueba 9. Captura elaborada por el autor.

Figura 43

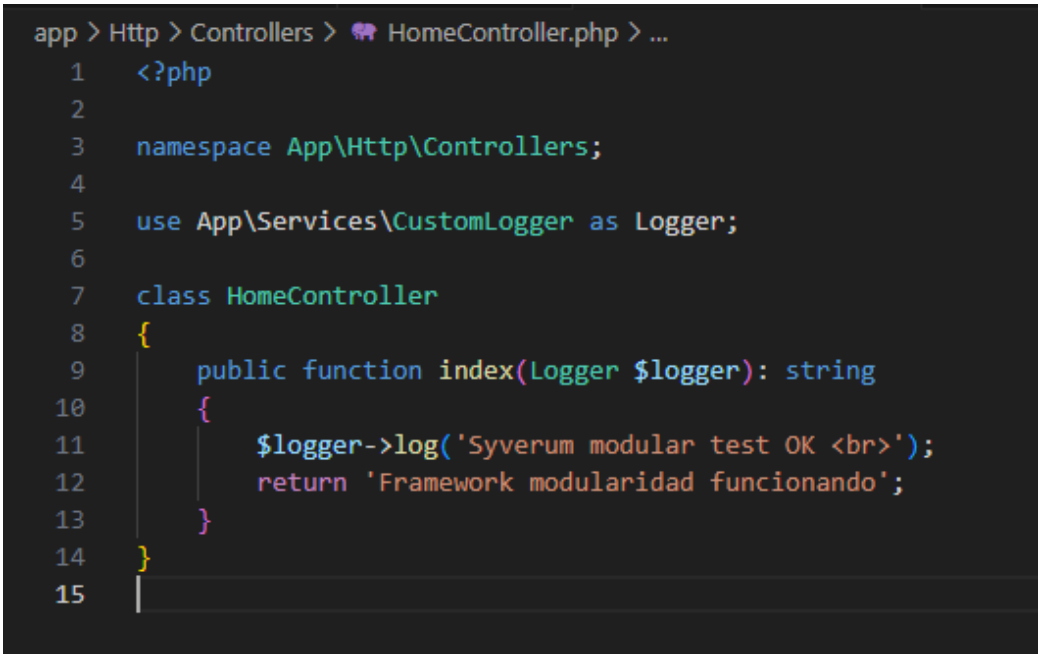
Definición de la Ruta Inicial del Framework hacia el Controlador de bienvenida.



Nota. Paso 3; testeo caso de prueba 9. Captura elaborada por el autor.

Figura 44

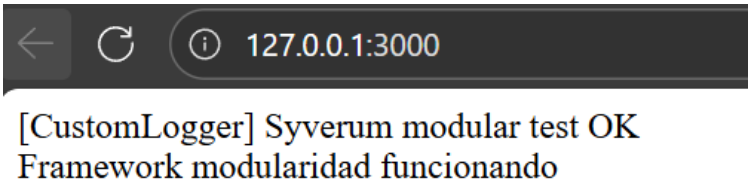
Uso del Servicio CustomLogger Dentro del HomeController.



Nota. Paso 4; testeo caso de prueba 9. Captura elaborada por el autor.

Figura 45

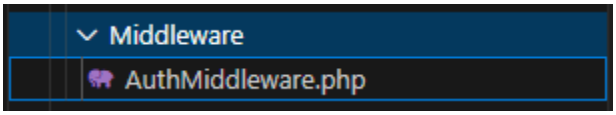
Ejecución del Framework Mostrando la Salida del CustomLogger



Nota. Paso 5; testeo caso de prueba 9. Captura elaborada por el autor.

Figura 46

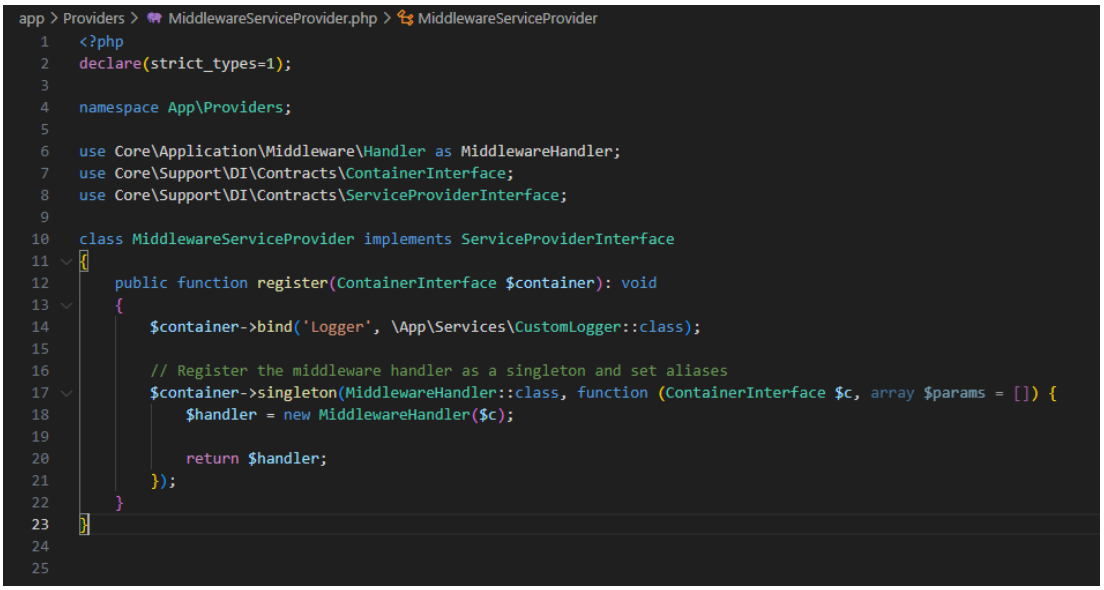
Archivo AuthMiddleware.php Presente por Defecto en el Framework.



Nota. Paso 6; testeo caso de prueba 9. Captura elaborada por el autor.

Figura 47

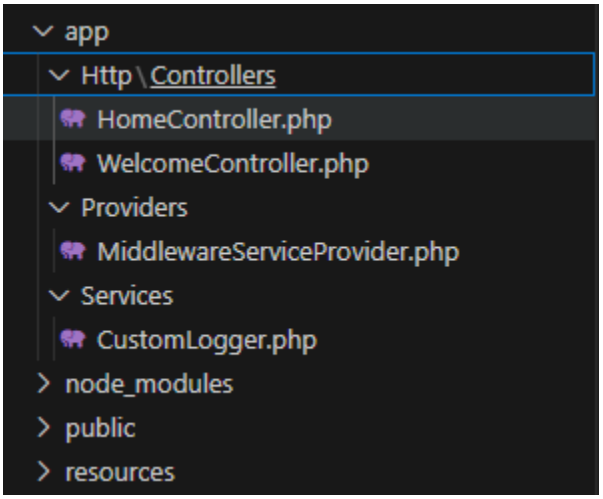
Eliminación de las Referencias a AuthMiddleware en el MiddlewareServiceProvider.



Nota. Paso 7; testeo caso de prueba 9. Captura elaborada por el autor.

Figura 48

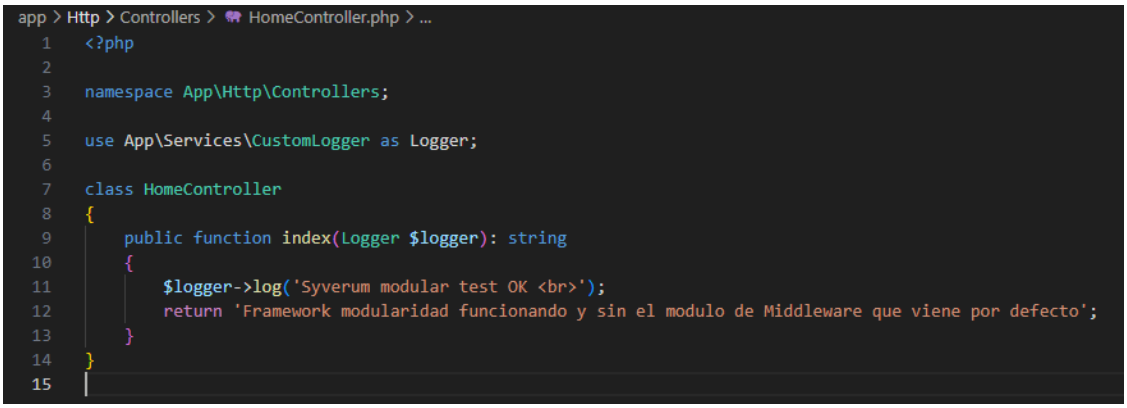
Estructura Final de la Carpeta app/ Sin el Módulo de Middleware por Defecto.



Nota. Paso 8; testeo caso de prueba 9. Captura elaborada por el autor.

Figura 49

Evidencia del Uso del CustomLogger en el Controlador Después de Eliminar el Middleware por Defecto.



Nota. Paso 9; testeo caso de prueba 9. Captura elaborada por el autor.

Figura 50

Salida en Navegador Confirmando que el Framework Funciona Sin el Middleware por Defecto.

[CustomLogger] Syverum modular test OK
Framework modularidad funcionando y sin el modulo de Middleware que viene por defecto

Nota. Paso 10; testeo caso de prueba 9. Captura elaborada por el autor.

Caso de Prueba 10 -REQ-NF03

Tabla 27

Caso de Prueba REQ-NF03: Instalación del Framework con Composer y CLI

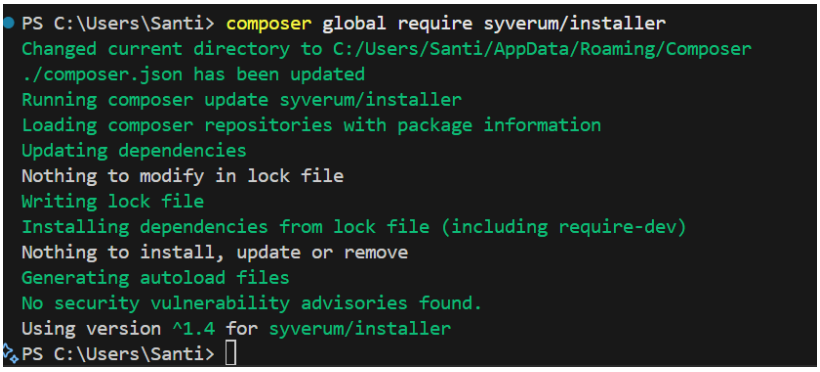
Caso de Prueba 10 -REQ-NF03	
Nombre	Instalación del framework mediante Composer y CLI
Precondiciones	Tener instalado PHP 8.x, Composer y acceso a la terminal del sistema. No debe existir previamente una carpeta con el nombre del proyecto.
Pasos de la Prueba	1. Ejecutar en la terminal el comando composer global require syverum/installer y crear un proyecto con el comando syverum new app-demo. 2. Acceder al directorio generado con cd nombre-proyecto. 3. Validar que el proyecto contiene la estructura base de Syverum (carpetas app/, public/, resources/). 4. Ejecutar el servidor embebido con composer dev.

	5. Acceder en el navegador a http://127.0.0.1:3000 y verificar que se muestra la página inicial del framework.
Resultado Esperado	La instalación se completa exitosamente con Composer, se genera la estructura de carpetas del framework y es posible iniciar el servidor con el comando CLI composer dev. La página inicial se muestra en el navegador sin errores.
Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	La instalación del framework mediante Composer se ejecutó correctamente, generando la estructura base del proyecto. El servidor se levantó con éxito y la página inicial se visualizó en el navegador, confirmando la correcta validación de los comandos CLI.

Nota. Este caso de prueba valida la facilidad de instalación del framework Syverum. A través de Composer se generó un nuevo proyecto y mediante los comandos CLI se comprobó que la ejecución es sencilla y funcional, cumpliendo con el requerimiento no funcional REQ-NF03.

Figura 51

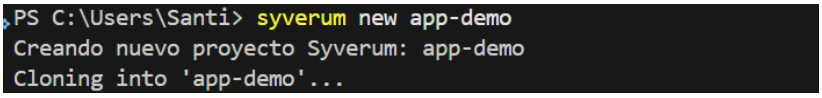
Instalación Global del Instalador de Syverum Mediante Composer.



Nota. Paso 1; testeo caso de prueba 10. Captura elaborada por el autor.

Figura 52

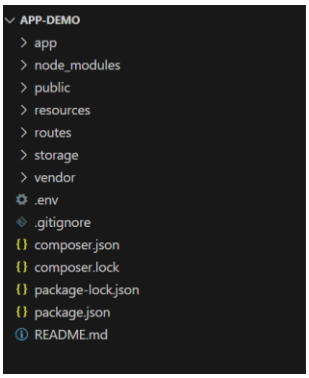
Creación de un Nuevo Proyecto con el Comando syverum new



Nota. Paso 2; testeo caso de prueba 10. Captura elaborada por el autor.

Figura 53

Estructura Base del Proyecto Generada Automáticamente.



Nota. Paso 3; testeo caso de prueba 10. Captura elaborada por el autor.

Figura 54

Ejecución del Servidor de Desarrollo Mediante composer run dev

Nota. Paso 4; testeo caso de prueba 10. Captura elaborada por el autor.

Figura 55

Página de Bienvenida del Framework Syverum Cargada Correctamente en el Navegador.

Nota. Paso 5; testeo caso de prueba 10. Captura elaborada por el autor.

Caso de Prueba 11 -REQ-NF04

Tabla 28

Caso de Prueba REQ-NF04: Mantenibilidad del Código con Principios SOLID y Desacoplamiento Arquitectónico

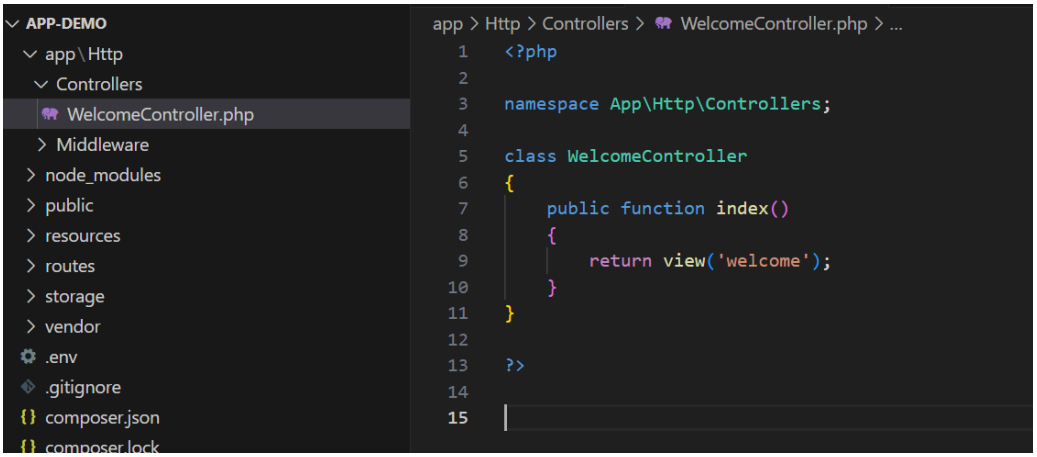
Caso de Prueba 11 -REQ-NF04	
Nombre	Mantenibilidad del código bajo principios SOLID
Precondiciones	Proyecto inicializado con Syverum. El código fuente debe estar organizado en capas (app/, Http/, Providers/).
Pasos de la Prueba	1. Revisar la implementación de clases y comprobar que cada una cumple con el Principio de Responsabilidad Única (ej. HomeController solo gestiona peticiones HTTP). 2. Validar la inyección de dependencias mediante el contenedor IoC, evitando instanciación directa de servicios. 3. Confirmar la existencia de interfaces para contratos (ej. ContainerInterface, ServiceProviderInterface) y su uso en lugar de implementaciones rígidas. 4. Verificar la separación entre capas de aplicación, dominio e infraestructura en la arquitectura del framework. 5. Ejecutar una auditoría de código (manual o con

	herramientas estáticas) para comprobar cumplimiento básico de SOLID.
Resultado Esperado	La arquitectura muestra un código desacoplado, fácil de mantener y extender. Se cumple con principios SOLID: responsabilidad única, inyección de dependencias, programación contra interfaces y separación de capas.
Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	El framework demostró mantener un diseño desacoplado y mantenible. Los controladores gestionan únicamente la capa HTTP, los servicios se inyectan por IoC, y existen interfaces para contratos. La revisión arquitectónica evidenció que el sistema cumple con los principios SOLID, garantizando bajo costo de cambios y facilidad de evolución.

Nota. Este caso de prueba valida que el framework Syverum está diseñado siguiendo principios SOLID y prácticas de desacoplamiento arquitectónico, asegurando la mantenibilidad y evolución futura del sistema.

Figura 56

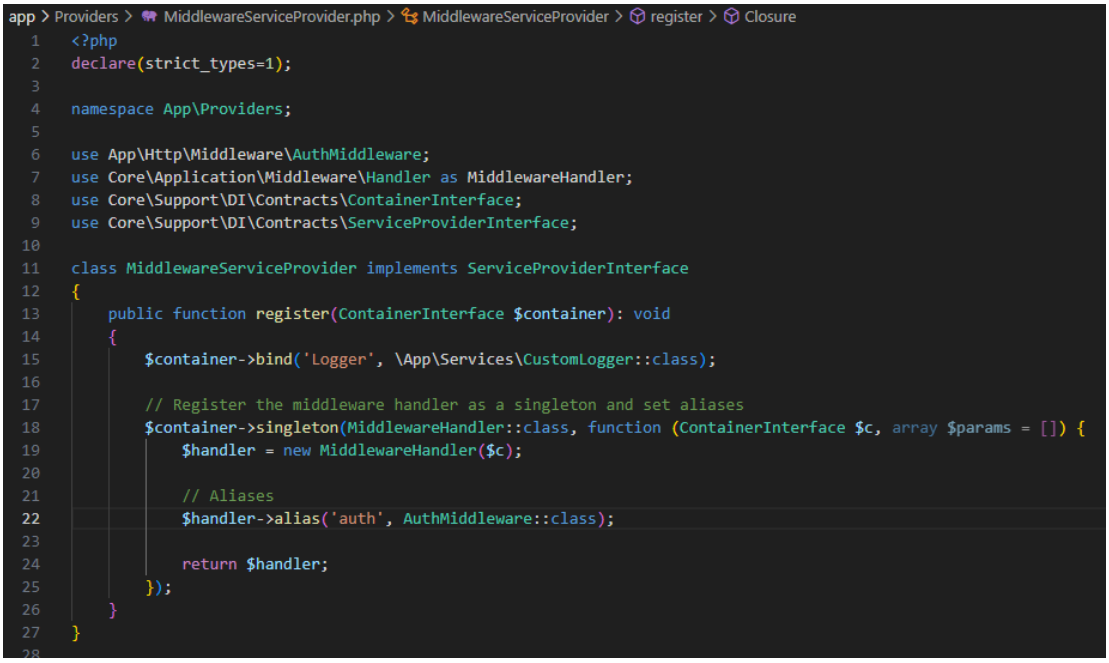
Controlador WelcomeController Aplicando el Principio de Responsabilidad Única (SRP).



Nota. Paso 1; testeo caso de prueba 11. Captura elaborada por el autor.

Figura 57

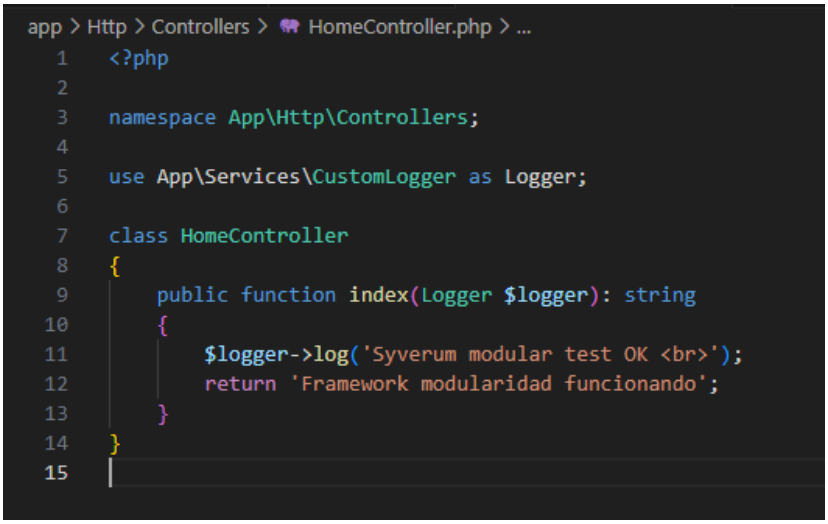
Uso de ServiceProviderInterface para Registrar Dependencias y Mantener el Desacoplamiento.



Nota. Paso 2; testeo caso de prueba 11. Captura elaborada por el autor

Figura 58

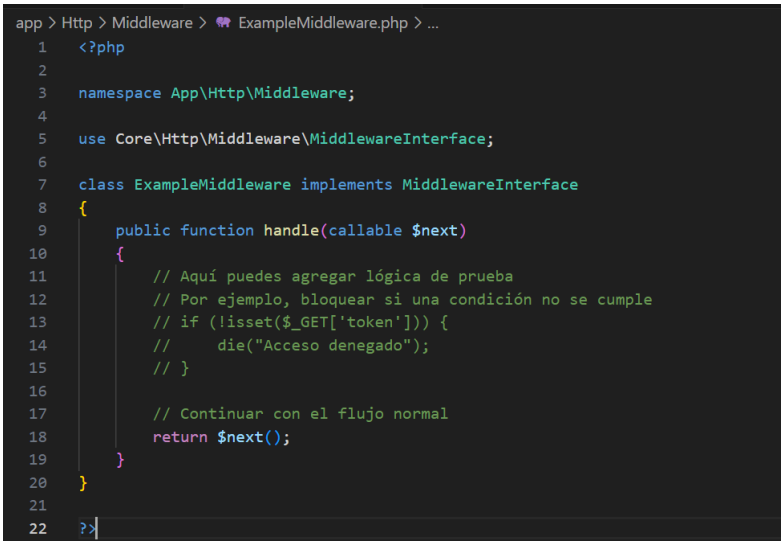
Inyección del Servicio CustomLogger en el HomeController, Demostrando el Uso de IoC.



Nota. Paso 3; testeo caso de prueba 11. Captura elaborada por el autor.

Figura 59

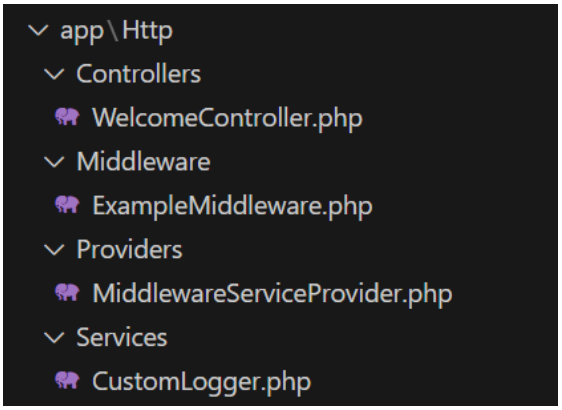
Ejemplo de Middleware Implementando MiddlewareInterface Como Evidencia de Programación Contra Interfaces.



Nota. Paso 4; testeo caso de prueba 11. Captura elaborada por el autor.

Figura 60

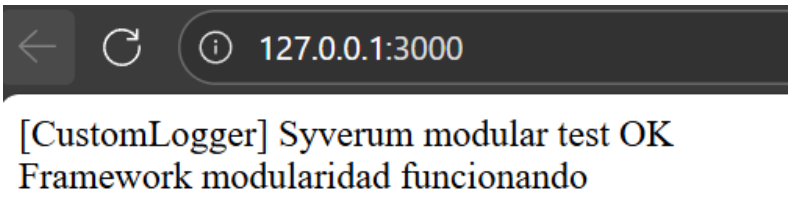
Estructura Modular del Proyecto Syverum con Separación Clara de Capas (Controllers, Middleware, Providers, Services).



Nota. Paso 5; testeo caso de prueba 11. Captura elaborada por el autor.

Figura 61

Ejecución del Framework Mostrando la Salida del CustomLogger, Validando la Correcta Integración Bajo Principios SOLID.



Nota. Paso 6; testeo caso de prueba 11. Captura elaborada por el autor.

Caso de Prueba 12 -REQ-NF05

Tabla 29

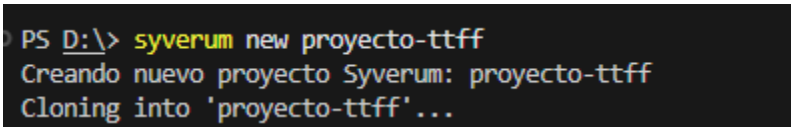
Caso de Prueba REQ-NF05: Tiempo Hasta Primera Funcionalidad (TTFF)

Caso de Prueba 11 -REQ-NF05	
Nombre	Tiempo hasta primera funcionalidad (TTFF)
Precondiciones	Instalador (Syverum/installer) y Composer configurados.
Pasos de la Prueba	1. Ejecutar syverum new proyecto-ttff 2. Correr composer run dev y abrir la vista inicial en navegador.
Resultado Esperado	En menos de 5 minutos se visualiza una respuesta HTTP con vista básica.
Fecha de Inicio de la Prueba	29/09/2025
Fecha Finalización de la Prueba	29/09/2025
Resultado Obtenido	El proyecto fue generado exitosamente y la vista inicial se cargó en el navegador en un tiempo inferior a 5 minutos, cumpliendo con el criterio de Tiempo hasta Primera Funcionalidad (TTFF).

Nota. Esta tabla documenta los resultados obtenidos al evaluar el tiempo requerido para visualizar la primera funcionalidad básica de un proyecto generado con Syverum Framework.

Figura 62

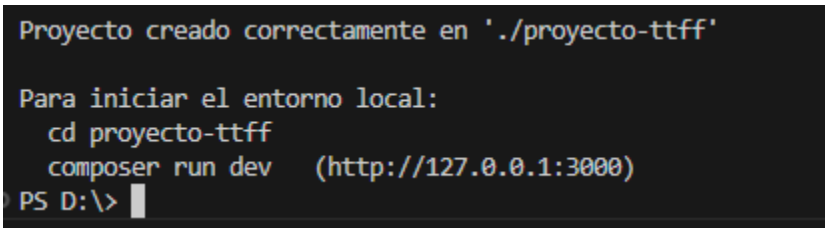
Creación del Proyecto Syverum Denominado Proyecto-ttff



Nota. Captura de pantalla propia de la ejecución del comando syverum new proyecto-ttff, donde se inicia el proceso de clonación del skeleton (2025).

Figura 63

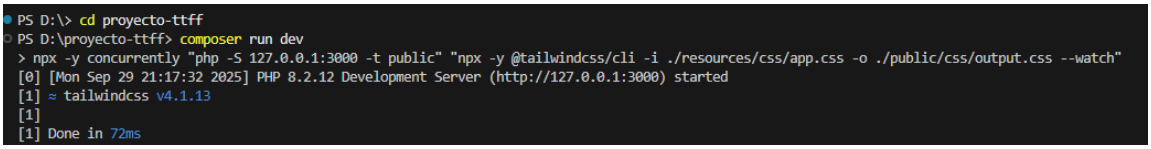
Confirmación de Ccreación del Proyecto y Comandos de Inicio



Nota. Captura de pantalla propia que muestra la finalización de la instalación del proyecto y las instrucciones para ejecutar el servidor local (2025).

Figura 64

Ejecución del Servidor de Desarrollo Mediante composer run dev



Nota. Captura de pantalla propia que evidencia la correcta inicialización del servidor PHP con TailwindCSS, listo para mostrar la vista inicial del framework (2025).

Figura 65

Vista de Bienvenida Generada Automáticamente por Syverum

Nota. Captura elaborada por el autor.

Despliegue e Inicialización del Proyecto con Syverum

En el caso de un framework como Syverum, el despliegue no se limita a un único procedimiento técnico, ya que este puede variar dependiendo del tipo de servidor o infraestructura utilizada (por ejemplo, un hosting compartido, un servidor dedicado o un contenedor). Por esta razón, en este apartado se plantea el despliegue más como el proceso de inicialización de un proyecto y los aspectos mínimos que deben considerarse para que la aplicación funcione de manera correcta.

Condiciones Previas y Requisitos Técnicos

Antes de iniciar un proyecto en Syverum es necesario garantizar que el entorno cumpla con ciertas condiciones básicas:

Versión de PHP: el framework requiere PHP 8.x o superior para aprovechar las características modernas del lenguaje.

Composer: gestor de dependencias que permite instalar Syverum y las librerías adicionales necesarias para el proyecto.

Node.js: requerido únicamente en proyectos que incluyan tareas de compilación o procesamiento de recursos de frontend (CSS, JS o assets).

Servidor HTTP: puede utilizarse Apache, Nginx, PHP-FPM o el servidor embebido de PHP para manejar las solicitudes entrantes.

Estas condiciones conforman los requisitos mínimos para la ejecución de proyectos desarrollados con Syverum. En un entorno productivo podrían sumarse otros componentes como sistemas de cache, balanceadores de carga o la orquestación mediante contenedores, pero en el marco de esta investigación se consideran suficientes los elementos anteriores.

Proceso de Instalación e Inicialización

Este es el flujo que un desarrollador seguirá para crear, configurar y lanzar un proyecto con Syverum:

Tabla 30

Proceso de Instalación e Inicialización con Syverum (Paso a Paso)

Paso	Acción	Descripción / Propósito
1	Instalar el instalador global	composer global require syverum/installer — Esto agrega un comando útil para generar nuevos proyectos Syverum de forma rápida.
2	Crear un nuevo proyecto	Ejecutar syverum new nombre-proyecto genera la estructura inicial del proyecto con todos los directorios y archivos base (controladores, rutas, vistas, configuración).
3	Ingresar al proyecto	Acceder al directorio del proyecto recién creado: cd nombre-proyecto
4	Ejecutar en desarrollo	Usar composer run dev para iniciar el entorno de desarrollo que sirve la aplicación localmente.
5	Configurar entorno	Modificar el archivo .env para ajustar variables como conexión a base de datos, claves secretas, URLs de servicios externos, etc.
6	Apuntar el servidor al directorio público	En producción, configurar el *document root* del servidor web al directorio public/, que contiene el punto de entrada index.php.
7	Instalar dependencias de producción	Ejecutar composer install --no-dev para instalar solo lo necesario para producción, sin dependencias de desarrollo.

Nota. Esta tabla describe el flujo recomendado para crear, configurar y poner en marcha un proyecto con Syverum

Trabajos Futuros

El desarrollo de Syverum ha demostrado la viabilidad de un framework liviano y modular en PHP, orientado a proyectos pequeños y medianos. No obstante, existen múltiples oportunidades para ampliar sus capacidades, fortalecer su arquitectura y fomentar su adopción en entornos reales. Este capítulo presenta las principales líneas de trabajo propuestas para futuras versiones del framework.

Ampliación de Funcionalidades del Framework

Entre las proyecciones de mejora para Syverum se plantea la ampliación de sus funcionalidades mediante nuevos módulos y herramientas. En primer lugar, se propone el soporte oficial para APIs REST y GraphQL, integrando librerías que faciliten la definición de contratos de API, el versionamiento y la generación automática de documentación con estándares como OpenAPI o Swagger. Esto ampliaría el alcance del framework en aplicaciones orientadas a servicios. También se contempla un módulo de autenticación y autorización que gestione el acceso mediante middleware nativo, con soporte para OAuth 2.0 y JWT, garantizando una protección robusta de rutas y una administración escalable de roles y permisos. Finalmente, se proyecta una CLI más avanzada, con comandos para generar migraciones, scaffolding, pruebas automatizadas y otras tareas del ciclo de desarrollo, fortaleciendo la experiencia del desarrollador y la adopción del framework.

Integración con Nuevas Tecnologías

Como parte de las mejoras proyectadas, se propone la integración de herramientas como Docker y Kubernetes para facilitar la creación de entornos replicables, portables y escalables. Esta incorporación permitiría a los desarrolladores desplegar instancias del framework en

cuestión de segundos, estandarizar configuraciones y preparar Syverum para entornos de producción distribuidos.

Asimismo, se contempla la implementación de integración continua y despliegue continuo (CI/CD) mediante pipelines automatizados con tecnologías como GitHub Actions o GitLab CI. Este enfoque posibilitaría la ejecución automática de pruebas, la validación de cambios y el despliegue de nuevas versiones del framework de forma segura y eficiente, contribuyendo al mantenimiento de la calidad y la estabilidad del software en proyectos colaborativos.

Por último, se plantea la compatibilidad con arquitecturas de microservicios, explorando la descomposición de aplicaciones en servicios autónomos que se comuniquen mediante sistemas de mensajería como RabbitMQ o Apache Kafka. Esta orientación permitiría que Syverum se utilice como base para la construcción de microservicios independientes, manteniendo la cohesión del dominio y garantizando la interoperabilidad entre componentes.

Fortalecimiento de la Arquitectura

La arquitectura de Syverum, fundamentada en principios de diseño como la separación de responsabilidades, la inversión de dependencias y la interoperabilidad, proporciona una base sólida para el desarrollo de aplicaciones web modulares y mantenibles. Sin embargo, se identifican oportunidades para fortalecer su estructura interna y ampliar su capacidad de adaptación a entornos más complejos y exigentes.

En este sentido, se plantea la incorporación de patrones arquitectónicos avanzados, como *Command Query Responsibility Segregation* (CQRS) y *Event Sourcing*, que resultan especialmente útiles en aplicaciones que demandan alta concurrencia, trazabilidad de eventos y una separación clara entre las operaciones de lectura y escritura. La adopción de estos enfoques

permitiría que Syverum evolucione hacia arquitecturas más robustas y escalables, sin comprometer la claridad ni la consistencia del dominio.

Asimismo, se propone la estandarización de adaptadores mediante la elaboración de guías formales que orienten el desarrollo de interfaces externas, tales como adaptadores HTTP, CLI o de colas de mensajería. Estas directrices garantizarían la coherencia estructural del framework, facilitando su extensión sin afectar el núcleo del sistema y asegurando la compatibilidad y mantenibilidad del código a largo plazo.

Pruebas Avanzadas y Validación externa

Con el fin de consolidar la confiabilidad y robustez de Syverum, se plantea la realización de pruebas avanzadas y procesos de validación en entornos reales. Estas acciones permitirán evaluar el comportamiento del framework bajo condiciones exigentes y compararlo con soluciones consolidadas dentro del ecosistema PHP.

En primer lugar, se prevé la ejecución de pruebas de carga y estrés mediante herramientas como Apache Benchmark (ab) y wrk, con el propósito de simular escenarios de alta concurrencia y tráfico sostenido. Este tipo de evaluaciones permitirá identificar posibles cuellos de botella, validar la eficiencia del sistema y establecer límites operativos en contextos reales.

De igual manera, se propone la realización de estudios comparativos entre Syverum y frameworks ampliamente utilizados como Laravel, Symfony y Slim, considerando aspectos como el tiempo de arranque, la claridad arquitectónica, el rendimiento, la curva de aprendizaje y la extensibilidad. Estos análisis contribuirán a posicionar a Syverum dentro del ecosistema PHP y a detectar oportunidades concretas de mejora.

Finalmente, se contempla la implementación del framework en aplicaciones reales, a través de proyectos piloto desarrollados en pequeñas empresas, instituciones educativas o

comunidades tecnológicas. Estas experiencias permitirán evaluar su comportamiento en condiciones reales de uso, recopilar retroalimentación directa de los usuarios y validar su aplicabilidad en distintos contextos de desarrollo.

Usabilidad y Comunidad

Uno de los objetivos estratégicos de Syverum es consolidarse como una herramienta útil, accesible y colaborativa tanto para desarrolladores en formación como para profesionales del área. Para alcanzar este propósito, se plantean diversas acciones orientadas a mejorar la usabilidad del framework y a fomentar el crecimiento de una comunidad activa en torno a su desarrollo.

En primer lugar, se propone la creación y publicación de una documentación pública y colaborativa, disponible en un portal accesible e inspirado en modelos exitosos como *Laravel Docs*. Esta documentación incluirá guías de instalación, ejemplos de uso, referencias de API y recomendaciones de buenas prácticas, además de permitir la colaboración abierta mediante *pull requests* y sugerencias directas de la comunidad.

Asimismo, se proyecta el desarrollo de ejemplos y casos de uso prácticos que faciliten la adopción del framework por parte de nuevos usuarios. Entre ellos, se consideran plantillas y proyectos de referencia orientados a la creación de blogs, tiendas en línea básicas, APIs educativas y sistemas de gestión, los cuales servirán como punto de partida y demostración de las capacidades técnicas de Syverum.

Finalmente, se plantea el fomento de una comunidad *open source* activa y participativa, promoviendo la colaboración a través de repositorios públicos en GitHub. Este enfoque permitirá que los desarrolladores externos puedan reportar errores (*issues*), proponer mejoras (*pull*

requests) y participar en discusiones técnicas. De esta manera, se busca fortalecer la legitimidad de Syverum como un proyecto de código abierto y enriquecer su evolución colectiva.

Conclusiones

Objetivo 1 — Levantamiento de requerimientos funcionales y no funcionales

Se consolidó un catálogo de requerimientos funcionales (REQ-F01…REQ-F07) y no funcionales (REQ-NF01…REQ-NF05), tomando como base necesidades recurrentes en el desarrollo web y referentes del ecosistema PHP. Este levantamiento permitió definir el alcance del framework y alinear las decisiones de diseño con criterios explícitos de utilidad, mantenibilidad y extensibilidad. La trazabilidad de los requisitos se integró en el índice y desarrollo del documento, estableciendo los criterios de verificación utilizados posteriormente en las pruebas.

Objetivo 2 — Diseño de la arquitectura con separación de núcleo, interfaces y adaptadores

Se modeló una arquitectura por capas (Domain / Application / Infrastructure) que respeta la Regla de Dependencias y desacopla el núcleo de negocio de los detalles de infraestructura y de las interfaces de entrada/salida. Esta estructura facilita la sustitución y extensión de componentes sin comprometer el dominio, y garantiza interoperabilidad mediante el uso de estándares PSR y un contenedor de inversión de control (DI/IoC). La documentación incluye los estilos arquitectónicos adoptados, diagramas de flujo de ejecución y arquitectura general, demostrando que la separación propuesta es coherente y auditable.

Objetivo 3 — Construcción del framework con interfaz MVC para modularidad y escalabilidad

El prototipo implementa una interfaz MVC clara (rutas, controladores, vistas), complementada por una CLI mínima para scaffolding, un esqueleto de proyecto y un contenedor de servicios. Esta estructura reduce el tiempo de desarrollo inicial (time-to-first-feature), manteniendo orden y capacidad de extensión. La propuesta se posiciona como una alternativa

pragmática entre micro-frameworks y soluciones full-stack, al combinar un núcleo reducido con módulos desacoplados y convenciones comprensibles para cualquier proyecto.

Objetivo 4 — Ejecución de pruebas unitarias, funcionales y de seguridad para validar la solución

Se diseñaron y ejecutaron casos de prueba vinculados a los requisitos definidos, incluyendo pruebas sobre el uso de la CLI, inyección de dependencias, configuración mediante archivos .env, y validaciones de modularidad y extensibilidad. Estas pruebas aportaron evidencia del funcionamiento correcto del sistema y permitieron verificar los criterios de calidad establecidos. Aunque las pruebas se realizaron en entornos controlados y no incluyeron benchmarks de alta concurrencia —limitación reconocida—, los resultados confirman la estabilidad del núcleo y la coherencia del modelo de extensibilidad. Se deja abierta la posibilidad de continuar con pruebas de carga y validaciones en escenarios de producción.

Referencias Bibliográficas

Alidoosti, R., Lago, P., Razavian, M., & Tang, A. (2022). *Ethics in software engineering: A systematic literature review*. [Journal/Publisher].

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

Brown, T. (2009). *Change by design: How design thinking creates new alternatives for business and society*. Harper Business.

Composer. (s. f.). *Introduction*. https://getcomposer.org/doc/00-intro.md

Cockburn, A. (2005). *Hexagonal architecture (Ports & Adapters)*. https://alistair.cockburn.us/hexagonal-architecture

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

Domain Language. (2015). *Domain-driven design reference* [PDF]. https://www.domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf

Evans, E. (2015). *Domain-driven design reference* [PDF]. 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*. https://martinfowler.com/articles/injection.html

Grupo Castilla. (2024, noviembre 20). *Licencia open source (código abierto): Tipos y ventajas*. https://www.grupocastilla.es/licencia-open-source/

Hernández, R., Fernández, C., & Baptista, P. (2020). *Metodología de la investigación científica* (6ª ed.). ESUP. https://www.esup.edu.pe/wp-content/uploads/2020/12/2.%20Hernandez,%20Fernandez%20y%20Baptista-Metodolog%C3%ADa%20Investigacion%20Cientifica%206ta%20ed.pdf

International Organization for Standardization, International Electrotechnical Commission, & Institute of Electrical and Electronics Engineers. (2018). *ISO/IEC/IEEE 29148:2018 — Systems and software engineering — Life cycle processes — Requirements engineering*. https://www.iso.org/standard/72089.html

International Organization for Standardization & International Electrotechnical Commission. (2011). *ISO/IEC 25010:2011 — Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. https://www.iso.org/standard/35733.html

Jiménez Beltrán, J. H. (2015). *Software libre para la gestión de peticiones, quejas, reclamos y felicitaciones con énfasis en la cocreación* [Tesis de maestría, Universidad Autónoma de Bucaramanga]. http://hdl.handle.net/20.500.12749/3388

Laravel. (s. f.). *Blade templates; Service container; Artisan*. https://laravel.com/docs/12.x/blade

Martin, R. C. (2017). *Clean architecture*. https://georgearisty.dev/posts/clean-architecture/

Martin, R. C. (2003). *Agile software development: Principles, patterns, and practices*.

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

Node.js. (s. f.). *About the documentation*. https://nodejs.org/api/documentation.html

Ore Quiroz, H., Aldana Juárez, W., Salazar Sandoval, C., & Pantoja-Tirado, L. (2021). Benchmarking como herramienta gerencial en las empresas: Revisión bibliográfica. *Revista de Investigación Científica y Tecnológica Llamkasun, 2*(2), 54–65.

PHP-FIG. (s. f. -a). *PSR-4: Autoloader*. https://www.php-fig.org/psr/psr-4/

PHP-FIG. (s. f. -b). *PHP Standards Recommendations (PSR)*. https://www.php-fig.org/

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

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

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

Sistemas Operativos. (2025). *Licencia MIT: Propósito, comparativas y características*. https://sistemasoperativos.info/software-libre/licencia-mit/

Tailwind CSS. (s. f.). *Documentación*. https://tailwindcss.com/docs

Universidad Nacional Abierta y a Distancia – UNAD. (2023). *Norma APA 7ª edición* [E-book]. Repositorio Institucional UNAD. https://repository.unad.edu.co/handle/10596/54824

vlucas/phpdotenv. (s. f.). *Repositorio oficial*. https://github.com/vlucas/phpdotenv

Apéndices

Apéndice A

Estructura del Repositorio Syverum Framework

Repositorio: https://github.com/Santiagomoo/syverum-framework

El repositorio syverum-framework contiene el núcleo principal del framework. En la carpeta src/core/ se encuentran los componentes esenciales como el sistema de rutas, controladores, vistas y middlewares. El archivo composer.json define las dependencias y la configuración del paquete. El archivo composer.lock registra las versiones exactas instaladas, y el README.md explica qué es Syverum y cómo usarlo. En conjunto, este repositorio conforma la base sobre la cual se construyen las aplicaciones que usan el framework.

Apéndice B

Estructura del Repositorio Syverum Installer

Repositorio: https://github.com/Santiagomoo/syverum-installer

El repositorio syverum-installer contiene el instalador del framework Syverum. Este paquete permite crear un nuvo proyecto mediante el comando syverum new [nombre-del-proyecto]. Incluye los archivos necesarios para generar la estructura inicial del proyecto y copiar los archivos base del esqueleto. Además, define la configuración para su distribución como paquete de Composer, lo que facilita su instalación global. Su propósito es automatizar la creación de proyectos con una sola línea de comando.

Apéndice C

Estructura del Repositorio Syverum Skeleton

Repositorio: https://github.com/Santiagomoo/syverum-skeleton

El repositorio syverum-skeleton representa la plantilla base que se genera al crear un nuevo proyecto con Syverum. Contiene las carpetas app/Http donde se definen los controladores y middlewares, la carpeta resources para las vistas, routes con las definiciones de rutas, y public donde se alojan los archivos estáticos del sitio. Incluye además archivos como .env para la configuración del entorno, composer.json con las dependencias, y package.json para manejar los recursos front-end. En resumen, este repositorio define la estructura inicial sobre la cual el desarrollador comienza a trabajar.

Apéndice D

Documentación Oficial de Syverum

Documentación: https://santiagomoo.github.io/syverum-docs/

La documentación oficial de Syverum explica cada aspecto del framework:

Getting Started (Inicio): introducción general con sus características, público objetivo y requisitos.

Instalación: cómo instalar Syverum usando Composer y el comando del instalador.

Estructura del proyecto: explica cómo se organiza un proyecto típico.

Rutas: enseña cómo definir rutas y aplicar middlewares.

Controladores: detalla la creación de controladores con el comando syverum make:controller.

Modelos: aborda el manejo de la base de datos y las entidades.

Vistas (Blade): describe el uso del motor de plantillas incluido.

Middlewares: explica cómo interceptar solicitudes y controlar el flujo.

Base de datos: muestra cómo conectar Syverum con MySQL y trabajar con migraciones.

Inyección de dependencias (DI): describe el contenedor de servicios del framework.

FAQ / Soporte: ofrece respuestas a preguntas frecuentes y recursos de ayuda.